decdatasystem

# cts300 system users guide

digital

# CTS-300

# System User's Guide

Order No. AA-C747C-TC

May 1980

This manual is a guide for users applying the DIBOL language and the CTS-300 utility programs in the CTS-300 environment.

**SUPERSESSION/UPDATE INFORMATION:** This document supercedes CTS-300 User's Guide (AA-C747B-TC).

**OPERATING SYSTEM AND VERSION:** RT-11 V4

**SOFTWARE VERSION:** CTS-300 V6

digital equipment corporation · maynard, massachusetts

First Printing, September 1977
First Revision: October 1978
Second Revision: May 1980

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

8/80-14

## CTS-300 REVISION HISTORY

With the release of the CTS-300 Version 6 software, this user's guide has been revised to reflect changes resulting from the software changes and to provide a more useful manual.

The primary software changes are the addition of CTSGEN which replaces TSDGEN; the ability of XMTSD to be run in the foreground (and the resulting communications now possible between foreground and background); the DIBOL keyboard editor, DKED; and the PRINTU report generator utility.

The entire manual has been restructured for ease of use. The preface explains the new organization and goals for the manual.

The highlights of the documentation changes are:

- The above software changes have been documented.
- The error messages are now listed in one place -- at the end of the manual in Appendices A and B.
- Chapters on ISAM, DDT, and SORT have been extensively reworked.
- Material has been added on CTS-300 file types.

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

# PREFACE

## GOALS

This manual supplies information specific to users of a CTS-300 system. It is not a tutorial manual nor is it purely a reference manual. However, one goal is to supply information so a new user can use the system and another goal is to provide enough easy-to-find detail so the experienced user can build and use a system for greater functionality and efficiency.

## ASSUMPTIONS

The ability to write simple DIBOL programs is assumed as is the ability to perform basic RT-11 operating system procedures. If you are a new user of CTS-300 without this background, you should refer to the related documentation listed at the end of this preface. Specifically recommended is the Introduction to CTS-300 and DIBOL (AA-5519A-TC).

## STRUCTURE

The body of the manual is divided into three sections:

### SECTION I Introduction to CTS-300

This section will be of primary interest to the new user. It discusses the general capabilities of the system and explains the general system components. The kinds of files the system uses are also covered.

### SECTION II Development

Information you need to build both a working system and programs to run under that system is in this section. This section covers the relationship of the CTS-300 System to the RT-11 SYSGEN and details the CTSGEN procedure. The DKED editor is presented and program compilation discussed in detail.

### SECTION III Utilities

All the CTS-300 utilities (except CTSGEN and DKED which are covered in Section II) are described in this section.

The remainder of the manual consists of two appendixes. Appendix A contains the CTS-300 System error messages. Messages in this appendix are listed in sequence by error number. Appendix B contains error messages for all the supplied utility programs.

## DOCUMENTATION CONVENTIONS

The DIGITAL Command Language (DCL) format is used for the majority of commands used in examples.

User input is shown printed in red.

The carriage return terminator is not shown at the end of command lines; its use is assumed. It is shown, however, where confusion might result if it were missing.

## RELATED DOCUMENTATION

| Document Title | Order Number |
|---|---|
| Introduction to CTS-300 and DIBOL | AA-5519A-TC |
| CTS-300 Release Notes and Installation Guide | AA-5697E-TC |
| CTS-300 Concepts and Facilities | AA-5495A-TC |
| DIBOL-11 Language Reference Manual | AA-1760F-TC |
| PDP-11 MACRO-11 Language Reference Manual | AA-5075B-TC |
| RT-11 Documentation Directory | AA-5285E-TC |
| Introduction to RT-11 | AA-5281B-TC |
| RT-11 System User's Guide | AA-5279B-TC |
| RT-11 System Installation and System Generation Guide | AA-H376A-TC |
| RT-11 Software Support Manual | AA-H379A-TC |
| RT-11 Master Index | AA-H380A-TC |
| RT-11 Keypad Editor User's Guide | AA-H366B-TC |
| RT-11 Programmer's Reference Manual | AA-H378A-TC |
| RT-11 System Message Manual | AA-5284C-TC |
| RT-11 Pocket Guide | AA-5287C-TC |
| RT-11 System Release Notes | AA-5286C-TC |
| DECFORM User's Guide | AA-5792D-TC |

## INTRODUCTION TO SECTION I

Section I contains material to aquaint the new user with CTS-300. Chapter 1 explains the relationship of CTS-300 to RT-11 and briefly highlights some of the major components of CTS-300 and how they relate to one another. Chapter 2 lists some of the RT-11 monitor commands more commonly required by the CTS-300 user. Also covered are file conventions and CTS-300 data files.

## INTRODUCTION TO SECTION I

Section I contains material to aquaint the new user with CTS-300. Chapter 1 explains the relationship of CTS-300 to RT-11 and briefly highlights some of the major components of CTS-300 and how they relate to one another. Chapter 2 lists some of the RT-11 monitor commands more commonly required by the CTS-300 user. Also covered are file conventions and CTS-300 data files.

# CHAPTER 1

# INTRODUCTION TO CTS-300

## 1.1 INTRODUCTION

The Commercial Transaction System 300 (CTS-300) is an operating system for the DIGITAL Datasystem 300 series of equipment.

CTS-300 consists of a group of programs, including an RT-11 monitor, RT-11 utility programs, and CTS-300 programs. This group of programs organizes the processor and peripheral devices into a working unit for the development and execution of DIBOL programs. DIBOL is DIGITAL's business-oriented language and is designed for ease of use in the business community. The result is an operating environment expressly suited to business data processing needs.

The most visible parts of a CTS-300 Operating System are the RT-11 monitor, if it is a single-user system, the CTS-300 Run-Time System (RTS), if it is a time-shared system, and the CTS-300 utility programs. All these are outlined in this chapter and explained later in this manual.

This manual is directed toward CTS-300 related subjects. RT-11 components are discussed in complete detail in the appropriate RT-11 documentation.

## 1.2 MONITORS

A monitor is a master control program that observes, supervises, controls, or verifies the operation of a computer system. It coordinates all activities in a system, including I/O supervision, resource allocation, program execution, and operator communication. There are three monitors associated with CTS-300. These three monitors are supplied by RT-11 and they are:

- Single-Job Monitor (SJ)
- Foreground/Background Monitor (FB)
- Extended Memory Monitor (XM)

### 1.2.1 SJ Monitor

The Single-Job (SJ) monitor runs in systems with up to 64 K bytes of memory. This monitor has limited capability when compared with the other two, but it has the advantage of being small.

### 1.2.2 FB Monitor

The Foreground/Background (FB) monitor runs in systems with 64 K bytes of memory and allows access to both the foreground and the background divisions of memory. Foreground/background operation provides a way to share the processor between two programs; one in the foreground and one in the background. For example, during program execution, the FB monitor could run the single-user line printer spooler program in the foreground part of memory for output while it accepts keyboard input in the background part of memory.

### 1.2.3 XM Monitor

The Extended Memory (XM) monitor supports the same capabilities as the FB monitor and, in addition, this monitor will run in systems with up to 256 K bytes of memory. With the XM monitor, it is also possible to run the DIBOL extended memory time-shared RTS program in the foreground.

## 1.3 RUN-TIME SYSTEMS

Run-time systems are a function of CTS-300. The purpose of the CTS-300 RTS is to provide facilities for interpreting DIBOL code. Additionally, in a multiple-user system, a run-time system provides executive control over the various programs that may be running.

There are three run-time systems:

- Single-User
- Time-Shared
- Extended Memory Time-Shared

### 1.3.1 The Single-User System

As its name implies, a Single-User DIBOL (SUD) RTS is for single-user, single-program execution. Programs developed for single-user operation automatically call the interpretive program from memory whenever such a single-user program is run. Single-user programs operate under the direct control of an RT-11 monitor.

### 1.3.2 The Time-Shared System

A Time-Shared DIBOL RTS (called TSD in the non-extended memory version) is for multiple-user, multiple-program execution. Application programs developed for multiple-user operation run under the control of a time-shared run-time program. This time-shared run-time program, developed by the user with supplied CTS-300 resources, contains the interpretive code that operates like the single-user system explained above and, in addition, exercises control over the DIBOL programs being run by the users. The time-shared run-time program, itself, runs under the control of an RT-11 monitor.

Although it is recommended that the TSD program be run under the SJ monitor, it can run under the FB or the XM monitor. In the latter case, it must always run in the background; and, of course, the extended memory capability is not available, even with the XM monitor.

### 1.3.3 The Extended Memory Time-Shared System

The Extended Memory Time-Shared System (XMTSD) is a special version of the time-sharing program to allow programs to load and execute above 64 K bytes. To operate above 64 K bytes, XMTSD must be used with the XM monitor.

