NonStop™ Systems

GUARDIAN™ Operating System User's Guide

Operating System Library

82396

NOTICE

Effective with the B00/E08 software release, Tandem introduced a more formal nomenclature for its software and systems.

The term "NonStop 1+™ system" refers to the combination of NonStop 1+ processors with all software that runs on them.

The term "NonStop™ systems" refers to the combination of NonStop II™ processors, NonStop TXP™ processors, or a mixture of the two, with all software that runs on them.

Some software manuals pertain to the NonStop 1+ system only, others pertain to the NonStop systems only, and still others pertain both to the NonStop 1+ system and to the NonStop systems.

The cover and title page of each manual clearly indicate the system (or systems) to which the contents of the manual pertain.

NonStop™ Systems

GUARDIAN™ Operating System User's Guide

Abstract
This manual describes the basic operating-system tasks that all users perform. Task-oriented instructions are presented for these utilities: COMINT, FUP, BACKUP, RESTORE, PERUSE, SPOOLCOM, and the spooler. This user's guide is for all users of Tandem NonStop systems.

Product Version
GUARDIAN B00

Operating System Version
GUARDIAN B00 (NonStop Systems)

Part No. 82396 A00

March 1985

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-2599

DOCUMENT HISTORY

Edition	Part Number	Operating System Version	Date
First Edition	82396 A00	GUARDIAN B00	March 1985

New editions incorporate all updates issued since the previous edition. Update packages, which are issued between editions, contain additional and replacement pages that you should merge into the most recent edition of the manual.

NEW AND CHANGED INFORMATION

With the B00 software release, the material in the old GUARDIAN Operating System Command Language and Utilities Manual has been divided into four new manuals. The material was divided according to software system and according to purpose. That is, information for NonStop systems is now separate from that for the NonStop 1+ system, and task-oriented material is now separate from general reference material. As a result, each software system now has both a reference manual and a user's guide for the GUARDIAN operating system.

Your new manuals are:

- GUARDIAN Operating System User's Guide for NonStop systems (Part No. 82396 A00)
- GUARDIAN Operating System Utilities Reference Manual for NonStop systems (Part No. 82403 A00)

Manuals for users of the NonStop 1+ system are:

- GUARDIAN Operating System User's Guide for the NonStop 1+ system (Part No. 82395 A00)
- GUARDIAN Operating System Utilities Reference Manual for the NonStop 1+ system (Part No. 82402 A00)

This GUARDIAN Operating System User's Guide contains basic task-oriented material for new users. The material that was printed on blue paper in the final edition of the Command Language and Utilities Manual (Sections 1, 2, 3, 5, 6, 7, 9, and 10) is now contained in the user's guide.

The GUARDIAN Operating System Utilities Reference Manual contains reference material for all users. This reference material includes complete syntax descriptions for the commands of each operating system utility described in this manual. Syntax is also given for several other utilities, including the Peripheral Utility Program (PUP), the Disc Space Analysis and Disc Space Compression utilities (DSAP and DCOM), DELAY, DIVER, ERROR, PEEK, and SPOOL.

System error and warning messages, which appeared in appendixes in the earlier manuals, now appear in the System Messages Manual for the NonStop 1+ system (Part No. 82408) and the System Messages Manual for NonStop systems (Part No. 82409).

SECTIONS 1 THROUGH 3, COMINT

Nine COMINT commands that perform privileged functions have been converted into separate programs. In most cases, each program retains the same name and uses the same syntax as the previously used command. The programs also perform the same functions as the commands that they replace.

For example, the new ADDUSER program now replaces the ADDUSER command in COMINT. To start an ADDUSER process, you can enter an ADDUSER command using the original command syntax.

Four of the new programs appear in this user's guide. They are:

ADDUSER	DEFAULT
PASSWORD	USERS

The programs that do not appear in this manual are BUSCMD, DELUSER, RCVDUMP, RELOAD, and RPASSWRD. For information about all nine new COMINT programs, see the GUARDIAN Operating System Utilities Reference Manual.

In addition, minor manual revisions, including reorganization of some material, have been made in these sections. Many examples have been expanded, and new examples have been added.

SECTIONS 4 THROUGH 6, FUP

Subsections containing introductory information on files, file names, and types of disc files have been added.

SECTIONS 7 AND 8, BACKUP AND RESTORE

Only minor changes have been made to these two sections. Two new utilities, BACKUP2 and RESTORE2, are available to work with files in the format of the new optional disc process (DP2). Full documentation of their function and command syntax is in the GUARDIAN Operating System Utilities Reference Manual.

SECTION 9, Spooler

There is a new spooler option for the header message that is specially designed for batch processing.

SECTION 10, PERUSE

You can now enter multiple PERUSE commands on the same line, separated by semicolons. The maximum length of the command line is 132 characters.

The description of PERUSE operations with TGAL has been expanded to include procedures for finding TGAL errors and for printing out portions of a job.

SECTION 11, SPOOLCOM

This section has been expanded to include SPOOLCOM tasks that all users can perform: obtaining the status of spooler components, changing the attributes of your own job, and bringing a device back online that is not ready or is out of paper.

SUMMARY OF CHANGES TO SPOOLER DOCUMENTATION

With the B00 software release, the information formerly in the two spooler manuals has been subdivided by audience (general users, system operators and managers, and application programmers) as well as by purpose (task-oriented information or reference information).

For spooler information specific to your system, see the following new or revised manuals:

Manual	Information
GUARDIAN Operating System User's Guide (Part No. 82396)	Introduction to the spooler. How to use SPOOLCOM and PERUSE.
GUARDIAN Operating System Utilities Reference Manual (Part No. 82403)	SPOOLCOM and PERUSE syntax, considerations, examples
System Operator's Guide (Part No. 82401)	Information that system operators and managers need to start, stop, and control the spooler
System Procedure Calls Reference Manual (Part No. 82359)	Complete syntax and considerations for spooler interface, utility, and print procedures
Spooler Programmer's Guide (Part No. 82394)	Information to help programmers control the spooler from application programs

CONTENTS

FIGURES

TABLES

PREFACE

This first edition of the GUARDIAN Operating System User's Guide presents task-oriented material for the principal utilities associated with the GUARDIAN operating system. As an introductory manual, this guide gets you started on the Tandem system and demonstrates basic operating system functions that every user needs to know. You are shown here all the most commonly used tasks, such as logging on to the system through COMINT, using FUP to duplicate or move files, using PERUSE to check the status of jobs you have sent to the spooler, and using BACKUP to make backups of files.

Because Tandem has two software systems that will now be documented separately, two versions of this user's guide are available. You have Part No. 82396 for your NonStop system. Users of the NonStop 1+ system should use Part No. 82395.

Before reading this user's guide, you should read the following manual as background:

Introduction to Tandem Computer Systems (Part No. 82503)

This user's guide is intended for all system users, including system operators, system and group managers, and application programmers. Each section of the manual presents tasks you can perform with a particular utility. You can read the sections in any order and skip sections that do not contain information you need. However, when you read a section, you should read or skim the entire section from beginning to end. Many examples use files or a set of conditions created earlier in the section. You will learn more if you have performed the earlier tasks.

This guide contains much of the task-oriented material from its predecessor, the GUARDIAN Operating System Command Language and Utilities Manual. Drawn from this source are the sections on COMINT, FUP, and BACKUP/RESTORE, as well as the section on security features of Tandem systems.

Also added is task-oriented material on the spooler and PERUSE as well as SPOOLCOM for ordinary users from the former System Operations Manual. Task-oriented material on SPOOLCOM for system operators, however, is in the new System Operator's Guide (Part No. 82401). This new guide, which replaces the System Operations Manual, contains system information and descriptions of tasks that system operators normally perform.

Complete syntax information for all the programs and commands described in this user's guide can be found in the GUARDIAN Operating System Utilities Reference Manual (Part No. 82403).

Programming considerations are discussed in the following manuals:

GUARDIAN Operating System Programmer's Guide (Part No. 82357)

System Procedure Calls Reference Manual (Part No. 82359)

Spooler Programmer's Guide (Part No. 82394)

With this release, the GUARDIAN Operating System Command Language and Utilities Manual (Part No. 82073) becomes obsolete.

SYNTAX CONVENTIONS USED IN THIS MANUAL

The following list defines the conventions for syntax notation used in the syntax summaries that comprise the appendixes for this manual.

Notation	Meaning
UPPERCASE LETTERS	Represent keywords and reserved words; you must enter these items exactly as shown.
<lowercase letters>	Angle brackets around lowercase letters represent variables that you must supply.
Brackets []	Enclose optional syntax items. A vertically aligned group of items enclosed in brackets represents a list of selections from which you may choose one or none.
Braces {}	Enclose required syntax items. A vertically aligned group of items enclosed in braces represents a list of selections from which you must choose only one.
Vertical bar |	Separates selections that appear on one line. A horizontally aligned group of items separated by vertical bars and enclosed by brackets or braces represents a list of optional or required syntax items from which you choose one item.
Ellipsis ...	Immediately following a pair of brackets or braces indicates that you can repeat the enclosed syntax items any number of times.
Percent Sign %	Precedes a number in octal notation.
Spaces	May be required or optional: If two syntax items are separated by a space, that space is required between the items. If one of the items is a punctuation symbol, such as a parenthesis or a comma, spaces are optional.
Punctuation (),;!.	Symbols or punctuation not described above must be entered precisely as shown. Any punctuation inside quotation marks must be entered as shown.
RETURN	Indicates pressing the RETURN key.

SECTION 1

INTRODUCTION TO COMINT

COMINT, the command interpreter for the GUARDIAN operating system, is the primary user interface to Tandem computer systems. Through COMINT, you can:

- Gain access to the system
- Use system utilities
- Start processes (that is, run programs)

WHO USES COMINT?

COMINT is used by everyone who needs direct access to the GUARDIAN operating system and its capabilities. These users include:

- Application programmers
- System programmers
- System managers
- System operators

To prevent unauthorized access to the system, COMINT is normally made inaccessible to users of application programs.

HOW DOES COMINT START?

COMINT is the name of a program file in the system subvolume ($SYSTEM.SYSTEM or $SYSTEM.SYS<nn>, where <nn> is an octal integer). From that file, many individual COMINT processes can be started. At system generation, the system manager or operator can start a COMINT process (or "run a COMINT program") for each terminal in the system. Then the user or users assigned to each terminal have only to log on to the COMINT process in order to access the system.

HOW IS COMINT USED?

COMINT is most often used interactively. In this case, a COMINT process controls a terminal connected to the system. After you log on, you can enter commands at the terminal keyboard. COMINT then performs the actions you request or invokes the program you name in your command. If you work on a network, you can start a COMINT process on another system, as described in Section 3.

Besides this interactive mode, COMINT can accept commands from a command file or an input file. You can use command files (or OBEY files) as described in Section 2; you can also use other devices for input files, as described in the COMINT command syntax description in the GUARDIAN Operating System Utilities Reference Manual.

Application programs can also use COMINT. See the GUARDIAN Operating System Programmer's Guide for more information.

SECTION 2

BASIC USES OF COMINT

This section contains information for new users of COMINT, the GUARDIAN operating system command interpreter. It describes how to:

- Enter COMINT commands
- Start and end a session with COMINT
- Set default values for volumes and file security
- Set and change your password
- Control processes
- Perform simple disc file operations
- Execute COMINT commands from a file
- Correct command errors

Selected COMINT commands are described in this section. For complete syntax and reference information on all COMINT commands and programs, see the GUARDIAN Operating System Utilities Reference Manual.

HOW TO ENTER COMINT COMMANDS

You can enter interactive COMINT commands only when a colon prompt (:) appears on the terminal screen. The colon prompt indicates that the COMINT process is ready to accept a command. Type your command after the prompt, then press the RETURN key:

```
:WHO <RETURN>
```

You must end every command by pressing the RETURN key. In examples in this manual, a RETURN is assumed at the end of each command line.

A COMINT command can contain up to 132 characters on one line. You can enter longer commands (up to 528 characters) by ending each line of the command with an ampersand character (&). COMINT then redisplays its colon prompt, and you can enter the rest of your command. For example, these two commands are equivalent:

```
:TAL / IN $MANUF.MYSUB.MYSRCE, OUT $LP / $MANUF.MYSUB.MYOBJ

:TAL / IN $MANUF.MYSUB.MYSRCE, OUT $LP / $MAN&
:UF.MYSUB.MYOBJ
```

GETTING STARTED

To get started as a new user, you need to know how to start a session with COMINT, how to select or change your logon password, and how to end a session with COMINT. The next three subsections describe these procedures.

Logging On (LOGON Command)

To gain access to the system, you must "log on." To do this, you must have a user name and user identification, which are normally assigned by a system manager.

User names have two parts. The first part of your user name is the name of your group; the second part is the name assigned to you within that group. Group and user names are separated by a period. For example, this is the user name assigned to Fred in the manufacturing group:

```
MANUF.FRED
```

In addition to a user name, each user can have a logon password. Your logon password is a string of characters that you must enter in order to gain access to the system. On most systems, you can select and change your own password, as described in the subsection "Changing Your Password."

Suppose MGBGT is Fred's password. To log on, Fred enters:

```
:LOGON MANUF.FRED, MGBGT
```

A comma must separate the user name and the password. COMINT ignores any spaces that occur immediately before or after the comma.
If FRED does not have a password, he can log on by typing LOGON followed by his user name, like this:

```
:LOGON MANUF.FRED
```

Using the Blind Password and Blind Logon Features

For security reasons, you might not want your user name or your password to appear on the terminal screen when you log on. Using the "blind password" and "blind logon" features, you can log on without displaying your password (blind password) or your user name and password (blind logon).

To log on using blind password, enter LOGON, your user name, and a comma, like this:

```
:LOGON MANUF.FRED,
```

This prompt appears:

```
PASSWORD:
```

Type your password. The characters you type are read by the operating system, but they are not visible on the terminal screen. Press the RETURN key, and your logon is complete.

If you make a mistake typing your user name or your password, this message appears:

```
INCORRECT PASSWORD OR USER NAME NOT FOUND
```

If you get this message, retype your password and press RETURN.

To log on using blind logon, enter LOGON and a RETURN:

```
:LOGON
```

A question mark prompt appears:

```
?
```

Now you can enter your user name and password (if you have one). The characters you enter after the question mark do not appear on the terminal screen.

Using User IDs

Like your user name, your user identification (or user ID) also has two parts:

- Your <group-id>, a number between 1 and 255 that uniquely identifies your group
- Your <user-id>, a number between 1 and 255 that uniquely identifies each user within a group

The angle brackets (< >) around these two terms indicate that these are syntax variables that you can use in commands (as described in the GUARDIAN Operating System Utilities Reference Manual). This form is used in this manual so that you can become accustomed to reading syntax terms.

If the <group-id> for the manufacturing group is 8, and Fred's <user-id> is 44, Fred's user ID is:

```
8,44
```

You use user IDs with the USERS and STATUS commands, as shown later in this section. User IDs are used by the GUARDIAN operating system for several types of authorization checking. See Section 12 for more information about user IDs.

Changing Your Password (PASSWORD Program)

You can select or change your password by entering a PASSWORD command containing your new password. A password can contain from one to eight letters, numbers, or other characters, with no blanks.

For example, after Fred logs on, he can change his password with the PASSWORD program:

```
:LOGON MANUF.FRED, MGBGT
:PASSWORD mozart
```

Now "mozart" (all lowercase) is Fred's new password. Note that COMINT distinguishes uppercase letters from lowercase letters in passwords. Therefore, each time Fred logs on, he must enter his password exactly as he did in his PASSWORD command.

To delete your password, enter a PASSWORD command without including a new password:

```
:PASSWORD
```

CAUTION

If you do not have a password, anyone can log on to your system by entering LOGON with your user name. Using a password can prevent unauthorized access to your files.

Logging Off (LOGOFF Command)

To end your session with COMINT, enter the LOGOFF command:

```
:LOGOFF
```

While you are logged on, you can log on again with the same or a different user name without logging off first. Simply enter a new LOGON command. Of course, in order to log on as another user, you must know the correct password for the other user name.

For example, while Fred is logged on as MANUF.FRED, he can log on as MANUF.MABEL (whose password is "brahms"), like this:

```
:LOGON MANUF.MABEL, brahms
```

Logging on again in this manner logs off the original user name. In this example, MANUF.FRED is logged off when Fred logs on as MANUF.MABEL.

USING FILE NAMES

A complete disc file name in a Tandem system has three parts:

- A volume name, the name of the disc volume where the file resides
- A subvolume name, the set of related files in the same disc volume
- A file name, the name of the individual file

A volume name always begins with a dollar sign ($) and can contain from one to seven alphanumeric characters. Both the subvolume name and the file name are strings of one to eight alphanumeric characters that must begin with a letter. You can give names of your choice to your files and subvolumes, but each volume name must be the name of an actual disc volume.

You can think of the three parts of a file name as the parts of a file cabinet. A volume in the system is like the cabinet itself, a subvolume is like a drawer in the cabinet, and a file is like an individual folder in the drawer.

A complete file name is one that has all three parts: volume, subvolume, and file name. A complete file name is also known as a fully qualified file name. A partial file name is one that omits at least one part. Separate the parts of a file name with periods.

Here is an example of a complete file name:

```
$DISC1.MYSUB.NEWFILE
```

$DISC1 is the volume name, MYSUB is the subvolume name, and NEWFILE is the name of the file.

Here are three examples of different types of partial file names:

```
NEWFILE              MYSUB.NEWFILE      $DISC1.NEWFILE
```

The first example (NEWFILE) contains only the name of the file, without the volume or subvolume names. The second example contains only the subvolume (MYSUB) and file names. The last example gives the volume name ($DISC1) and the file name but omits the name of the subvolume. (When you give a partial file name, the operating system assumes that the omitted parts are the "current defaults," as described in the next subsection).

In addition, in systems that have a name, including all systems that are part of a network, a file name can include the name of the system in which it resides. Names of systems always begin with a backslash (\). If a file resides in the current default system, you can omit the system name.

Here is an example of a complete file name in system \NY:

```
\NY.$DISC1.MYSUB.NEWFILE
```

SETTING DEFAULT VOLUMES FOR DISC FILES

For each user, GUARDIAN maintains two separate sets of default values: logon defaults and current defaults. Each set of defaults includes a value for:

- System (initially, the system where you log on)
- Disc volume
- Subvolume
- File security (described in the chapter on security features)

Your logon defaults are in effect when you log on. These initial default values are your starting point in the system; they are your first current defaults. You can change your logon defaults with the DEFAULT program, as described in the next subsection.

Your current default values always define your "location" or frame of reference in the system. During a session with COMINT, you can "relocate" yourself or change your frame of reference by changing your current defaults. To do this, you use the VOLUME or SYSTEM command, as described in the subsection "Changing Your Current Defaults."

Your current defaults serve an important function. When you give a partial file name in a command, the operating system uses the current default values for the missing parts of the file name. This process of adding parts to file names is known as file-name expansion.

File-name expansion works like this: When you specify a file name in a command, you can omit the volume and subvolume names if the file resides in your current default subvolume and volume. You must include the volume and subvolume if they are not your current defaults. Similarly, if the file resides on the current default volume but in another subvolume, you must

include the subvolume name with the file name, but you can omit the volume name because it is the current default. The operating system inserts your current defaults for any parts of the file name that you omit.

For example, suppose that your current default volume is $GROOVE, and your default subvolume is SUBSAND. To purge the file $GROOVE.SUBSAND.SOURCE, you can enter:

```
:PURGE SOURCE
```

Because of file-name expansion, the operating system assumes that the complete file name is $GROOVE.SUBSAND.SOURCE.

Setting Your Logon Defaults (DEFAULT Program)

You use the DEFAULT program to set the default volume and subvolume that are in effect when you log on. DEFAULT can also be used to specify your logon default security for files you create. (For a complete discussion of file security, see Section 12.)

For example, you can change your logon default volume to $SPIN and your logon default subvolume to FRED by entering:

```
:DEFAULT $SPIN.FRED
```

After you enter a DEFAULT command, the old logon default settings remain in effect until the next time you do either of these: Log on, or enter the VOLUME command without specifying a volume or subvolume, as described in the following subsection. Then the logon defaults you specify in your DEFAULT command will be in effect each time you log on.

When you log on, the default system name is always the name of the system where you are logged on.

Changing Your Current Defaults (VOLUME and SYSTEM Commands)

The operating system uses your current defaults for volume, subvolume, and system when performing file-name expansion on partial file names. During a session with COMINT, you can change your current defaults as often as you like. Use the VOLUME command to change your current default volume, subvolume, or file security. Use the SYSTEM command to change your current default system.

For example, you can change your current default disc volume to $DISC2 by entering:

```
:VOLUME $DISC2
```

To change your current default subvolume to ANOTHER, enter:

```
:VOLUME ANOTHER
```

To change both at once, enter:

```
:VOLUME $DISC2.ANOTHER
```

After you enter this command, any file name you enter without a volume and subvolume name is assumed to be:

```
$DISC2.ANOTHER.<file-name>
```

Entering VOLUME alone restores your logon default settings (or the settings you set with the DEFAULT program if you have used DEFAULT in this session):

```
:VOLUME
```

You can check your current defaults with the WHO command, as described in the next subsection.

The SYSTEM command changes your current default system name. Just enter SYSTEM followed by the name of the system that you want to use as your default. For example, this command sets the system \SANFRAN as the current default:

```
:SYSTEM \SANFRAN
```

After you enter this command, any file name you specify without a system name is assumed to reside in system \SANFRAN. (Note, however, that the SYSTEM command does not log you onto the \SANFRAN system. To log on to a remote system, you must first start a COMINT process in that system, as described in Section 3.)

To reset the current default system name to be the system where you logged on (or the system you named in a DEFAULT command in this session), enter the SYSTEM command alone:

```
:SYSTEM
```

NOTE

Length limits affect your ability to specify default system and volume names with the SYSTEM and VOLUME commands and with the DEFAULT program. You cannot specify a new default system name if the current default volume name contains seven characters after the dollar sign ($). Also, you cannot specify a new default volume name if the current default system name contains seven characters after the backslash (\).

GETTING INFORMATION ABOUT USERS (WHO COMMAND AND USERS PROGRAM)

You can get information about your status as a user and about the status of other users. Use the WHO command to display your current defaults, and use the USERS program to display information about any user in the system.

The WHO command tells you:

- The name of your home terminal (such as $FRED)
- The name of your COMINT process (such as $C106)
- Your current default volume and subvolume (such as $GROOVE.SUBSAND)
- The CPU number of the processor where your COMINT process is running (such as 05), and the number of the CPU where your backup COMINT process, if any, is running
- Your user ID (such as 8,44)
- Your user name (such as MANUF.FRED)
- Your current default file security (such as "NUNU") (discussed further in Section 12)

For example, if you were Fred, entering the WHO command would give you this display:

```
:WHO

HOME TERMINAL: $FRED
COMMAND INTERPRETER: \TS.$C106 PRIMARY CPU: 05  BACKUP CPU: 04
CURRENT VOLUME: $GROOVE.SUBSAND
USERID: 008,044   USERNAME: MANUF.FRED      SECURITY: NUNU
```

The USERS program can give you information about a single user or a group of users on your system. For information about a single user, enter USERS followed by either the user name or the user ID of a given user.

For example, you can get information about Fred in the manufacturing group by entering either of these commands:

```
:USERS MANUF.FRED

:USERS 8,44
```

COMINT then displays information such as this:

GROUP . USER	I.D. #	SECURITY	DEFAULT VOLUMEID
MANUF.FRED	008,044	NUNU	$GROOVE.SUBSAND

This display tells you the following:

- GROUP . USER lists the user name of the user whose user ID you entered
- I.D. # lists the <group-id> and <user-id> of that user
- SECURITY gives the logon default file security setting for that user
- DEFAULT VOLUMEID lists the logon default volume and subvolume for that user

You can also get information on all users in a group by entering the name of the group followed by an asterisk (*) as the user name:

```
:USERS MANUF.*
```

For information on all users in your own group, enter USERS followed by only an asterisk:

```
:USERS *
```

CONTROLLING PROCESSES

Processes are programs that are running. COMINT allows you to start, stop, and get information about processes. These operations are described in the next three subsections.

Starting a Process (RUN Command)

Starting a process is the same as running a program. You can start processes with the RUN command of COMINT. Enter RUN followed by the name of the file in which the object program resides.

For example, you can run the object program contained in the file MYPROG in your current default volume and subvolume by entering:

```
:RUN MYPROG
```

Many important system programs reside in the system subvolume ($SYSTEM.SYSTEM or $SYSTEM.SYS<nn>, where <nn> is a two-digit octal integer; SYS<nn> is the subvolume containing the GUARDIAN operating system image currently in use.) You can start system processes by simply entering the program name. COMINT searches the system subvolume for the program file with that name and starts up a new process from that file. As an example, you can run the text editor program by entering:

```
:EDIT
```

You can control the execution of a program by including one or more options in the RUN command. With different options, you can specify:

- The name of a file containing input for the program (the IN <file-name> option)
- The name of a file where the program directs its output (the OUT <list-file> option)

- The name to be given to the process (NAME $<process-name>)
- The execution priority of the process (PRI <priority>)
- The processor where the process executes (CPU <cpu-number>)

The syntax description of the RUN command in the GUARDIAN Operating System Utilities Reference Manual gives a complete list of these options (<run-option>). In a RUN command, the run options follow the name of the program file; you separate them from each other by commas. The group of run options must begin and end with slashes (/).

For example, you can run the program $SOFT.DATA.TEST and specify a file containing input (DATA1) and a file to store the results of the program (LISTING1) by entering:

```
:RUN $SOFT.APPS.TEST / IN DATA1, OUT LISTING1 /
```

One useful run option is NOWAIT, which allows you to continue entering COMINT commands while the new process runs. If you do not include the NOWAIT option, you cannot enter more COMINT commands until the process you started has finished.

For example, to start a TGAL process that formats the text in the file $INFO.INSTRX.CHAP1 and sends its output to the spooler without making you wait for the process to complete, enter:

```
:TGAL / IN $INFO.INSTRX.CHAP1, OUT $S.#SPOOL, NOWAIT/
```

Getting Information about Processes (STATUS Command)

You can get information about processes with the STATUS command. To get information about the last process started from your terminal (excluding the original COMINT), enter the STATUS command with no parameters. If the process is still executing, you receive information about the process.

For example, suppose that the last process you started was the text editor. Here is an example of the information displayed by the STATUS command:

```
:STATUS

                                SYSTEM  \TS
PID     PRI PFR %WT USERID       MYTERM        PROGRAM FILE NAME

07,034 148      000 008,044     $FRED       $SYSTEM .SYSTEM  .EDIT
```

In this example:

- PID is the process ID assigned to the EDIT process by the operating system. (The process ID consists of the CPU number where the process is executing and the number of the process in that CPU. The process ID, or PID, is represented in syntax as <cpu>,<pin>.)
- PRI is the execution priority of the process. (The priority can be any whole number between 1 and 199; processes with higher-numbered priorities are executed first.)
- MYTERM is the name of the terminal where the process was started. (Terminal names, like process names, always start with a dollar sign.)
- PROGRAM FILE NAME is the name of the file containing the program being run.

For a complete description of the STATUS command display, see the GUARDIAN Operating System Utilities Reference Manual.

You can display information about all processes that were started from your terminal by including the asterisk and the TERM option:

```
:STATUS *, TERM
```

You can also get information about processes that were started from another terminal or processes started by another user. For example, you can get information about all the processes started from terminal $KLEAN by entering:

```
:STATUS *, TERM $KLEAN
```

With the USER option, you can display information about all processes that you as a user started from any terminal in the system:

```
:STATUS *, USER
```

You can get information about processes started by the user with user ID 1,4 by entering:

```
:STATUS *, USER 1,4
```

Stopping a Process (STOP and PAUSE Commands and BREAK Key)

A process can execute in one processor or simultaneously in two processors as a NonStop process pair. If a process you started is executing in only one processor, you can stop it by entering the STOP command followed by the process ID number.

For example, you can stop the EDIT process shown in the previous example of the STATUS command by entering:

```
:STOP 7,34
```

To stop a NonStop process pair, enter STOP followed by the name of the process. (A process executing as a NonStop process pair has both a process name and a process ID; you can display the name of a process with the PPD command, which is described in the GUARDIAN Operating System Utilities Reference Manual.) Also see the description of the STOP command in that manual for more information about stopping NonStop process pairs.

The BREAK key on terminals can also play a role in process control. For most programs supplied by Tandem, pressing the BREAK key has the following effect:

- If the program is waiting for a command, pressing BREAK allows COMINT to regain control of the terminal. The COMINT colon prompt (:) then reappears on the terminal screen. The other program continues to run even after COMINT regains control.

- If the program is executing a command, pressing BREAK halts the execution of the command. The program then waits for a new command.

The function of the BREAK key is defined by each program, and it can vary from program to program. You should consult the documentation for each program to find out exactly how the BREAK key is used. In all cases, however, if COMINT regains control of the terminal after BREAK is pressed, you can return control of the terminal to the system program by entering the PAUSE command:

```
:PAUSE
```

DISC FILE OPERATIONS

You can create, rename, and purge disc files with COMINT. All the operations described here, and more complex operations, can also be performed with the File Utility Program (FUP). See Sections 4, 5, and 6 for more information.

Creating Files (CREATE Command)

When you run a program, the files needed by that program must already exist. For example, if you want to send the output from a program to a disc file, rather than to your terminal, the disc file must already exist.

You can use the CREATE command in COMINT to create unstructured disc files. (To create other types of files, see the sections on FUP in this manual. For a complete description of the different types of files in Tandem systems, see the ENSCRIBE Programming Manual.)

For example, suppose you want to send the output from the STATUS *, USER command to a disc file. You first create an unstructured file named STAT in your current default subvolume by entering:

```
:CREATE STAT
```

Now you can send the output of the STATUS command to the file STAT, instead of to your terminal, by entering:

```
:STATUS / OUT STAT / *, USER
```

If you want to read an unstructured file such as STAT with EDIT, the Tandem text editor, the file must be in the file format of the EDIT program. An EDIT file is a special type of unstructured file. You can create EDIT files directly with the EDIT program (see the EDIT Manual for more information). You can also convert unstructured files into EDIT files with this command:

```
:EDIT <unstruct> PUT <edit-file>
```

<unstruct> is the name of the unstructured file to be converted, and <edit-file> is the name of the new EDIT file to be created.

As an example, to convert the file STAT created in the previous example to an EDIT file, enter:

```
:EDIT STAT PUT STATS
```

STATS is the name of the EDIT file created by the text editor.

Note that the default size of files you create with the CREATE command might not be large enough for some applications. For information about how to create larger files, see the descriptions of the CREATE command and the FUP CREATE command in the GUARDIAN Operating System Utilities Reference Manual.

Displaying File Names (FILES Command)

To display the names of the files in your current default volume and subvolume, enter the FILES command, like this:

```
:FILES
```

You can get a list of all files in a specific subvolume by entering the FILES command followed by the name of the subvolume. For example, suppose you want to get the names of all files in subvolume $GROOVE.SUBSAND. Enter:

```
:FILES $GROOVE.SUBSAND
```

COMINT then displays a list of the files in subvolume $GROOVE.SUBSAND, in alphabetical order:

```
$GROOVE.SUBSAND

ACRONYMS ARCHIVE CALCSRC CLS      DAY1     ENCRYPT FLYBOY  GRAPHS
HUMOR    L       LOOSELF LORALORA MAIL     NONO    OTHELLO PLM
RECIPES  RELEASE REQUEST S1       SINEWAVE STYLE   TALREAD TELCON
TESTPROC TMAIL   TNAME   TOOLS    TOWORLD  WHAT    WHO
```

Renaming Files (RENAME Command)

You can rename files with the RENAME command. With this command, you can specify a new subvolume, a new file name, or both. Enter RENAME followed by the old file name, a comma, and the new file name.

For example, you can change the name of the file MAIL in your current default volume to PROJECT.IDEAS by entering:

```
:RENAME MAIL, PROJECT.IDEAS
```

After you enter this command, the file MAIL no longer exists in your current default subvolume. Instead, MAIL has become the file IDEAS in subvolume PROJECT.

You cannot change the volume name of a file with this command. RENAME can change the file name or the subvolume of a file but not the physical volume location of a file. To give a new volume name to a file, you must make a copy of the file in the new disc volume using the FUP DUPLICATE command. The FUP sections in this manual give information about duplicating files.

Purging Files (PURGE Command)

You can purge or delete files from the system by entering PURGE followed by the name of the file to be purged.

Suppose you want to purge the file OLDFILE in subvolume $DATA.COMMENTS. Enter:

```
:PURGE $DATA.COMMENTS.OLDFILE
$DATA.COMMENTS.OLDFILE  PURGED
```

If OLDFILE resides in your current default subvolume and volume, you can purge it by entering:

```
:PURGE OLDFILE
$YRVOL.YRSVOL.OLDFILE  PURGED
```

EXECUTING COMMANDS FROM A FILE (OBEY COMMAND)

If you are going to enter a long list of commands or if you frequently enter a single lengthy command, you may save time and energy by using a command file for entering the commands. Files that contain COMINT commands are known as command files. You can put one or more COMINT commands in a file and execute them using the OBEY command. To execute the commands in a command file, enter OBEY or O followed by the name of the file.

For example, suppose you want a daily listing of the processes you started in three different systems. Use the following procedure to create an EDIT file named ALLSTATS that contains

this series of SYSTEM and STATUS commands. (You can place only one command to a line in a command file.)

```
:EDIT ALLSTATS!
*a
1       SYSTEM \LONDON
2       STATUS *, USER
3       SYSTEM \PARIS
4       STATUS *, USER
5       SYSTEM \ROME
6       STATUS *, USER
7       SYSTEM
8       //
*e
:
```

Whenever you want to execute these commands, enter:

```
:OBEY ALLSTATS
```

or

```
:O ALLSTATS
```

(This OBEY command assumes that ALLSTATS is in the current default subvolume. If the file is in another subvolume, include the subvolume name; if the file is in another volume and subvolume, you must fully qualify the file name.)

COMINT displays each command from the command file and then executes the command.

CORRECTING COMMAND ERRORS (FC COMMAND)

You can use the "fix command," FC, to change the last command that you entered. With it, you can insert, delete, or replace characters in the command. FC allows you to correct errors in commands, to reexecute commands, and to execute a series of similar commands.

After you enter the FC command, the previous command you entered is redisplayed. On the line below the command, a period appears as a prompt, followed by the cursor. The line that starts with the period prompt is a "correction line." Here you fix the previous command: you can type a character string that replaces the characters in the command you are fixing, or you can type subcommands to signal the beginning of a string that will replace characters in the command or be inserted into it.

For example, suppose you misspell a subvolume name in a FILES command and want to correct it:

```
:FILES $DATA.TABLE
FILE SYSTEM ERROR 014
:FC
:FILES $DATA.TABLE
.
```

Insertion (The I Subcommand)

You can insert characters by typing the I (or i) subcommand at the position in the line where the insertion should begin. Then type the characters you want to insert. Press RETURN when you finish making your insertion. The characters are then inserted before the character above the "I" or "i," like this:

```
:FC
:FILES $DATA.TABLE
.              iXA
:FILES $DATA.TAXABLE
.
```

A new period and correction line then appear. Each time you press RETURN after making changes on a correction line, the edited version of the command is redisplayed with a new period prompt. You have a chance to make additional changes. If the redisplayed line is correct, enter RETURN to execute the command.

Deletion (The D Subcommand)

To delete characters, type the D (or d) subcommand under the characters to be deleted:

```
:FC
:FILES $DATA.TAXABLE
.              ddd
:FILES $DATA.TALE
.
```

Replacement (The R Subcommand)

To replace characters, use either of these two methods:

- Type the R (or r) subcommand below the first character to be replaced. The next characters you type replace the characters in the original command, starting with the character above the "R" or "r"; for example:

```
:FC
:FILES $DATA.TALE
.             rRECORDS
:FILES $DATA.RECORDS
.
```

- If the new characters do not begin with an uppercase or lowercase I, D, or R, you can simply type them under the characters to be replaced:

```
:FC
:FILES $DATA.RECORDS
.             MONTHLY
:FILES $DATA.MONTHLY
.
```

Multiple Corrections

With one exception, if you want to perform more than one operation at the same time, you must separate the commands with two slashes (//). Here is an example of an FC command that makes multiple corrections:

```
:FC
:TGAL / IN $DATA.TABLE.OCT, OUT $S.#SPOOL/
.                       NOV//              i, NOWAIT
:TGAL / IN $DATA.TABLE.NOV, OUT $S.#SPOOL, NOWAIT/
.
```

The one exception occurs when any number of uppercase or lowercase D's begin a command. Another command can follow the D's without the intervening slashes (//). The following example illustrates how you can use this feature to replace character strings with another string of any length within a command:

```
:FC
:TGAL / IN $DATA.TABLE.NOV, OUT $S.#SPOOL, NOWAIT/
.                 dddddiRECS
:TGAL / IN $DATA.RECS.NOV, OUT $S.#SPOOL, NOWAIT/
.
```

Reentering Commands with FC

You can also use the fix command to reenter commands. Enter the FC command. When the period appears, do not make any changes. Simply press RETURN:

```
:FC
:FILES $DATA.TABLE
.<RETURN>
```

The command is then reexecuted. For a series of long commands that differ only slightly, you can use FC rather than retype each command; for example:

```
:TGAL / IN $ZDISC.MYGROUP.FILE1, OUT $S.#LP1, NOWAIT /
:FC
:TGAL / IN $ZDISC.MYGROUP.FILE1, OUT $S.#LP1, NOWAIT /
.<RETURN>                          2
:TGAL / IN $ZDISC.MYGROUP.FILE2, OUT $S.#LP1, NOWAIT /
.<RETURN>
:FC
:TGAL / IN $ZDISC.MYGROUP.FILE2, OUT $S.#LP1, NOWAIT /
.<RETURN>                          3
:TGAL / IN $ZDISC.MYGROUP.FILE3, OUT $S.#LP1, NOWAIT /
.<RETURN>
:
```

If, after you enter the FC command, you decide you don't want to reenter the previous command or a new version of it, simply press the BREAK key before entering the final RETURN. Execution of the fix command stops, and COMINT's colon prompt (:) reappears.

SECTION 3

ADVANCED USES OF COMINT

This section is intended for users who are already familiar with the basic uses of COMINT. It describes how to:

- Start a remote COMINT process
- Restart a COMINT process

You can find complete information about starting COMINT processes in the GUARDIAN Operating System Utilities Reference Manual. For information about using COMINT in application programs, see the GUARDIAN Operating System Programmer's Guide.

STARTING A REMOTE COMINT PROCESS

When Tandem systems are linked together to form a network, access to a file can be restricted to either of these general groups:

- Users on the system where the file resides
- Users on any system in the network (for systems that are part of a network)

If a file is available only to users on the system where it resides, you must be logged onto the system in order to gain access to it. To log onto a system other than the one where your current COMINT process is running, you first start a remote COMINT process in that system.

Before you can start a COMINT process on any remote system, you must be established as a user on that system and have remote passwords set up between your own system and the remote system.

Remote passwords and other system security features are discussed in Section 12.

To start a COMINT process in another system, you enter a COMINT command that specifies the name of the system followed by the COMINT program name. (Complete information about starting COMINT processes, including the syntax of the COMINT command and descriptions of available options, can be found in the GUARDIAN Operating System Utilities Reference Manual.)

For example, if your local system is part of a network that includes the \VIENNA system, you can start a COMINT process on the \VIENNA system by entering:

```
:\VIENNA.COMINT
```

A new remote COMINT process such as the one on \VIENNA does not have a backup process. If you want the remote COMINT to run as a NonStop process pair, you must do the following:

- Give the COMINT process a name by including the NAME <run-option> in your command to start the remote COMINT. If you include the NAME option but do not specify a name, the operating system assigns a name to the new process. See the description of the COMINT program in the GUARDIAN Operating System Utilities Reference Manual for more information about the NAME option.

- Specify the processors where the primary and backup COMINT processes are to execute. You specify the processor for the primary process with the CPU option. See the GUARDIAN Operating System Utilities Reference Manual for a description of the CPU option. You specify the processor where the backup process is to execute by including the CPU number of the processor after the slash character (/) following the last <run-option> in your command to start the COMINT process.

For example, you can start a remote COMINT process in the \BUDA system that runs as a NonStop process pair by giving a name for the COMINT process ($PEST), the processor where the primary process will execute (4), and the processor where the backup process will execute (5). The following command does all this:

```
:\BUDA.COMINT / CPU 4, NAME $PEST / 5
```

When the remote COMINT process begins executing, the COMINT colon prompt reappears on the screen. You can then log on to the remote system using the procedure you use on your local system.

To stop a remote COMINT process, enter:

- LOGOFF if you are logged on to the remote system
- EXIT (followed by "y" or "yes") if you did not log on

You then return to your home system.

RESTARTING A COMINT PROCESS

If a command interpreter that controls a terminal stops, you can restart it by running the COMINT program. Complete syntax of the command to start a COMINT process can be found in the GUARDIAN Operating System Utilities Reference Manual.

When you restart a COMINT process, you can maintain the same distribution of system resources by specifying the same options for the new COMINT process as for the previous one. You need the following information to restart the COMINT process for a particular terminal:

- The name of the COMINT process that controls the terminal (such as $C007)
- The name of the terminal the COMINT process controls (such as $TERM2)
- The CPU numbers of the processors that are running the primary and backup COMINT processes (such as 03 and 05)

You can obtain this information while the COMINT process is running by entering the WHO command at any terminal controlled by a COMINT process. If the COMINT process is not running and you need this information, see your system manager.

Your command to restart an interactive command interpreter must specify:

- As both the IN and OUT options, the name of the terminal where you want the COMINT process to run
- As the NAME option, the name of the COMINT process
- As the CPU option, the CPU number of the processor to run the primary process
- After the final slash (/) around the run options, the CPU number of the processor that will run the backup process

For example, suppose that you receive the following information for a terminal:

```
:WHO

HOME TERMINAL: $MABEL
COMMAND INTERPRETER: \CAL.$C186  PRIMARY CPU: 02  BACKUP CPU:  03
CURRENT VOLUME: $DRUM2.PLANNING
USERID: 008,027   USERNAME: MANUF.MABEL   SECURITY: NUNU
```

If this COMINT process is halted, you can restart it by entering:

```
:COMINT / IN $MABEL, OUT $MABEL, NAME $C186, CPU 2, NOWAIT / 3
```

SECTION 4

INTRODUCTION TO FUP

The File Utility Program (FUP) is a program you can use to create and manage disc files. With FUP, you can:

- Create new files
- Assign or preset file characteristics
- Display, print, or store information about files
- Duplicate files
- Rename files
- Set file security
- Load data into files
- Display the contents of a file
- Purge (delete) files

WHO USES FUP?

Anyone with access to COMINT (the GUARDIAN operating system command interpreter) can use FUP. These users include:

- Application programmers
- System programmers
- System managers
- System operators

Although most FUP commands are available to all users, the system enforces security requirements for accessing or altering files. For example, you cannot PURGE a file with FUP unless you have purge access to that file, and you cannot LICENSE a program file that contains privilege code unless you are logged on with the super ID. For more information about security requirements, see Section 12 and see the FUP section in the GUARDIAN Operating System Utilities Reference Manual.

HOW IS FUP USED?

You can enter FUP commands in any of three ways: through COMINT, interactively through FUP itself, or through a command file. Section 5 describes each method of entering FUP commands.

In addition, the Tandem Data Definition Language (DDL) compiler generates FUP commands for creating new data base files. See the Data Definition Language (DDL) Reference Manual for more information.

SECTION 5

BASIC USES OF FUP

This section contains information for new users of the File Utility Program, FUP. The first two subsections present a short summary of background information about the file system. The rest of this section describes the basic tasks that you can perform with FUP, including how to:

- Enter FUP commands
- Control FUP execution
- Get information about files
- Duplicate, rename, and purge files
- Change the security of files
- Give files to another user

Before reading this section, you should be familiar with the basic tasks you can perform with the GUARDIAN command interpreter (COMINT), as described in Section 2, "Basic Uses of COMINT."

If you want to see complete, detailed reference information and command syntax for FUP or COMINT, see the GUARDIAN Operating System Utilities Reference Manual.

FILES AND FILE NAMES

Before you use FUP to create or manage disc files, you should be familiar with the file system of the GUARDIAN operating system. This subsection summarizes the definition of a file and a file name.

In the GUARDIAN system, "files" include:

- Disc files (containing data, code, or text)
- Nondisc devices (such as terminals, printers, or tape drives)
- Processes (programs that are running)

You refer to a file by its file name. For disc files, the person or process who creates the file gives it a name. Nondisc devices have assigned names in the system. Processes can be named by their creator or can be assigned a name by the operating system.

An example of a disc file name is $VOL.SUBVOL.FILENAME. The parts of this file name are:

- $VOL is the name of the volume where the disc file resides. The volume name must begin with a dollar sign ($) followed by an alphabetic character; volume names cannot be longer than seven characters.
- SUBVOL is the name of the subvolume containing the disc file. (Subvolume names are not usually assigned to nondisc devices.) A subvolume name must begin with an alphabetic character; it can contain up to eight letters and/or numbers, but no punctuation or special characters (such as !, ?, or &).
- FILENAME is the name of the file itself. Like subvolume names, file names must begin with an alphabetic character, and can contain a maximum of eight letters and/or numbers, but cannot contain punctuation or special characters.

TYPES OF DISC FILES

You can create disc files to store data bases, coded programs, or text. For most uses, you should selectively create a specific type of file that best fulfills your purpose. Also, even if you do not create files of your own, you may work with various types of files.

ENSCRIBE, the Tandem data-base record manager, supports these four types of disc files:

- Unstructured files
- Key-sequenced files
- Entry-sequenced files
- Relative files

The first file type--unstructured--is an array of bytes of data. The organization of an unstructured file is determined by the creator, not by the file structure. Unstructured files often contain program code or text. Although it is possible to organize the contents of an unstructured file as a data base, the operating system cannot access the data as it does with structured files.

The last three file types are all structured files. A structured file is designed to contain a data base. The data base itself is made up of logical records (individual sets of data about separate items or people). Each type of structured file has a different organization for data records. See the ENSCRIBE Programming Manual for a full description of the three types of structured files.

When you create a file, you can use FUP to specify the structure of the file. You select the file's structure to match the type of data you want to store in the file. Section 6 of this manual demonstrates how to select file characteristics and create both structured and unstructured files with FUP.

HOW TO ENTER FUP COMMANDS

You can enter FUP commands in any of these ways:

- By entering complete FUP commands at the COMINT prompt
- By starting a FUP process and entering commands interactively at the FUP prompt
- By starting a FUP process that takes as its input a command file containing FUP commands

The following three subsections describe these three methods.

Entering FUP Commands through COMINT

You can enter single FUP commands at COMINT's colon prompt (:). Type FUP followed by the name of the FUP command. After FUP executes the command, control of the terminal returns to COMINT.

For example, you can enter the FUP HELP command by typing:

```
:FUP HELP
```

FUP performs the HELP command by listing the syntax of the FUP HELP command on the screen. Then the COMINT prompt reappears.

When you enter FUP commands through COMINT, you can enter only one command at a time. A new FUP process starts (and completes) for each command you enter. For this reason, when you have a series of FUP commands to enter, you might want to use a faster method. You can use either of the methods described in the next two subsections.

Entering FUP Commands Interactively through FUP

If you start a FUP process, you can enter FUP commands interactively at the FUP prompt. To start a FUP process, enter FUP alone (with no command name or other options). The File Utility Program then displays a banner message and its prompt, a hyphen (-):

```
:FUP
GUARDIAN FILE UTILITY PROGRAM - T9074B00 - (18MAR85)
-
```

A FUP process now controls the terminal. The hyphen indicates that the FUP process is ready to accept a command. You can enter a FUP command interactively by typing it after the hyphen, like this:

```
-HELP ALL
```

The FUP process performs the HELP ALL command (listing all the FUP commands) and then redisplays its prompt. FUP continues to accept and execute commands until you enter the EXIT command:

```
-EXIT
```

After you enter EXIT, the FUP process returns control of the terminal to the command interpreter. Another way to stop a FUP process is to enter a CTRL/Y character. CTRL/Y is a control

character used to mark the end of a file (EOF). When FUP reads CTRL/Y from its input file (in this case, your terminal), it stops execution. You can enter CTRL/Y at a terminal by pressing the CTRL and Y keys simultaneously.

The examples in this manual, except for the next subsection, use the interactive method of entering commands through FUP itself.

Entering FUP Commands from a Command File

A command file is an unstructured disc file that contains a series of commands (or a single command). Different command files can contain commands for different programs. For example, you might create one command file that contains a series of FUP commands, and another command file that contains COMINT commands. Command files are often useful when you must enter a series of commands or enter a long command more than once.

Every time you start a FUP process, your command can include any <run-option> available with the RUN command in COMINT (see Section). For example, two common run options are IN <file-name> (for naming input files) and OUT <list-file> (for naming output files). To execute a series of commands contained in a particular FUP command file, you specify the name of the command file as the IN option for the FUP process, as described below.

Sending Input to FUP from a Command File

If you specify a command file as the input file (the IN <file-name>) when you start a FUP process, the FUP process executes the commands contained in that file.

For example, suppose that you used the EDIT program to create a command file that gives you a list of the subvolumes in three different disc volumes--$DISC1, $DISC2, and $DISC3. The file is named ALLSUBS and contains these FUP commands:

```
SUBVOLS $DISC1
SUBVOLS $DISC2
SUBVOLS $DISC3
```

To execute these commands, enter a FUP command naming ALLSUBS as the IN option:

```
:FUP / IN ALLSUBS /
```

Control of the terminal returns to COMINT after FUP executes the last command in the ALLSUBS file.

You can add comment lines within a command file to identify the file and to explain the operations being performed. FUP comment lines must begin with a less-than symbol (<). Any characters on the line following the < are ignored by FUP. Here is the ALLSUBS command file with comment lines added:

```
< FUP Commands for Obtaining a List of
< All Subvolumes in $DISC1, $DISC2, and
< $DISC3
<
< Last Modified 1/17/85 13:24
<
SUBVOLS $DISC1     < Contains manufacturing files
SUBVOLS $DISC2     < Contains administrative files
SUBVOLS $DISC3     < Contains all other files
```

Sending Output from FUP to a File

When you start a FUP process, you can use the OUT option to send the process output to a file. An output file or list file can be either a disc file or a peripheral device such as a printer. If you do not specify an output file, FUP sends its output by default to your terminal.

Suppose that you want to save the output from a FUP operation in a disc file. If the output file doesn't already exist, you must begin by creating one. You can create an unstructured file with the CREATE command in COMINT (see Section 2), and you can create structured or unstructured files with the FUP CREATE command (see Section 6). Next, you start a FUP process.

In your command to start a FUP process, specify the command file (if you are using one) as the IN option, and the output file as the OUT option. For example, if ALLSUBS is a command file, and SUBINFO is a file you are using for the output, you can enter:

```
:FUP / IN ALLSUBS, OUT SUBINFO /
```

You can also send output to a printer. Specify the name of the printer with the OUT option in your FUP command, like this:

```
:FUP / IN ALLSUBS, OUT $LP /
```

Just as you can use an output file in a command to start a FUP process, you can include an output file with several individual FUP commands. The OUT file you specify in an individual FUP command overrides any OUT file you specified when you invoked FUP.

For example, you can send the results of three SUBVOLS commands to different output files by entering:

```
-SUBVOLS / OUT INFO1 / $DISC1
-SUBVOLS / OUT INFO2 / $DISC2
-SUBVOLS / OUT INFO3 / $DISC3
```

After these three commands are executed, the subvolumes in $DISC1 are listed in the file INFO1, and the subvolumes in $DISC2 are listed in INFO2, and the subvolumes in $DISC3 are listed in INFO3. Of course, these files must already exist before you can send command output to them.

GETTING HELP FROM FUP (HELP COMMAND)

If you are familiar with a FUP command but have forgotten its syntax, or if you want a list of available commands, the FUP HELP command provides a way to quickly get the information you need.

Entering HELP ALL lists all the FUP commands. This example demonstrates how to enter the command and shows the information that you receive:

```
-HELP ALL
ALLOCATE     ALLOW      ALTER         BUILDKEYRECORDS   CHECKSUM
COPY         CREATE     DEALLOCATE    DUP               EXIT
FILES        GIVE       HELP          INFO              LICENSE
LISTOPENS    LOAD       LOADALTFILE   PURGE             PURGEDATA
RENAME       RESET      REVOKE        SECURE            SET
SHOW         SUBVOLS    SYSTEM        VOLUME
```

To display the syntax of a particular FUP command, enter HELP followed by the name of a command, like this:

```
-HELP GIVE
GIVE <fileset list> , <group id> , <user id> [ , PARTONLY ]
```

CONTROLLING FUP

You can control the execution of a FUP process in several ways. These methods are described in the following two subsections.

Using the BREAK Key

When a FUP process controls a terminal, the BREAK key does the following:

- It allows COMINT to regain control of the terminal if FUP is waiting for a command. When this happens, the COMINT colon prompt reappears on the terminal screen. The FUP process, however, continues to execute even after COMINT regains control (see the following comment).
- It halts the execution of a FUP command that provides information. This includes the COPY, FILES, INFO, LISTOPENS, SHOW, and SUBVOLS commands. FUP then redisplays its hyphen prompt and waits for a new command. If, however, FUP is executing any other command, it continues to execute the command but returns control of the terminal to COMINT. The COMINT colon prompt then reappears on the terminal screen.

If COMINT regains control of the terminal after you press BREAK, you can return control of the terminal to FUP by entering the PAUSE command in COMINT:

```
:PAUSE
```

If you want to stop the FUP process rather than return control of your terminal to FUP, enter the STOP command in COMINT. For example, if FUP is the last process you started, you can stop it by entering:

```
:STOP
```

For more information about stopping processes, see Section 2.

Changing System and Volume Defaults (SYSTEM and VOLUME Commands)

Each FUP process maintains default values for volume and subvolume, as well as a default system in systems that are part of a network. The default system and volume for FUP are separate from the default values kept by the command interpreter.

Like COMINT, FUP expands file names by adding the current default value for any part of a file name that you omit. Thus, when you enter a file name as part of a FUP command, you need to include only those parts of the name that are different from the current default values kept by FUP.

Initially, the default values for a FUP process are the same as the current COMINT default values that are in effect when you start the FUP process. You can change these values with the FUP SYSTEM and VOLUME commands, as shown in the following examples.

To change the current default system, enter SYSTEM followed by the name of the system:

```
-SYSTEM \VENICE
```

You can return to the initial default system by entering SYSTEM without a system name:

```
-SYSTEM
```

You can change the default volume, the default subvolume, or both with the FUP VOLUME command. To change only the default volume, enter VOLUME followed by the name of the new default volume:

```
-VOLUME $DISC99
```

To change the default subvolume, enter VOLUME followed by the name of the new default subvolume:

```
-VOLUME MAYFLY
```

To change both the default volume and subvolume at once, enter VOLUME followed by the volume name, a period, and the subvolume name. For example, the changes made by the commands in the last two examples can be made at one time by entering:

```
-VOLUME $DISC99.MAYFLY
```

You can restore the initial default system, volume, and subvolume values by entering VOLUME alone:

```
-VOLUME
```

GETTING INFORMATION ABOUT SUBVOLUMES AND FILES

FUP commands can give you:

- A list of the subvolumes in a disc volume (SUBVOLS command)
- A list of the files in a subvolume (FILES command)
- Information about an individual file or set of files (INFO command)

Getting Information about Subvolumes (SUBVOLS and FILES Commands)

To get a list of the subvolumes in a given disc volume, enter the SUBVOLS command followed by the name of the volume:

```
-SUBVOLS $DISC78
```

In systems that are part of a network, you can specify a volume in another system, like this:

```
-SUBVOLS \DETROIT.$DISC45
```

If you enter the SUBVOLS command alone, you receive a list of the subvolumes in the current default system and volume:

```
-SUBVOLS
$DISC33
    MICK      JANIS     JIMIT
```

You can use the FILES command to get a list of the files in a subvolume. Enter the FILES command followed by the name of the subvolume, like this:

```
-FILES \NY.$APPLE.JACK
```

With the FUP FILES command, as with the FILES command in COMINT, you need to include the system name, volume name, or subvolume name only if it differs from the current default value. For example, you can get a list of the files in the subvolume SUBTLE in the current default system and volume by entering:

```
-FILES SUBTLE
```

In addition to a list of the files in one subvolume, you can get a list of the files in all the subvolumes on a disc volume. Enter the FILES command with an asterisk (*) in place of the subvolume name:

```
-FILES $SYSTEM.*
```

Getting Information about Single Files (INFO Command)

The INFO command is one of the most useful FUP commands. You can use INFO to find out many of the characteristics of a file or a set of files. Some common uses of the INFO command are described in this subsection. For additional information, see Section 6 and the description of the FUP INFO command in the GUARDIAN Operating System Utilities Reference Manual.

To get information about a single file, enter the INFO command followed by the name of the file. Remember that FUP expands file names using its own current default values. The next example illustrates how to get information on a file (here, the file DICTALT) in the current default system, volume, and subvolume:

```
-INFO DICTALT
          CODE      EOF      LAST MODIF  OWNER RWEP  TYPE REC BLOCK
$JUMBO.PATHWAY
 DICTALT   201     2048   17AUG84 12:22 8,44  NUNU    K     38 1024
```

In this example, the first line consists of listing headers or labels that identify the information displayed below that line. Under the listing headers is the name of the subvolume containing the listed file ($JUMBO.PATHWAY).

The headers in the FUP INFO listing indicate the following:

- CODE is the file code. Codes 100-999 are reserved by Tandem Computers Incorporated for use as system codes. For example, code 100 is assigned to all program files, and code 101 is assigned to files in the format used by the EDIT program. You can assign file codes other than 100-999 when you create files.
- EOF shows the current length of the file in bytes.
- LAST MODIF is the date and time when the file was last modified. If the file was last modified on the day you enter the INFO command, only the time is displayed.

- OWNER shows the user ID of the owner of the file. Each file is owned by only one user on the system. When a file is created, it is owned by the user who created it.

- RWEP stands for Read/Write/Execute/Purge. It shows the current security settings for the file. The security of a file is set by its owner. For information about file security, see Section 12.

- TYPE, REC, and BLOCK show information about structured files. See the description of the FUP INFO command in the GUARDIAN Operating System Utilities Reference Manual for details.

Getting Information about File Sets (INFO Command)

You can get information about more than one file with a single FUP INFO command. Like several other FUP commands, INFO allows you to specify:

- A <fileset> (a set of one or more files)

- A <fileset-list> (a list that includes one or more <fileset>s)

The specification for a file set is much like a single file name, except that a file set can be more than one file. A file set can be a list of file names separated by commas and enclosed in parentheses. A file set can also be a single file name.

You can specify the name of a system, volume, or subvolume where a file set resides, just as you would for the name of a single file. If any of these are omitted, FUP expands the file name or names using the current default values. With a file set, however, you can also include an asterisk (*) in place of a volume or subvolume name, with the following results:

- As the subvolume name, the asterisk applies the command to all the subvolumes in the volume you specify.

- As the file name, the asterisk applies the command to all files in the subvolume or subvolumes you specify. When you use an asterisk in place of the subvolume name, you must use an asterisk in place of the file name as well.

For example, you can get information about all the files in the current default subvolume by entering:

```
-INFO *
```

You can get information about all the files in the volume $MANUF by entering:

```
-INFO $MANUF.*.*
```

A file-set list can be a single file set, or a list that includes more than one file set. To include more than one file set in a file-set list, you must do the following:

- Enclose the <fileset-list> within parentheses.
- Include a comma after each <fileset> except the last.

For example, you can get information about the files in both the current default subvolume and in the volume $MANUF with this INFO command:

```
-INFO (*, $MANUF.*.*)
```

For a complete description of <fileset> and <fileset-list>, see the FUP command syntax summary in the GUARDIAN Operating System Utilities Reference Manual.

You can also use the INFO command to get information about files owned by a particular user on the system. Enter the INFO command followed by a <fileset> or a <fileset-list>, a comma, and a user name or user ID, as in the following examples:

```
-INFO (*.*, $SYSTEM.PROG1.*), USER MANUF.MABEL

-INFO *, USER 8,44
```

FUP then displays information about the files in each file-set list that are owned by the specified user.

PERFORMING COMMON FILE OPERATIONS

This section describes these common tasks that are performed with FUP:

- Duplicating files
- Renaming files
- Changing the security of files
- Giving ownership of files to other users
- Deleting files from the system

Duplicating Files (DUPLICATE Command)

You can duplicate a single file or a set of files with the DUPLICATE command. To duplicate a single file, enter DUPLICATE (or the abbreviated form DUP) followed by the name of the file to be copied, a comma, and the name of the new file. If you omit the volume, subvolume, or system name, FUP expands the file name by inserting the current default names.