When XMTSD is run under the XM monitor (as it should always be), it can be run in the foreground and this gives it additional important capabilities that are discussed in Chapter 5.

## 1.4 DEVELOPMENT

CTS-300 is supplied to users who are interested in developing application programs to solve specific business problems. Development is divided into two broad categories: program development and system development. Both categories are explained primarily in Section II of this manual.

### 1.4.1 Program Development

The resources provided for DIBOL program development are the DIBOL editor, DKED, the debugging utility, DDT, the DIBOL compiler, DICOMP, and the object module linker, LINK.

DKED, which will operate under control of a time-sharing program, allows you to create and/or modify source files or data files.

DICOMP compiles source files and produces object files ready for linking. DICOMP also provides listings and symbol tables.

DDT, the debugging utility, is documented in Section III of this manual.

Linking is the final step in program development, and this is done under the control of the RT-11 linker utility. Specific examples of linking are provided, where appropriate, throughout the manual.

### 1.4.2 System Development

Before program development can be accomplished, both the hardware and software must be matched to the specific needs of the user. In this manual, this is called system development. The RT-11 and the CTS-300 systems must each undergo this process. For RT-11, there is a special RT-11/CTS-300 SYSGEN, and for CTS-300, there is CTSGEN. Chapter 6, System Development, describes the process.

## 1.5 DIBOL UTILITIES AND PROGRAMS

In addition to the resources already mentioned, there are several programs that are useful, either by themselves or in conjunction with the run-time system. Utilities are described in Section III of this manual.

### 1.5.1 Supplied Utilities

#### Line Printer Spoolers

LPTSP1 and LPTSPL are the line printer spooler utility programs for CTS-300. They are used to output a data file to one or more line printers. These programs function similarly but are different in loading and starting procedures, as well as in error messages. They are documented in Chapter 8.

PRINTU

PRINTU is a utility that provides a means to generate a simple report program to present information in a data file. PRINTU is documented in Chapter 9.

ISAM (ISMUTL)

ISMUTL is a CTS-300 utility program that creates, reports the status of, or reorganizes ISAM files. ISAM files are data files that are accessed through the indexed sequential access method. ISAM files and ISMUTL are documented in Chapter 10.

SORT/MERGE

The SORT/MERGE utility is supplied in the form of two files. SORTG, along with a user-generated control file as input, generates the data division of a DIBOL program which is then compiled with SORTM to produce an object file. The object file is linked to produce the SORT/MERGE program. SORT/MERGE is documented in Chapter 11.

STATUS

STATUS is a utility program that provides information on active time-shared jobs and on the parameters of the time-shared RTS. In addition, while running STATUS, there is an option which allows an active job to be terminated. STATUS is documented in Chapter 12.

REDUCE

REDUCE is a utility that is used to remove the unnecessary blocks from DIBOL run-time programs that result from the way time-shared programs are linked. REDUCE is documented in Chapter 13.

### 1.5.2 Other DIBOL Programs

There are other programs related to CTS-300 systems that could be useful in your application. Among these are DECFORM and RDCP. DECFORM is an application package that is part of the CTS-300 distribution. RDCP is a communications package.

# CHAPTER 2

# BASIC COMMANDS AND FILE CONVENTIONS

## 2.1 INTRODUCTION

There are basic operating commands and file conventions that must be understood and observed if efficient, trouble-free use of the CTS-300 system is to be obtained. It is not the intent of this chapter to explain all the RT-11 commands or to discuss files in detail, but rather to make you aware of everyday requirements.

## 2.2 COMMANDS

Special function keys and keyboard commands provide communication with the keyboard monitor (KMON). These keys and commands allocate system resources, manipulate memory images, start programs and enable file maintenance. After the RT-11 Operating System has been brought up, any RT-11 keyboard command is legal provided the appropriate software module(s) exists on your system. The readiness of KMON to receive a command is indicated by its period (.) prompt character. For the sake of convenience, this chapter contains some of the more commonly used special keys and commands.

All the commands presented here are explained in detail in the RT-11 **System User's Guide.**

### 2.2.1 Special Character Functions

The special functions of some characters on the keyboard are noted below. These special functions are obtained by simultaneously pressing the key marked CTRL and the character noted (C, O, S, Q, U, or Z).

CTRL/C — If CTRL/C trap is not set (see the FLAGS external subroutine in the DIBOL-11 Language Reference Manual), the following action occurs:

A single CTRL/C aborts the program or transaction, clears task-oriented information, and returns control to KMON.

A double CTRL/C operates the same as a single CTRL/C.

~~If~~ a program is accepting input and CTRL/C trap is set, then:

A single CTRL/C has no immediate effect but is stored in the input buffer. If the program accepts input later, the single CTRL/C aborts the program or transaction, clears task-oriented information, and returns control to KMON.

A double CTRL/C aborts the program and returns control to KMON.

| | |
|---|---|
| CTRL/O | A single CTRL/O suppresses terminal output while permitting program execution. A second CTRL/O reenables terminal output. Terminal output is lost between the first CTRL/O and the second. |
| CTRL/S | A single CTRL/S suspends terminal output. |
| CTRL/Q | A single CTRL/Q continues terminal output after suspension by CTRL/S. Terminal output resumes from the point at which it was suspended. No output is lost. |
| CTRL/U | A single CTRL/U erases the entire current line of terminal input. The current line is defined as being the characters between the last line feed (or CTRL/C or CTRL/Z) and the present cursor position. |
| CTRL/Z | Indicates end-of-file (EOF) when the terminal is used as an input file device. |
| RUBOUT | This key erases the character preceding the cursor. This operation continues, right-to-left, up to the beginning of the current line. |

### 2.2.2 Basic Command Rules

The following are general rules that apply to all keyboard commands.

- Keyboard commands can be abbreviated as long as the command remains unique. Throughout the manual optional characters in each keyboard command are noted by brackets [ ].
- Keyboard command syntax requires a minimum of one space between the command and the first argument.
- Each command line must be terminated by pressing the RETURN key. The system accepts only one such command per line.

### 2.2.2.1 Commands to Allocate System Resources

The following are some of the more common commands used to allocate system resources. A brief explanation is given for each. See the RT-11 System User's Guide for details.

| | |
|---|---|
| DATE | The DATE command enters the indicated date into the system and allows you to determine what date (if any) is presently in the system. The system date is assigned to newly created files, new device directory entries, and listing output. |
| TIME | The TIME command allows you to determine the current time of day as kept by the system, or to enter a new time of day. If the KW11-L clock option is not present on the system or if the time is entered incorrectly, an error message is generated. |
| ASSIGN | The ASSIGN command assigns a user-defined logical name as an alternate name for a physical device. Only one logical name can be assigned per ASSIGN command, but several ASSIGN commands can be used to assign different names to the same physical device. |
| DEASSIGN | The DEASSIGN command removes the logical name(s) previously assigned to a device. |
| LOAD | The LOAD command makes a device handler resident. Program execution is faster when a handler is resident, even though memory area for the handler must be allocated. |
| UNLOAD | The UNLOAD command makes previously loaded handlers nonresident, thus freeing the memory they occupied. |
| SET | The SET command changes device handler characteristics and system configuration parameters. |

### 2.2.2.2 Commands to Start a Program

The following two commands are used to start programs from the keyboard. See the RT-11 System User's Guide for details.

| | |
|---|---|
| RUN | The RUN command loads the specified memory image file into memory and starts execution. |
| R | This command is similar to the RUN command except that the file specified must be on the system device (SY:). |

### 2.2.3 Monitor Error Messages

Monitor error messages are produced by the RT-11 Operating System. See the **RT-11 System Message Manual** for these messages.

## 2.3 FILE CONVENTIONS

### 2.3.1 File Naming

CTS-300 system conventions call for a standard two-character device name. Devices may be assigned logical names which take precedence over physical names. File names consist of one to six characters followed by an optional combination of a period with from one to three characters. Those optional characters and period specify the file name extension and usually indicate the format of the file. If an extension is not specified, most system programs assign default extensions to identify that file. See the following section for common file name extensions for the various types of files.

### 2.3.2 File Types

There are several kinds of files used in an RT-11/CTS-300 system. These fall into the general categories of: system files (monitors, device handlers, command files, and other internal files), program files (source, object, and executable files), and data files. Data files are covered in Section 2.3.3.