For example, suppose that you want to duplicate the file BAKE in your current default subvolume. You want the name of the new file also to be BAKE, but you want it to reside in the subvolume $PISMO.CLAM. Enter:

```
-DUP BAKE, $PISMO.CLAM.BAKE
```

If the file $PISMO.CLAM.BAKE already exists, FUP stops execution of the command and responds with an error message. If you want to overwrite an existing file with DUP, you must include the PURGE option, which is described in the GUARDIAN Operating System Utilities Reference Manual.

To duplicate more than one file with a single command, enter DUP followed by a <fileset> or <fileset-list> to be copied, a comma, and a destination. .

To specify a destination, you must do the following:

- Include a system or volume name if either one differs from the current default value.
- Specify either of these:
  - --A single subvolume name, if you want all the new files to reside in that subvolume
  - --An asterisk (*) in place of the subvolume name, which specifies that the subvolume name of each new file is to be the same as the subvolume name of the file from which it was copied
- Include an asterisk (*) in place of a file name. The names of the new files will be the same as those of the old files.

As an example, suppose that you display the files in the subvolumes $ALPHA.SOUP and $COUNT.DOWN:

```
-FILES $ALPHA.SOUP
$ALPHA.SOUP
    A          B
-FILES $COUNT.DOWN
$COUNT.DOWN
    BLASTOFF
```

To duplicate the files in both $ALPHA.SOUP and $COUNT.DOWN to the subvolume $COUNT.UP, enter:

```
-DUP ($ALPHA.SOUP.*, $COUNT.DOWN.*), $COUNT.UP.*
```

Next display the contents of subvolume $COUNT.UP:

```
-FILES $COUNT.UP
$COUNT.UP
    A          B          BLASTOFF
```

To duplicate the same files to volume $DUKE and retain the original subvolume names, enter:

```
-DUP ($ALPHA.SOUP.*, $COUNT.DOWN.*), $DUKE.*.*
```

Now volume $DUKE has a SOUP and a DOWN subvolume, each containing the files copied from $ALPHA.SOUP and $COUNT.DOWN:

```
-FILES $DUKE.SOUP
$DUKE.SOUP
    A          B
-FILES $DUKE.DOWN
$DUKE.DOWN
    BLASTOFF
```

Renaming Files (RENAME Command)

You can rename a file or a set of files by using the RENAME command. To rename a single file, enter RENAME followed by the current file name, a comma, and the new file name; for example:

```
-RENAME FRED.INFO, MABEL.ARCHIVE
```

In this example, MABEL.ARCHIVE is the new name of the file. You can change the file's subvolume name and its file name, but not its volume name, with the RENAME command. A file that is renamed

remains in the same disc volume. To duplicate a file to another disc volume, use the FUP DUPLICATE command.

To rename a set of files, you specify a file set with RENAME just as you do with the DUPLICATE command. When you RENAME a file set, however, all the file names must remain the same; only the subvolume name can change. Enter RENAME followed by a file set or file-set list to be renamed, a comma, and a destination. For the destination you must specify the following:

- A subvolume name different from the subvolume names in the file set or file-set list to be renamed; the renamed files will reside in this different subvolume
- An asterisk (*) in place of the file name; the names of the renamed files will be the same as those of the original files

Suppose you want to rename the files in subvolumes $BIG.DOCUMNTS and $BIG.PROGDOCS. The renamed files must still reside in volume $BIG, but you would like to put all of them in the subvolume ALLDOCS. Enter:

```
-RENAME ($BIG.DOCUMNTS.*, $BIG.PROGDOCS.*), $BIG.ALLDOCS.*
```

If you omit system names or volume names in the file set or file-set list, FUP assumes the current default values. Also, remember that you cannot change the volume names of files with the RENAME command. This means that if you include a system or volume name in the file set or file-set list to be renamed, the same system or volume name must appear in the destination.

Changing a File's Security (SECURE Command)

If you own a file, you can change its security with the SECURE command. To do this, you assign a new security string to the file. A security string is a string of four characters that specify, respectively, who can read, write, execute, and purge a file. For information about file security and security strings, see Section 12.

Enter SECURE followed by the name of a file, a file set or a file-set list, a comma, and a security string. The security string is then assigned to all of the files you specify. As an example, suppose you want to assign the security string "NONO" to all the files in the subvolumes $FIDO.SPOT and $FIDO.ROVER. Enter:

```
-SECURE ($FIDO.SPOT.*, $FIDO.ROVER.*), "NONO"
```

Giving Files to Other Users (GIVE Command)

Only one user can own a file. If you own a file, you can give ownership of the file to another user with the GIVE command. Enter GIVE followed by the name of a file, a file set, or a file-set list, a comma, and the user ID of the new owner.

As an example, suppose you want to give the files in the subvolume FORTLIB to user RESRCH.JACK, whose user ID is 3,12. Enter:

```
-GIVE FORTLIB.*, 3,12
```

When you give a file to another user, you can get it back only if the new owner or a user with super ID uses the GIVE command to give it back to you.

Deleting Files from the System (PURGE Command)

You can delete individual files or sets of files with the PURGE command. Enter PURGE followed by a file name or a file set, as in the following example:

```
-PURGE IDEAS.*
```

After you enter the command, the screen displays information about each file and a prompt; for example:

```
-PURGE IDEAS.*
          CODE     EOF      LAST MODIF    OWNER  RWEP TYPE REC BLOCK
$CORP.IDEAS
  BAD      101    998732          14:32    8,44  CUCU
PURGE?
```

If you want to purge the file shown ($CORP.IDEAS.BAD), enter "Y" or "y" after the PURGE? prompt:

```
          CODE     EOF      LAST MODIF    OWNER  RWEP TYPE REC BLOCK
$CORP.IDEAS
  BAD      101     48732          14:32    8,44  CUCU
PURGE? y
  GOOD     101      2048 18MAY84 10:10    8,44  CUCU
PURGE?n
  INDIFF   101    998129 03MAR85 13:46    8,44  CUCU
PURGE?
```

FUP does not prompt you for permission to purge files if you place an exclamation point (!) at the end of your PURGE command. In this way, you can purge an entire subvolume of files with one command; for example:

```
-PURGE IDEAS.*!
3 FILES PURGED
```

CAUTION

Be careful when you include the exclamation point (!) in a command. The exclamation point means that the change you request will be made without further prompting. The results may be irreversible.

You can specify more than one file or a file set in a PURGE command. If you do, you must separate the elements with commas, like this:

```
-PURGE IDEAS.*, NOVELS.*
```

SECTION 6

ADVANCED USES OF FUP

This section contains information for users who are familiar with both the basic uses of the File Utility Program (FUP, described in Section 5) and with ENSCRIBE, the Tandem data-base record manager. This section describes how to:

- Set and display file-creation parameters
- Create different types of disc files
- Maintain disc files with FUP

For more information about file structures in Tandem systems, see the ENSCRIBE Programming Manual.

CREATING FILES

You can create both structured and unstructured disc files with FUP. However, using FUP is only one of several ways to create files. You can create unstructured files with the CREATE command in COMINT as well as with FUP. You can also create files with a program by calling the file-system CREATE procedure. For information about file-system procedures, see the ENSCRIBE Programming Manual. For information about the CREATE command in COMINT, see Section 2 of this manual and Section 2 of the GUARDIAN Operating System Utilities Reference Manual.

To create files with FUP, perform these four steps:

1. Assign values to file-creation parameters with the SET command. FUP maintains a table of current file-creation parameters. The values of these parameters can determine the attributes of any file you create with FUP. (You can also override the current settings by specifying different values in your CREATE command.)

2. Check the values of file-creation parameters with the SHOW command. The SHOW command allows you to check the current values of file-creation parameters, to ensure that the values are correct, before creating a new file.

3. Create the file with the CREATE command. When you enter the CREATE command, FUP consults its table of file-creation parameters and, if the current values will result in a legal file, creates a file whose attributes are based on these values.

4. Restore one or more file-creation parameters to their original values with the RESET command. You are now ready to create another new file.

Figure 6-1 shows the four steps for creating a file with FUP.

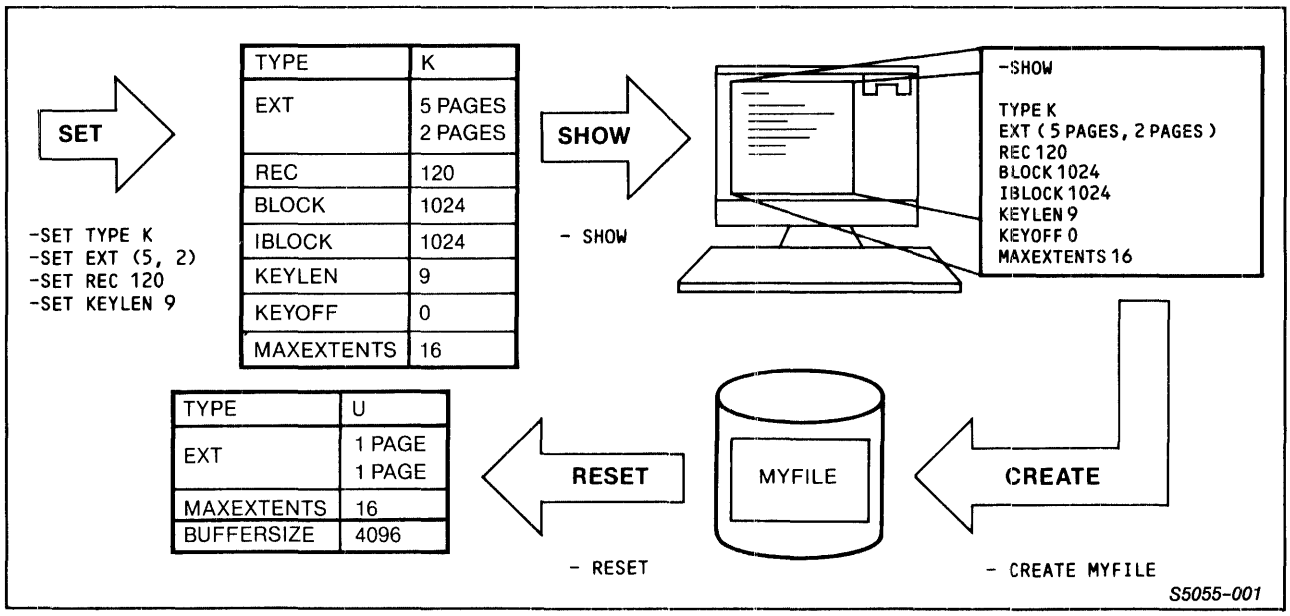


Figure 6-1. Steps for Creating a File with FUP

Table 6-1 lists and defines the options that you can control with the SET command; these are the file-creation parameters that determine the default characteristics of the file you create. The same parameters can be included in your CREATE command to override the current value displayed by the SHOW command. The examples in this section demonstrate the use of many of the file-creation parameters, and all of the parameters are fully described in the GUARDIAN Operating System Utilities Reference Manual.

Table 6-1. Options for the FUP SET Command

OPTION	SETS
TYPE	File type (key-sequenced, relative, etc.)
CODE	File code (to identify files)
EXT	Primary and secondary extent sizes
LIKE	Parameters to match an existing file
REFRESH	Automatic file-label refreshment
AUDIT	File auditing by TMF
REC	Record length
BLOCK	Data-block length
IBLOCK	Index-block length
COMPRESS	Data and index compression
DCOMPRESS	Data compression
ICOMPRESS	Index compression
KEYLEN	Primary-key length
KEYOFF	Primary-key offset
ALTKEY	Alternate-key specifications
ALTFILE	File name of an alternate-key file
ALTCREATE	Automatic creation of alternate-key files
PART	Secondary-partition specifications
PARTONLY	Creation of individual partitions
ODDUNSTR	Reading and writing of odd-numbered bytes
MAXEXTENTS	(DP2 files) Maximum number of file extents
BUFFERED	(DP2 files) Mode of handling write requests
BUFFERSIZE	(DP2 files) Size of internal buffer for unstructured files
AUDITCOMPRESS	(DP2 files) Mode of auditing by TMF
VERIFIEDWRITES	(DP2 files) Mode for disc writes
SERIALWRITES	(DP2 files) Selection of serial or parallel mirror writes

You can also create structured files with the aid of the Tandem Data Definition Language (DDL). DDL is a data-management tool for describing large data bases. You first describe the files in a DDL source schema, then compile the schema with the DDL compiler to produce both a data-base dictionary and a file containing FUP file-creation commands. To create the files, you run FUP, specifying as an IN file the command file created by DDL. For more information about DDL, see the Data Definition Language (DDL) Reference Manual.

Using the SET, SHOW, and CREATE Commands

When you start a FUP process, the default file-creation parameters are in effect. You can display these initial default values by entering the SHOW command just after you start FUP:

```
:FUP
-SHOW
    TYPE U
    EXT ( 1 PAGES, 1 PAGES )
    MAXEXTENTS 16
    BUFFERSIZE 4096
```

The TYPE parameter determines the file type of a file. Here, it has a value of U (for "Unstructured"). The two parts of the EXT parameter determine the sizes of primary and secondary file extents, respectively. Both extent sizes are set to "1 PAGE" (a unit of 2048 bytes). MAXEXTENTS and BUFFERSIZE are attributes that appear for DP2 files only. MAXEXTENTS controls the maximum number of extents that can be allocated to a file. BUFFERSIZE sets the size in bytes of the buffer used for unstructured files. (For information about DP2 file attributes, see the FUP section of the GUARDIAN Operating System Utilities Reference Manual.)

If you enter the CREATE command now, you will create an unstructured file whose primary and secondary extent sizes are one page each:

```
-CREATE FILE1
CREATED - $MANUF.FREDFILE.FILE1
```

You can see the attributes of the newly created file by entering the INFO command with the DETAIL option:

```
-INFO FILE1, DETAIL
$MANUF.FREDFILE.FILE1                10/23/84 20:23
    TYPE U
    EXT ( 1 PAGES, 1 PAGES )
    MAXEXTENTS 16
    BUFFERSIZE 4096
    OWNER 8,44
    SECURITY (RWEP): NUNU
    MODIF: 10/23/84 20:21
    EOF 0 ( 0.0% USED)
    EXTENTS ALLOCATED: 0
```

Suppose, however, that you want to create a key-sequenced file. To do this, you must specify the file type and the length of the primary key with SET commands:

```
-SET TYPE K
-SET KEYLEN 9
```

Now if you enter the SHOW command, this information is displayed:

```
-SHOW
    TYPE K
    EXT ( 1 PAGES, 1 PAGES )
    REC 80
    BLOCK  1024
    IBLOCK 1024
    KEYLEN 9
    KEYOFF 0
    MAXEXTENTS 16
```

Note that this SHOW display lists more attributes than those you assigned with your SET command. These values are additional defaults that apply to structured files (including key-sequenced files, files with alternate keys, and partitioned files). For a complete description of file-creation parameters, see the syntax of the SET command in the GUARDIAN Operating System Utilities Reference Manual.

Before you create a file, you should use SHOW to check that the values of the parameters shown in the SHOW display are the ones you want. If they are, and if they will result in a legal file, you can create a new file.

Using the RESET Command

After you create each file (and before you create others), you might want to restore some or all of the parameters to their default values. To do this, enter the RESET command followed by the parameters to reset. You can reset more than one parameter with a single RESET command by separating the parameters with commas, like this:

```
-SHOW < displays the current values before you enter RESET
    TYPE K
    EXT ( 5 PAGES, 5 PAGES )
    REC 80
    BLOCK  1024
    IBLOCK 1024
    KEYLEN 9
    KEYOFF 0
    DCOMPRESS, ICOMPRESS
    MAXEXTENTS 16
-RESET EXT, COMPRESS
-SHOW  < displays the new parameter values after RESET
    TYPE K
    EXT ( 1 PAGES, 1 PAGES )
    REC 80
    BLOCK  1024
    IBLOCK 1024
    KEYLEN 9
    KEYOFF 0
    MAXEXTENTS 16
```

You can reset all file-creation parameters at once by entering RESET with no parameters. This restores the parameter values that were in effect when you started the FUP process:

```
-RESET
-SHOW
    TYPE U
    EXT ( 1 PAGES, 1 PAGES )
    MAXEXTENTS 16
    BUFFERSIZE 4096
```

File-Creation Examples

The following examples demonstrate the creation of all the types of files: unstructured, entry-sequenced, relative, and key-sequenced files. Examples are also given for creating files with alternate keys, partitioned files, and files that match the attributes of an existing file. Each example shows the series of commands needed to create a particular type of file. The logon default file-creation attributes are in effect at the start of each example.

Creating an Unstructured File. Unstructured files are essentially arrays of bytes. They are normally used to store object programs and text input with EDIT, the Tandem text editor. If you do not change the logon default file-creation attributes, or if you create a file with the CREATE command in COMINT, the new file is an unstructured file.

To create an unstructured file named $MY.LIL.UN whose primary extent size is 10 pages (20,480 bytes), whose secondary extent size is 2 pages (4096 bytes), and whose file code is 999, enter:

```
-SET EXT (10,2)      < You can specify extent sizes in pages,
                     < bytes, records, and megabytes.  If you
                     < do not specify a unit, FUP assumes that
                     < the unit is pages.
-SET CODE 999        < Set the file code (used to identify
                     < the file).
-SHOW                < Show the current parameter values.
    TYPE U
    CODE 999
    EXT ( 10 PAGES, 2 PAGES )
    MAXEXTENTS 16    < Appears only for DP2 files.
    BUFFERSIZE 4096  < Appears only for DP2 files.
-CREATE $MY.LIL.UN   < Create the file.
CREATED - $MY.LIL.UN
```

Creating an Entry-Sequenced File. For an entry-sequenced file, the following is true:

- New records are added to the end of the file.
- Records are searched sequentially from the beginning of the file.
- Records added to the file can vary in length, but once a new record is added, its length cannot change.
- Records in the file can be updated but not deleted.

Figure 6-2 shows the structure of an entry-sequenced file.

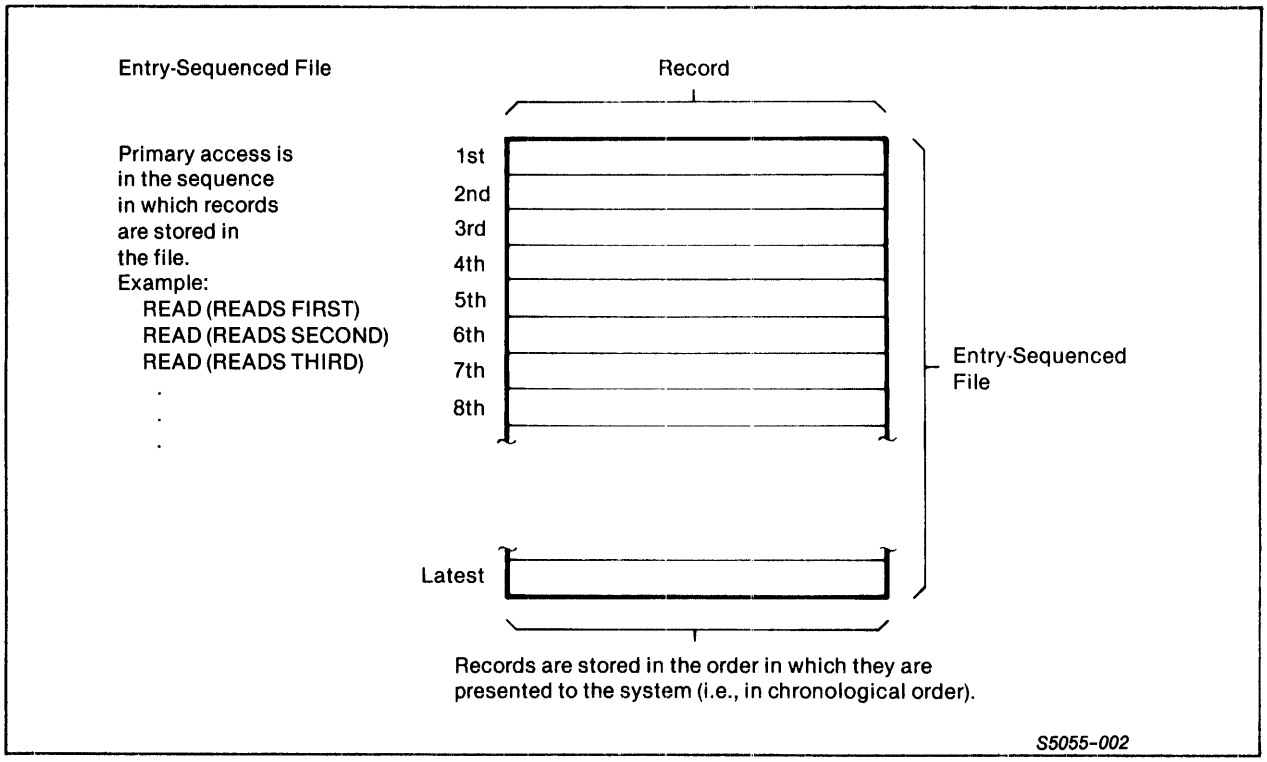


Figure 6-2. Structure of an Entry-Sequenced File

You can create an entry-sequenced file named $MY.EN.SEQ whose primary extent and secondary extent sizes are four pages each, and whose data-block length is 2048 bytes by entering:

```
-SET TYPE E          < Set file type to entry-sequenced.
-SET EXT 4           < You can specify extent sizes in pages,
                     < bytes, records, and megabytes.  If you
                     < do not give a unit, FUP assumes pages.
-SET BLOCK 2048      < Set the data-block length.
-SHOW                < Show the current parameter values.
    TYPE E
    EXT ( 4 PAGES, 4 PAGES )
    REC 80
    BLOCK  2048
    MAXEXTENTS 16    < Appears only for DP2 files.
-CREATE $MY.EN.SEQ   < Create the file.
CREATED - $MY.EN.SEQ
```

Creating a Relative File. For a relative file, the following is true:

- All physical records are the same length.
- Records are stored by record number. Record numbers give the position of a record relative to the first record in the file.
- You can retrieve records randomly using record numbers.

Figure 6-3 shows the structure of a relative file.

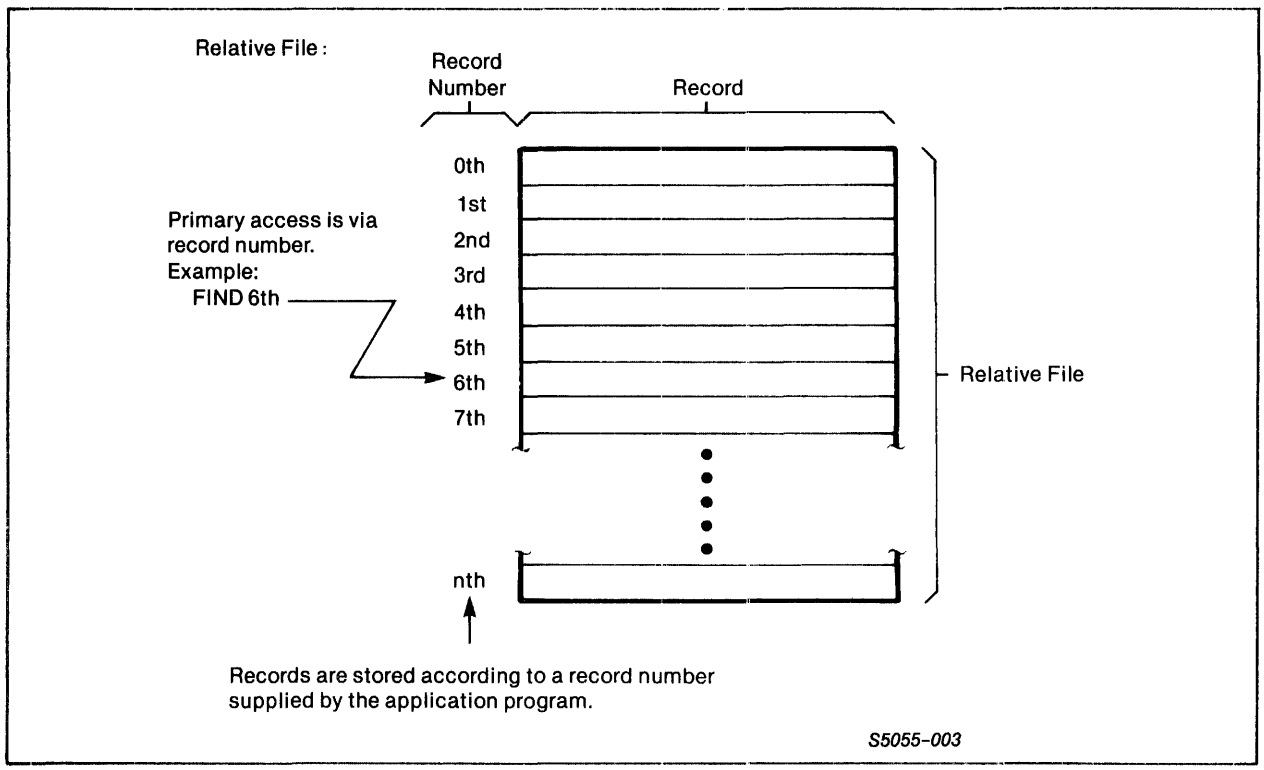


Figure 6-3. Structure of a Relative File

You can create a relative file named $MY.OLD.REL whose primary extent size is 5 pages, whose secondary extent size is 2 pages, and whose record length is 120 bytes by entering:

```
-SET TYPE R          < Set file type to relative.
-SET EXT (5,2)       < You can specify extent sizes in pages,
                     < bytes, records, and megabytes.  If you
                     < do not give a unit, FUP assumes pages.
-SET REC 120         < Set the record length to 120 bytes.
-SHOW                < Show the current parameter values.
    TYPE R
    EXT ( 5 PAGES, 2 PAGES )
    REC 120
    BLOCK  1024
    MAXEXTENTS 16    < Appears only for DP2 files.
-CREATE $MY.OLD.REL < Create the file.
CREATED - $MY.OLD.REL
```

Creating a Key-Sequenced File. In a key-sequenced file, records are stored by the values of their primary keys. A primary key is a field within a record that uniquely identifies the record. Figure 6-4 shows the structure of a key-sequenced file.

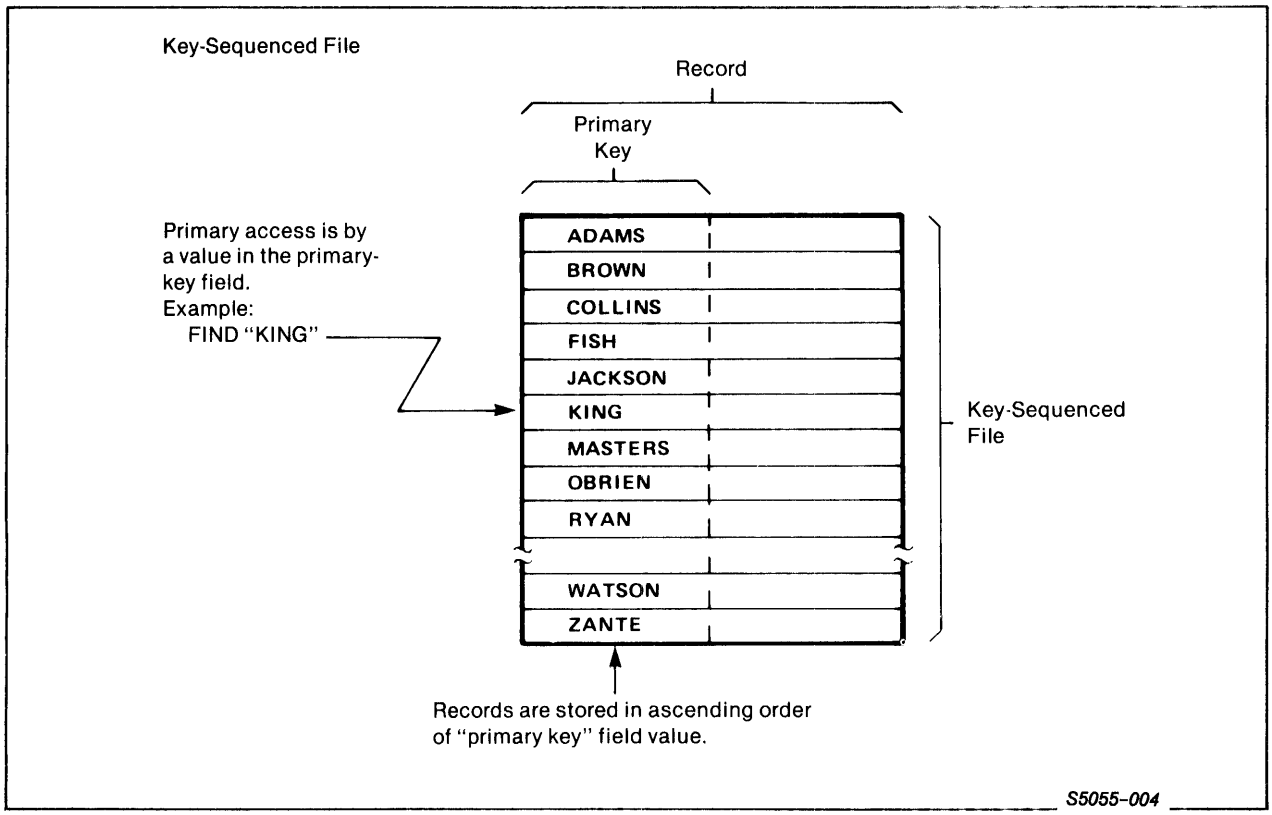
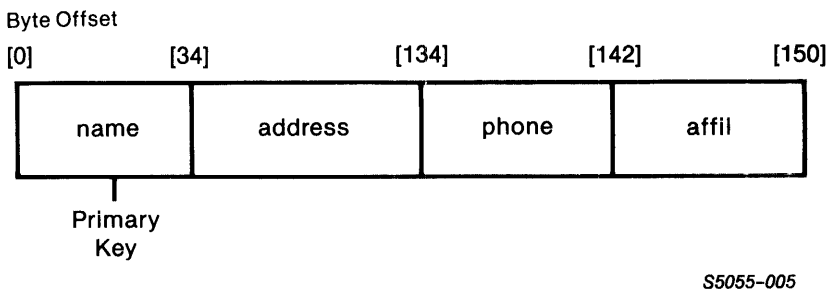


Figure 6-4. Structure of a Key-Sequenced File

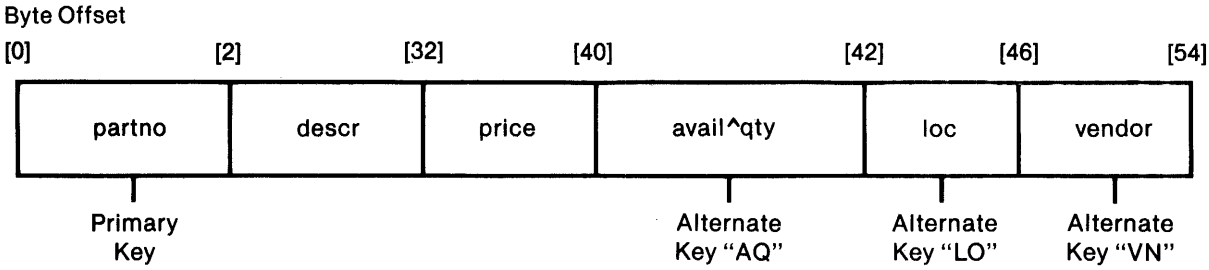
Here is a possible format for a record in a key-sequenced file:



You can create a file for records in the preceding format by entering the following FUP commands in a disc file and then starting a FUP process that takes input from this file:

```
VOLUME $VOL1.SVOL       < Set the default volume and
                        < subvolume to the desired values.
SET TYPE K              < For file type, specify key-sequenced.
SET CODE 1000           < You may optionally specify a file
                        < code to identify the file.
SET EXT (16, 1)         < Set appropriate primary and secondary
                        < extent sizes for the application.
SET REC 150             < Set the record length and block size.
SET BLOCK 2048
SET COMPRESS            < If desired, you may specify data and
                        < index compression.
SET KEYLEN 34           < You must specify a primary-key length
                        < for key-sequenced files.
SET IBLOCK 2048         < You can also specify the size of
                        < index blocks.
SHOW                    < Display the current parameter values.
   TYPE K
   CODE 1000
   EXT ( 16 PAGES, 1 PAGES )
   REC 150
   BLOCK  2048
   IBLOCK 2048
   KEYLEN 34
   KEYOFF 0
   DCOMPRESS, ICOMPRESS
   MAXEXTENTS 16        < Appears only for DP2 files.
CREATE MYFILE           < Create the file.
```

Creating a Key-Sequenced File with Alternate Keys. Besides having a primary key, a key-sequenced file can have one or more alternate keys. Here is a possible format for a record in a key-sequenced file with alternate keys:



S5055-006

To create a file to contain data in this record format, first enter the following FUP commands in a file. Then run FUP, specifying the command file as the input file with the IN option:

```
VOLUME $VOL1.SVOL          < Set the default volume and
                           < subvolume to the desired values.
SET TYPE K                 < Set the file type to key-sequenced.
SET CODE 1001              < You can specify a file code to
                           < identify the file, if desired.
SET EXT (32,8)             < Set appropriate primary and
                           < secondary extent sizes for the
                           < application.
SET REC 54                 < Set the record length and block
                           < size.
SET BLOCK 4096
SET IBLOCK 1024            < You can also specify the size of
                           < index blocks.
SET KEYLEN 2               < You must specify a primary-key
                           < length for key-sequenced files.

SET ALTKEY ("AQ", KEYOFF 40, KEYLEN 2)  < Specify the alternate
SET ALTKEY ("LO", KEYOFF 42, KEYLEN 4)  < keys and the name and
SET ALTKEY ("VN", KEYOFF 46, KEYLEN 8)  < number of the SET
SET ALTFILE (0, INVALT)                 < alternate-key file.
                                        < If the FILE parameter
                                        < is not specified in
                                        < the SET ALTKEY
                                        < command, the
                                        < alternate-key file
                                        < number is set to the
                                        < default value of 0.
```

```
SHOW                          < Show the current parameter values.
   TYPE K
   CODE 1001
   EXT ( 32 PAGES, 8 PAGES )
   REC 54
   BLOCK  4096
   IBLOCK 1024
   KEYLEN 2
   KEYOFF 0
   ALTKEY ( "AQ", FILE 0, KEYOFF 40, KEYLEN 2 )
   ALTKEY ( "LO", FILE 0, KEYOFF 42, KEYLEN 4 )
   ALTKEY ( "VN", FILE 0, KEYOFF 46, KEYLEN 8 )
   ALTFILE ( 0, $SVOL1.SVOL.INVALT )
   ALTCREATE
   MAXEXTENTS 16              < Appears only for DP2 files.

CREATE INV                    < Create the file.
```

Creating a Key-Sequenced Partitioned File. You do not need to keep all the data in a disc file on the same disc volume. By partitioning a file, you can store data in a file on as many as 16 different disc volumes. Partitioning allows you to create files that can be larger than those which reside on only one disc. Also, because the disc heads for each disc can be repositioned at the same time, access to data can be faster. Figure 6-5 shows the structure of a partitioned file.

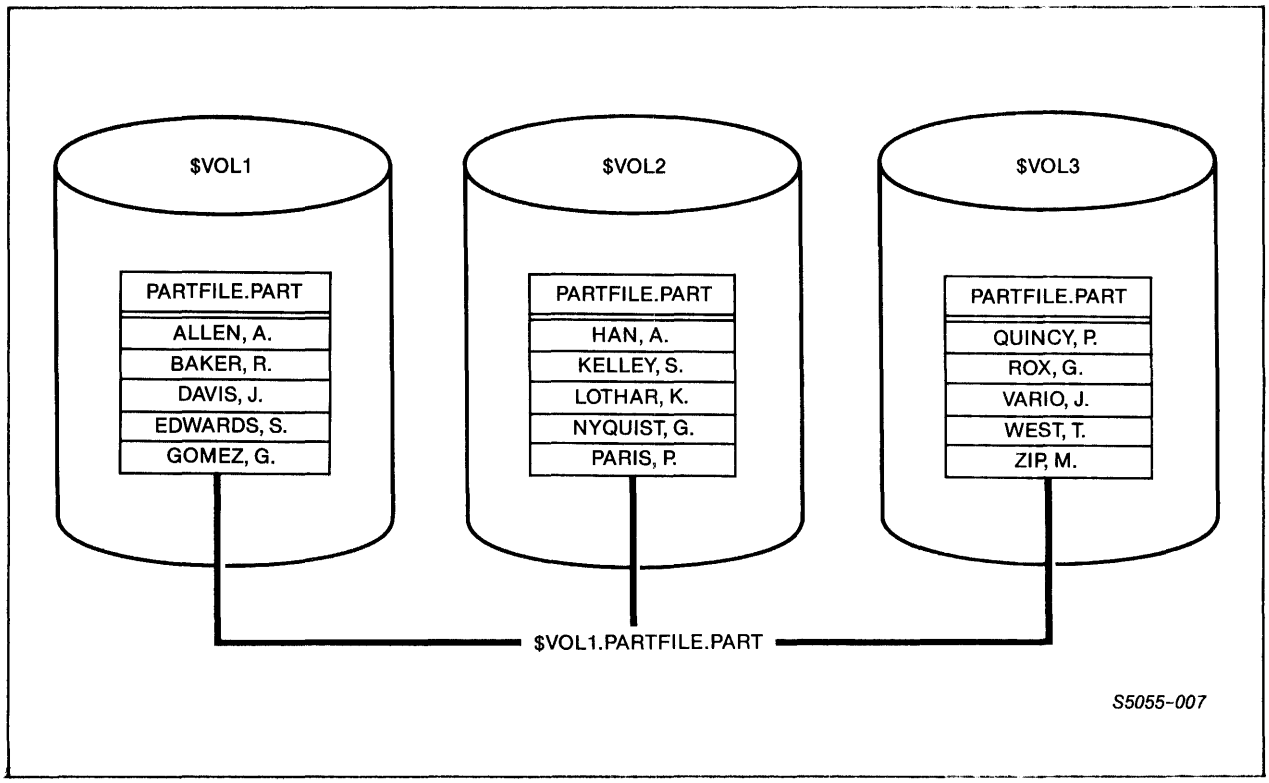


Figure 6-5. Structure of a Partitioned File

Suppose you want to create a partitioned file whose primary partition resides in the disc volume $VOL1. You want the name of the file to be $VOL1.PARTFILE.PART You want the primary partition to contain the first record and all subsequent records up to, but not including, records whose primary key begins with "HA".

You also want to create secondary partitions for the file. The first will reside on $VOL2, while the second will reside on $VOL3. The first secondary partition, on $VOL2, contains records whose primary key begins with "HA" and subsequent records up to, but not including, records whose primary keys begin with "QU".

Creating a Key-Sequenced Partitioned File. You do not need to keep all the data in a disc file on the same disc volume. By partitioning a file, you can store data in a file on as many as 16 different disc volumes. Partitioning allows you to create files that can be larger than those which reside on only one disc. Also, because the disc heads for each disc can be repositioned at the same time, access to data can be faster. Figure 6-5 shows the structure of a partitioned file.

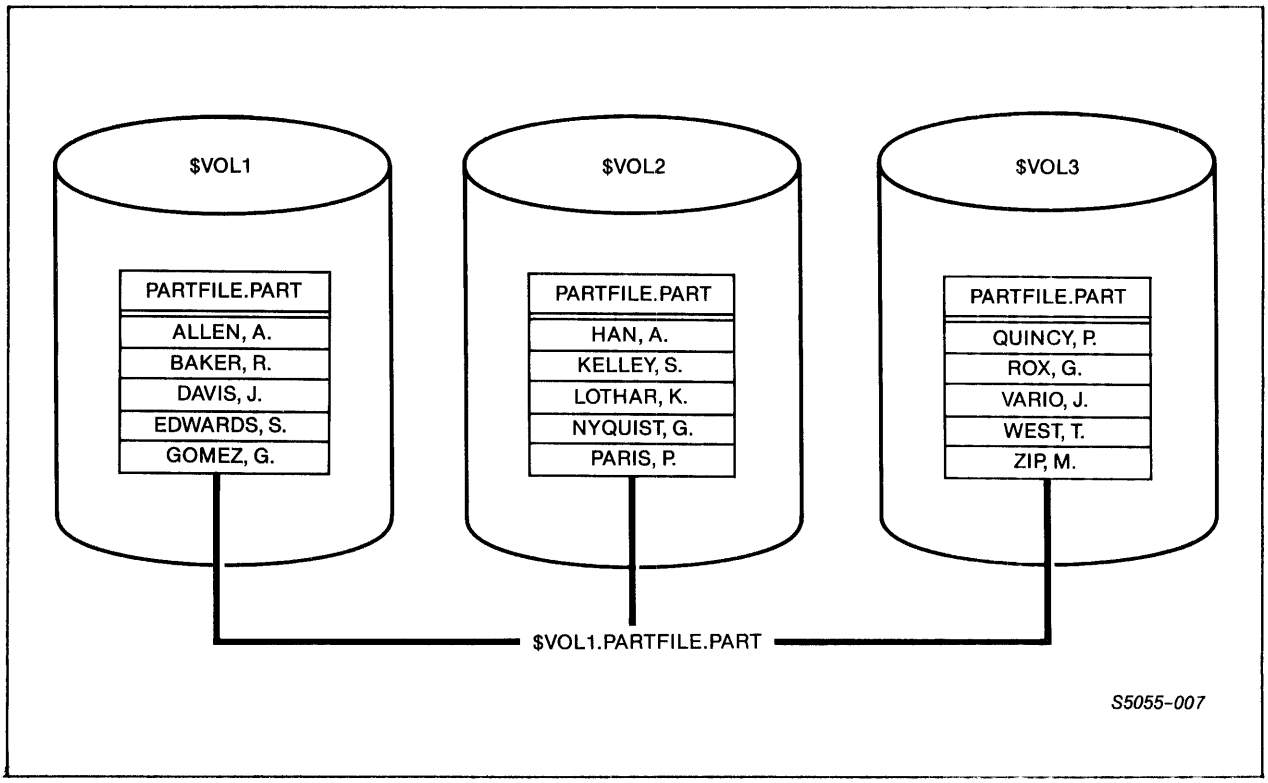


Figure 6-5. Structure of a Partitioned File

Suppose you want to create a partitioned file whose primary partition resides in the disc volume $VOL1. You want the name of the file to be $VOL1.PARTFILE.PART You want the primary partition to contain the first record and all subsequent records up to, but not including, records whose primary key begins with "HA".

You also want to create secondary partitions for the file. The first will reside on $VOL2, while the second will reside on $VOL3. The first secondary partition, on $VOL2, contains records whose primary key begins with "HA" and subsequent records up to, but not including, records whose primary keys begin with "QU".

All other records (that is, those records whose primary keys begin with "QU" and all subsequent records) will reside in the second secondary partition on $VOL3.

These FUP commands create a key-sequenced partitioned file named $VOL1.PARTFILE.PART:

```
VOLUME $VOL1.PARTFILE          < Set the default volume and
                               < subvolume to the desired values.
SET TYPE K                     < Set the file type to
                               < key-sequenced.
SET CODE 409                   < You may optionally specify a
                               < file code to identify the file.
SET EXT (64,8)                 < Set appropriate primary and
                               < secondary extent sizes for the
                               < application.
SET REC 128                    < Set the record length.
SET KEYLEN 20                  < Specifying the primary-key
                               < length is required for
                               < key-sequenced files.
SET PART ( 1, $VOL2, 64, 8, "HA" )  < Specify the partitions,
SET PART ( 2, $VOL3, 64, 8, "QU" )  < their volumes, primary
                                    < and secondary extent
                                    < sizes, and the partial-
                                    < key values.
                               < Show the current parameter
SHOW                           < values.
   TYPE K
   EXT ( 64 PAGES, 8 PAGES )
   REC 128
   BLOCK  1024
   IBLOCK 1024
   KEYLEN 20
   KEYOFF 0
   PART ( 1, $VOL2, 64, 8, "HA" )
   PART ( 2, $VOL3, 64, 8, "QU" )
   MAXEXTENTS 16  < This attribute appears only for DP2 files.
CREATE PART                     < Create the file.
```

Creating a File with the Attributes of an Existing File. If a file already exists with all or most of the attributes you want a new file to have, you can use the LIKE option of the SET command to duplicate these attributes.

You can display the attributes of a file by entering the INFO command and specifying the DETAIL option, as shown in the following example:

```
-INFO $CLEAN.BERKELEY.AIR, DETAIL
$CLEAN.BERKELEY.AIR                    8/24/84  0:31
    TYPE K
    EXT ( 5 PAGES, 5 PAGES )
    REC 80
    BLOCK  1024
    IBLOCK 1024
    KEYLEN 9
    KEYOFF 0
    DCOMPRESS, ICOMPRESS
    MAXEXTENTS 16
    OWNER 8,44
    SECURITY (RWEP): NUNU
    MODIF:  8/24/84  0:29
    EOF 0 (0.0 % USED)
    EXTENTS ALLOCATED: 0
    FREE BLOCKS 0
    INDEX LEVELS: 0
```

Make sure the parameters for the existing file have the values you want for the new file. Then enter the SET command with the LIKE option. FUP then sets its current file-creation characteristics to match those of the file you specify. New files you create will have the same attributes:

```
-SET LIKE $CLEAN.BERKELEY.AIR
-SHOW
    TYPE K
    EXT ( 5 PAGES, 5 PAGES )
    REC 80
    BLOCK  1024
    IBLOCK 1024
    KEYLEN 9
    KEYOFF 0
    DCOMPRESS, ICOMPRESS
    MAXEXTENTS 16
-CREATE $CLEAN.SANJOSE.AIR
CREATED - $CLEAN.SANJOSE.AIR
```

MAINTAINING FILES

Besides creating files with FUP, you can use FUP to:

- Load data into files
- Purge data from files
- Rename and move files with alternate keys
- Move files to a backup volume
- Add alternate keys to files
- Modify partitioned files

These operations are described in the next subsections.

Loading Data into Files

FUP can move data between files in three different ways. First, the DUP command duplicates an entire file at a time. With DUP, both your original file and the copy must be disc files.

Second, the COPY command moves data one record at a time (see the description of COPY in the GUARDIAN Operating System Utilities Reference Manual). Unlike the DUP command, COPY allows you to copy part of a file. It also allows you to copy records to and from media other than discs, including tape volumes and terminals.

Finally, the LOAD command moves data into a structured file. With the LOAD command, data is transferred one record at a time from the source file and moved a block at a time into the destination file.

There are two advantages to the LOAD command:

- Loading files does not affect alternate-key values.
- Data can be written a block at a time. LOAD is faster than COPY.

To load a file, enter LOAD followed by the name of the file that contains the data, a comma, and the name of the file to be loaded.

The next example illustrates how you can load a file with data stored on tape. The name of the file to be loaded is:

```
$VOL1.SVOL.PART
```

PART is a key-sequenced file with three partitions. The secondary partitions are on volumes $VOL2 and $VOL3.

The first secondary partition, on $VOL2, contains records whose primary key begins with "HA" and subsequent records up to, but not including, records whose primary keys begin with "QU". All other records (that is, those records whose primary keys begin with "QU" and all subsequent records) will reside in the second secondary partition on $VOL3.

The records to be loaded into this file are 128 bytes in length and are on magnetic tape in unsorted order. There is one record for each block on the tape.

To load the file, enter:

```
-VOLUME $VOL1.PARTFILE.PART
-LOAD $TAPE, PART
```

When the LOAD command is executed, the records from tape are first read, then sorted by primary key. After the records are sorted, FUP loads them into the partitions using the key specifications contained in the file label of PART.

The LOAD command also allows you to specify a number of useful options. See the description of the LOAD command in the GUARDIAN Operating System Utilities Reference Manual for more information.

Purging Data from Files

With the PURGEDATA command, you can purge data from a file without deleting the file. When you enter the PURGEDATA command, the end-of-file (EOF) pointer is set to 0, and other file-label values indicating the size of the file are reset to indicate that the file is empty. PURGEDATA does not, however, change the attributes of the file.

Enter PURGEDATA followed by the name of the file, a <fileset>, or a <fileset-list>, as in this example:

```
-PURGEDATA ($VOL1.XDATA.FILE0, $VOL2.XDATA.FILE1)
```

Data is purged from all files you specify.

Renaming and Moving Files with Alternate Keys

Renaming or moving a file with alternate keys is complicated by the fact that the names of alternate-key files are attributes of the primary-key file. The following examples show how to modify files to reflect the new names.

Renaming a File and Its Alternate-Key File

In this example, assume that you want to rename a structured file named $VOL1.SVOL.PRIFILE. You want the file to be named OLDSVOL.PRIFILE. The file has one alternate-key file named $VOL1.SVOL.AFILE, which you want to rename OLDSVOL.AFILE.

To change the names of the two files, enter:

```
-VOLUME $VOL1.OLDSVOL
-RENAME SVOL.PRIFILE,PRIFILE
-RENAME SVOL.AFILE,AFILE
```

Now you must alter the file label of PRIFILE so that it includes the new name of the alternate-key file. You do this with the ALTER command:

```
-ALTER PRIFILE, ALTFILE ( 0, AFILE )
```

Moving a File with Alternate Keys to a New Volume

In this example, assume that you have a structured file named $VOL1.SVOL.PRIFILE. It has one alternate-key file named $VOL1.SVOL.AFILE. You want to move the files to a new volume, $NEW, without changing the file names. To move them, enter:

```
-VOLUME $VOL1.SVOL
-DUP ( PRIFILE, AFILE ), $NEW.*.*
-VOLUME $NEW
-ALTER PRIFILE, ALTFILE ( 0, AFILE )
```

The ALTER command changes the name of the alternate-key file in the file label of PRIFILE from $VOL1.SVOL.AFILE to $NEW.SVOL.AFILE.

Moving Files to a Backup Volume

When you want quick access to backup files, you can back up disc files onto another disc volume. Use disc backups in addition to, not in place of, using BACKUP and RESTORE to back up files onto magnetic tape. This subsection shows how you can use FUP to perform a disc-to-disc backup operation.

Assume that you want to duplicate all the files in disc volume $VOL1. You want the copies to reside in disc volume $BACKUP. To copy the files, enter:

```
-DUP $VOL1.*.*, $BACKUP.*.*, PARTONLY, SAVEALL
```

After FUP executes this command, a duplicate (backup) copy of each file on $VOL1 exists on $BACKUP. Each backup file has the same file name and subvolume name as the corresponding original file. Because you include the PARTONLY option, only primary or secondary partitions residing on $VOL1 are duplicated, while partitions on other volumes are not. The SAVEALL option preserves the user ID, timestamp, and security setting for each file.

Adding Alternate Keys to Files

As your data bases grow and change, you may need to add new keys to existing files. The steps for adding an alternate key to a file that already has alternate keys, and to a file that does not have alternate keys, are shown in the following examples.

Adding an Alternate Key in an Existing Alternate-Key File

For this example, suppose that you have a file named $VOL1.SVOL.PRIFILE. This file has an alternate-key file named $VOL1.SVOL.AFILE. You want to add the alternate-key records for the new key field to this file.

The key specifier for the new key is "NM", the key offset in the record is 4, and the key length is 20.

To add the new alternate key, enter:

```
-VOLUME $VOL1.SVOL
-ALTER PRIFILE, ALTKEY ( "NM", KEYOFF 4, KEYLEN 20 )
-LOADALTFILE 0, PRIFILE, ISLACK 10
```

The LOADALTFILE command loads the alternate-key records for key specifier "NM" and any other alternate keys into the alternate-key file. To allow for future growth of the file, you can reserve empty space in index blocks by specifying a percentage of slack space with the LOADALTFILE ISLACK option.

Note, however, that when you add a new alternate key, the length of the key cannot be longer than the longest key already in the alternate-key file unless you:

- Use the SET LIKE command to duplicate the attributes of the old alternate-key file
- Specify the new alternate key with the ALTER command
- Specify a new record and key length for the alternate-key file that is:

```
2 + length of the primary key
  + length of the longest alternate key
```

For example, suppose that the file PRIFILE in the previous example has one alternate key that is 15 bytes long. The primary-key length is 40 bytes. The name of the alternate-key file is $VOL1.SVOL.ALTFILE. You can add a new alternate key, "NM", with a length of 20 bytes by entering:

```
-VOLUME $VOL1.SVOL
-ALTER PRIFILE, ALTKEY ( "NM" , KEYOFF 15 , KEYLEN 20 )
-SET LIKE ALTFILE
-PURGE ALTFILE!
-SET REC 62
-SET KEYLEN 62
-CREATE ALTFILE
-LOADALTFILE 0, PRIFILE
```

Here, the new record and key lengths must be 62 bytes (2 for the key specifier, plus 20 for the longest alternate key, plus 40 for the primary key).

Adding an Alternate Key in a New Alternate-Key File

For this example, assume that you have an entry-sequenced file named $VOL1.SVOL.FILEA. This file does not have an alternate-key file. To add an alternate key to the file, you need to create a new alternate-key file; this new file is to be named $VOL1.SVOL.FILEB.

The key specifier for the new key is "XY", the key offset in the record is 0, and the key length is 10.

To add the new alternate key, enter:

```
-VOLUME $VOL1.SVOL
-CREATE FILEB, TYPE K, REC 16, KEYLEN 16
-ALTER FILEA, ALTFILE (0, FILEB), ALTKEY ("XY", KEYLEN 10)
-LOADALTFILE 0, FILEA
```

You use the CREATE command to create the alternate-key file $VOL1.SVOL.FILEB. For non-unique alternate keys, the record length and key length are 16 bytes (2 for key specifier, plus 10 for the alternate-key field lengths, plus 4 for the primary key length). For unique alternate keys (specified by including the UNIQUE option in the SET ALTKEY command), the key length is 12 bytes (2 for key specifier, plus 10 for alternate-key field lengths), and the record length is 16.

Next, use the ALTER command to change the file label for FILEA so that it specifies FILEB as the alternate-key file and contains the key specifier "XY".

Finally, use the LOADALTFILE command to load the alternate-key records into the alternate-key file. Note that an index-block slack percentage of 0 is the default value.

Modifying Partitioned Files

Changing partitioned files involves making changes to specific partitions, then modifying the attribute of the file to reflect these changes. The examples that follow illustrate how to:

- Add new partitions
- Move partitions
- Reload partitions
- Increase the extent sizes of existing partitions

Moving a Partition to a New Volume

For this example, assume you have a partitioned file named $VOL1.SVOL.PARTFILE. Secondary partitions of this file reside on volumes $VOL2 and $VOL3. You want to move the secondary partition on $VOL2 to the volume $NEW.

To move the partition, enter:

```
-VOLUME $VOL1.SVOL
-DUP $VOL2.PARTFILE, $NEW.*, PARTONLY
-ALTER PARTFILE, PART ( 1, $NEW )
-PURGE $VOL2.PARTFILE
```

The DUP command creates a new copy of the secondary partition named $NEW.SVOL.PARTFILE. You can copy a secondary partition only if you include the PARTONLY option in your DUPLICATE command.

You use the ALTER command to change the file label of the primary partition. After the ALTER command is executed, the file label indicates that the secondary partition resides in the file $NEW.SVOL.PARTFILE.

Loading a Partition of an Alternate-Key File

For this example, suppose you have a key-sequenced, partitioned file named $VOL1.SVOL.PRIFILE. The file has alternate keys. The length of its primary-key field is 10. It has three alternate-key fields with key specifiers "F1", "F2", and "F3". The length of each alternate-key field is 10 bytes.

All the alternate-key records are contained in one alternate-key file that is partitioned over three volumes. (To create such an alternate-key file, you must enter the SET NO ALTCREATE command to prevent automatic creation of an alternate-key file, and then create the partitioned alternate-key file separately.) Each volume contains the alternate-key records for one alternate-key field. This is possible because the key specifier for each alternate-key field is also the partial-key value for the secondary partitions.

The primary partition of the partitioned, alternate-key file is $VOL1.SVOL.AFILE. It contains the alternate-key records for the key specifier "F1".

Partitions of the alternate-key file AFILE also reside in volumes $VOL2 and $VOL3. $VOL2.SVOL.AFILE contains the alternate-key records for the key specifier "F2". $VOL3.SVOL.AFILE contains the alternate-key records for the key specifier "F3".

To load the alternate-key records for the key specifier "F2" into the file $VOL2.SVOL.AFILE, enter:

```
:FUP
-VOLUME $VOL1.SVOL
-CREATE TEMP, EXT 30
-BUILDKEYRECORDS PRIFILE, TEMP, "F2", RECOUT 22, BLOCKOUT 2200
-LOAD TEMP, $VOL2.AFILE, PARTOF $VOL1, RECIN 22, BLOCKIN 2200
-PURGE ! TEMP
```

The CREATE command creates the disc file to be used for output from the BUILDKEYRECORDS command. Next, BUILDKEYRECORDS generates alternate-key records which can then be loaded into the new file. The BUILDKEYRECORDS BLOCKOUT option specifies record blocking to improve the efficiency of disc write operations.

You finish by using the LOAD command to load the secondary partition $VOL2.SVOL.AFILE. Because you did not include the SORTED option, records are first sorted before they are loaded. When loading the file, you must use the RECIN option to specify the same record blocking you specified with the RECOUT option of the BUILDKEYRECORDS command.

Increasing the Extent Size of a Partition

This example uses a key-sequenced partitioned file ($VOL1.PARTFILE.PART) with secondary partitions in volumes $VOL2 and $VOL3.

To increase the extent size of the partition in $VOL2, enter:

```
-VOLUME $VOL1.PARTFILE
-ALTER PART, PART ( 1, $VOL2, 120, 12 )
-RENAME $VOL2.PART, $VOL2.TEMP, PARTONLY
-SET LIKE $VOL2.TEMP
-SET EXT ( 120, 12 )
-CREATE $VOL2.PART
-DUP $VOL2.TEMP, $VOL2.PART, OLD, PARTONLY
-PURGE $VOL2.TEMP
```

The ALTER command changes the file label of the primary partition so that it includes the new extent size of the secondary partition in $VOL2.

Next, the RENAME command keeps the data from the secondary partition in a temporary file.

The SET LIKE command recreates the file-creation parameters of the original secondary partition. Then you change the extent size with a SET EXT command. Your CREATE command recreates the secondary partition with a larger extent size.

Finally, you move the data from the temporary file to the newly created partition with the DUP command.

Note that while you can rename a file with the RENAME command if it is open for read-write or write-only access, you cannot duplicate the file with the DUPLICATE command. Thus, you must ensure that the partition is not being written to if the preceding sequence of operations is to succeed. (File-access modes are discussed in the ENSCRIBE Programming Manual.)

Adding Partitions

You can add partitions to relative and entry-sequenced files that do not already have them, but not to key-sequenced files. For this example, you have a nonpartitioned relative file named $VOL1.SVOL.RELFILE. To add a partition to this file, enter:

```
-VOLUME $VOL1.SVOL
-SET LIKE RELFILE
-SET PARTONLY
-CREATE $VOL2.RELFILE
-SHOW EXT
    EXT ( 100 PAGES, 10 PAGES )
-ALTER RELFILE, PART ( 1, $VOL2 , 100, 10 )
```

First, enter a SET LIKE command to set the creation parameters to those of the original file, $VOL1.SVOL.RELFILE. Enter SET PARTONLY to specify that any file you create is to be a secondary partition. The new partition is created on volume $VOL2.

You enter SHOW EXT to display the extent sizes of the original file. Use the ALTER command to change the file label of $VOL1.SVOL.RELFILE to show that it is the primary partition of a partitioned file with a secondary partition in the volume $VOL2.

If you now want to add a third partition to the file that will reside in the volume $VOL3, enter:

```
-CREATE $VOL3.RELFILE
-ALTER RELFILE, PART ( 2, $VOL3, 100, 10 )
```

SECTION 7

INTRODUCTION TO BACKUP AND RESTORE

BACKUP is a utility program that copies disc files onto magnetic tape. The RESTORE utility program returns files stored with BACKUP from tape to disc. There are also two new utilities, BACKUP2 and RESTORE2, that work with files recognized by the existing disc process (DP1) and also files recognized only by the optional new disc process (DP2). The complete syntax for all these utilities is in the GUARDIAN Operating System Utilities Reference Manual.

Also refer to the DP1-DP2 File Conversion Manual for information on how to use BACKUP2 and RESTORE2 to convert files from DP1 format to DP2 format and vice versa.

WHY USE BACKUP AND RESTORE?

Together, BACKUP and RESTORE allow you to:

- Safeguard important information stored in disc volumes

  (If the original files are damaged or destroyed in a major system malfunction or castastrophe, you can replace them with the tape copies made with BACKUP.)

- Save disc space

  (You can use BACKUP to save files that are not being used and to preserve seldom-used information in tape archives. You can also use BACKUP and RESTORE for compacting data on discs.)

WHO USES BACKUP AND RESTORE?