As explained in the preceding section, files are assigned an extension that usually indicates the file format and function. Below are listed some of the more common RT-11 file extensions and some extensions unique to CTS-300.

| Extension | Meaning |
|---|---|
| .BAD | file with unreadable blocks |
| .BAK | edit (K52 and DKED) backup file |
| .BAT | batch command file |
| .COM | indirect command file |
| .DDF | DIBOL data file |
| .DBL | DIBOL source file |
| .DIR | directory listing file |
| .ISM | ISAM file |
| .LOG | log file |
| .LST | listing file |
| .MAC | MACRO source files |
| .MAP | output (linker) file |
| .OBJ | object code file |
| .REL | relocatable image file |
| .SAV | memory image file |
| .SYS | monitor or device handler file |
| .TMP | temporary file |
| .TSD | program file linked to run in a DIBOL time-shared system |

### 2.3.3 CTS-300 Data Files

There are three kinds of data files used in CTS-300: sequential files, random access sequential files, and ISAM files.

#### 2.3.3.1 Sequential and Random Access Sequential Files

DIBOL sequential files are composed of fixed-length records containing ASCII characters. As each record is stored, a carriage return / line feed character is automatically appended. When the file is closed (after being opened in output (O) mode) a CTRL/Z character (decimal 026) is written by CTS-300 following the last record. With the exception of multivolume files, discussed below, the CTRL/Z character is required in any sequential file accessed by CTS-300.

#### 2.3.3.2 Multivolume Sequential Files

The CTS-300 system can create a logical file that consists of one or more volumes. Each volume is a single, separately named RT-11 file that can reside on either the same or a different device unit or device type. For example, a CTS-300 data file might consist of six volumes: the first three residing on RK0 as DATA1.DDF, DATA2.DDF, and DATA3.DDF; the fourth residing on RK1 as DATA4.DDF; the fifth, DATA5.DDF, residing on a disk pack that is not presently mounted; and the last on MT0 as DATA6.DDF. This arrangement allows files to be virtually unlimited in length.

As with single-volume sequential files, the last record in the last volume of a file is always followed by a CTRL/Z character. And, as with single-volume sequential files, the file must have been opened in output mode for the CTRL/Z character to have been inserted upon execution of the CLOSE statement. This CTRL/Z character is interpreted by the CTS-300 RTS as a logical end-of-file (EOF).

However, by using one of the FLAGS options (see the **DIBOL-11 Language Reference Manual**), multivolume files can be disabled. In this case, for an input file, an EOF will be returned when the last block is read or a CTRL/Z is detected; whichever is first. For an output file, an error message indicating that the output file is full will appear if the space allocated for the file is exceeded. When the output file is closed, the unused portion of the final block is cleared to nulls, but no CTRL/Z is appended.

It is the responsibility of any DIBOL-11 program and the user to distinguish between the volumes of a multivolume sequential file, since neither the CTS-300 RTS nor the RT-11 Operating System can do so. Each volume is simply another file to RT-11. Further, as far as the run-time system is concerned, each volume is identical except for the one that contains the CTRL/Z character which is detected as an EOF condition. The RT-11 files which comprise the volumes of the logical file may be located anywhere there is room on a disk.

During sequential output (FORMS, WRITES, or DISPLAY statements), whenever RT-11 detects the physical EOF (that is, the last available block is reached) before a CLOSE statement is issued, the run-time system assumes that another volume of the logical file is to be created. The run-time system displays the following message at the console terminal and waits for a response:

```
MOUNT SUCCESSOR TO dev:filnam.ext FOR OUTPUT
```

where dev:filnam.ext is the file whose physical EOF was just found. If another volume is being created, type in the new file specification. For example, if DATA1 had been the file on the first volume, you would type the following:

```
MOUNT SUCCESSOR TO DK1:DATA1.DDF FOR OUTPUT
DK1:DATA2.DDF
```

This new file becomes the next volume in the logical file; the old output file is closed; and the processing continues until either the last available block is used or a CLOSE is issued.

If there is to be no successor to this volume, or if a premature termination is desired, you may respond to the message by typing a CTRL/Z, which causes the OUTPUT FILE FULL error condition. The data that caused the overflow will not be written.

During input (ACCEPT statement with alphanumeric argument or READS statement), if a physical EOF is detected before a logical EOF (CTRL/Z), it is assumed that the logical file continues elsewhere in another volume (RT-11 file). The run-time system displays the following message at the console terminal and waits for your response:

```
MOUNT SUCCESSOR TO dev:filnam.ext FOR INPUT
```

where the meaning and response are similar to the message for the output file above. That is, if another volume is available for input, type in the new file specification, and program processing will continue until a physical EOF is again detected or until programming ends. If there is no successor to this volume, or if premature termination is desired, you can respond to the message by typing CTRL/Z, which causes an EOF error condition, or a branch to the EOF label specified in the ACCEPT or READS statement.

When the next volume of the logical input or output file is to be continued on the same device drive with a different disk or tape, the CTS-300 system must wait while the old disk or tape is exchanged for the new one. This is done by including a /W at the end of the file specification which was given in response to the mount successor message. For example, using the above example for an output successor (only this time with a new device) type:

```
MOUNT SUCCESSOR TO DK1:DATA1.DDF FOR OUTPUT
DK2:DATA2.DDF/W
```

CTS-300 closes the file on the old unit and displays the message:

```
WAITING FOR DK2:DATA2.DDF...
```

You may now exchange the new disk for the old one. When the new disk or tape is ready, type

```
     <CR> or <LF>
```

to open the next volume of the logical file on the new disk or tape and continue processing.

When performing input using the ACCEPT statement with a decimal argument, the program operates as described above except that CTRL/Z is treated as any other character, and termination of the file can be accomplished only by a CTRL/Z response to the mount successor message.

When performing random access operations using READ and WRITE statements, the program must choose the correct volume to access. This is because a record in a random access file is located in relation to the beginning of the file on the current volume being accessed, not to the entire logical file that comprises all volumes. In this mode of operation, it is often useful for a number of volumes to be kept open simultaneously, each on a separate channel, so the program can more easily find the volume containing the desired record.

### 2.3.3.3 ISAM Files

An ISAM (Indexed Sequential Access Method) file is a multivolume file in which the records are stored and accessed according to an index file and a linking scheme that are a part of the ISAM file structure. All volumes of an ISAM file are open simultaneously, so access to the proper device is automatically maintained.

There are some differences between an ISAM file and a sequential file. One difference is that in an ISAM file the EOF marker is not a CTRL/Z but rather an entire record in which each bit is set to a one. Another difference is that in a sequential file a carriage return/line feed is appended to each record.

CTS-300 ISAM files are discussed in detail in Chapter 10.

## INTRODUCTION TO SECTION II

Section II covers the CTS-300 facilities that are provided for system and program development.

Program development begins with a source file consisting of DIBOL program statements. This file can be created with the DIBOL keyboard editor, DKED, described in Chapter 3. The source file is compiled with the DIBOL compiler, DICOMP, described in Chapter 4. The final step in producing an executable program is linking. Linking is illustrated throughout the manual to show specific requirements and is documented in the RT-11 System User's Guide.

Before any program (RT-11 or CTS-300) can be run, however, the total hardware/software system must be configured for its intended use. This is accomplished via an RT-11 SYSGEN and a CTS-300 CTSGEN, both described in Chapter 6. In order to know what the CTS-300 system capabilities are, both for the CTSGEN and for later day-to-day use, the CTS-300 Run-Time Systems are discussed in Chapter 5.

# CHAPTER 3

# DIBOL KEYBOARD EDITOR (DKED)

## 3.1 INTRODUCTION

The DIBOL keyboard editor (DKED) is designed to emulate the RT-11 Keyboard editor, K52. DKED is available for both Single-User Systems (DKED.SAV) and Extended Memory Time-Shared Systems (DKED.TSD). This makes it possible to create and edit source files under either SUD (with EIS) or XMTSD.

The assumption is made that you know how to use K52 (if you do not, see the **RT-11 Keypad Editor User's Guide**). This chapter simply explains the areas where the operating characteristics of DKED are different from those of K52.

## 3.2 USING DKED

Differences between DKED and K52 are discussed in this section.

The more minor differences are:

- DKED is designed for use with the VT52. When a VT100 is used, it is automatically set to the VT52 mode and remains in this mode when you exit from DKED. The keypad functions of the VT100 are similar to those of the VT52.
- If you are operating a single-user system, you must:

  ```
  SET TT SCOPE
  SET TT NOCRLF
  ```

- Messages appear at the top of the display.
- DEL WORD and UNDEL WORD are not implemented.
- CTRL/U operates the same as DELLIN.
- There is no displayed end-of-file (EOF) character.
- You can not open a file of zero length.
- The operator is queried for confirmation of a QUIT command or a YANK command (YANK is a new command; see Section 3.2.2).
- Every command issued must be followed by the <RETURN> key.

Differences that are explained in more detail in the following sections are:

- DKED is organized on a page basis.
- DKED has a new command: YANK.
- There are some new command response messages.
- Default file extensions are different.
- The search mode supports wildcards.
- The HELP command has been expanded.
- CTRL/Z is a valid character in DKED.