The BACKUP and RESTORE programs are used by anyone who needs to keep data in archives or who needs to transfer data between systems that are not part of a network. BACKUP and RESTORE are also commonly used by system operators as part of routine preventive maintenance. For information about the use of BACKUP and RESTORE in system operations, see the System Operations Manual for your system.

SECTION 8

USING BACKUP AND RESTORE

This section contains information for new users of the BACKUP and RESTORE programs. It describes:

- How to enter BACKUP and RESTORE commands
- When to use BACKUP and RESTORE command options

You should be familiar with the information in Section 2, "Basic Uses of COMINT," before reading this section. For detailed reference information on BACKUP and RESTORE as well as on BACKUP2 and RESTORE2, see the GUARDIAN Operating System Utilities Reference Manual.

ENTERING BACKUP COMMANDS

You enter the BACKUP command through COMINT, the GUARDIAN operating system command interpreter. To run the BACKUP program, enter BACKUP followed by the name of the magnetic tape drive you want to use, a comma, and a list of the files you want to copy.

Here is an example of a BACKUP command:

```
:BACKUP $TAPE1, $DATA.YOURBABY.*
```

This command backs up all the files in the subvolume $DATA.YOURBABY onto tape, using the tape drive $TAPE1.

The syntax term for the files you list in a BACKUP or RESTORE command is <fileset-list>. You can use <fileset-list> with many other utilities and COMINT commands. This section discusses in detail the syntax rules for specifying a file-set list.

Your BACKUP command can also include any <run-option> listed in the description of the RUN command in COMINT (see Section 2 of this manual and Section 2 of the GUARDIAN Operating System Utilities Reference Manual). Among these options is the OUT <list-file> option, which sends listings from the BACKUP program to an existing disc file or printer.

For example, if you enter this BACKUP command, the listing from BACKUP is sent to the file LISTFILE in the current default subvolume:

```
:BACKUP / OUT LISTFILE / $TAPE2, $DATA.OURBABY.*
```

You can also include command options that are specific to the BACKUP and RESTORE programs. These options are fully described in the Section 4 of the GUARDIAN Operating System Utilities Reference Manual. Several of the options are also described later in this section. For example, to generate a list of the files you are backing up, you include the LISTALL option of the BACKUP program:

```
:BACKUP / OUT LFILE2 / $TAPE, $DATA.FLYBABY.*, LISTALL
```

A COMINT command can contain at most 132 characters on a single command line or at most 528 characters if the command continues beyond one command line. For this reason, it may not be possible to include all the files that you want to back up in a BACKUP command. In this case, you may want to use an input file to enter your BACKUP command parameters (the parts of the BACKUP command that follow the run option list enclosed within slash characters). Instead of typing the parameters (such as a very long file-set list) when you enter your BACKUP command, you can store the parameters in a file and name the file as an IN option when you type the BACKUP command.

For example, you can start a BACKUP process using as parameters the options contained in the file FRED.FLIST by entering:

```
:BACKUP / IN FRED.FLIST /
```

For more information about BACKUP and RESTORE input files, see the description of the IN option in the syntax description for BACKUP in the GUARDIAN Operating System Utilities Reference Manual.

SPECIFYING A FILE-SET LIST FOR BACKUP

You can back up any files to which you have read access. (See Section 12 for a discussion of file-access modes.) To list the files you want to copy, you specify a file-set list in your BACKUP command. A file-set list is a list of one or more file sets. Both terms are defined and discussed in this subsection.

A file set is a set of one or more files. A specification for a file set is very much like a file name. You can specify the name of a system, volume, or subvolume where the file set resides, just as you would for a single file. As in expanding a partial file name, if any of these are omitted, BACKUP assumes the current default values. However, you can also include the following:

- An asterisk (*) in place of a volume name or a subvolume name

  If you use an asterisk for a volume name, BACKUP copies all files from all disc volumes in the system. If you include an asterisk in place of a subvolume name, BACKUP copies files from all subvolumes in the volume or volumes you specify.

  Note that if you use an asterisk for the volume name, you must also use an asterisk for the subvolume name.

- An asterisk (*) in place of the file name

  BACKUP then copies all files in the volumes or subvolumes you specify in the file set.

  Note that if you use an asterisk for the subvolume name, you must use an asterisk for the file name as well.

For example, you can copy all the files in your current default subvolume by entering:

```
:BACKUP $TAPE, *
```

Because no volume or subvolume name is given in this example, BACKUP uses the current default values for volume and subvolume.

You can copy all the files in the volume $MANUF by entering:

```
:BACKUP $TAPE, $MANUF.*.*
```

You can copy all the files in the system by entering:

```
:BACKUP $TAPE, *.*.*
```

A file-set list can be a single file set or several file sets. To include more than one file set in a file-set list, do the following:

- Enclose the file-set list within parentheses.
- Separate the file sets in the file-set list with commas.

For example, you can copy the files in your current default subvolume and all the files in the volume $MANUF by entering:

```
:BACKUP $TAPE, (*, $MANUF.*.*)
```

When you refer to the syntax descriptions in the GUARDIAN Operating System Utilities Reference Manual, remember that a <fileset> or a <fileset-list> can be the name of a single file. You can copy a single file or a set of files using BACKUP.

USING BACKUP COMMAND OPTIONS

You can tailor the BACKUP program by specifying one or more BACKUP command options. Some of the more commonly used options are described in this subsection. In addition to these options, other BACKUP options allow you to do the following:

- Back up open files and request a prompt when BACKUP encounters an open file (OPEN and MSGONLOCK options)
- Give new volume and subvolume names to the files backed up on tape (VOL option)
- Ignore any data errors encountered (IGNORE option)
- For partitioned files, back up only those partitions defined in the file-set list you specify (PARTONLY option)
- Back up files that are audited by the Transaction Monitoring Facility (TMF) (AUDITED option)
- Select a starting point in the file-set list where the backup is to begin (START option)
- Verify the checksums for the tape files (VERIFYTAPE option)

For more information about BACKUP command options, see Section 4 of the GUARDIAN Operating System Utilities Reference Manual.

Using the DENSITY and BLOCKSIZE Options

If you use a Tandem Model 5106 tape drive when backing up files, you can specify one of three recording densities:

- 6250 bpi (bytes per inch)
- 1600 bpi
- 800 bpi

To specify a recording density, include the DENSITY option with a density setting in the BACKUP command, as in this example:

```
:BACKUP $TAPE1, $RIVER.RUN.*, DENSITY 1600
```

The tape drive then produces a tape with the recording density you specify. If you do not specify a density with the DENSITY option, the current density setting of the tape drive determines the recording density.

You can also specify the size of the blocks written to tape with the BLOCKSIZE option. For the NonStop system, the default block size is 8. However, for the NonStop 1+ system, the block size is always 2, regardless of the size specified with the BLOCKSIZE option. If you want to back up files with a NonStop system and restore them to a NonStop 1+ system, you must specify a block size of 2. Because of the difference in block sizes, you cannot use BACKUP and RESTORE to transfer files from a NonStop 1+ system to a NonStop system.

For example, to back up all the files on the $SYSTEM volume, and to specify a block size of 2, enter:

```
:BACKUP $TAPE, $SYSTEM.*.*, BLOCKSIZE 2
```

Using the LISTALL Option

The BACKUP utility produces a listing or display that includes the names of files being backed up. By default, the BACKUP listing includes only those files which have generated error or warning messages. You can, however, get a list of all the files that have been backed up, as well as any error messages which occur, by including the LISTALL option in a BACKUP command.

BACKUP displays its listing at the OUT file you name in your command; if you omit the OUT option, BACKUP uses your terminal. Regardless of the listing device you specify, messages (including errors and instructions) are also sent to the terminal where you entered the BACKUP command.

This example shows a listing given by the BACKUP program without the LISTALL option:

```
   :BACKUP $TAPE, *

GUARDIAN FILE BACKUP PROGRAM - T9024B00 - (18MAR85)   SYSTEM \ZOO
   VOLNAME SVOLNAME FILENAME REEL ERROR  ADDRESS 28 AUG 83 17:37

   $SYSTEM  SYSTEM  COPY      01   120              *** NOT DUMPED
   $SYSTEM  SYSTEM  GPLIB     01   012              *** NOT DUMPED
   $SYSTEM  SYSTEM  PRIVDECS  01   120              *** NOT DUMPED

FILES DUMPED=00014    FILES NOT DUMPED=00003
```

Here, three attempts to copy files result in errors. The ERROR column shows code numbers that identify file-system errors.

BACKUP with the LISTALL option gives you a complete listing of the files backed up and the files that generate errors or warnings.

The following example illustrates the output from BACKUP when you specify the LISTALL option:

```
:VOLUME $SYSTEM.SYSTEM
:BACKUP $TAPE, *, LISTALL

GUARDIAN FILE BACKUP PROGRAM - T9024B00 - (18MAR85)   SYSTEM \ZOO
   VOLNAME SVOLNAME FILENAME REEL ERROR  ADDRESS 28 AUG 84 17:22

   $SYSTEM  SYSTEM  COPY      01   120             *** NOT DUMPED
   $SYSTEM  SYSTEM  DUMP      01
   $SYSTEM  SYSTEM  EEDIT     01
   $SYSTEM  SYSTEM  FLYFILE   01
   $SYSTEM  SYSTEM  GPLIB     01   012             *** NOT DUMPED
   $SYSTEM  SYSTEM  GPLIBDEC  01
   $SYSTEM  SYSTEM  LITE      01
   $SYSTEM  SYSTEM  OLDEDIT   01
   $SYSTEM  SYSTEM  OTAL      01
   $SYSTEM  SYSTEM  PRIVDECS  01   120             *** NOT DUMPED
   MOUNT NEXT REEL <cr>
   $SYSTEM  SYSTEM  PUP       02
   $SYSTEM  SYSTEM  TAL       02

   FILES DUMPED=00009   FILES NOT DUMPED=00003
```

In this example, the BACKUP operation required two reels. The "MOUNT NEXT REEL" prompt appeared when the first reel of tape was full. After mounting another tape, the user pressed RETURN to restart the BACKUP process. The file $SYSTEM.SYSTEM.PRIVDECS was listed twice because part of it was copied onto the first reel, while the rest was copied onto the second.

With the LISTALL option, you can create a permanent record of the files that were copied. To do this, specify both the LISTALL option and an OUT file in your BACKUP command. You can then print a copy of the output file and use it to label the backup tape.

Using the NOT Option

If there are files in the file-set list for your BACKUP command that you do not want to copy, you can specify them with the NOT option. You use the same form to specify files to ignore that you use for specifying the file-set list you want to back up.

As an example, suppose you want to back up all the files in subvolume $CAFFE.MED except MITSLAG and MOKA. Enter:

```
:BACKUP $TAPE1, $CAFFE.MED.*, NOT (MITSLAG, MOKA)
```

Using the PARTIAL Option

BACKUP with the PARTIAL option selectively backs up files. You can back up those files which were modified after a specified date and time by including the PARTIAL option. Enter PARTIAL followed by the date (the first three letters of the month, the day of the month, and the year, with spaces separating them), a comma, and the time (using a 24-hour clock).

Suppose that today is May 25, 1985. You can back up all files in the system that were modified after 6 p.m. yesterday by entering:

```
:BACKUP $TAPE1, *.*.*, PARTIAL MAY 24 1985, 18:00
```

ENTERING RESTORE COMMANDS

You enter RESTORE commands, like BACKUP commands, through COMINT. The form of the RESTORE command is nearly identical to that of the BACKUP command. Enter RESTORE followed by the name of the magnetic tape drive you want to use, a comma, and a file-set list that names the files you want to restore. Note that you must have write access to a file currently on disc if you want to restore a file with the same name.

You can include RESTORE options in your command, as described in the syntax description in the GUARDIAN Operating System Utilities Reference Manual. You can also include any run option for the RUN command in COMINT; and a RESTORE command, like a BACKUP command, can name an IN file that contains the command parameters.

Here is an example of a RESTORE command:

```
:RESTORE / OUT LIST / $TAPE1, (*, $DATA.BIGBABY.*), LISTALL
```

Here, you use the tape drive $TAPE1 to restore to disc all files with your current default subvolume name and files with the subvolume name $DATA.BIGBABY. A listing of all the files that are backed up and any errors which occur is sent to the file LIST on your current default subvolume.

USING RESTORE COMMAND OPTIONS

This section describes some of the commonly used RESTORE command options. Besides the options discussed here, other RESTORE options allow you to:

- Select the starting point in the file-set list where the restoration is to begin (START option)
- Restore the latest versions of the files on the first backup tape in a set of tapes (REBUILD option)
- Verify the restored data (VERIFY option)
- Ignore any data errors encountered (IGNORE option)
- Restore files that were open at backup (OPEN option)
- For partitioned files, restore only those partitions that are defined in the file-set list (PARTONLY option)
- For partitioned files, restore only those partitions whose primary partitions reside on a specific volume (PARTOF option)
- Rewind the tape and leave it online at the end of the restoration (NOUNLOAD option)
- Restore files that were audited by the Transaction Auditing Facility (TMF) (AUDITED option) and restore audited files as nonaudited files (TURNOFFAUDIT option)

For descriptions of all RESTORE command options, see Section 4 of the GUARDIAN Operating System Utilities Reference Manual.

Using the LISTALL Option

The RESTORE program listing, like that of BACKUP, lists only those files for which errors or warnings are generated. You can get a listing of all the files restored to disc by including the LISTALL option in the RESTORE command. In addition, if you include the LISTALL option but do not specify a file-set list to be backed up, RESTORE lists all the files on the tape without restoring any files.

For example, to place a list of the files on $TAPE1 in the file L5 without restoring the files, enter:

```
:RESTORE / OUT L5 / $TAPE1,,LISTALL
```

Using the KEEP Option

The KEEP option allows you to preserve files that are currently on disc when you restore files with the same name. Using KEEP means that if a disc file has the same name as a file which is being restored, the file already on disc is preserved. If you do not include the KEEP option, such a disc file is replaced by the file on the tape.

Using the TAPEDATE and MYID Options

If you include the TAPEDATE option, the timestamp of a file restored to disc is the timestamp in effect when the file was backed up. If you omit TAPEDATE, the timestamp of restored files is the time when the RESTORE was performed.

The MYID option allows you (the user who runs the RESTORE program) to become the owner of the files that are restored. If you do not include the MYID option, each file that is restored belongs to the user who owned it when it was backed up.

Using the NOT Option

If there are files in the file-set list that you do not want to restore, specify them with the NOT option in the same way you specify the file-set list to be backed up or restored.

For example, the next command does the following:

- Restores all the files in volume $ECLAIR except the files in subvolumes LINZER and PETIT4
- Keeps any files currently on disc that have the same name as files in the RESTORE command file-set list
- Retains the backup-timestamp of the files being restored
- Gives you ownership of the restored files

```
:RESTORE $TAPE1, $ECLAIR.*.*, NOT (LINZER.*, PETIT4.*),&
:KEEP, TAPEDATE, MYID
```

Using the VOL Option

By default, the RESTORE program restores files to the volume and subvolume where they resided when they were backed up. You can, however, restore files to a new volume, a new subvolume, or both by including the VOL option. Any files you restore will reside in the volume or subvolume you specify in the VOL option. For files that are restored, any parts of the file name you do not specify with the VOL option remain the same.

Suppose you want to restore this file-set list to disc:

```
($TOLSTOY.NOVELS.*, $PROUST.*.*)
```

You want the files you restore to reside in the subvolume $QUICK.READING. Enter:

```
:RESTORE $TAPE, ($TOLSTOY.NOVELS.*, $PROUST.*.*), VOL &
:$QUICK.READING
```

If, by using the VOL option, more than one file to be restored has the same file name, only the last file with the same name is restored. As an example, consider the following file-set list:

```
($TOLSTOY.NOVELS.BIG1, $PROUST.RTP.BIG1)
```

Suppose you attempt to restore both of the files in this file-set list to the subvolume $GIANT.NOVELS by entering:

```
:RESTORE $TAPE, ($TOLSTOY.NOVELS.BIG1, $PROUST.RTP.BIG1)&
:, VOL $GIANT.NOVELS
```

Only one file named $GIANT.NOVELS.BIG1 is restored--the file originally named $PROUST.RTP.BIG1.

SECTION 9

INTRODUCTION TO THE SPOOLER

WHAT IS THE SPOOLER?

The Tandem spooler is a set of programs that act as an interface between the users and their applications (programs), and the print devices of a system.

The spooler receives output from an application and stores it on disc. This output can be material in EDIT format, such as memos or reports, or object code from a compiled program. When the designated print device becomes available, the output is printed.

Features of the Tandem spooler include:

- Continuous operation--the spooler keeps working even in the event of processor failure.
- A flexible routing structure--you can change the destination of a job after it enters the spooler.
- No programming necessary--you can send your output to the spooler simply by specifying the spooler as your OUT file.
- Interactive control--using the PERUSE program, you can inspect or alter the status of your job, examine the data you have sent to the spooler, and change the destination of your output.
- Operator control--the SPOOLCOM program permits operators to initialize and modify all of the spooler components.

WHY USE THE SPOOLER?

The spooler, because it offers a consistent interface to all devices in the system, is simple and easy to use. Application programs can write data to the spooler as easily as they write to a disc file. Tandem programs (for example, TAL, TGAL, COBOL) can have their output directed to the spooler with the OUT <listfile> option of the RUN command.

The spooler ensures maximum efficient use of the print devices in the system, while insulating the application from device-dependent considerations. Many programs print only a few pages over their entire execution. It would not be very wise to allow those programs to have exclusive access to a line printer, while other applications wait their turns. Similarly, if that particular printer is unavailable, the program would have to redirect its output to a different device and handle all of the device-dependent protocols.

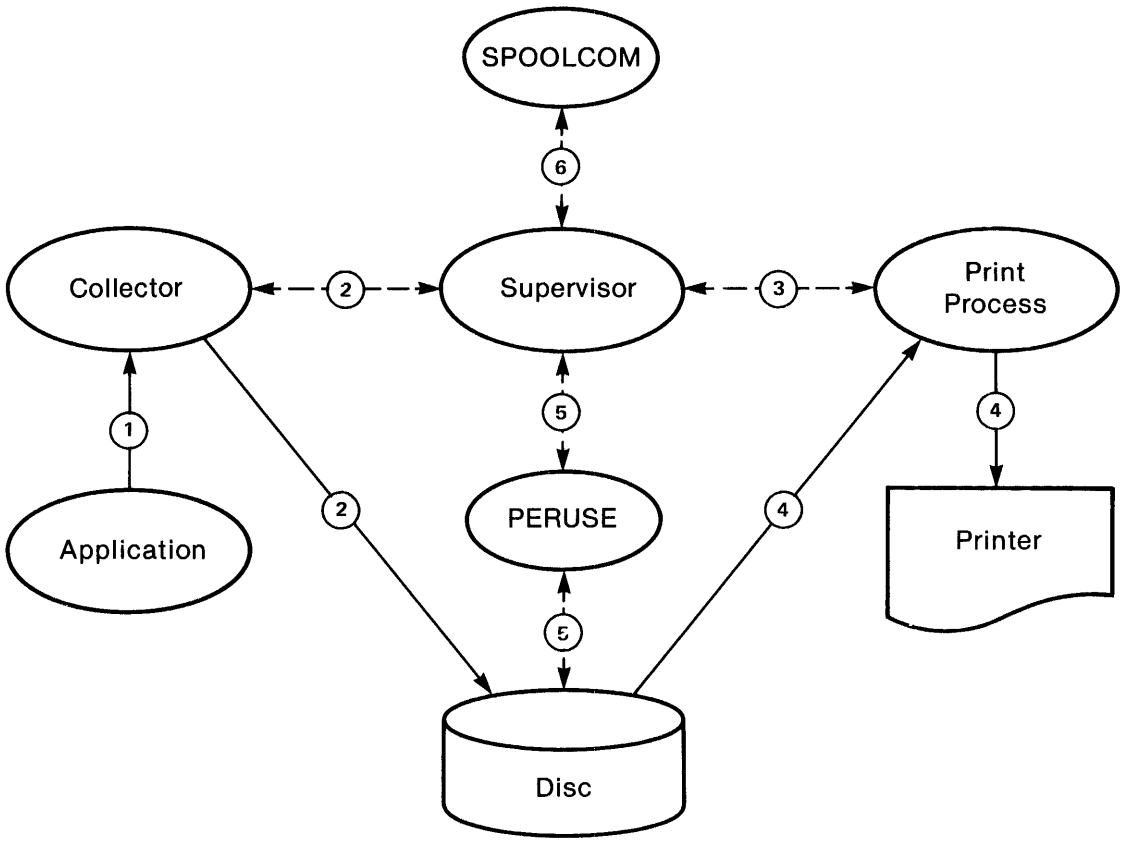
The spooler allows many different programs to send their data to the same printer at the same time. It keeps the data separate and prints the output of each program in an orderly manner.

There are alternatives to using the Tandem spooler, and for some special applications they may be more efficient. For most users, however, the spooler is the best solution to the problem of producing hard-copy output.

SPOOLER COMPONENTS

The Tandem spooler consists of the following programs that interact in order to perform all of the operations of the spooler. They are illustrated below in Figure 9-1.

- Supervisor--monitors and communicates with the other programs and decides when and where to print jobs. There is only one supervisor in each spooler.
- Collectors--accept output from applications and store the output on disc. There can be any number of collectors associated with a given spooler.
- Print processes--retrieve the output stored on disc by the collector and print it on a device. Each print device in the spooler system has a print process associated with it.



Dotted lines represent interprocess messages. Solid lines represent the path taken by spooled data.

1. *Application* sends its output to the collector process.
2. *Collector* writes data to disc, and informs the supervisor that it has accepted a job.
3. *Supervisor* informs the print process where the job can be found on the disc.
4. *Print process* reads the job from disc, and writes the job to the printer.
5. *PERUSE* obtains job information from the supervisor, or lets you examine data on the disc.
6. *SPOOLCOM* enables system operators to modify the status of spooler components.

S5055-008

Figure 9-1. Spooler Components

- SPOOLCOM--is your interface to the spooler system.

  You can use SPOOLCOM interactively to get the status of all spooler components, start offline devices, and cause a device to skip pages when printing their jobs. Appendix E contains a brief syntax of the SPOOLCOM commands that can be executed by any user. A complete description of SPOOLCOM syntax, considerations, and examples is contained in the GUARDIAN Operating System Utilities Reference Manual.

  System operators use SPOOLCOM to create and initialize the components of the spooler system.

- PERUSE--allows you to control and monitor your job.

  You run PERUSE interactively from a terminal, and it accesses the spooler to translate your commands into messages to the supervisor, which then carries out your instructions. Appendix F contains a brief syntax of the PERUSE commands. A complete description of PERUSE syntax, considerations, and examples is contained in the GUARDIAN Operating System Utilities Reference Manual.

SPOOLCOM OR PERUSE--WHICH SHOULD YOU USE?

Many commands in SPOOLCOM and PERUSE are equivalent to each other. For example, both DEV commands display the status of devices; many of the PERUSE commands seem to be portions of the SPOOLCOM JOB command (COPIES, DEL, FORM, HOLDAFTER, OWNER, PRI). So the beginning user may wonder which one to use. Is one better than the other, depending on the application? The following are only suggestions; you will develop your own preferences and rules of thumb.

PERUSE offers more flexibility for monitoring and changing your own jobs. When you enter it, you immediately see what jobs you have in the spooler system, identified by their number and accompanied by other important information. You can examine your job page by page or line by line, and you can use the LIST command to locate specific word strings. With fewer keystrokes than SPOOLCOM, you can change a job attribute and send your job to a device to print, or you can delete it.

SPOOLCOM is meant primarily as a tool for system operators to monitor and control the spooler system, and for programmers to move their programs through the spooler system in the manner they desire. Its primary use for other users is to provide status information on the spooler components.

SPOOLCOM DEV is useful when you want to know device status or queue length. It is quicker to noninteractively use SPOOLCOM DEV on a specified device than it is to enter PERUSE and do a DEV there, because you don't have to wait to enter PERUSE and display its startup message. For example, in SPOOLCOM the sequence is:

```
:SPOOLCOM DEV $HT1
 DEVICE                STATE          FLAGS    PROC   FORM
 $HT1                  JOB 1732         T      $SPLA

    JOB  LOCATION        DEVICE                SEQ    COPY  PAGE
    1732 #LP1.LP1        $LP1                  PRINT  1     10
    1845 #LP1.LP1        $LP1                  2      1     30
:
```

After entering PERUSE, you must wait for it to display its startup message before executing the DEV command and exiting:

```
:PERUSE
PERUSE - T9101C12 - (18MAR85)    SYSTEM  \TS

  JOB   STATE  PAGES  COPIES  PRI  HOLD  LOCATION   REPORT
  1732  READY  1      1       4          #LP1       ACCTG JANE

_DEV $LP1

DEV STATE: PRINTING      FORM:

  JOB   OWNER    PAGES  WAIT       FORM
  1732  033,019  10     00:01:58
  1845  021,080  30     00:03:45

_E
:
```

Note, however, that PERUSE DEV supplies two pieces of information that SPOOLCOM DEV does not: the owner of each job and the estimated waiting time for that job to finish printing.

SPOOLER JOBS AND JOB ATTRIBUTES

When you request the spooler to print some information at a print device, the request is called a job. The spooler assigns each job a job number in the range of 1 to 4095. In addition to a number, jobs have six primary attributes: priority, copies, report name, form name, state, and location. A description of these attributes follows; location is described below under "Routing Structure."

Job Priority

Job priority determines when a job will print in relation to other jobs queued for the same device. The spooler maintains a device queue for all print devices. Higher-priority jobs are placed near the front of the queue, while lower-priority jobs are placed near the end of the queue.

Job Copies

Job copies specifies the number of copies of the job that the spooler should print.

Job Report Name

The job report name is the name the Tandem print process prints in banner-head letters in the header message of the job. The header message is described below under "Devices and Device Attributes."

Job Form Name

The form name of a job allows you to guarantee that your job is printed only on a device that has the same form name. Form name can be used to ensure that jobs requiring special handling print on devices that are appropriately equipped.

For example, an application program may produce a job that fills out W-2 forms. To be useful, the job must be printed on a device loaded with special W-2 form paper. If the job has a form name of "W2," it will only print on a device with the same form name.

The form name "W2" is assigned to that device at the time special paper is loaded. This allows the "W2" job to print and prevents jobs with a different form name from printing on that device.

Job State

Job status is described by the job state. A job is always in one of four states: OPEN, READY, PRINT, or HOLD. These states are illustrated in Figure 9-2.

During the OPEN state, a job is being added to the spooler. For example, if you send the output from a TGAL execution to the spooler, the job is in the OPEN state until TGAL completes execution.

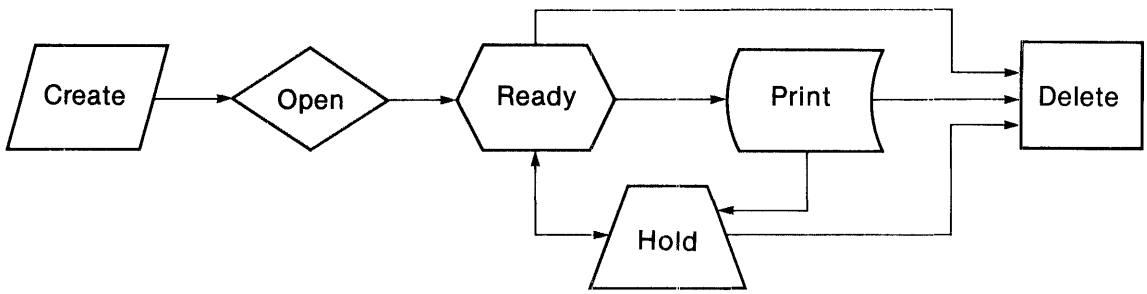
In the READY state, a job is ready to print, but it has not yet begun to print, usually because another job is printing.

A job is in the PRINT state while it is being printed. This is usually the last state before the job is deleted from the spooler. Using PERUSE to set the hold-after-printing flag prevents the spooler from deleting a job after printing.

You can put a job in the HOLD state by using a PERUSE request. The job will not be printed while HOLD is in effect. A job on HOLD will remain in the spooler indefinitely until you delete it or remove the HOLD.

You can put a job on hold at any time. If the job is in the READY or PRINT states, it is placed on hold immediately. If it is still OPEN, it is placed on hold after the job has been completely spooled.

The life cycle of a job starts at the OPEN state while the application is writing the data to the collector. The collector stores the data in a disc file. When the application is done sending data, the job is in the READY state. Finally, the job enters the PRINT state. When it has finished printing, the spooler deletes it, unless the hold-after-printing flag is on.



S5055-009

Create: You create a job when your application opens a file to a collector and performs a write; for example:

TGAL / IN <filename>, OUT $s.<device> /

Open: The job is open while the application is writing to the collector, which in turn writes the data to a disc file.

Ready: The job enters the READY state when the application closes the file to the collector.

Hold: You can place a job on hold only when it is in the ready or print state. This removes it from the device queue.

Print: The job waits in the device queue until it is ready to print. While in the PRINT state, the job is printing on the output device. If you place the job on hold, it immediately stops printing.

Delete: If the hold-after-printing flag is not on, the job leaves the spooler after it has finished printing. You can also delete a job from the spooler using a specific request. If the job is printing when you make that request, the job immediately stops printing and leaves the spooler.

Figure 9-2. The Life Cycle of a Job

DEVICES AND DEVICE ATTRIBUTES

A device is a print device that produces a hard-copy listing of your job. Every device is controlled by a print process. Devices (and print processes) have four attributes that you should be aware of: form name, header message, device state, and selection algorithm.

Device Form Name

The form name of a device defines the type of job that can be printed on it. Only jobs whose form name is the same as that of a given device can be printed on the device. An example of how form name might be used is shown above under "Spooler Jobs and Job Attributes."

Device Header Message

The header message of the Tandem print process includes each job report name, location, job number, form name, and date and time of printing. The header message for a device can be turned on or off by the operator. When the header message is turned on, it prints on the first page of the job. The report name and location are printed in big banner-head letters, as shown in Figure 9-3.

If the system operator specifies a batch header, the job information prints out on two of the three trailer pages as well as on the first two pages of each job. The trailer pages have printing over the page folds, enabling jobs printed on accordion-fold paper to be separated easily. The two-page header message always appears on the top page, regardless of how the job is folded.

If the header message is turned off, jobs print consecutively with only a new page to indicate the beginning of the next job.

The actual header message produced depends on the print process controlling the device. The headers described above are produced by the Tandem print process. If a device is controlled by a user-written print process, it can produce almost any kind of a header or none at all.

Figure 9-3. Sample Header Page

Device States

The device state describes the status of the device. There are six possible device states:

Printing	The device is currently printing a job.
Waiting	The device is idle and waiting for a job to print.
Offline	The device is not available for printing.
Suspended	The device is in the process of printing a job but has been suspended by the operator for some reason (for example, to change ribbons).
Deverror	The device has produced a file-system error while printing. Operator intervention is required.
Procerror	The supervisor has determined that the print process for that device is not working correctly. Operator intervention is required.

Selection Algorithm

The spooler maintains a queue for each device which contains entries for each job that is to be printed on that device. The next job to be printed on a given device is the one at the head of that device queue.

As a rule, higher-priority jobs print sooner than those with a lower priority. However, the selection algorithm affects the order in which jobs print within the same priority level. If the selection algorithm is FIFO ON, jobs are placed at the end of the queue and wait their turn to be printed. If the selection algorithm is FIFO OFF, the spooler also allows for short jobs to print before longer jobs of the same priority.

For a complete description of the spooler job-selection algorithm, refer to "Queue Ordering" in the System Operator's Guide.

ROUTING STRUCTURE

The function of the routing structure is to direct jobs to print devices. The routing structure consists of a set of locations and print devices. Figure 9-4 illustrates the association of locations with print devices.

A location is the logical destination of a job; a print device is its physical destination. This distinction permits great flexibility when routing jobs. The spooler assigns each job a location at the time it enters the spooler system, and the job eventually prints on the device associated with that location.

Location names have two parts: a <group-name> and a <destination-name>. The group name is always preceded by a crosshatch symbol (#). Examples of location names are:

```
#LP.LPEAST
#LP.LPWEST
```

#LP is the name of a group, while LPWEST and LPEAST are names of destinations in #LP.

Broadcast and Nonbroadcast Groups

If you specify only the group name #LP as the location, the spooler supplies the destination. If the group is a nonbroadcast group, then the spooler routes the job to the destination that can print the job soonest. If the group is a broadcast group, then the job is routed to all of the destinations in the group and prints on all the devices associated with the group.

For example, assume that #LP.LPEAST is associated with a line printer on the east side of the machine room, and #LP.LPWEST is associated with a line printer on the west side of the machine room. If #LP is a broadcast group, then a job routed to #LP prints at both line printers, and two copies of the job are printed. If #LP is a non-broadcast group, then the first available line printer prints the job, and only one copy of the job is printed.

In either case, a job routed to #LP.LPWEST will print once at the line printer at the west end of the machine room. Your system operator should be able to tell you which locations are available to you, which print devices are associated with those locations, and which groups are broadcast groups.

Default Routing

The spooler has a special location, #DEFAULT. This is the location used when you do not specify a location for a job. For example, TGAL / IN DAYREPRT, OUT $S/ (which is equivalent to TGAL / IN DAYREPRT, OUT $S.#DEFAULT /) causes the spooler to send the output to #DEFAULT.

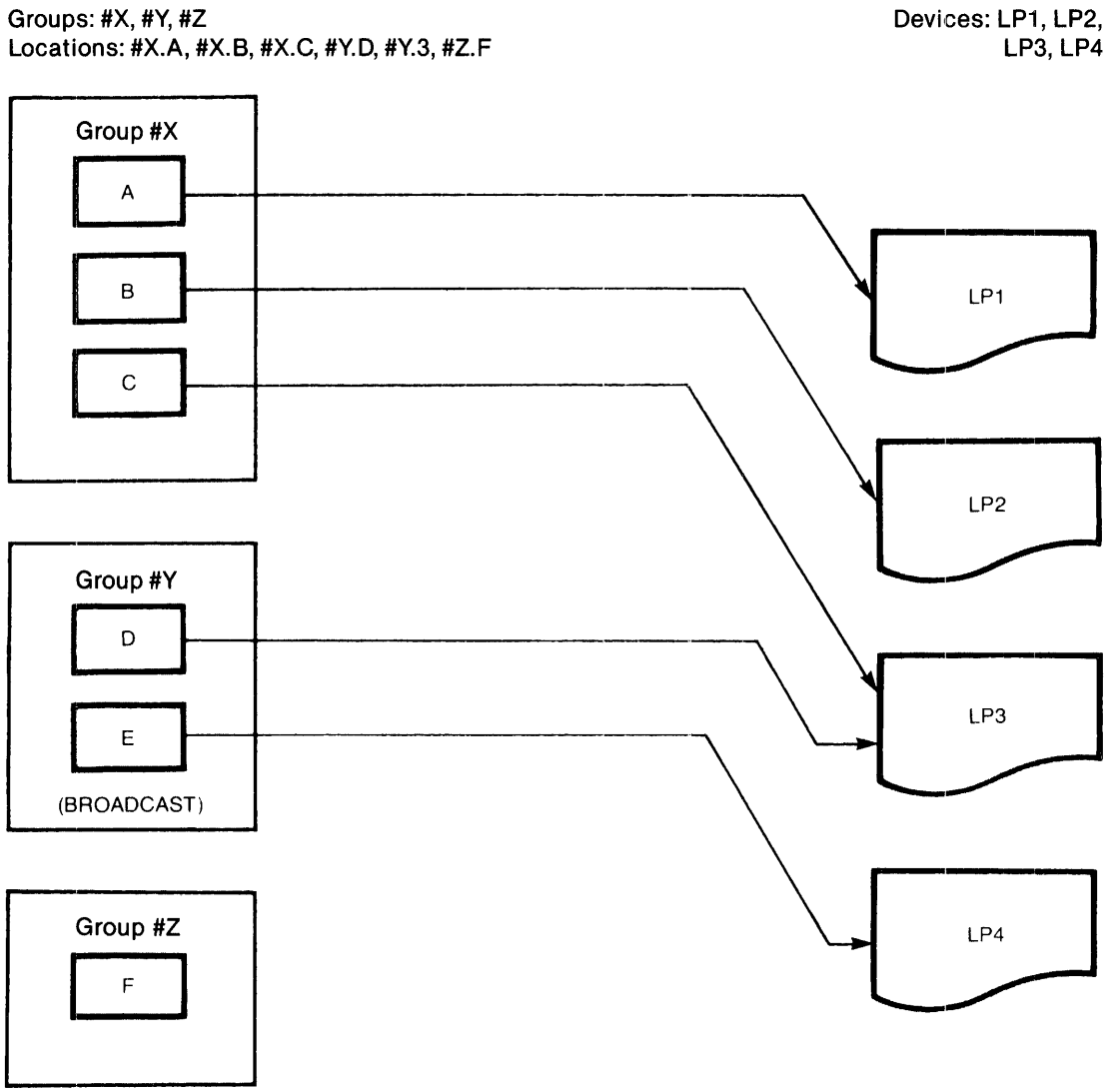
You should consult your system manager to find out which physical device or devices are associated with #DEFAULT.

Implicit Route Creation

When jobs are routed to nonexistent locations or groups, the spooler implicitly creates routes according to the following rules:

- When a job is routed to #X.1, where either the group #X doesn't exist or #X exists but #X.1 doesn't exist, the spooler creates the location #X.1.
- A job routed to a nonexistent group, #X, results in the creation of location #X.DEFAULT.

Many users choose the group #HOLD as a holding location so that they can examine their jobs with PERUSE before printing. You can use #HOLD or any nonexistent location for this purpose. However, if you want to print the job, you must change the location.



Note that a printer can be connected to several locations, but that each location is connected to at most one printer.

Jobs routed to #X print on one of LP1, LP2, or LP3.
Jobs routed to #X.B print on LP2.
Jobs routed to #Y print on both LP3 and LP4.
Jobs routed to #Z stay in the spooling system indefinitely. They can be printed by changing the location.

Connections are established or changed by the system operator.

Figure 9-4. The Routing Structure

HOW TO USE THE SPOOLER

The easiest way to spool a job is to designate a collector and a location as the OUT file when you run a program. For example, the following command line spools the output from a TGAL execution:

```
:TGAL /IN MYFILE, OUT $S.#LP.LPWEST/
```

Assuming that the collector $S exists, it creates the job, assigns a job number, and stores the data coming from TGAL in a disc file. When TGAL is finished, the spooler puts the job in the READY state. If there is a print device associated with #LP.LPWEST, then the job is printed when that device is free. (If there is no print device at that location, then the job sits in the spooler until you delete it or send it to an output device.)

As mentioned above, you do not have to specify the entire location name. The following example would also work:

```
:TGAL /IN MYFILE, OUT $S.#LP/
```

Assuming #LP is a nonbroadcast group, the job is printed on the first available print device associated with the group. In some systems, each group has only one print device associated with it, so the full location name is unnecessary.

The example below shows the TGAL command when the location is not specified:

```
:TGAL /IN MYFILE, OUT $S/
```

The job is sent to #DEFAULT and printed on one of the devices associated with that group.

SECTION 10

HOW TO USE PERUSE

WHAT IS PERUSE?

PERUSE is an interactive program that allows you to examine and change the attributes of a spooled job, as well as examine your job while it is in the spooler system. With PERUSE you can:

- Examine a job (for example, a lengthy compiler listing) before deciding whether to print it
- Display a job while it is being spooled
- Monitor changes in the status of a job
- Alter the attributes, such as location, number of copies, or report name of a job
- Print out specified pages rather than the entire file of a job that has been spooled

To perform these functions, PERUSE communicates with the spooler supervisor process and accesses the spooler disc files.

HOW TO ENTER PERUSE

To start a PERUSE process, enter "PERUSE" alone. PERUSE then displays a banner message:

```
:PERUSE
PERUSE  -  T9101B00  -  (18MAR85)     SYSTEM  \EAST
```

T9101B00 is the version of PERUSE.

(18MAR95) is the release date for this version of PERUSE.

\EAST is the system it is running on.

If you have any jobs in the spooler system when PERUSE begins execution, it lists them using the format shown in the example below:

```
JOB   STATE  PAGES  COPIES  PRI  HOLD  LOCATION   REPORT
456   PRINT  16     1       4          #LPP       ACCOUNT BARB
555   OPEN         1       4    B     #DEFAULT   ACCOUNT BARB
1435  READY  30     1       4     A    #LPRMT3    ACCOUNT BARB
```

Each line in the display stands for a different job. The meanings for each column are:

JOB shows the job number of each job, as assigned by the supervisor.

STATE shows the status of each job:

- OPEN — Job is still being collected by the spooler.
- READY — Spooler has finished collecting; job is queued and waiting to print.
- HOLD — Job has been placed on hold with the HOLD command.
- PRINT — Job is currently printing.

PAGES shows the number of pages in each job (OPEN jobs are still being collected, so the number of pages is not known).

COPIES shows the number of copies to be made of the job.

PRI shows the priority of the job. When a job is first sent to the spooler, its default priority is 4. The range is 0-7, with 0 being the lowest.

HOLD shows the hold status of the job:

A Hold-after-printing flag is on.
B Hold flag is on, but the job cannot be placed in HOLD (the job is OPEN).
X For some reason the job is in error (for example, TGAL abended while spooling the job).

LOCATION shows the location the job, which can be either real (a printing device) or not real (held in the spooler for PERUSE examination before sending to a printing device).

REPORT shows the report name that is printed in the job header message.

The display described above can be produced at any time during a PERUSE session with the JOB command.

Entering PERUSE Commands

After PERUSE displays your jobs, it issues a prompt character (_) to signal that it is ready to execute your commands. You can enter commands one to a line or several on the same line, separated by semicolons. The maximum length of the PERUSE command line is 132 characters. Each line is terminated with a carriage return; for example:

```
_J 123
_DEL
_EXIT
```

is the same as

```
_J 123; DEL; EXIT
```

Declaring the Current Job

Most PERUSE commands explicitly affect the current job, but at the beginning of a PERUSE session, there is no current job. You can declare a current job in three ways:

- Press the RETURN key or any of the function keys to set the most recently spooled job to the current job.
- Set the current job with the JOB command.
- Specify the current job implicitly: If at the time you enter PERUSE you issue a command that affects the current job,

PERUSE makes the job most recently spooled the current job. For example, if you give PERUSE a LIST command before you set the current job, then PERUSE makes the most recently spooled job the current job and lists data from that job.

The only exception to this rule is the DEL command. For safety reasons, PERUSE requires that you specifically set the current job prior to deleting it.

Displaying a Job

There are three ways to display a job. These are described more fully in the GUARDIAN Operating System Utilities Reference Manual. Briefly they are:

- Use the LIST command of PERUSE to list a page at a time.
- Press the RETURN key, which causes your job to scroll up until you release the key.
- Use the function keys to list lines from the job: the higher the number of the function key, the more lines that are listed to the screen.

The BREAK Key

Typing the BREAK key while PERUSE is listing lines from a job or producing a status display causes it to stop what it is doing and return the PERUSE prompt (_). Typing the BREAK key while PERUSE is waiting for a command causes the GUARDIAN command interpreter to wake and prompt for a command (:). Subsequently issuing a GUARDIAN command interpreter PAUSE command restarts PERUSE.

COMMAND SUMMARY

Table 10-1 is a PERUSE command summary. Appendix D contains a syntax summary of all PERUSE commands. Refer to the GUARDIAN Operating System Utilities Reference Manual for the complete syntax, examples, and considerations of all PERUSE commands.

Following Table 10-1 are two detailed examples of how PERUSE can help TGAL and TAL users view and get a hard copy of a file.

Table 10-1. PERUSE Command Summary

Command	Function
COPIES	alters the number of copies for the current job.
DEL	deletes the current job.
DEV	displays the status of a device.
EXIT	terminates the PERUSE session.
FC	allows you to fix and resubmit a PERUSE command.
FIND	finds an occurrence of a string in the current job.
FORM	changes the form name of the current job.
HELP	displays the syntax and meaning of PERUSE commands.
HOLD	sets the hold flag for the current job.
HOLDAFTER	sets hold-after-printing flag for the current job.
JOB	displays job information and sets the current job.
LIST	lists pages from the current job to the screen or to an output device.
LOC	changes the routing location of the current job.
NUMCOL	sets the number of columns displayed by LIST.
OPEN	specifies a new spooler supervisor.
OWNER	changes the owner of the current job.
PAGE	changes and displays the page position of the current job.
PRI	changes the printing priority of the current job.
REPORT	changes the report name of the current job.
STARTCOL	sets the first column to be displayed by LIST.
STATUS	monitors and displays the status of spooled jobs.

EXAMPLE OF PERUSE OPERATION WITH TGAL

This example shows how a TGAL user might take advantage of some of the capabilities of PERUSE when using the spooler.

Examining a Job

In this TGAL exercise, you specify the temporary holding file #HOLD as the <listfile>.

You can choose any name for a temporary holding file as long as it is a legal spooler location name (as described in the SPOOLCOM LOC command in the GUARDIAN Operating System Utilities Reference Manual) and not already connected to a device. For instance, you can use your initials:

```
:TGAL /IN MEMO, OUT $S.#JLS/
```

Using a temporary holding file allows you to examine the job, using any of the three ways discussed earlier under "Displaying a Job." You can also modify the job attributes before changing its location to one that has a device associated with it.

When you enter PERUSE, there is only this one job in the spooler:

```
:TGAL /IN MEMO, OUT $S.#HOLD/
:PERUSE
 PERUSE  -  T910C12  -  (18MAR85)     SYSTEM  \SYSNAME

 JOB   STATE  PAGES  COPIES  PRI  HOLD  LOCATION  REPORT
 534   READY  20     1       4          #HOLD     USER YOU
 -
```

Finding TGAL Errors

The simplest way to find TGAL errors in your document is to page through the spooled file by holding down the RETURN key.

However, if you have a large document and you have sequentially numbered your pages with the TGAL SECT command, you may wish to try this more efficient way to find TGAL errors. When the job is in the READY state in PERUSE, enter a LIST LAST command and read the page number that scrolls up. Then enter a LIST <that-page-num>. If the same page scrolls up again, you have no

TGAL errors in your document. If an earlier page appears, then TGAL error pages account for the difference.

Return to the beginning of your file, and LIST pages in increasing numbers until you get one whose page number is less than your LIST page number. Use LIST to isolate the TGAL error page, and make a written note of the TGAL error and the line number. Continue in the file, remembering to mentally add a page for each TGAL error page, and writing down the line number of each TGAL error. When finished, delete the file, exit PERUSE, enter EDIT, and search to the line numbers of the TGAL errors to make your corrections.

This can be a fast and effective way of generating an error-free (as far as TGAL is concerned, that is) document.

Finding a Key Phrase in a Job

You want to send to another printer a page containing your most recent correction to the file. You can use the FIND and PAGE commands to determine the exact page number to request from the spooler. If you know there is only one occurrence in this report of the string "16", you can use the FIND command to locate it and the PAGE command to tell you what page it is on:

```
_F /16.  /
        16.  Raw Material Resources

_P
PAGE: 14      LINE: 12
```

Altering Job Attributes

Now you perform several changes: you increase the number of copies to be printed and change the location from the holding file. You also change the report name so that the header message will stand out.

```
_COPIES 10; LOC #HT09; REPORT MEMO
```

The JOB command shows that the changes have been made to the job:

```
_JOB
  JOB    STATE   PAGES   COPIES   PRI   HOLD   LOCATION   REPORT
J 534    READY   20      10       4             #HT09      MEMO
_
```

Printing Out a Portion of a Job

You also want to send the first eight pages of the report to be printed out on the same printer. Without exiting PERUSE, you can LIST these pages out to that printer, including the parameter "C" so that the printer will obey any TGAL OV commands you may have in your EDIT file. The JOB command shows this newly spooled job and the location you have specified:

```
_LIST /OUT $S.#LP3/ 1/8, 14 C
_JOB
  JOB   STATE  PAGES  COPIES  PRI  HOLD  LOCATION  REPORT
J 534   READY  20     10      4           #HT09     MEMO
  560   OPEN         1       4           #LP3      MEMO

_
```

Checking the Status of a Print Device

Next you want to see how long the job will take to print out on a specific printer. The DEV command shows the device status; $HT09 refers to the print process that controls the device associated with the location #HT09. Notice that the queue is very long and that the your twenty-one-page job has been placed at the end of the queue:

```
_DEV $HT09
   DEV STATE: PRINTING      FORM:

     JOB   OWNER    PAGES  WAIT        FORM
     123   008,005  41     00:05:11
    1113   001,013  8      00:06:36
    1112   011,011  10     00:07:21
      10   017,002  21     00:09:56
     576   008,001  22     00:11:07
    1324   001,013  40     00:15:32
     344   006,012  21     00:17:07
     534   001,001  21     00:19:42

_
```

You need to get your job printed in less time than 19 minutes, so you decide to alter its priority. Changing the priority to 7 moves the job to the head of the queue. Note that job 123 is currently printing and will not be stopped by a job of higher priority:

```
_PRI 7;DEV $HT09
   DEV STATE: PRINTING      FORM:

     JOB    OWNER     PAGES  WAIT        FORM
     123    008,005   41     00:04:01
     534    001,001   21     00:07:36
    1113    001,013   8      00:08:26
    1112    011,011   10     00:09:06
      10    017,002   21     00:11:46
     576    008,001   22     00:13:57
    1324    001,013   40     00:17:22
     344    006,012   21     00:19:57

-
```

EXAMPLE OF PERUSE OPERATION WITH TAL

This example PERUSE session shows some of the features that programmers can take advantage of.

The TAL execution includes the NOWAIT option so that the TAL compilation runs concurrently with this PERUSE session. When the status of your job is displayed, the job created by the TAL execution is OPEN--that is, not yet completed spooling:

```
:TAL /IN PROG, OUT $S.#LP, NOWAIT/
:PERUSE

 PERUSE  -  T910C12  -  (18MAR85)     SYSTEM  \SYSNAME

  JOB   STATE  PAGES  COPIES  PRI  HOLD  LOCATION  REPORT
  135   OPEN           1       4          #LP      TAL USER

-
```

Monitoring Changes in Job Status

If you want to know when your job has finished compiling, you can use the STATUS command, which displays a constantly updated list of your jobs and notifies you with a beep when the compilation is finished. Then, to return control to PERUSE, you press the BREAK key:

```
_STATUS

  JOB    STATE  PAGES  COPIES  PRI  HOLD  LOCATION  REPORT
J 135    OPEN          1       4    B     #LP       TAL USER
 (BEEP!)
  JOB    STATE  PAGES  COPIES  PRI  HOLD  LOCATION  REPORT
JC135    READY  129    1       4         #LP       TAL USER
 (BREAK key)

-
```

Finding Errors in a TAL Listing

However, instead of waiting for the compilation to finish, you decide to search for errors. Since all TAL compilation errors begin with the string "**** E", you can use the FIND, PAGE, and LIST commands to find any errors in your program:

```
_F /**** E/
**** ERROR 27 **** Illegal syntax
-
```

Since you have found an error in the program, you won't want a hard copy. Set the hold flag so that you can continue searching for errors:

```
_HOLD;J
  JOB    STATE  PAGES  COPIES  PRI  HOLD  LOCATION  REPORT
J 135    HOLD          1       4    B     #LP       TAL USER
-
```

Continue using the FIND command to search for the same string of words. If you don't specify a new string, FIND continues to search for the next occurrence of the most recently specified string. When an error appears on your screen, you can determine which page it is on by using the PAGE command. You can also examine the context in which the error occurred by using the LIST command to list the entire page on which the error is located:

```
_F
**** ERROR 54 **** Illegal reference parameter
_P *
PAGE: 8      LINE: 12
_L *
```

Continue using the FIND command, searching for the same string until there are no more occurrences of it in the program:

```
_F
**** ERROR 72 **** Indirection must be supplied
_F
_
```

After finding all the errors in the program, you are through with this TAL listing, so you can delete the job and exit PERUSE:

```
_DEL;E
:
```

Then enter EDIT, correct the errors in your source listing, and execute TAL again.

In this next compilation, you are a little more conservative and use #LOOK as the location. Because you did not specify NOWAIT, the job is READY when you enter PERUSE:

```
:TAL /IN PROG, OUT $S.#LOOK/
:PERUSE

 PERUSE  -  T910C12  -  (18MAR85)     SYSTEM  \SYSNAME

  JOB   STATE  PAGES  COPIES  PRI  HOLD  LOCATION  REPORT
  136   READY  129    1       4          #HOLD     TAL USER

_
```

Using the LIST command to display the last page of the TAL compilation shows the trailer message, which indicates that there were no warning or error messages generated during the compilation:

```
_J *; LIST L

Object file name is $VOL1.ADMIN.BILL
This object file will run on either a TNS or a TNS/II system
Number of errors = 0
Number of warnings = 0
Primary global storage = 44
Secondary global storage = 1120
Code size = 1888
Data area size = 40 pages
Code area size = 2 pages
Maximum symbol table space available = 24892, used = 4064
Maximum extended symbol table space available = 0, used = 0
Number of source lines = 3224
Elapsed time - 00:11:05
```

Since no errors were found, you can route the program to the location that can print you a hard copy:

```
_LOC #LP;J

  JOB   STATE  PAGES  COPIES  PRI  HOLD  LOCATION  REPORT
  136   PRINT  129    1       4          #LP       TAL USER
_E
```

SECTION 11

HOW TO USE SPOOLCOM

SPOOLCOM provides both interactive and noninteractive control of the spooler, allowing you:

- To display the status of collectors, devices, jobs, print processes, routing structure, and the spooler itself
- To change the location, state, or any attribute of a job belong to you or to delete it from the spooler system
- To restart a device that has has gone offline with a device error

Other SPOOLCOM commands have great impact on the entire spooler system. For this reason, only system operators and members of the super group (user ID: 255,nnn) can perform these tasks, which are described in the System Operator's Guide.

HOW TO ENTER SPOOLCOM COMMANDS

You can enter SPOOLCOM commands in any of these ways:

- By entering complete SPOOLCOM commands at the COMINT prompt
- By starting a SPOOLCOM process and entering commands interactively at the SPOOLCOM prompt
- By starting a SPOOLCOM process that takes as its input another source

The following three subsections describe these three methods.

Entering SPOOLCOM Commands through COMINT

You can run SPOOLCOM noninteractively by specifying a command when you run SPOOLCOM. SPOOLCOM executes the command (or commands if you specify more than one) and terminates, returning you to the command interpreter; for example:

```
:SPOOLCOM DEV $LP1, STATUS

 DEVICE                  STATE              FLAGS    PROC     FORM
 $LP1                    WAITING              T      $SPLA

:
```

The above command causes SPOOLCOM to display the status of the device $LP1 and then to return your COMINT prompt.

Interactive Use of SPOOLCOM

You can run SPOOLCOM interactively by not specifying the "IN" or "OUT" run options (or by specifying the same terminal for both), and by omitting any <command-list> commands.

SPOOLCOM responds by sending its startup message to the OUT terminal and a close parenthesis ")" as a prompt character:

```
:SPOOLCOM
SPOOLCOM - T9101B00 - (18MAR85)    SYSTEM  \EAST
)
```

T9101B00 is the version of the spooler.

(18MAR85) is the release date for this version of SPOOLCOM.

\EAST is the system it is running on.

You can then enter a command line and SPOOLCOM executes the command (or displays an error if the command is illegal or cannot be executed). After executing the command or displaying the error message, SPOOLCOM sends a prompt and waits for another command. This interactive "command cycle" repeats until you terminate SPOOLCOM with the EXIT command.

Entering SPOOLCOM Commands from Another Source

SPOOLCOM accepts commands from a process, an unstructured disc file, a command file (also known as an OBEY file), or a file in EDIT format.

When reading commands from a process, SPOOLCOM performs a WRITEREAD operation on the process at the ")" prompt, and expects a command line to be returned. The process should perform a READUPDATE to its $RECEIVE file to get the prompt (which may be ignored) and should reply with a command line. Interprocess communication is fully explained in the GUARDIAN Operating System Programmer's Guide and is discussed with special regard to the spooler in the Spooler Programmer's Guide.

When the command file is a disc file, SPOOLCOM considers each record to be a command line. The records are read one at a time until an EXIT command is found or an end-of-file condition occurs.

If you specify a command file (described earlier in Section 5) as the input file, the SPOOLCOM process executes the commands contained in that file. For example, suppose that you use the EDIT program to create a command file thst gives you a list of the collectors, printers, and devices that define a spooler. The file is named $SYSTEM.SYSTEM.SPLCONF and contains these SPOOLCOM commands:

```
COLLECT $SPOOL, FILE $SYSTEM.SYSTEM.CSPOOL, &
   DATA $MKT.SPOL.DATAFILE, UNIT 4, CPU 1, BACKUP 0, PRI 146
PRINT $USERP, FILE $USER.USER.USER,PRI 145, CPU 2
DEV $FAST, PROCESS $STAND, SPEED 900
```

To execute these commands, enter a SPOOLCOM command naming SPLCONF as the IN option:

```
:SPOOLCOM / IN $SYSTEM.SYSTEM.SPLCONF /
```

Control of the terminal returns to COMINT after SPOOLCOM executes the last command in the SPLCONF file.

You can add comment lines within a command file to identify the file and to explain the operations being performed. SPOOLCOM comment lines must begin with the COMMENT command. Any characters on the line following COMMENT are ignored by SPOOLCOM. Below are some comment lines from the SPLCONF file:

```
COMMENT  -- THIS IS $SYSTEM.SYSTEM.SPLCONF

COMMENT  -- THIS CONTROL FILE USES SPOOLCOM TO CONFIGURE THE
COMMENT     SYSTEM TO BE COLD STARTED, AND PASSES THIS
COMMENT     INFORMATION TO THE SUPERVISOR.
```

The complete example of this command file can be found in the System Operator's Guide under the task, "Cold Starting the Spooler."

SPOOLCOM SECURITY

SPOOLCOM has complete control of the spooler. To protect the spooler from damage at the hands of inexperienced or malicious users, it has a security system.

Any user can use SPOOLCOM to find the status of the spooler components: jobs, devices, collectors, print processes, routing structures, and the spooler itself. You can also alter the state and attributes of your own jobs. However, any command that would affect the jobs of other users or the character of the spooler system is reserved for system operators. The spooler does not execute certain SPOOLCOM commands unless they are submitted by a system operator (group ID = 255), who can execute any command or subcommand.

SPOOLCOM COMMANDS

Whether command lines are entered from a terminal or read from a disc file, the maximum length is 132 characters. You can enter two or more SPOOLCOM commands on the same line if you separate them with semicolons; for example, the following command obtains the status of job number 43, then exits from SPOOLCOM:

```
)JOB 43, STATUS; EXIT
```

SPOOLCOM commands consist of a command word possibly accompanied by a parameter, followed by zero or more subcommands. The command and its parameter are separated from the subcommands by commas. Subcommands are also separated from each other by commas.

For example, to specify the report name TAL COMPILE for job number 35, the user enters this SPOOLCOM command:

```
)JOB 1635, HOLD, REPORT TAL COMPILE, START
```

JOB is the command, and the parameter 1635 indicates that job number 1635 is being referenced. In order to change a job report name, you must put the job in the HOLD state. Use the subcommand REPORT to specify the report name TAL COMPILE for the job. (The spooler converts all keyed input to uppercase.) After changing the report name, use the subcommand START to put the job back in the device queue. (In the comparable statement in PERUSE, the HOLD and START of a job are invisible to the user.)

SPOOLCOM allows you to enter commands affecting a job, a collector, or any other spooler component on a single line or separate lines. However, each command line must be complete. For example, to enter the above subcommands on three separate lines, you must repeat the command JOB and the parameter 1635 on each line:

```
)JOB 1635, HOLD
)JOB 1635, REPORT TAL COMPILE
)JOB 1635, START
```

These three commands have the same effect as the single command line shown above. In fact, each subcommand can be viewed as a separate command. Each subcommand is processed left to right, with each being completely processed before the next subcommand is executed. The only exception to this is DRAIN, since it can take some time for a component to completely drain. SPOOLCOM puts the component in the drain state but does not wait for the drain to complete.

COMMAND SUMMARY

A summary of commands available to all users is given in Table 11-1, followed by a summary in Table 11-2 of all SPOOLCOM commands including those available only to super-group users (user ID: 255,nnn). Note the differing functions of these commands, depending on the qualification of the user. Appendix E contains syntax summaries of SPOOLCOM commands.

You can find the complete syntax, considerations, and examples of SPOOLCOM, for both super-group users and others, in the GUARDIAN Operating System Utilities Reference Manual.