### 3.2.1 Page Format

The major difference between DKED and K52 is that DKED is organized on a page basis. A page in DKED is defined as either the area between two form-feed characters or as all the characters preceding a form-feed character that can be contained in the text buffer.

The following messages do not indicate an error condition. They are displayed simply to provide you with additional information. Any valid function or command can be entered.

When a form feed is encountered in a file, the editor sounds the audible alarm and informs you with the message:

```
Page (Ctrl/l) detected
```

If the text buffer should become filled before a CTRL/L is encountered, the editor sounds the audible alarm and informs you with the message:

```
EXCESSIVE PAGE LENGTH.
```

If the EOF is encountered, the editor sounds the audible alarm and informs you with the message:

```
EOF ends current page
```

The page organization has the following additional effects on data insertion and manipulation:

- DKED allows you to finish the last line you are working on, even if the text buffer is full. You must use the PAGE or REOPN command to complete the editing. Maximum line size, including continuation, is 130 characters.

- You can not return to the previous page (or pages) within the file. You must use the REOPN command to return to the beginning of the file.
- Because of the page format, you can not CUT an area that spans a page break.

### 3.2.2 Commands

#### 3.2.2.1 New Commands

| | |
|---|---|
| YANK | The YANK command results in deletion of the present page. When YANK is selected, the editor queries you for confirmation before doing the deletion with the message: ?DKED-W-Kill present page [Y,N] (Y) |
| REOPN | The REOPN command closes a file, reopens it, and places the cursor at the beginning of the file. |

#### 3.2.2.2 Command Response Messages

The following are the characteristics of the command response messages for DKED.

- The QUIT command is unchanged from K52; however, the confirmation message is now:

  ```
  ?DKED-W-Purge output file?  [Y,N] (Y)
  ```

- If you have entered an improperly structured command, a message will appear:

  ```
  ?CSI-F-Illegal command
  ```

- If you have specified an input file that does not exist, a message appears:

  ```
  ?DKED-F-Unable to open input file
  ```

- If there is an output file of the same name as the one you specify, you will receive a message:

  ```
  ?DKED-W-Output file exists- continue?  [Y,N] (N)
  ```

- In a time-shared system, if either the input or output file is currently being used by another user, you will receive one of the following messages:

  ```
      ?DKED-W-Input file being edited by another user
  or
      ?DKED-W-Output file being edited by another user
  ```

- When you attempt to open a file that already exists and then respond in a manner that would cause destruction of that file, a message appears:

  ```
      ?DKED-F-Illegal command sequence.  Please start over.
  ```

  The specific circumstances that will cause this message are:

  a) You open (or REOPN) a file

  ```
      OUTFIL=INFIL or OUTFIL=
  ```

  and the output file already exists. Your response is then Y (continue even though the file exists) followed by a QUIT command.

  b) You open (or REOPN) a file

  ```
      INFIL or INFIL=INFIL
  ```

  and INFIL.BAK exists and the QUIT command is used.

### 3.2.3 File Extensions

When you open a file under DKED, the default extension is .DDF. That is, if you opened a file as

```
    A=
```

the file, after editing and exiting, would be

```
    A.DDF
```

If, however, you opened the file as

```
    A.DBL=
```

the resulting file would have a .DBL extension.

If the file were opened as:

```
    A.=
```

the resulting file would have a null extension.

### 3.2.4 Search

The search function under DKED operates somewhat differently from K52; to some extent, this is due to the page construction. The following explanations apply to searches using the FIND, FIND NEXT, REPLACE, and SUBSTITUTE commands.

#### Model

The model is the search string specifier. It is supplied for the FIND command and serves for the subsequent FIND NEXT, REPLACE, and SUBSTITUTE commands. The model must be changed by another FIND command.

If the first three characters of the model specification are \P\, the search will page until the target is found, or an EOF is encountered. This applies in the forward mode only; see Mode, below. As an example of a search, consider the following models:

| **Model** | **Meaning** |
|---|---|
| \P\ABC | Searches for ABC in the present page and continues to successive pages until the search is successful or the EOF is encountered. |
| ABC | Searches for ABC in the present page and stops when the search is successful or the end of the page is reached. |

#### Mode

If you are in advance mode, the search starts from the present position and is done by line, from left to right, in a forward direction. Whether you search beyond the present page depends on how you specify the model (see above). If you are in backup mode, the search is done by line, from right to left, from the present position, in a backward direction. The backward search remains in the present page; it is not possible to search past the beginning of the present page in the backup mode.

#### SET WILDCARDS ON/OFF

If you select SET WILDCARDS ON, you indicate the presence of certain characters in the target that need not be compared for a successful match. There are two valid wildcard characters: a dot (.) and an asterisk (*).

If you select SET WILDCARDS OFF (the default), wildcard characters are treated like any other character, and there is no wildcard capability.

**The dot wildcard**

The dot (.) indicates that any character in the dot's position will be accepted for a match. That is:

```
     A.B, matches    AXB
                     AYB
                     AZB
```

**The asterisk wildcard**

The asterisk (*) is equivalent to an indeterminate number of dot wildcards and indicates that a match will be made with any character string (including a string of zero length) in the asterisk's position. That is:

```
     A*B, matches    AB
                     AXB
                     AXXXXXXXXB
```

An asterisk wildcard is useful for finding a target when only the first and last characters are known.

**Additional wildcard rules:**

- A model can not begin or end with a wildcard.
- Asterisks and dots can be combined to select a target in which the number of characters in some positions is important and in which the number of characters in other positions is not important.

  That is:

  ```
          A.B*C, matches  AXBC
                          AYBC
                          AXBXC
                          AYBXXXXC
  ```

- Adjacent asterisks are allowed; however, the meaning is equivalent to one asterisk.
- Adjacent asterisks and dots force the indeterminate asterisk string to have a minimum length determined by the number of dots.
- If wildcards are in use, a search can not be made for either the asterisk (*) or dot (.) character.

## 3.2.5 HELP

The HELP frame received in response to the HELP command has changed slightly from the one provided by K52. See Figure 3-1 for the DKED HELP frame. The YANK command is added, and there are two notes whose meaning will be discussed.

```
                DKED V06
                 Keypad Layout for VT52
                 Lower Function is GOLD
!-----------------------------------------------!
!           !           ! DELIN -> !     ^     !   DELETE       RUB CHAR<-
!   GOLD    !   HELP    !          !     !     !   CTRL/U       RUB LINE->
!           !           ! UNDELIN  !  REPLACE  !   CTRL/W       SCREEN
!-----------------------------------------------!   <GOLD>NNNN  REPEAT
!   PAGE    ! FIND NEXT !          !     !     !
!           ! (Note #1) !  BLANK   !     V     !
! COMMAND  7! * FIND   8!         9!  SECTION  !
!-----------------------------------------------!   #1) \P\model = paged
!  ADVANCE  !  BACKUP   ! DELCHR -> !   ---->  !                 search
!           !           !          !           !
!  BOTTOM  4!    TOP   5! UNDLCH  6!   YANK    !   * = Survives commands
!-----------------------------------------------!   set command list by:
!   WORD    !    EOL    !   CUT    !   <----   !   <GOLD><COMAND> and
!           !           !          !           !   respond: "HELPC"
! CHNGCASE 1! DELEOL-> 2! * PASTE 3!  APPEND   !
!-----------------------------------------------!
!     BEGIN OF LINE     !  SELECT  !   ENTER   !
!                       !          !           !
!       OPEN LINE       !  RESET   !   SUBS    !
!-----------------------------------------------!
```

Figure 3-1 DKED HELP Frame

Note 1.

This note in the FIND block reminds you that "\P\" must be specified for a multiple page search (see Section 3.2.4).

**Asterisk (*)**

The asterisk in the FIND and the PASTE blocks indicates that more information pertaining to these two commands is made available by selecting <GOLD><COMMAND> and responding HELPC. The statement "survives command" means the PASTE buffer and search model will survive even if you exit from the file and open another file. HELPC lists a summary of the options and operations related to searching and pasting. The response is a display:

```
1.)  EXIT
2.)  REOPN
3.)  QUIT
4.)  SET SEARCH BEGIN (default)        or:SSB
5.)  SET SEARCH END                    or:SSE
6.)  CLEAR PASTE                       or:CP
7.)  SET EXACT CASEFLAG                or:SEC
8.)  SET EXACT NONE (default)          or:SEN
9.)  SET WILDCARDS OFF (default)       or:SWOFF
10.) SET WILDCARDS ON                  or:SWON
11.) HELP WILDCARDS                    or:HELPW
12.) HELP COMMANDS                     or:HELPC
13.) SEARCH RULES                      or:RULES
14.) MODEL (displays the current search model)

Type <return> to continue:
```

This display shows:

- Items 4 through 13 show the abbreviated commands available with DKED.
- Items 1 and 3 through 8 above are exact duplicates of K52 commands.
- Item 2, the REOPN command (see Section 3.2.2), is useful to place yourself at the beginning of the file.
- Item 12, HELP COMMANDS (or HELPC), is the command to display this list.
- Item 13, the SEARCH RULES (or RULES) command, causes the following display:

```
SET SEARCH BEGIN (or END)
SET EXACT NONE (or CASEFLAG)
SET WILDCARDS OFF (or ON)
ADVANCE (or BACKWARD)
Model: xxx
Paging: No(Yes)
Length: (  n)

Type <return> to continue:
```

This display summarizes for you the current search model options (search begin/end, case exact/none, wildcards on/off, advance/backward search, and paging yes/no). The model you have selected for the search is shown (xxx) and its length indicated by the value of n.

- Item 14, MODEL command, causes the following display:

  ```
  xxx

  Paging:  No(Yes) Length:  ( n) Type <return> to continue:
  ```

  This display is a summary of the current model parameters showing: model (xxx), paging (yes/no), and model length (n).

- Item 11, HELP WILDCARDS (or HELPW) command, causes a description of the wildcard rules to be displayed. The display is self-explanatory.

### 3.2.6 CTRL/Z

CTRL/Z is interpreted in either of two ways by DKED, depending on when it is entered.

**CTRL/Z in a Command String**

If you enter a CTRL/Z when DKED is in the command string interpreter mode (a command is being entered in response to DKED's asterisk prompt), CTRL/Z operates like a CTRL/C; that is, DKED operation is terminated and control returns to the monitor.

**CTRL/Z as a Character in a File**

CTRL/Z can be entered like any other valid character in a file being created with DKED. The response is a ^Z. This allows you to insert the required CTS-300 terminating character in files you create or to add it to existing files. Once you have entered a CTRL/Z, the next time it is encountered DKED will interpret it as the EOF of the file being edited.

## 3.3 ERROR MESSAGES

DKED uses essentially the same error messages as K52. See Appendix B, Table B-1, in this manual and the RT-11 Keypad Editor User's Guide for an explanation of these messages.

# CHAPTER 4

# DIBOL COMPILER (DICOMP)

## 4.1 INTRODUCTION

The DIBOL compiler (DICOMP.SAV) is a CTS-300 program that is run under control of the RT-11 Operating System. It accepts source files that may have been created by the user and creates object files (.OBJ files) that are ready to be linked to produce an executable DIBOL program. This compiler may be invoked via DCL command (DIGITAL's Command Language; see the **RT-11 System User's Guide**); however, that mode does not support the /G, /P:N, and /S options (see Sections 4.2.2.5, 4.2.2.8, and 4.2.2.9).

### 4.1.1 Characteristics

Some DICOMP features:

- Can produce two output files: an object file and a listing file.
- Outputs object code to the single object file.
- Accepts up to six DIBOL source code files.
- Accepts several options which govern the nature of the files produced, the listings, and warning messages.

### 4.1.2 Chapter Organization

The remainder of this chapter is comprised of two sections: Section 4.2, Using DICOMP, and Section 4.3, Error Messages.

## 4.2 USING DICOMP

### 4.2.1 Running DICOMP

DICOMP is executed with the RUN command, like any other program, in response to the RT-11 monitor's prompt. DICOMP responds with an asterisk prompt.

```
.R[U] DICOMP
*[outfil][,listfil]=infil1[,infil2,...infil6][/n][/n]
```

where:

outfil — is the output of the compiler in the general form dev:filnam.ext.

- If not specified, the default device is DK:.
- The default extension is .OBJ.
- If the output file is not specified, no object file is generated.

,listfil — is the listing file in the general form dev:filnam.ext.

- The default device is DK:. Most often the listing file is directed to either the terminal (TT:) or the line printer (LP:) by specifying just the device. This could also be used with just a file name to store the listing on a disk.
- The default extension is .LST.
- If no list file is specified, none is produced; if an object file is specified, only that file is generated.
- If only a listing file is desired with no object file, specify only a listing file device and file specification preceded by the comma. The comma indicates there is no object file.

infil1[infil2...infil6] — are the DIBOL source files supplied as input to the compiler. These files are specified in the general form dev:filnam.ext.

- The default device is the system device DK:.
- The default extension is .DBL.

/n — is the compiler switch option. The options are discussed in detail in the following section. More than one option can be specified. They are separated by slashes.

### 4.2.2 Options

The valid options for DICOMP are listed below in alphabetical order.

#### 4.2.2.1 /A

The /A switch alphabetizes the symbol and label tables. Therefore, symbols and labels do not appear in the order they are encountered by the compiler.

#### 4.2.2.2 /B

The /B switch selects single buffering for I/O. The compiler normally uses two buffers to increase I/O speed. Single buffering slows down I/O time but provides more space for compilation. This additional space may be necessary during program development in an extended memory time-shared environment (see Chapter 5).

#### 4.2.2.3 /C

The /C switch causes a Cross Reference (CREF) listing to be generated. This listing indicates, by line number, where symbols are used and defined. (See Section 4.2.4.)

**NOTE**

The file CREF.SAV must be on the system device in order to obtain a CREF listing.

#### 4.2.2.4 /D

The /D switch causes a symbol table to be generated for use by the DIBOL Debugging Program (DDT). This switch must be specified if you intend to use DDT later.

#### 4.2.2.5 /G

The /G switch creates a log file of the errors generated during compilation. If there were no listing file selected, the log file also contains the statements that are in error. The log file is placed on the system device and takes the name of the first module in the compiler command line. This file always has an extension of .LOG. The user is responsible for removing this file when it is no longer needed.

The main purpose of the log file is to provide a place for error information when a compilation is requested by an operator using XMTSD in the foreground. (See Chapter 5)

#### 4.2.2.6 /L

The /L switch suppresses a program listing if one has been specified in the command string, but it does not suppress the symbol table.

#### 4.2.2.7 /O

The /O switch suppresses the generation of line numbers in the program listing. This has the advantage of using less memory, increasing program speed, and reducing the program size. This switch should be used only with debugged programs; otherwise, the line numbers in the error messages will be meaningless.

#### 4.2.2.8 /P:N

The /P switch allows you to override the default listing page length of 66 lines. The number of lines is selected with the value of N which may be either octal or decimal.

The value of N is interpreted as decimal if it is followed by a decimal point. Every page of a listing has two header lines followed by a blank line. Therefore, a value of N of 16 (/P:16.) would result in a listing page having two header lines, one blank line, and 13 lines of source code.

#### 4.2.2.9 /S

The /S switch suppresses the symbol table and label table listings if a listing file were requested.

#### 4.2.2.10 /W

The /W switch suppresses the warning messages. Since many of the warning messages do not interfere with running the program, they are not always needed.

### 4.2.3 Standard Listings

If a listing file is indicated in the command string, and if neither the program nor the symbol table has been suppressed, the normal output consists of three segments. These are the program listing, itself, the symbol table segment, and the label table segment with an error report appended to the latter.

#### 4.2.3.1 Program Listing

The compiler produces a complete listing of the source program. Sequential line numbers are assigned to most lines in the program. The exceptions are:

- START statements
- Compiler directive statements (for example, IFDEF)
- Continued lines
- Blank lines
- Comment lines

The line numbers are also referenced by the symbol table, the label table segment, and the CREF listing (if selected). Line numbers are especially useful in debugging DIBOL programs.

Errors in the source program also appear in the program listing (see Section 4.3).

#### 4.2.3.2 Symbol Table

The symbol table segment consists of a listing of all the data and literals referenced in the program. The table contains four columns which are entitled NAME, TYPE, DIMENSION, and SIZE. These titles are explained below:

| | |
|---|---|
| NAME | Contains a list of all data field names and literals referenced by the program. |
| TYPE | Contains the field type of the data field and literals found in the column entitled NAME. Field types are: ALPHA, DECIMAL, RECORD, or IMDEF (Improper Definition). Symbol table entries for COMMON variables have a C preceding the type of symbol it is:<br><br>C- ALPHA<br>C- DECIMAL<br>C- RECORD |

| | |
|---|---|
| DIMENSION | Contains the dimension of the data fields and literals listed under the NAME column. This is the array element count defined for the field given under NAME. A dimension of zero is always assigned to any data field or literal which is improperly defined (that is, IMDEF). |
| SIZE | Contains the size, in terms of 8-bit bytes, of the data fields and literals given under NAME. For a field this corresponds to the size of the field or array element. |

#### 4.2.3.3 Label Table

The label table segment contains a listing of all the labels and external subroutines used within the program. The table contains five columns which are entitled NAME, #, TYPE, LINE #, and ORIGIN. The columns headed # and ORIGIN are of little importance to the DIBOL-11 programmer and are not discussed. The headings used in the label table listing are explained below:

NAME — Contains the name of each label and external subroutine referenced by the program.