Table 11-1. Command Summary for All Users

Command	Function
COLLECT	obtains the status of the spooler collectors.
COMMENT	designates a comment to be ignored by SPOOLCOM.
DEV	controls and obtains the status of devices in the spooler system. While a device is printing a job, the owner of that job can issue a SKIP and SKIPTO subcommand on that device. Anyone can start an offline device that has not been put offline by a DRAIN command issued by the operator.
EXIT	terminates a SPOOLCOM session.
FC	allows edit and reexecution of a command line (same as the FC command in the GUARDIAN command interpreter).
HELP	displays the syntax of SPOOLCOM commands for user reference.
JOB	alters attributes and changes the state of your own jobs; also obtains the status of any job in the spooler system.
LOC	displays the status of the spooler routing structure.
OPEN	specifies the particular spooler system to which the other SPOOLCOM commands refer.
PRINT	obtains the status of the spooler print processes.
SPOOLER	obtains the status of the spooler.

Table 11-2. Command Summary for Super-Group Users

Command	Function
COLLECT	specifies attributes, obtains status, and changes the state of collectors.
COMMENT	designates a comment to be ignored by SPOOLCOM.
DEV	specifies attributes, obtains status, and changes the state of devices.
EXIT	terminates a SPOOLCOM session.
FC	allows edit and reexecution of a command line (same as FC command in the GUARDIAN command interpreter).
HELP	displays the syntax of SPOOLCOM commands for user reference.
JOB	specifies attributes, obtains status, and changes the state of a job.
LOC	defines and modifies the spooler routing structure.
OPEN	specifies the particular spooler system to which the other SPOOLCOM commands refer.
PRINT	specifies attributes, obtains status, and changes the state of print processes.
SPOOLER	starts, stops, and obtains the status of the spooler.

TASKS FOR ALL USERS

Briefly, SPOOLCOM allows all users to perform the following operations, which are more fully described below:

- Obtain status of — your job, collector, device, print process, routing structure, spooler
- Change attributes of — your job
- Restart (bring back online) — device

How to Obtain the Status of Spooler Components

From the description of the spooler in Section 9, "Introduction to the Spooler", you know that the important spooler components are its collectors, devices, jobs, print processes, and routing structure; the supervisor that controls communication; and the SPOOLCOM and PERUSE programs that allow you to interface with the supervisor.

SPOOLCOM enables you to find the status of collectors, devices, print processes, the spooler routing structure, and the spooler itself, as well as the status of any job you own. Each of these components is controlled by a SPOOLCOM command:

Spooler Component	SPOOLCOM Command
Collector	COLLECT
Device	DEV
Job	JOB
Print Process	PRINT
Routing Structure	LOC
Spooler	SPOOLER

You can execute any one of these commands without any qualifying subcommands or parameters:

```
:SPOOLCOM <SPOOLCOM-command> [ , STATUS ]
```

This command displays on your screen information about all the collectors, devices, and so on that exist on your spooler system. Then, because this is noninteractive access to SPOOLCOM, it returns your COMINT prompt (:).

Since your spooler system probably has many devices, jobs, and related routing structures, the STATUS subcommand may return a lot of information. You may want to specify the particular entity you are interested in, such as a job of a specific number, or a device or location of a specific name. You can use PERUSE to tell you which jobs you own that are currently in the spooler system, as well as the location, and then use SPOOLCOM LOC to associate the location with the device that is to print your job.

For example, after spooling your job and entering PERUSE, you see that your job is numbered 566 at location #HT4:

```
JOB   STATE  PAGES  COPIES  PRI  HOLD  LOCATION       REPORT
566   OPEN          1       4          #HT4           MKTG BOB
```

Exit PERUSE, and execute the LOC command to find both the location that is connected to that device and the jobs in the device queue:

```
:SPOOLCOM LOC #HT4

LOCATION            FLAGS  DEVICE
#HT4.DEFAULT               $HT4

  JOB  LOCATION          DEVICE              SEQ    COPY  PAGE
  364  #HT4.DEFAULT      $HT4                PRINT  1     6
  513  #HT4.DEFAULT      $HT4                2      1     6
  566  #HT4.DEFAULT      $HT4                3      1     4
```

Be careful that you use "#" with the LOC, not "$", or else you get an error message:

```
:SPOOLCOM LOC $HT4
OPEN $SPLS    ; LOC $HT4
                    ^
INVALID COMMAND PARAMETER
```

How to Change Your Job

SPOOLCOM lets you handle your job in the same way as PERUSE. In addition to finding the status of your job, you can alter the number of copies, the form name, the owner name, the location, and the printing priority. You can also put your job on hold and remove it from hold, and delete it from the spooler system. Refer to the GUARDIAN Operating System Utilities Reference Manual.

How to Restart a Device

If a device has gone offline due to a device error, you can see this with both PERUSE and SPOOLCOM.

SPOOLCOM tells you both the GUARDIAN file-system error number (in this case it is error 100, NOT READY) and the job numbers in the device queue:

```
:SPOOLCOM DEV $LP1

DEVICE                    STATE          FLAGS  PROC    FORM
$LP1                      DEV ERROR 100    T    $SPLP

  JOB  LOCATION          DEVICE              SEQ   COPY  PAGE
  571  #LP1.DEFAULT      $LP1                1     1     40
  596  #LP1.DEFAULT      $LP1                2     1     43
```

By way of comparison, PERUSE also identifies the error number, plus the owner ID and the printing times for the jobs queued up to print. The "+" following these times indicates that the device must come back online before these print times have any meaning:

```
_DEV $LP1

DEV STATE: ERROR 100     FORM:

  JOB   OWNER    PAGES  WAIT        FORM
  571   008,013  40     00:02:32+
  596   008,013  43     00:03:39+
_E
```

SPOOLCOM allows users to restart a device that has gone offline due to a device error. Some of the more common device errors you can encounter and correct are shown in Table 11-3.

Table 11-3. Common Device Errors That All Users Can Correct

DEV ERROR	Meaning	Recovery
14	DEVICE DOES NOT EXIST	Check if it exists. Do :FILES /OUT <device-name>/. If COMINT lists the device name, then it exists and you should restart it. If COMINT does not list the device name, then the device does not exist.
100	NOT READY	Press the READY button on the device or otherwise make it ready.
102	PAPER OUT	Reload with paper, and make the device ready.

To restart a device that has gone offline, correct the device error, and issue the command:

```
:SPOOLCOM DEV $<dev-name>, START
```

SECTION 12

SECURITY FEATURES OF TANDEM SYSTEMS

This section describes the security features of the GUARDIAN operating system as they are specified through COMINT and FUP commands.

Security as it is controlled programmatically is detailed in the GUARDIAN Operating System Programmer's Guide.

Some of the detail in this section is useful primarily to system managers. It is presented here so that other interested users can gain a reasonably full comprehension of the security system, including an understanding of items that appear in the output from some FUP and COMINT commands.

The GUARDIAN operating system provides security for both local and network environments in the following areas:

- Logon security to prevent unauthorized access to systems
- Disc file security to prevent unauthorized access to disc files
- Process security to prevent interference with running processes

The security system is designed so that it will not interfere with application design in systems where security is not desired.

Additional security may be provided by the application programs.

INTERFACE TO THE SECURITY SYSTEM

User interface to the security features of the operating system can be established through any of these:

- The COMINT process
- The File Utility Program (FUP)
- System procedure calls in user programs

Command Interpreter Interface

The following COMINT commands and programs provide the user interface to the security system:

ADDUSER program
DEFAULT program
DELUSER program
LOGOFF command
LOGON command
PASSWORD program
REMOTEPASSWORD command
RPASSWRD program
USERS program
VOLUME command
WHO command

These commands and programs are fully described in the GUARDIAN Operating System Utilities Reference Manual.

FUP Interface

The following FUP commands provide the user interface to the security system:

GIVE command
INFO command
LICENSE command
REVOKE command
SECURE command

These commands are described in the GUARDIAN Operating System Utilities Reference Manual.

Programmatic Interface

The procedures that provide the interface between user programs and the security system are listed and described in the GUARDIAN Operating System Programmer's Guide.

SYSTEM USERS

There are four classes of system users: (1) standard users, (2) group managers, (3) system operators, and (4) users with super ID.

- Standard users can do the following:

  --Establish communication with the GUARDIAN command interpreter by logging on to the system

  --Display system status

  --Assign themselves a logon password

  --Set default volume and subvolume names

  --Set file security for files they own

  --List names of disc files

  --Create, rename, and purge disc files

  --Run, debug, and stop their own processes

  --Start a backup process for their COMINT process

  --Switch primary control of their COMINT process to the backup process

  --List all groups and the users in them

  --Set remote passwords

  --Log off the system

- Group managers can do the following:

  --Perform all functions that standard users can perform

  --Add new users to their own group

  --Delete users from their own group

  --Log on as any user in their group without knowing that user's password (which means that the group manager also has access to the user's files)

- System operators can do the following:

  --Perform all functions that standard users can perform

  --Monitor processor usage

  --Reload processor modules

  --Set the current date and time of day for the system

  --Alter bus availability states (hardware paths)

- A super ID user can:

  --Perform all functions that standard users, group managers, and system operators can perform

  --Add new groups to the system

  --Delete groups from the system

  --Add new users to the system in any group

  --Delete users from the system

  --Debug and stop any user's processes

  --Debug privileged programs

  --Log on as any user in any group without knowing that user's password

Note that the preceding lists deal only with functions that group managers, system operators, and super ID users can perform using FUP and COMINT. Other utilities (not discussed in this section) add to the capabilities of these users--notably, the Peripheral Utility Program (PUP), discussed in the System Operator's Guide for your system.

Identifying Users

Each user has a unique user name and a corresponding unique user ID. A user name has the form <group-name>.<user-name>, where <group-name> is the name of the group to which the user belongs, and <user-name> is a name identifying the individual user within the group. Similiarly, a user ID is a pair of integers in the form <group-id>,<user-id>, where <group-id> identifies the user's group and <user-id> identifies the user within the group. The two integers in a user ID must be between 0 and 255, inclusive.

The user names and user IDs for all users on the system are kept in a system file. During a logon, the system checks that the user name supplied in the LOGON command is valid, and that the correct password, if any, is supplied (see "Logging On" in this section for an explanation of passwords).

The classes of special users (users who have capabilities beyond those of a standard user) and their corresponding user IDs are:

User Class	Typical User Name	User ID
Super ID user	SUPER.SUPER	255,255
System operator	SUPER.OPERATOR	255,n
Group manager	groupname.MANAGER	n,255

where n is a nonnegative integer less than 255, and <group-name> is the name of a user group known to the system.

Adding New Users and Groups

When a new system is initialized, only two users exist: a user with super ID and the null user. The null user is a standard user having the user name NULL.NULL and the user ID 0,0.

The super ID user uses the ADDUSER program to create new groups and to add new users to existing groups. A group manager can also add new users to a particular group with the ADDUSER program. For each new user, a user name and corresponding user ID must be specified.

For example, assume that a new system was just initialized for the administration department. The system manager (who is the super ID user) now wants to create a user group called ADMIN, with group ID 6.

With the ADDUSER program, the system manager first creates a group manager:

```
:ADDUSER ADMIN.MANAGER, 6,255
```

Once the super ID user creates a group manager for a new group, both the super ID and the group manager can add new users to that group. For example, after the super ID user creates the ADMIN.MANAGER user, ADMIN.MANAGER can enter the following ADDUSER commands to add other users to the group:

```
:ADDUSER ADMIN.JOAN, 6,11
:ADDUSER ADMIN.CHRIS, 6,12
:ADDUSER ADMIN.JOHN, 6,13
:ADDUSER ADMIN.NADINE, 6,14
:ADDUSER ADMIN.MIKE, 6,15
```

As many as 256 groups, with up to 256 users each, can be created for each system.

LOGGING ON

Before a user can gain access to the system, that user must log on. Logging on is done by entering the previously defined user name in the LOGON command of COMINT (see Section 2).

Passwords

Each user name can be protected by a password to prevent unauthorized individuals from accessing the system. When a new user is created with the ADDUSER program, that user has no password. Users can define their own password with the PASSWORD program. Passwords are stored in a system file along with the user names. During subsequent logons, COMINT consults this file to determine whether a password is established and then to verify the password. Users can change or remove their passwords by issuing a new PASSWORD command.

FILE SECURITY AND ACCESS

Each file has an owner (initially the user who created the file) and a file security. The owner is identified by a user ID, which is initially the same as the creator's user ID. The owner of a file can transfer ownership to another user by means of the FUP GIVE command.

Each user has a logon default security string that is automatically assigned to any files created during that session, unless the user specifies a different security. During a session, the user can change the default security string as outlined in the next subsection.

Setting File Security

Four types of access are allowed for a file: read, write, execute, and purge (R,W,E,P). These accesses are defined as follows:

- Read--the ability to examine or copy the file's contents, and the ability to execute a command file using the OBEY command in COMINT
- Write--the ability to modify the contents of the file
- Execute--the ability to execute the file as a process using the RUN command in COMINT (applies to program code files)
- Purge--the ability to delete the file, rename it, or alter its definition

The file's owner can specify one of seven levels of security for each of the four types of access.

You can set file security with the FUP SECURE command or, in a program, you can use the file system's SETMODE or SETMODENOWAIT procedure (see the GUARDIAN Operating System Programmer's Guide).

Table 12-1 shows the seven levels of security. You use alphabetic code to specify security with the DEFAULT program and the VOLUME command in COMINT and the FUP SECURE command (however, the hyphen (-) can be specified only with FUP SECURE). You use the corresponding numeric values when you specify security in a program (you can also specify numeric values with the FUP SECURE command). "Local" refers to access within a single system; "remote" refers to access between systems (nodes) in a network (see "Allowed File Access" in this section for details).

Table 12-1. Levels of Security

FUP Code	Program Values	Access
-	7	Local super ID only
U	6	Owner (local or remote), that is, any user with owner's user ID
C	5	Member of owner's group (local or remote) that is, any member of owner's community
N	4	Any user (local or remote)
O	2	Owner only (local)
G	1	Member of owner's group (local)
A	0	Any user (local)

Allowed File Access

Accessors, or openers, of a file are classified as either local or remote with respect to that file. A local user is one who is logged on to the system where the file resides; a remote user is one who is logged on to a different system. The security level of the opener of a file is determined by two things:

- The ID of the opener--that is, whether the opener is the owner of the file, a member of the owner's group, or a member of another group
- The location of the opener--that is, whether the opener is local or remote with respect to the file

When a user or program attempts to open a file, the opener's security level is determined and checked against the file's security level, as defined in Table 12-1, for the requested

access mode (read, write, execute, or purge). Table 12-2 lists the accesses that are allowed for files based on these two security levels.

In the top row of Table 12-2, find the file's security level for the type of access you are considering (read, write, execute, or purge). Then read down that column and find the Y or hyphen that corresponds to the opener's security level. A Y means the access is allowed; a hyphen means the access is not allowed.

Table 12-2. Allowed File Accesses

	File Security Level									
	-	U	C	N	O	G	A			
Accessor's Security Level										
Super ID user, local access	Y	Y	Y	Y	Y	Y	Y			
Super ID user, remote access		Y	Y	Y	-	-	-			
Owner or owner's group manager, remote access	-	Y	Y	Y	-	-	-			
Member of owner's group, remote access	-	-	Y	Y	-	-	-			
Any other user, remote access	-	-	-	Y	-	-	-			
Owner or owner's group manager, local access	-	Y	Y	Y	Y	Y	Y			
Member of owner's group, local access	-	-	Y	Y	-	Y	Y			
Any other user, local access	-	-	-	Y	-	-	Y			

Note that the ID of the opener of the file is actually determined by comparing the opener's process accessor ID (generally the same as the opener's user ID or the user ID of the owner of a process making the access) with the user ID of the owner of the file. Process accessor IDs are explained further under "Process Security" later in this section.

For example, assume that a file owned by ADMIN.BILL was secured by the FUP SECURE command as follows:

```
-SECURE BILLFILE, "AGNU"
```

This security means that any local user can read from the file, only local members of the ADMIN group can write to the file, any network user can execute the file, and only the owner (whether logged on locally or remotely) can purge it.

If user ADMIN.ANN were operating on a network from a remote system, she could do nothing more than execute the file, but if she were logged on locally, she would also have read and write access.

PROCESS SECURITY

This subsection defines and describes the use of security system features that protect and restrict access to running processes. These features include process accessor IDs and creator accessor IDs and describes their use in the security system. Also described are procedures for licensing programs and for adopting the user ID of a program file's owner as that program's process accessor ID.

Process and Creator Accessor IDs

Two identifications are associated with a process: the creator accessor ID and the process accessor ID. The creator accessor ID identifies the user who initiated the creation of the process. The process accessor ID (which is normally the same as the creator accessor ID) identifies the process and is used to determine if the process has the authority to make requests to the system, for example, to open a file, stop another process, and so on.

Figure 12-1 illustrates a chain of process creations in which the process accessor ID of the original process is passed to other generations of processes. In this figure, CI is a COMINT process, p1 and p2 are processes created by CI, and p3 is a process created by p2.

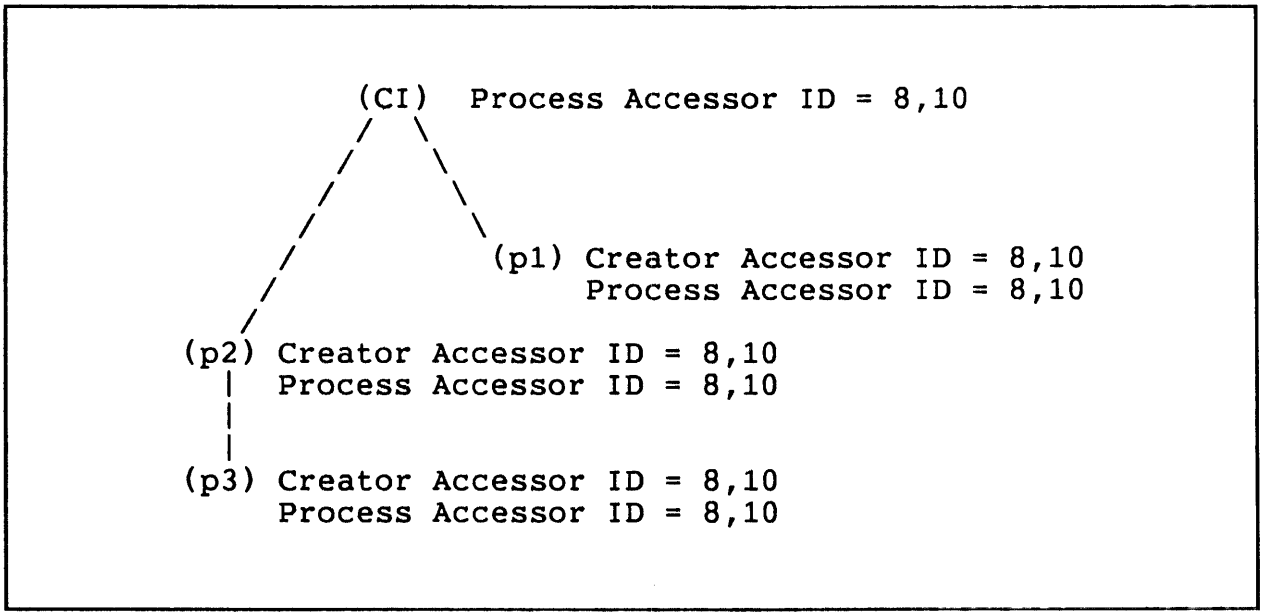


Figure 12-1. Passing of Accessor IDs

A process can find out its creator accessor ID and process accessor ID using the CREATORACCESSID and PROCESSACCESSID procedures, respectively (see the GUARDIAN Operating System Programmer's Guide).

The process accessor ID is used by the security system to determine if file access should be allowed (see "File Security and Access"). In addition, the process accessor ID is used to determine whether certain security-restricted operations, such as STOP and DEBUG, can be performed if the requester is not the process's creator or does not have a super ID.

Security-restricted operations can be performed by:

- The super ID user
- A process with a process accessor ID equal to the group manager's
- A process with a process accessor ID which is the same as that of the target process's creator
- A process with a process accessor ID equal to the target process's accessor ID

When a process is created, the operating system passes the appropriate process accessor ID to the descendant process. This ID becomes the creator accessor ID of the new process. The

process accessor ID of the new process can come from either of two sources: the process accessor ID of its creator (this is the usual case), or the owner ID of a process's program file (if file adoption was specified with the FUP SECURE PROGID attribute).

Adopting a Program File's Owner ID

Program file owner ID adoption allows the owner of a program file (or the super ID) to specify that the process accessor ID of any process created by running that program file is to be the same as the program file's owner ID rather than the creating process's process accessor ID. This option allows the program file's owner to control the files that the new process can access and to control the operations that can be performed on or by the process. Program file ID adoption is specified with the FUP SECURE command (PROGID option) or the SETMODE or SETMODENOWAIT procedure.

Figure 12-2 shows several generations of processes and demonstrates how creator accessor and process accessor IDs can change when the PROGID attribute is set on. CI is a COMINT process with process accessor ID 8,10. P1 is a process created by CI, and p2 is a process created by p1.

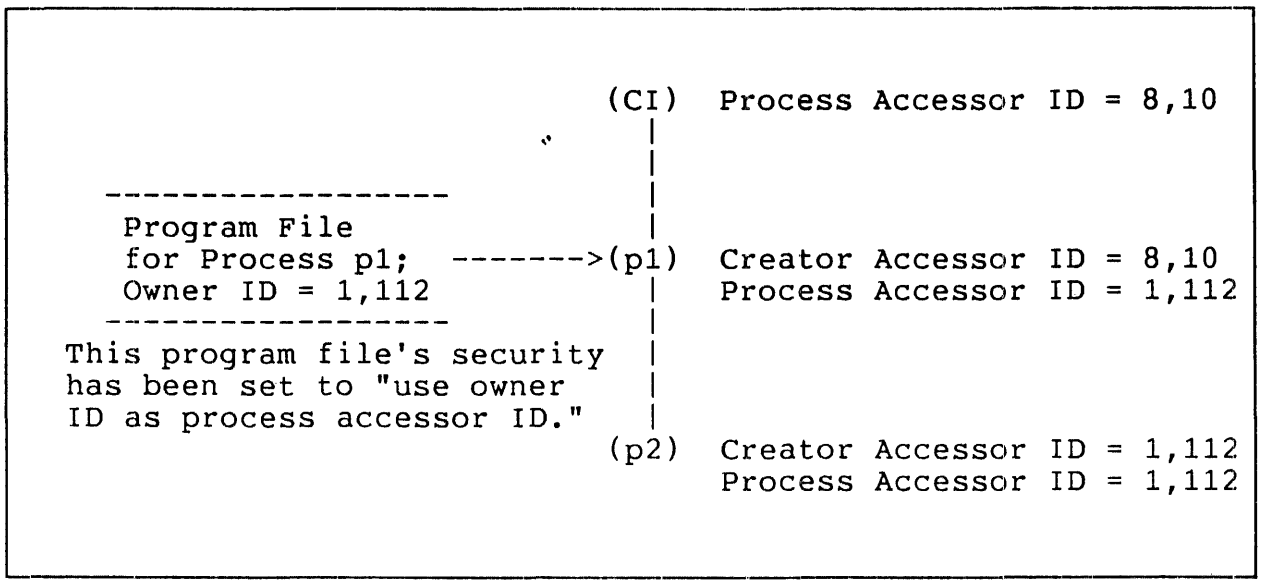


Figure 12-2. Effect of Adopting a Program File's Owner ID

Controlled Access with Program File ID Adoption

In any application, there might be data files that require a controlled type of access--such as allowing many users access to certain records, while denying the same users access to some records that are considered sensitive. For example, an employee file might contain such data as employees' identification numbers, names and addresses, and some sensitive information such as the employees' salaries. This information might be in a record format such as this:

emp #	emp name	address	benefits	salary	.....etc.

The following example shows how a user can control the access to such a data file and also control any future file accesses or program functions.

An employee data file is owned by user 1,112 and is secured for local owner access only ("OOOO"). This means that only the file owner (or the local super ID) has direct access to the file. However, a controlled form of file access is allowed using a query program that has been written to return only nonsensitive information. The program file is owned by user 1,112 and is secured so that any local user can execute the process ("OOAO"). Additionally, program file ID adoption has been specified (use owner ID as process accessor ID).

As shown in Figure 12-3, user 8,10 (process accessor ID of 8,10) executes the query program, which returns "limited data views" only. The query process adopts the owner ID of the program file (1,112), which becomes its process accessor ID. (If the query program were to create another process, that process would inherit 1,112 as both its creator accessor ID and its process accessor ID.)

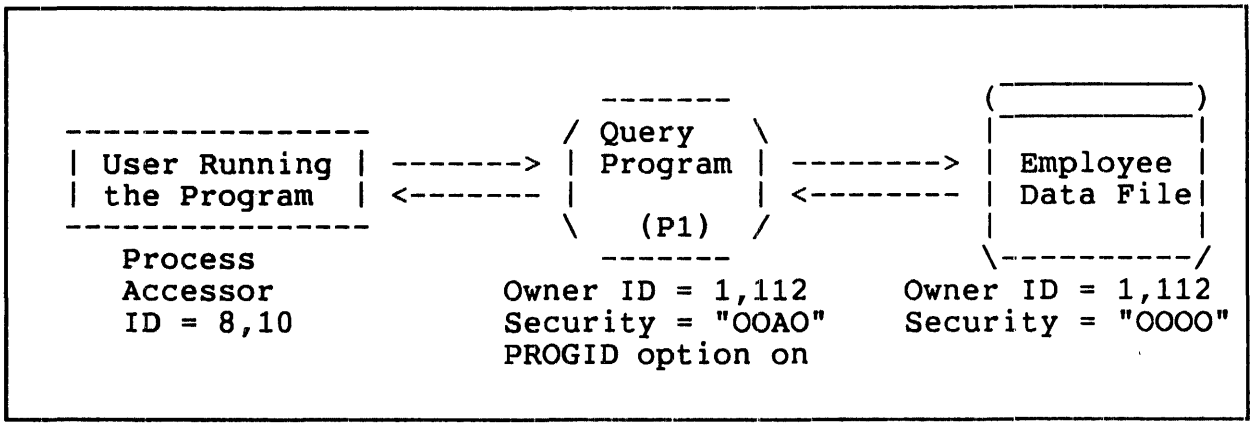


Figure 12-3. Controlled Access to a Data File

Licensing Programs

If a program contains privileged procedures (procedures having the CALLABLE or PRIV attribute), it must be licensed before it can be run by any user other than the super ID. Only a super ID user can license a file; licensing is performed with the FUP LICENSE command.

Programs running in the privileged mode have total freedom to access operating system tables and to execute privileged instructions and procedures, so it is possible for such programs to circumvent the file security checks and thereby gain access to any file. However, some privileged programs are needed in the system (for example, the command interpreter is a privileged program). Through licensing, the installation can run privileged programs that it has authorized, but users may not run unauthorized privileged programs.

Note that if a licensed file is opened with write access or read-write access, the file becomes unlicensed.

For example, a privileged program called PRIVPROG exists in a software development group. PRIVPROG is owned and licensed by the super ID so that all members of the group can execute it. A programmer in the group has developed a revision to the PRIVPROG program and wants to replace the object program with the revision.

Provided that the super ID user also gives the programmer write access to the program file, the following TAL compilation replaces the program with the revision and causes the program to become unlicensed:

```
RUN TAL /IN source/ PRIVPROG
```

This means that no users except super ID users (excluding even the programmer who replaced the program) are allowed to execute the program. When PRIVPROG is debugged and ready for use, the super ID can license it so that others in the group can run it.

NETWORK SECURITY

A user at system \X who wants to access a file (disc file, device, or process) on system \Y must satisfy these three requirements:

- The user on system \X must also be established as a user on system \Y.
- The user must have matching remote passwords set up at both system \X and system \Y.
- The user on system \X must have the authority to access the particular file on system \Y as a remote accessor, if it is a disc file.

Global Knowledge of User IDs

Each user is known to the local system by a user name and a user ID, for example, ADMIN.BILL and 6,14 (respectively). A user can access files on a remote system only if that person's user name and user ID are also known to that specific system. If ADMIN.BILL wants to access a file on a remote system, that system must also have a user identified as ADMIN.BILL with a user ID of 6,14.

Establishing Remote Passwords

Once the user IDs of network users are added to relevant nodes of the network, a system of remote passwords is used to specify whether remote access is permitted. Remote passwords are specified with the REMOTEPASSWORD command in COMINT or the RPASSWRD program.

For example, consider two systems in a network named \A and \B. A user identified as ADMIN.BILL, with a user ID of 6,14, was defined at both systems.

At system \A, the following commands are entered to establish an allow-access remote password to system \A:

```
:LOGON ADMIN.BILL
:REMOTEPASSWORD \A, SHAZAM
```

ADMIN.BILL's allow-access password for \A from all other systems is SHAZAM.

At system \B, the following is entered:

```
:LOGON ADMIN.BILL
:REMOTEPASSWORD \A, SHAZAM
```

The user at system \B entered the matching password and now has remote access to system \A as ADMIN.BILL.

ADMIN.BILL, logged on at system \B, does not have quite the same status at \A as the ADMIN.BILL at \A. Since ADMIN.BILL at \B is a remote accessor of \A, he cannot access disc files on \A that are secured for local access only. Also, if ADMIN.BILL on \B creates a process on \A that attempts to access the home terminal on \B, the attempt will fail because remote passwords to allow access from \A to \B were not established.

For ADMIN.BILL to gain access to \B from \A, an allow-access password must be defined for ADMIN.BILL at \B, matched by a request-access password at \A. For example, the following is entered first at \B and then at \A:

```
:LOGON ADMIN.BILL
:REMOTEPASSWORD \B, AARDVARK
```

Now users logged on as ADMIN.BILL at either system \A or system \B have access to both systems.

The following considerations apply to remote passwords:

- Once matching remote passwords are established at both systems, users do not need to specify the remote password to gain access to the remote system. Furthermore, the super IDs for the various nodes in a network can set up the appropriate allow-access and request-access passwords for all users so that the users themselves need not be concerned with REMOTEPASSWORD commands. Once the appropriate passwords are set up for a given user, that user can automatically access files remotely without having to be aware of the network passwords.

- As shown previously, the absence of an allow-access password prevents remote access by anyone acting as that user. Thus, if MARKETNG.SUE does not supply an allow-access password, no remote user with the same user ID can access MARKETNG.SUE's home system as MARKETNG.SUE.

- A remote password, once defined, remains in effect until modified by a subsequent REMOTEPASSWORD command. This command removes any previously designated remote password for the corresponding \system-name:

  ```
  REMOTEPASSWORD \system-name
  ```

  This command removes all of the user's remote passwords:

  ```
  REMOTEPASSWORD
  ```

- Request-access passwords and allow-access passwords can be specified at any time. Remote access is permitted as soon as both remote passwords are defined (provided they match).