TYPE — Contains the type of name listed under NAME. Three types are identified:

- LABEL for labels
- EXSUB for subroutines
- IMDEF for both improperly defined labels and improperly defined subroutines.

LINE # — Contains the line number on which a label is defined. This line number is zero for each external subroutine called, and it is also zero for each improperly defined or improperly referenced label or external subroutine.

### 4.2.4 CREF Listing

When a listing file is specified with the /C switch in the compiler string, a Cross Reference (CREF) listing is generated and appended to the end of the compiler listing.

DICOMP normally places a CTRL/Z (^Z) at the end of a DIBOL listing; however, it is not done when a CREF listing is appended. When CREF is specified, a temporary file is opened on the system device.

The CREF listing adds as many as four separate sections to the listing file. A discussion of sections and their functions follows.

#### 4.2.4.1 Symbol Cross Reference Table

Lists, in alphabetical order, each of the user-defined symbols, followed by a series of numbers. The first number, followed by the # symbol, indicates the line number in which the symbol is defined. All other numbers indicate line numbers which reference the symbol. Each page of this section is numbered S-n, where n is the page number in the section.

#### 4.2.4.2 Label Cross Reference Table

Lists, in alphabetical order, each of the program labels, followed by a series of numbers. The first number, followed by the # symbol, indicates the line number in which the label is defined. All other numbers indicate line numbers which reference the label. Each page of this section is numbered L-n, where n is the page number in the section.

#### 4.2.4.3 External Subroutine Cross Reference Table

Lists, in alphabetical order, each of the external subroutines that is called, followed by a series of numbers. Each line number indicates a line number in which that subroutine is called. Each page of this section is numbered X-n, where n is the page number in the section.

#### 4.2.4.4 COMMON Cross Reference Table

Lists, in alphabetical order, each of the user-defined COMMON symbols, followed by a series of numbers. The first number, followed by the # symbol, indicates the line number in which the symbol is defined. All other numbers indicate line numbers which reference the symbol. Each page of this section is numbered C-n, where n is the page number in the section.

## 4.3 ERROR MESSAGES

DICOMP detects three classes of mistakes in the source language files. A mistake is presented to the user as either a warning, an error, or a fatal error.

A warning is issued by the compiler when a statement is potentially a problem in the program. Otherwise, the statement is properly defined and executable. Warning messages do not affect compilation. An error usually indicates a mistake in definition or syntax. Fatal errors indicate that it is impossible to continue processing.

All fatal error messages are printed on the terminal. Two kinds of fatal errors are possible: those which return to the DICOMP prompt and those which return to the RT-11 monitor. A special case is that in which the compiler detects no PROC statement. In this case, entry for the binary output file is removed from the directory, and a message indicating that there is no PROC statement is displayed on the terminal.

Error messages usually contain helpful information to find and correct the problem. A pointer (circumflex or up arrow, depending on the terminal) is displayed which points to the approximate position of the first error detected in the line. If the statement contains an error, two asterisks are placed to the left of the assigned statement number. If the statement contains only a warning, no asterisks appear. The error message appears below the line containing the error.

Below is an example of error reporting for an error in the data portion of a source file:

```
1 RECORD A
2 D1,D2,00
3 D2,D2,00
4 RECORD,X
5 ,A2,'AB'
    ^

INITIAL VALUE NOT ALLOWED
```

The circumflex under line five indicates the point at which the error occurred; the text below the line in error specifies the nature of the error.

Whenever a statement line contains an improperly defined symbol, the error is flagged only on the line where it first occurs. Two asterisks appear to the left of the assigned statement number, with the error message below the line. All subsequent occurrences of the symbol are flagged by the two asterisks only, without the message.

As many as three error and/or warning messages are printed per line. A total error and warning message count, which includes the printed messages and any undefined labels, is printed on the terminal.

Appendix B, Table B-2, contains a complete list of the error messages for DICOMP.

# CHAPTER 5

# CTS-300 OPERATING SYSTEMS

## 5.1 INTRODUCTION

This chapter describes both the single-user and the multi-user (time-shared) environments and presents some of their system requirements. It also contains information on preparing programs and explains how to run programs in either of the two environments.

### 5.1.1 Operating Systems Characteristics

The function of any CTS-300 operating system is to allow you to run DIBOL programs. It provides the interface between a DIBOL program and the RT-11 Operating System. Although there are some special time-shared related instructions, the run-time environment is not a major consideration when writing a DIBOL program. A program becomes a single-user or a multi-user program as a result of how it is linked.

### 5.1.2 General System Requirements

Whether yours is a single-user or a multi-user (time-shared) system, there are preliminary requirements that must be met before programs can be run. These requirements are satisfied by properly running the RT-11 Operating System Generation program (SYSGEN) and the CTS-300 Operating System Generation program (CTSGEN). SYSGEN allows you to build an RT-11 Operating System software environment that is compatible with your hardware configuration. CTSGEN allows you to build a CTS-300 Operating System that is compatible with both the RT-11 system as structured by SYSGEN and the intended DIBOL-11 program requirements. SYSGEN and CTSGEN are discussed briefly in this chapter and in detail in Chapter 6.

### 5.1.3 Chapter Organization

The remainder of this chapter is comprised of six sections: Sections 5.2 through 5.7. Section 5.2, The Single-User Environment, describes the operation of a single-user system. The remaining sections are devoted to time-shared operation. Section 5.3, The Time-Shared Environment, is general in its discussion of time-sharing requirements and characteristics. The time-shared user without extended memory hardware will be interested in Section 5.4, TSD Characteristics, which covers the TSD system. The extended memory time-shared user will find, in Section 5.5, XMTSD Characteristics, a discussion of the XMTSD system and its variations. Section 5.6, Terminating Time-Shared Operation (RTEXIT), will be of interest to all time-shared users. Section 5.7, Utilizing Resources on a Small System, is intended to provide some useful guidelines for users with a floppy-based system. Section 5.8, Error Messages, is a reference to time-shared error messages.

## 5.2 THE SINGLE-USER ENVIRONMENT

### 5.2.1 System Requirements for SUD

For SUD programs you can SYSGEN for either the SJ, FB, or the XM monitor. The SJ monitor is recommended unless you intend to use the SUD print spooler (LPTSP1.REL); then the FB or XM monitor must be used.

The DIBOL compiler produces interpretive code. Therefore, the requirement for DIBOL programs to run under the RT-11 Operating System is that there be a run-time code interpreter available. This interpreter which can be thought of as an executive or a run-time system is produced as a result of the CTSGEN process. It is assigned the name SUD.RTS by CTSGEN and must always reside on the system disk.

If you plan to use either ISAM or the DIBOL debugging program, DDT, you must make an appropriate selection in CTSGEN.

The single-user run-time code interpreter requires approximately 12.4 KB of memory without ISAM or DDT.

### 5.2.2 Preparing Programs

Program editing and compilation of DIBOL-11 programs for SUD systems are performed using any valid editor (EDIT, TECO, K52, DKED, etc.) and the DIBOL compiler. DKED and the DIBOL compiler (DICOMP) programs are described in Chapters 3 and 4 of this manual.

If you plan to use DDT, you must compile with the debugging option.

### 5.2.3 Linking Programs

The object (OBJ) file resulting from compilation must be linked to the DIBOL library (DIBOL.OBJ). The following command string could be used to link the program TEST.OBJ and subroutines SUB.OBJ and SUB1.OBJ for SUD operation:

For programs without DDT:

```
    .LINK TEST,SUB,SUB1,DIBOL
```

Results in a SUD program named TEST.SAV.

For programs with DDT:

The following command string illustrates linking for DDT:

```
    .LINK TEST,SUB,SUB1,TSDDT,DIBOL
```

### 5.2.4 Running Programs

DIBOL programs in a SUD system are run like any other Save file:

```
    .R PROG or .RU dev:PROG
```

The run-time interpreter SUD.RTS is automatically brought into memory whenever a single-user DIBOL program is run.

### 5.2.5 SUD Memory Allocation

The allocation of memory for a SUD System is shown in Figure 5-1.