- Remote passwords are independent of local passwords. In the preceding example, ADMIN.BILL could prevent unauthorized persons from logging on as ADMIN.BILL by entering this command at either system:

  ```
  PASSWORD local-password
  ```

Process Access

Several security considerations relate to remote processes:

- With respect to a given system, each process in the network is either "local" or "remote." The following rules apply:

  --A process is remote if it is running on a remote system.

  --A process is remote if its creator is on a remote system.

  --A process is remote if its creator is remote.

  By the last two rules, even a process that is running in a given system can be remote with respect to that system. These rules prevent a remote process from creating another process to access a file whose security specifies local access only.

- A remote process cannot suspend or activate a local process. A remote process cannot stop a local process, unless the stop mode of the local process is 0 (anyone may stop it).

- A remote process cannot put a local process into a debug state.

Using a Remote COMINT Process to Gain Local Access

Openers of a file are classified as either local or remote with respect to that file. A local user is one who is logged on to the system on which the file resides. A remote user is one who is logged on to a different system in the same network.

A remote accessor of a system can become a local accessor by running a COMINT process in the remote system and logging on. For example, if remote passwords have been established so that user ADMIN.BILL at \A can access system \B, he can issue the following commands:

```
:\B.COMINT
:LOGON ADMIN.BILL
```

User ADMIN.BILL is now logged on as the local ADMIN.BILL on system \B. Therefore, he can access disc files on \B owned by ADMIN.BILL even if they are secured "O" (local owner), as well as other files that are only accessable locally.

This remote session can be terminated with either an EXIT or a LOGOFF command. If ADMIN.BILL terminates the COMINT process on system \B with an EXIT command, COMINT asks:

"ARE YOU SURE YOU WANT TO STOP YOUR process-name CI?"

process-name is either the process name or the process ID of the remote COMINT process, including the system name. Reply YES or Y to stop the remote COMINT process.

If ADMIN.BILL enters a LOGOFF command, COMINT terminates and displays this message:

"EXITING FROM CI ON SYSTEM \B"

Remote users can be prevented from becoming local users in a number of ways. One method is for the local super ID to specify "A" (any local user) as the execute security for the COMINT program file, thereby preventing anyone in any remote system from running a COMINT program in system \B.

Also, a user who has the same user name as a user in another system cannot log on to that system unless he knows the local password for that user name. For example, ADMIN.BILL on system \A cannot log onto system \B if ADMIN.BILL at \B has a local password that is unknown to ADMIN.BILL at \A.

Global Passwords

In a large network, it is sometimes not desirable for all users to have access to the network, but it is desirable to allow network access to certain users without forcing them to type or even know all of the required REMOTEPASSWORD commands.

At each node, a user called NET.ACCESS can be established, and the following commands issued:

```
:LOGON NET.ACCESS
:PASSWORD local-password
:REMOTEPASSWORD \A, global-password
:REMOTEPASSWORD \B, global-password
:REMOTEPASSWORD \C, global-password

:REMOTEPASSWORD \n, global-password
```

For this example, the global remote password is the same for all systems and is known only to the system managers. The local password is different for each system and is given to those users who are allowed to access the network.

Only those users who know the local password can log on as NET.ACCESS. While logged on as NET.ACCESS, these users can access remote files. For example, the following command allows the users to access remote files secured so that NET.ACCESS (remote) can get to them:

```
:LOGON NET.ACCESS, local-password
```

Subnetworks

In a large network, it is sometimes desirable to allow users to access some nodes but not others. For example, users on system \SANFRAN may be allowed to access systems \LA, \SEATTLE, and \CUPRTNO, but not the \NEWYORK and \CHICAGO systems.

In this case, the ideas used in the preceding examples can be extended to allow access to any number of subnetworks (any collection of individual nodes). A user such as NET.WEST is established at each node of the subnetwork, and a password scheme like the one used in the previous example is used to allow certain users to log on as NET.WEST.

Subnetworks implemented in this manner can be allowed to overlap or include one another. \CHICAGO might be accessible from \NEWYORK by logging on as NET.EAST, and from \PHOENIX by logging on as NET.MIDWEST. Similarly, each node in the network might have a user called NET.GLOBAL, who is allowed to access every other node.

Capabilities of a Remote Super ID User

On a single system, a super ID user can access any file. On a network, the powers of the super ID can be defined as local, global, or somewhere in between.

To make SUPER.SUPER exclusively a local super ID user, do not issue REMOTEPASSWORD commands for SUPER.SUPER at any node.

To make SUPER.SUPER a global super ID, issue REMOTEPASSWORD commands (as defined in the "Global Passwords" example) at every node, and give every SUPER.SUPER the same password. In this case, if a disc file is secured A, G, O, or -, a remote super ID user can still gain access to the file by running a COMINT process in that system and logging on as the local SUPER.SUPER.

To make SUPER.SUPER a super ID who is somewhere in between, issue REMOTEPASSWORD commands (as defined in "Global Passwords") at every node, but give each SUPER.SUPER a distinct password. This way, any disc file can be protected from remote access by giving it A, G, O, or - security. (The remote SUPER.SUPER can then access files secured N, C, or U.) A remote SUPER.SUPER cannot log on as a local super ID user because the local super ID's password is unknown.

APPENDIX A

COMINT COMMAND SYNTAX SUMMARY

This appendix gives the complete syntax for all the COMINT commands. Explanations for the special terms and punctuation used here can be found in "Syntax Conventions Used in This Manual." For complete definitions of all the syntax terms, see the command descriptions in the GUARDIAN Operating System Utilities Reference Manual.

```
[\<system-name>.]COMINT
  [ / <run-option> [ , <run-option> ]... / ]
  [ <backup-cpu-number> ]
```

For <run-option> see the RUN command.

```
ACTIVATE [[\<system-name>.]{$<process-name>}]
                           {<cpu>,<pin>    }
```

```
ADDDSTTRANSITION <start-date-time> , <stop-date-time> , <offset>
```

```
ADDUSER <group-name>.<user-name> , <group-id> , <user-id>
```

```
ALTPRI [[\<system-name>.]{$<process-name>}] , <priority>
                         {<cpu>,<pin>    }
```

```
ASSIGN [<logical-unit> [ , <actual-file-name> ]]
       [[ , <create-open-spec> ]...           ]

   <logical-unit> is either of these two:

                  * .<logical-file>
      <program-unit> .<logical-file>

   <create-open-spec> is any one of these six:

      <extent-spec>
      CODE <file-code>
      <exclusion-spec>
      <access-spec>
      REC <record-size>
      BLOCK <block-size>

         <extent-spec> is either of these two:

            EXT [ ( ] <pri-extent-size> [ ) ]
            EXT ( [ <pri-extent-size> ] , <sec-extent-size> )

         <exclusion-spec> is any one of these three:

            EXCLUSIVE
            SHARED
            PROTECTED

         <access-spec> is any one of these three:

            I-O
            INPUT
            OUTPUT


BACKUPCPU [ <cpu-number> ]


BUSCMD  { X | Y } , { DOWN | UP } , <from-cpu> , <to-cpu>


CLEAR  {ALL [ { ASSIGN | PARAM } ]}
       {ASSIGN <logical-unit>     }
       {PARAM <parameter-name>    }


COMMENT [ <comment-text> ]


CREATE <file-name> [ , <extent-size> ]
```

```
DEBUG [ <process> ] [ , TERM [ \<system-name>.]<terminal-name> ]
```

See the PPD command for <process>.

```
DEFAULT  {<default-names> [ , "<default-file-security>" ]}
         {, "<default-file-security>"                    }
```

See the VOLUME command for <default-names>.

```
DELUSER <group-name>.<user-name>
```

```
EXIT
```

```
FC
```

```
FILES [ / OUT <list-file> / ] [ <subvol-set> ]
```

```
HELP [ / OUT <list-file> / ] [<command-name>]
                             [ALL           ]
```

```
INITTERM
```

```
LIGHTS [ON          ]  [ , <sys-option>  ... ] [ , BEAT ]
       [OFF         ]  [ , ALL ]
       [SMOOTH [n]  ]
```

<sys-option> is any one of these three:

```
DISPATCHES
SYSPROCS
PAGING
```

```
LOGOFF
```

```
LOGON [ <group-name>.<user-name> [ , <password> ] ]
```

```
O[BEY] <command-file>
```

```
PARAM [ <param-name>  <param-value>
      [ , <param-name>  <param-value> ]... ]
```

```
PASSWORD [ <new-password> ]

PAUSE   [[\<system-name>.]{$<process-name>}]
        [                 { <cpu>,<pin>   }]

PMSG { ON | OFF }

PPD [ / OUT <list-file> / ] [ <process> ]

   <process> is either of these two:

      [ \<system-name>.]$<process-name>
      [ \<system-name>.]<cpu>,<pin>

PURGE <file-name> [ , <file-name> ]...

RCVDUMP <dump-file> , <cpu> , { X | Y }

RECEIVEDUMP / OUT <dump-file> / <cpu> , <bus>

RELOAD  [ <cpu-set> [ ; <cpu-set> ... ] ]
        [ HELP                          ]

   <cpu-set> is any one of these three:

      { <cpu-range>          } [ , <option> [ , ... ] ]
      { ( <cpu-range> , ...  } [ , <option> [ , ... ] ]
      {  *                   } [ , <option> [ , ... ] ]


      <cpu-range> is either of these two:

         <cpu-num>
         <cpu-num>-<cpu-num>

      <option> is any one of these three:

         $<vol>[.SYS<nn>.OSIMAGE]
         <bus>
         NOSWITCH

REMOTEPASSWORD [ \<system-name> [ , <password> ] ]
```

```
RENAME <old-file-name> , <new-file-name>


RPASSWRD [ \<system-name> [ , <password> ]]


[ RUN[D] ] <program-file>
  [ / <run-option> [ , <run-option> ]... / ] [ <param-string> ]

   <run-option> is any one of these eleven:

        INSPECT {OFF      }
                {ON       }
                {SAVEABEND}

        IN [ <file-name> ]
        OUT [ <file-name> ]
        NAME [ $<process-name> ]
        CPU <cpu-number>
        PRI <priority>
        MEM <num-pages>
        NOWAIT
        TERM $<terminal-name>
        LIB [ <file-name> ]
        SWAP [ <file-name> ]


SET <attribute> { ON | OFF | SAVEABEND }

   <attribute> is INSPECT.

SETTIME {<month-name> <day>}  <year>, <hour>:<minute>
        {<day> <month-name>}
       [ LCT | LST | GMT ]


SHOW [ / OUT <list-file> / ]
     [ <attribute> [ , <attribute> ]... ]

   <attribute> is any attribute controlled by the SET command.
```

```
STATUS [ / OUT <list-file> / ] [ <range> ] [ , <condition> ]...

   <range> is any one of these four:

      [ \<system-name>.]<cpu>,<pin>
      [ \<system-name>.]<cpu-number>
      [ \<system-name>.]$<process-name>
      [ \<system-name>.]*

   <condition> is any one of these five:

      TERM [ $<terminal-name> ]
      PRI  [ <priority> ]
      USER [ <user> ]
      PROG <program-file-name>
      DETAIL


STOP  [[ \<system-name>.]{$<process-name>}]
                         { <cpu>,<pin>   }


SUSPEND  [[ \<system-name>.]{$<process-name>}]
                            { <cpu>,<pin>   }


SWITCH


SYSTEM [ \<system-name> ]


SYSTIMES


TIME


USERS [ / OUT <list-file> / ] [ <range> ]

   <range> is any one of these six:

       (blank)
       <group-id>,<user-id>
       <group-id>,*
       <groupname>.<username>
       [ <groupname>.]*
       *.* or *,*
```

```
VOLUME [ <default-names> ] [ , "<default-file-security>" ]

   <default-names> is any of these three:

      <subvol-name>
      $<volume-name>
      $<volume-name>.<subvol-name>


WAKEUP { ON | OFF }


WHO [ / OUT <list-file> / ]


{X|Y}BUSDOWN <from-cpu> , <to-cpu>


{X|Y}BUSUP <from-cpu> , <to-cpu>
```

APPENDIX B

FUP COMMAND SYNTAX SUMMARY

This appendix gives the complete syntax for all the File Utility Program (FUP) commands. Special terms and punctuation used here are explained in "Syntax Conventions Used in This Manual." All the FUP command parameters are fully defined in the descriptions in the GUARDIAN Operating System Utilities Reference Manual.

```
FUP [ / <run-option> [ , <run-option> ]... / ] [ <command> ]
```

See the RUN command in Appendix A for <run-option>.

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

```
ALLOW <num> { ERRORS | WARNINGS }
```

```
ALTER <file-name> { , <alter-option> } ...

   <alter-option> is any one of these seventeen:

      CODE <file-code>                  ! {0:65535}
      [ NO ] REFRESH
      DELALTKEY <key-specifier>
      DELALTFILE <key-file-number>
      ALTKEY ( <key-specifier> { , <altkey-param> }... )
      ALTFILE ( <key-file-number>, <file-name> )
      PART ( <secondary-partition-num>
        , [ \<system-name>.]$<volume-name>
          [ , <pri-extent-size> [ , <sec-extent-size> ] ] )
      [ NO ] AUDIT
      ODDUNSTR
      PARTONLY
      MAXEXTENTS <maximum-extents>
      BUFFERSIZE <unstructured-buffer-size>
      [ NO ] BUFFERED
      [ NO ] AUDITCOMPRESS
      [ NO ] VERIFIEDWRITES
      [ NO ] SERIALWRITES
      RESETBROKEN

      <altkey-param> is any one of these seven:

         FILE <key-file-number>     ! {0:255}
         KEYOFF <key-offset>        ! {0:2034}
         KEYLEN <key-length>        ! {1:255}
         NULL <null-value>          ! { "<c>" | {0:255} }
         NO NULL
         [ NO ] UNIQUE
         [ NO ] UPDATE


BUILDKEYRECORDS <primary-file-name> , <out-file-name>
                , <key-specifier-list> [ , <out-option> ] ...

   <out-option> is any one of these seven:

      RECOUT <out-record-length>
      BLOCKOUT <out-block-length>
      PAD "<pad-character>"
      EBCDICOUT
      [ NO ] UNLOADOUT
      [ NO ] REWINDOUT
      SKIPOUT <num-eofs>
```

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

   See the INFO command for <fileset-list>


COPY <in-file-name> [ , [ <out-file-name> ]
     [ , <copy-option> ]... ]


   <copy-option> is any one of these seven:

      FIRST  {<ordinal-record-num>                    }
             {KEY { <record-spec> | <key-value> }     }
             {<key-specifier> ALTKEY <key-value>      }
      COUNT <num-records>
      UPSHIFT
      UNSTRUCTURED
      <in-option>
      <out-option>
      <display-option>


      <in-option> is any one of these ten:

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


      <out-option> is any one of these nine:

         RECOUT <out-record-length>
         BLOCKOUT <out-block-length>
         PAD "<pad-character>"
         EBCDICOUT
         VAROUT
         FOLD
         [ NO ] UNLOADOUT
         [ NO ] REWINDOUT
         SKIPOUT <num-eofs>
```

<display-option> is any one of these six:

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

```
CREATE <file-name> [ , <create-param> ]...
```

See the SET command for <create-param>.

```
DEALLOCATE <fileset-list> [ , PARTONLY ]
```

See the INFO command for <fileset-list>.

```
DUP[LICATE] <from-fileset-list>, <to-fileset>
            [ , <rename-option> ... ]
            [ , <file-conversion-option> ... ]
[ , NEW    ] [ , PARTONLY ] [ , SAVEID     ]
[ , OLD    ]                [ , SOURCEDATE ]
[ , PURGE  ]                [ , SAVEALL    ]
[ , KEEP   ]
```

See the INFO command for <fileset-list> and <fileset>.

<rename-option> can be either of these two:

```
PART ( <sec-partition-number>
     , [ \<system-name>.]$<volume-name>
     [ , <pri-extent-size> , [ <sec-extent-size> ] ] )
ALTFILE ( <key-file-number> , <file-name> )
```

<file-conversion-option> can be any one of these four:

```
EXT  {<extent-size>                                 }
     {( <pri-extent-size> , <sec-extent-size> )     }
SLACK <percentage>
DSLACK <percentage>
ISLACK <percentage>
```

```
EXIT
```

```
FC
```

```
FILES [ / OUT <list-file> / ] [ <subvolset> ]

GIVE <fileset-list>, <group-id> , <user-id> [ , PARTONLY ]

   See the INFO command for <fileset-list>.

HELP [ / OUT <list-file> / ] [ <command-name> ]
                             [ ALL            ]

INFO [ / OUT <list-file> / ] <fileset-list>
   [ , DETAIL                                    ]
   [ , STAT[ISTICS] [ , PARTONLY ]               ]
   [ , EXTENTS                                   ]
   [ , USER  {<group-id>,<user-id>     }         ]
   [         {<group-name>.<user-name> }         ]

   <fileset-list> can be either of these:

      <fileset>
      ( <fileset> [ , <fileset> ]... )


   <fileset> is any one of these three:

      [\<system>.][$<volume>.][<subvol>.]<file-name>
      [\<system>.][$<volume>.][<subvol>.]*
      [\<system>.][$<volume>.]*.*


LICENSE <fileset-list>

   See the INFO command for <fileset-list>.


LISTOPENS [ / OUT <list-file> / ] <fileset-list>
          [ , SCRATCH <scratch-file-name> ]


LOAD <in-file-name> , <destination-file-name>
     [ , <load-option> ]...


   <load-option> can be any one of these five:

      SORTED
      <in-option>
      <key-seq-option>
      PARTOF $<volume-name>
      PAD "<pad-character>"
```

See the COPY command for <in-option>.

<key-seq-option> can be any one of these five:

```
MAX <num-records>
SCRATCH <scratch-file-name>
SLACK  <percentage>
DSLACK <percentage>
ISLACK <percentage>
```

```
LOADALTFILE <key-file-number>, <primary-file-name>
            [ , <key-seq-option> ] ...
```

See the LOAD command for <key-seq-option>.

```
PURGE [ ! ] <fileset> [ , <fileset> ] ... [ ! ]
```

See the INFO command for <fileset>.

```
PURGEDATA <fileset-list> [ , PARTONLY ]
```

See the INFO command for <fileset-list>.

```
RENAME <old-fileset-list>, <new-fileset> [ , PARTONLY ]
```

See the INFO command for <fileset-list> and <fileset>.

```
RESET [ <create-spec> [ , <create-spec> ]... ]
```

<create-spec> is any one of these:

```
TYPE
CODE
EXT
REC
REFRESH
AUDIT
BLOCK
IBLOCK
COMPRESS
DCOMPRESS
ICOMPRESS
KEYLEN
KEYOFF
```

```
      ALTKEY [ <key-specifier> ]
      ALTKEYS
      ALTFILE [ <key-file-number> ]
      ALTFILES
      ALTCREATE
      PART [ <partition-num> ]
      PARTS
      PARTONLY
      MAXEXTENTS
      BUFFERSIZE
      BUFFERED
      AUDITCOMPRESS
      VERIFIEDWRITES
      SERIALWRITES
      ODDUNSTR
```

```
REVOKE <fileset-list> [ , <secure-option> ]...
```

See the INFO command for <fileset-list>.

See the SECURE command for <secure-option>.

```
SECURE <fileset-list> [ , [ <security> ]
       [ , <secure-option> ] ... ]
```

See the INFO command for <fileset-list>.

<security> is either of these:

```
   <security-num>
   "<security-string>"
```

<secure-option> is any one of these:

```
    PROGID
    PARTONLY
    CLEARONPURGE
```

```
SET <create-param> [ , <create-param> ]...

   <create-param> is any one of these:

      LIKE <file-name>
      TYPE <file-type>      ! { U | R | E | K | 0 | 1 | 2 | 3 }
      CODE <file-code>      ! {0:65535}
      EXT {<extent-size>                                    }
          {( <pri-extent-size>, <sec-extent-size> )         }
      [ NO ] REFRESH
      [ NO ] AUDIT
      REC <record-length>                 ! {1:4072}
      BLOCK <data-block-length>           ! {1:4096}
      IBLOCK <index-block-length>         ! {1:4096}
      [ NO ] COMPRESS
      [ NO ] DCOMPRESS
      [ NO ] ICOMPRESS
      KEYLEN <key-length>                 ! {1:255}
      KEYOFF <key-offset>                 ! {0:2034}
      ALTKEY ( <key-specifier> { , <altkey-param> }... )
      ALTFILE ( <key-file-number>, <file-name> )
      [ NO ] ALTCREATE
      PART ( <secondary-partition-num>
             , [\<system-name>.]$<volume-name>
             [ , <pri-extent-size> [ , [ <sec-extent-size> ]
             [ , <partial-key-value> ] ] ] )
       [ NO ] PARTONLY
       ODDUNSTR
       MAXEXTENTS <maximum-extents>
       BUFFERSIZE <unstructured-buffer-size>  ! {1:4096}
       [ NO ] BUFFERED
       [ NO ] AUDITCOMPRESS
       [ NO ] VERIFIEDWRITES
       [ NO ] SERIALWRITES

   See the ALTER command for <altkey-param>.


SHOW [ / OUT <list-file> / ]
     [ <create-spec> [ , <create-spec> ]... ]

   See the RESET command for <create-spec>.


SUBVOLS [ / OUT <list-file> / ]
        [ [\<system-name>.]$<volume-name> ]
```

```
SYSTEM [ \<system-name>[.$<volume-name>][.<subvol-name> ] ]

VOLUME [ [ \<system-name>.]{<subvolume-name>                    } ]
       [                    {$<volume-name>[.<subvolume-name>]} ]
```

APPENDIX C

BACKUP AND RESTORE COMMAND SYNTAX SUMMARY

This appendix contains a summary of the syntax of the commands to start BACKUP and RESTORE programs. The special symbols, terms, and punctuation used here are defined in "Syntax Conventions Used in this Manual." All BACKUP and RESTORE command parameters are fully described in Section 4 of the GUARDIAN Operating System Utilities Reference Manual.

```
BACKUP [ / <run-option> [ , <run-option> ]... / ]
       [ <backup-parameters> ]

   <backup-parameters> are:

      [ \<system-name>.]$<tape-file> , <fileset-list>

      [ , AUDITED                                    ]
      [ , BLOCKSIZE <data-record-size>               ]
      [ , DENSITY <density>                          ]
      [ , IGNORE                                     ]
      [ , LISTALL                                    ]
      [ , MSGONLOCK                                  ]
      [ , NOT <not-fileset-list>                     ]   ...
      [ , OLDFORMAT                                  ]
      [ , OPEN                                       ]
      [ , PARTIAL <partial-dump-date>                ]
      [ , PARTONLY                                   ]
      [ , START <fileset-name>                       ]
      [ , VERIFYTAPE                                 ]
      [ , VOL {[ $<new-vol>.]<new-svol>}             ]
      [       {  $<new-vol>            }             ]
```

$<tape-file> is either of these:

```
$<device-name>
$<logical-device-number>
```

<fileset-list> is either of these:

```
<fileset>
( <fileset> [ , <fileset> ]... )
```

<fileset> is any one of these four:

```
[ $<volume>.][<subvol>.]<file-name>
[ $<volume>.][<subvol>.]*
[ $<volume>.]*.*
*.*.*
```

See the RUN command in COMINT for <run-option>.

Display Form of RESTORE:

```
RESTORE [ \<system-name>.]$<tape-file> [ , , VERIFYTAPE ]
```

Restore Form of RESTORE:

```
RESTORE [ / <run-option> [ , <run-option> ]... / ]
        [ \<system-name>.]$<tape-file> [ , <fileset-list> ]

        [ , AUDITED [ , TURNOFFAUDIT ]                    ]
        [ , IGNORE                                        ]
        [ , KEEP                                          ]
        [ , LISTALL                                       ]
        [ , MYID                                          ]
        [ , NOT <not-fileset-list>                        ]
        [ , NOUNLOAD                                      ]
        [ , OPEN                                          ]...
        [ , PARTOF $<volume-name>                         ]
        [ , PARTONLY                                      ]
        [ , REBUILD                                       ]
        [ , START <fileset-name>                          ]
        [ , VOL {[ $<new-vol>.]<new-svol>}                ]
        [       {  $<new-vol>            }                ]
        [ , TAPEDATE                                      ]
        [ , VERIFY                                        ]
```

See the RUN command in COMINT for <run-option>.

See the BACKUP program for $<tape-file>.

See the BACKUP program for <fileset-list>.

APPENDIX D

PERUSE SYNTAX SUMMARY

The following is a summary of the syntax for all PERUSE commands.

```
PERUSE [ / <run-options>  / ] [ <supervisor> ]

COPIES <number-of-copies>
DEL
DEV $<device-name>
E[XIT]
FC
F[IND] B[OTH] [ / <find-string> / ]
FORM [ <form-name> ]
HELP [ / <OUT listfile> / ] [ <command-name> | ALL ]
HOLD [ ON | OFF ]
HOLDAFTER [ ON | OFF ]
J[OB] [ <job-num> | * | S[TATUS] | $<location-name> ]
```

```
L[IST] [ / OUT <listfile> / ]

       <page-range> [ , <page-range> ] ...

   <page-range> is A[LL] | <page> [ / <page> ] [ C ] [ O ]

   <page> is { F | L | * | <number> } [ + <number> ]
                                      [ - <number> ]

LOC [ #<location-name> ]

NUMCOL <number-of-columns>

OPEN $<supervisor-name>

OWNER { <group-name>.<user-name> }
      { <group-num> , <user-num> }

PAGE [ <number> | F | L | * ]

PRI <priority>

REPORT [<report-name>]

STARTCOL <starting-column>

S[TATUS] [<delay>]
```

APPENDIX E

SPOOLCOM SYNTAX SUMMARY

The following is a summary of the syntax of SPOOLCOM commands that can be executed by members of the super group (user ID 255,nnn). For the complete syntax, considerations, and examples of SPOOLCOM, refer to the GUARDIAN Operating System Utilities Reference Manual.

```
SPOOLCOM [ / <run-options> / ] [ <supervisor> [ ; ] ]

         [ <command> ] [ ; <command> ] ...
```

where <command> is:

```
COLLECT [ $<process-name> ] [ , <subcommand> ] ...
```

where <subcommand> is one of:

```
BACKUP <backup-cpu>
CPU <cpu>
DATA <data-filename>
DELETE
DRAIN
FILE <program-filename>
PRI <process-priority>
START
STATUS [ / OUT <filename / ] [ DETAIL ]
UNIT <unit-size>
```

```
COMMENT <any-text>
```

```
DEV [ $<dev-id> ] [ , <subcommand> ] ...

   where <subcommand> is one of:

      ALIGN
      CLEAR [ DEL ]
      DELETE
      DRAIN
      EXCLUSIVE [ OFF [ ! ] ]
      FIFO [ OFF ]
      FORM [ <form-name> ]
      HEADER [ ON | OFF | BATCH ]
      JOB <job-num>
      PARM <parameter>
      PROCESS <process-name>
      RESTART [ OFF | <interval> ]
      RETRY <interval>
      SKIP [ - ] <num-pages>
      SKIPTO <page-num>
      SPEED <lines-per-minute>
      START
      STATUS [ / OUT <filename>/ ] [ DETAIL ]
      SUSPEND
      TIMEOUT <num-retries>
      TRUNC [ OFF ]
      WIDTH <device-width>
      XREF [ /OUT filename/ ]


EXIT


FC


HELP [ / OUT <filename> / ] [ command | ALL ]
```

```
JOB [ <job-num> ] | ( <qualifier> [ , <qualifier> ] ... ) ]

    [ , <subcommand> ] ...

       where <qualifier> is one of:

          COLLECT <name>
          DATE { FROM <time> [ THRU <time> ] }
               { THRU <time>                 }
          FORM <name>
          LOC <group>[.<destination>]
          OWNER { <groupname>.<username> }
                { <group-#> , <user-#>   }
          PAGES { > | < } <pages>
          REPORT <name>
          STATE <job-state>

       and <subcommand> is one of:

          COPIES <num-copies>
          DELETE [ ! ]
          FORM [ <form-name> ]
          HOLD
          HOLDAFTER [ OFF ]
          LOC [ #<location> ]
          OWNER { <groupname>.<username> }
                { <group-#> , <user-#>   }
          REPORT [ <report-name> ]
          SELPRI <selection-priority>
          START
          STATUS [ / OUT <filename> / ] [ DETAIL ]

LOC #<group> , BROADCAST [ OFF ]

LOC [ #<group>[.<dest> ] ] , DELETE

LOC [ #<group>.]<dest> , DEV [ <device-name> ]

LOC [ #<group> | <dest> | #<group>.<dest> ] ,

    STATUS [ / OUT <filename> / ] [ DETAIL ]

LOC [ #<group>[.<dest> ] ] , XREF [ / OUT <filename> / ]

OPEN $<spool>
```

```
PRINT [ $<process-name> ] [ , <subcommand> ] ...

   where <subcommand> is one of:

      BACKUP <backup-cpu>
      CPU <cpu>
      DEBUG [ OFF ]
      DELETE
      FILE <program-filename>
      PARM <parameter>
      PRI <execution-priority>
      START
      STATUS [ / OUT <filename> / ] [ DETAIL ]
      XREF [ / OUT <listing-device> / ]

SPOOLER [ , <subcommand> ] ...

   where <subcommand> is one of:

      DRAIN
      ERRLOG <filename>
      START
      STATUS [ / OUT <filename> / ] [ DETAIL ]
```

INDEX

READER COMMENT CARD

Tandem welcomes your comments on the quality and usefulness of its software documentation. Does this manual serve your needs? If not, how could we improve it? Your comments will be forwarded to the writer for review and action, as appropriate.

If your answer to any of the questions below is "no," please supply detailed information, including page numbers, under Comments. Use additional sheets if necessary.

- ▶ Is this manual technically accurate? Yes ☐ No ☐
- ▶ Is information missing? Yes ☐ No ☐
- ▶ Are the organization and content clear? Yes ☐ No ☐
- ▶ Are the format and packaging convenient? Yes ☐ No ☐

Comments

Name Date

Company

Address

City/State Zip

GUARDIAN™ Operating System User's Guide

NonStop™ Systems

82396 A00

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

B U S I N E S S R E P L Y M A I L

FIRST CLASS PERMIT NO. 482 CUPERTINO, CA, U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

Tandem Computers Incorporated
Attn: Manager—Software Publications
Location 01, Department 6350
19333 Vallco Parkway
Cupertino CA 95014-9990

TAPE

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-2599