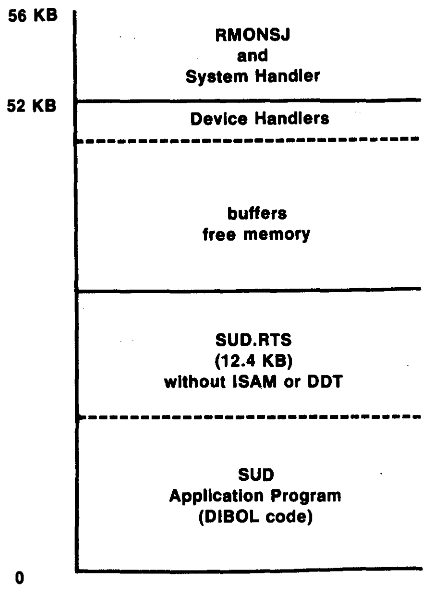


Figure 5-1 SUD Memory Allocation

## 5.3 THE TIME-SHARED ENVIRONMENT

A DIBOL-11 Time-Shared RTS or executive, is a program that runs under the control of the RT-11 monitor. Its purpose is to control time-shared DIBOL user programs. There are two versions available:

- The normal version, identified as TSD, runs under the RT-11 Single-Job (SJ) monitor, the Foreground/Background (FB) monitor, or the Extended Memory (XM) monitor. Of course, TSD can not access extended memory even with the Extended Memory monitor. The SJ monitor is recommended because its small size leaves more memory for user programs.
- The extended memory version, identified as XMTSD, runs under the RT-11 Extended Memory monitor only.

A time-shared RTS permits properly compiled and linked DIBOL-11 programs to be independently loaded and executed in a time-shared environment. The time-shared system communicates with all active terminals to provide interactive program control. Simple keyboard commands, explained below, permit you to initiate and terminate program operation. During execution, each program appears to have at its disposal the full resources of the CTS-300 system. Further, programs can share data files and I/O devices. The time-shared system provides complete facilities for:

- Program loading and execution
- Allocation of memory resources
- Program scheduling
- Detached program operation
- Error detection and reporting
- Sending/Receiving messages between programs (see the **DIBOL-11 Language Reference Manual**)
- Timed wait of program execution (SLEEP statement; see the **DIBOL-11 Language Reference Manual**)
- Line printer spooling (see Chapter 8)

The information in this section applies to both TSD and XMTSD systems.

### 5.3.1 System Requirements for Time-Sharing

As with the single-user system, the DIBOL programs in a time-shared environment require a run-time code interpreter, and, as with the single-user environment, CTSGEN produces this code. However, the CTSGEN for a time-shared system also includes the specification (and resulting construction) of the code to process the time-shared related statements. If you plan to use either ISAM or the DIBOL debugging program, DDT, you must make an appropriate selection in CTSGEN. In addition, active terminals to be recognized by the run-time system are selected in CTSGEN. Special SYSGEN requirements are presented under the TSD and XMTSD sections.

The following table indicates the approximate memory requirements of the two commonly used RT-11 monitors and the two CTS-300 time-shared run-time systems. The time-shared systems both have multiterminal support. The sizes of ISAM and DDT are also shown.

| **Component** | **Approximate Size** |
|---|---|
| SJ Monitor | 3.5 - 4.5 KW |
| XM Monitor | 7.0 - 9.0 KW |
| SUD Interpreter (SUD.RTS) | 6.2 KW |
| TSD Run-Time System | 9.0 KW |
| XMTSD Run-Time System | 12.0 - 16.0 KW |
| ISAM | 2.1 KW |
| DDT | 0.6 KW |

### 5.3.2 Dynamic Memory Allocation

The time-shared RTS dynamically manages the allocation of the free memory used by DIBOL programs. This means that memory space is allocated automatically on a demand basis. When you request that a program be executed, the run-time system loads the program into the first free area of memory that it finds large enough to contain the program. When a program terminates execution, the memory space that it occupied becomes available for reallocation to another program. When free memory becomes fragmented to the extent that no contiguous area of memory is large enough to contain the program to be loaded, the run-time system attempts to relocate currently existing programs to make the necessary room. The free memory spaces are concatenated and, if there is then enough space, the program is loaded.

### 5.3.3 Scheduling

To effect time-sharing operation, the run-time system performs program scheduling (time sharing) on both a DIBOL-11 interpretive instruction basis and an I/O request basis. This means that a program will run until a predetermined number (usually 64) of DIBOL-11 interpreter instructions are executed or until an I/O statement (READ, WRITE, etc.) is executed. When either of these conditions occurs, the run-time system suspends execution of the current program and schedules another program to run.

Program scheduling on an instruction count basis can be controlled within a DIBOL program by means of the SLICE external subroutine. See the DIBOL-11 Language Reference Manual.

### 5.3.4 Detached Program Operation

A program that is running under the control of a time-sharing system program, and that does not require the constant use of a terminal, can disconnect itself from the terminal by using the DETACH statement. Detached operation disconnects a program either from the terminal that initiated the program's operation or from the terminal to which it was last attached.

When a DETACH statement is executed by a program, the message shown below is displayed:

```
DETACHING

TSD VERSION VBnn-nn  (or XM-TSD VERSION VC nn-nn)
*
```

At this point you may issue either a RUN command or an ATTACH command. See Section 5.3.9.

The detached program runs to completion as long as either no further terminal I/O is required or no error condition occurs that requires a displayed message. If terminal I/O is required, the program's operation is suspended until an ATTACH command is issued from an active terminal.

### 5.3.5 Data File Management

Files created in the time-shared environment are compatible with those created in the single-user environment and conform to the file conventions discussed in Chapter 2 of this manual. Under the time-shared RTS, these data files can also be shared by two or more programs, thus providing the capability of creating a common data base. In addition, programs can also share certain I/O devices with one another.

#### 5.3.5.1 File Sharing

All programs that are to share a file must open the file with the OPEN statement in either update (U) or in input (I) mode. When a program opens a file for either update or input, the file can subsequently be opened by other programs (in I or U modes). For example, one or more programs can be reading a file sequentially (READS statement) while other programs can be directly accessing the file (READ or WRITE statement).

When a record is read using the direct access READ statement in update mode, the blocks within which the record wholly or partially resides are automatically locked. This means that the record cannot be read by another program until it is released. Also, other adjacent records may be locked if they happen to reside wholly or partially within the block(s) that contains the record being read. An attempt to access (READ or WRITE statement) a locked record will cause an error message to be displayed. The block(s) that contains the record is unlocked when the program that originally read the record does one of the following:

- Rewrites the record
- Reads another record from the same file
- Issues an UNLOCK statement (When a program that accesses a file using two or more channels issues an UNLOCK, the lock condition is cleared for all channels.)
- Issues a CLOSE to the channel
- Terminates operation

### 5.3.5.2 Device Sharing

Mass storage direct access devices, such as disks and DECtape, are sharable. Sequential access devices (magtape) are nonsharable devices. However, in the case of magtapes, different device unit numbers are considered to be different devices. Line printers and other non-mass storage devices are also nonsharable.

There is a restriction on the use of the ASSIGN command when devices are being shared. If the ASSIGN command is used to assign a logical name to a disk, then this assigned name must be unique throughout all programs that refer to that device. That is, the device must be referred to by its assigned name and that name only. To do otherwise creates an ambiguous condition for the run-time system with the following results:

- The run-time system will not perform I/O mode checking correctly. If two programs attempt to open (OPEN statement) the same file using different logical device designations, the run-time system will not be able to detect the error condition that should be detected if a file is opened simultaneously for I mode and for O mode.
- The automatic lock feature will not be invoked to prevent simultaneous reading or writing of the same record.

### 5.3.6 Preparing Programs

A program that runs under a time-shared RTS is identical to a program that runs in the single-user environment except that the time-sharing capabilities of certain language statements become operable. Specifically these are: READ, WRITE, READS, WRITES, SEND, RECEIVE, DETACH, and SLEEP. The SLEEP command, while it operates in a single-user system, is primarily a time-sharing statement. Details on the use of these statements are provided in the **DIBOL-11 Language Reference Manual**.

All program preparation operations, including compilation and linking, are identical for the TSD or the XMTSD Run-Time Systems.

Program editing and compilation are performed using any valid editor (EDIT, TECO, K52, DKED, etc.) and the DIBOL compiler (DICOMP). DKED is the only editor you can use under a DIBOL time-sharing RTS. DKED and DICOMP are described in Chapters 3 and 4 of this manual.

If you plan to use DDT, you must include the /D switch in your DICOMP command string.

### 5.3.7 Linking

#### 5.3.7.1 Linking to the Time-Shared DIBOL Library

The object (OBJ) file resulting from compilation must be linked with the time-shared RTS library (TDIBOL.OBJ). The following command string could be used to link, for time-shared operation, the program TEST.OBJ and subroutines SUB.OBJ and SUB1.OBJ:

```
.LINK/EXE:TEST.TSD TEST,SUB,SUB1,TDIBOL/BOT:100000
```

It is recommended that the file name extension .TSD be used for the program file output by the linker. This has two important benefits:

- A file linked for time-sharing operation can be easily distinguished from a file linked for single-user operation. This naming convention also prevents a .SAV file from being overwritten if the time-sharing file has the same file name.
- No extension need be specified in the RUN command for a time-shared program with the .TSD extension.

To save disk storage space, the utility program, REDUCE, should be used to remove the unused blocks resulting from the linking process. See Chapter 13.

### 5.3.7.2 Linking for DDT Use

If the DDT utility program is to be used with a program running in the time-shared environment, the file TSDDT.OBJ must be included in the link command string. The following command string, assuming the program was compiled for DDT, could be used to link (for DDT operation) the program and subroutines shown in the previous example:

```
.LINK/EXE:TEST.TSD TEST,SUB,SUB1,TSDDT,TDIBOL/BOT:100000
```

- The file TSDDT must precede the file TDIBOL in the linker's command string; otherwise, a linker error will result.
- DDT operation requires the time-shared system to be a version in which DDT was selected during CTSGEN. See Chapter 6 in this manual.

In order to save disk storage space, the utility program, REDUCE, should be used to remove the unused blocks resulting from the linking process. See Chapter 13.

## 5.3.8 Creating Overlays

Programs and their subroutines that are to be run in a time-shared environment can be formed into a series of overlays using the same procedures used with any other program. See the **RT-11 System User's Guide.** The main object module, TDIBOL, and TSDDT (if used) must all reside in the root segment of the overlay.

The example below illustrates linking for overlays using the same modules as with linking for DDT, above. In this example, subroutines SUB and SUB1 are linked for non-simultaneous operation in overlay region one. The remaining modules are in the root segment.

```
.LINK/PROMPT/EXE:  TEST.TSD TEST,TSDDT,TDIBOL/BOT:100000
*SUB/O:1
*SUB1/O:1
*//
```

## 5.3.9 Commands

All time-shared run-time systems accept at least two commands: the RUN command and the ATTACH command. Either of these commands can be terminated by a carriage return. When XMTSD is run as a foreground program, it supports additional commands. See Section 5.5.2. The time-sharing program indicates its readiness to receive a command with an asterisk prompt.

#### 5.3.9.1 The RUN Command

A RUN command issued from any active terminal loads and starts execution of a DIBOL-11 program.

The command has the form:

```
     *R filnam[.ext] or *RU dev:filnam[.ext]
```

where:

dev: is the name of the device where the program resides. If not specified, the device DK: is assumed.

filnam[.ext] is the name of the program that is to be run.

- If no extension is specified, the extension .TSD is assumed.
- Programs with overlays must reside on the system device.

Examples:

```
     *R ACCT01
```

Loads and executes program ACCT01.TSD from the device DK:.

```
     *RU RL1:LIST
```

Loads and executes program LIST.TSD from device RL1:.

Program execution continues until one of the following conditions occurs:

- The program issues a STOP or END statement.
- A trappable run-time error occurs that was not trapped by an ONERROR statement.
- A fatal (nontrappable) run-time error occurs.
- A detached program requires the use of a terminal. See ATTACH command, described in the following section.
- The user types a CTRL/C command while the program is attached to the terminal.
- The program is terminated via the kill command in the STATUS program.

#### 5.3.9.2 The ATTACH Command

The ATTACH command connects a program running in the detached state to the terminal from which the command was issued.

The command has the form:

```
     *ATTACH [dev:][filnam.ext]
```

or

```
     *A [dev:][filnam.ext] for XMTSD operation (See Section 5.5)
```

where:

| | |
|---|---|
| dev: | is the name of the device where the program resides. If not specified, the device DK: is assumed. |
| filnam.ext | is the name and extension of the program that is to be attached to the terminal from which the command is entered.<br>• If no extension is specified, the extension .TSD is assumed.<br>• If no file name is specified, the ATTACH command lists all the current detached jobs in the system. |

Example:

```
     .R TSD
     TSD VERSION VBnn-nn
     *ATTACH
     NO DETACHED JOBS
     *R LPTSPL
     TIME SHARED DIBOL LINE PRINTER SPOOLER VB0X-0X
     DO YOU WANT TO RUN DETACHED?
     Y
     DETACHING
     TSD VERSION VBnn-nn
     *ATTACH
     JOBS RUNNING DETACHED


     DK:LPTSPL.TSD
                 .
                 .
                 .
```

### 5.3.10 Programmed Startup

Time-shared programs can be executed with the RUN command discussed above, or their execution can be initiated by a program which is running under a time-shared system. There are three methods by which this can be accomplished: forced job startup, chain mode startup, and implicit job startup.

### 5.3.10.1 Forced Job Startup

A forced job startup occurs when the XCALL statement is used with the RUNJB subroutine in the manner presented here. Forced job startup is used to start up a program that is not already running.

The format is:

```
XCALL RUNJB ('filspec',n)
```

where:

| | |
|---|---|
| filspec | is the CTS-300 file specification for the program to be started up. |
| n | is a number with the following meaning: |

- If n is positive (including 0), it is the number of the terminal to which the program is to be attached.
- If n is -1, it indicates that the program is to be started up in a detached state.

If the program is started up detached and it requires a response, it is necessary to supply this response with a SEND statement prior to the XCALL for the forced startup. The general form is:

```
SEND ('msg','filspec')
        .
        .
        .
XCALL RUNJB ('filspec',-1)
```

where:

| | |
|---|---|
| msg | is a message sent to the program before it is run. This message is required for some DIBOL programs. |
| filspec | is, in both lines, the CTS-300 file specification for the program to be started up. |
| -1 | indicates the program is started in a detached state. |

### 5.3.10.2 Chain Mode Startup

Chain mode is used when it is desired to execute a program following the normal termination of a first program.

The format is:

```
     STOP 'filspec'
```

where:

| | |
|---|---|
| filspec | is the CTS-300 file specification for the desired next program to run. |

If the calling program were detached, the program started in chain mode will also be detached. If this second program requires an operator response, it is necessary to supply this response with a SEND statement prior to chaining. The general form is:

```
     SEND ('msg','filspec')
           .
           .
           .
     STOP 'filspec'
```

where:

| | |
|---|---|
| msg | is a message sent to the program before it is run. This message is required for some DIBOL programs that would normally query the operator. |
| filspec | is, in both cases, the CTS-300 file specification for the program to be started up. |

### 5.3.10.3 Implicit Job Startup

Implicit job startup is used to send a message to a program and to ensure that the program will be automatically started up detached if not already running.

The form is:

```
     SEND ('msg','filspec',n)
```

where:

msg — is the message to be sent.

filspec — is the CTS-300 file specification for the program to be started up.

n — has a value of -2 or -3:

- A -2 indicates that if the program is not currently running, the time-shared RTS will start up the job in a detached state.
- A -3 indicates that a copy of the program will be started up in a detached state. The copy started is in addition to any others that may be currently running.

### 5.3.11 Stopping Programs

A program normally continues until it runs to completion and a STOP or END statement is executed. You can prematurely terminate program operation by typing a CTRL/C at the keyboard of the terminal to which the program is currently attached. The CTRL/C function can be disabled by the CTRL/C trap. It is wise to be certain that the trap is not set if you are planning to use the CTRL/C to terminate program execution. If the program is running in a detached state, the ATTACH command must first be used to connect the terminal to the program before CTRL/C can be used to stop it. When the program stops, control returns to the run-time system.

CTRL/C normally causes immediate program termination. If the program is not waiting for keyboard input (that is, executing a READS statement to the terminal), two CTRL/Cs typed in succession are needed to terminate program operation. Files opened for output (O) mode are lost since they cannot be closed.

Following are some of the implications of the CTRL/C command:

- If keyboard input is pending, or if no I/O operation is in process, CTRL/C stops the program immediately.
- If an I/O operation is in process, CTRL/C stops the program only after the I/O operation is completed.
- Two CTRL/Cs in succession will cause an immediate, unconditional stop, regardless of any I/O operation that may be pending.
- CTRL/C has the following effect on files: Files opened in the output (O) mode are lost, since they cannot be closed. Files opened in the update (U) mode may not have all changes incorporated in them.
- The CTRL/C command can be disabled from within a DIBOL program by means of the external subroutine FLAGS. See the DIBOL-11 Language Reference Manual.

## 5.4 TSD CHARACTERISTICS

This section applies only to the nonextended memory time-sharing program identified as TSD.

### 5.4.1 System Requirements for TSD

For TSD, you must SYSGEN for either the SJ, FB, or the XM monitor. However, even if you use TSD with the XM monitor, you will not be able to access extended memory.

## 5.4.2 Running the TSD System Program

Once the TSD program has been properly built via CTSGEN, time sharing can begin. Load and execute the time-sharing program (TSD RTS) by using the monitor's RUN command in the form:

```
    .R filnam.ext or .RU dev:filnam.ext
```

where:

| | |
|---|---|
| dev: | is the name of the device where the TSD RTS program resides. If not specified, the system device is assumed. |
| filnam.ext | is the name and extension of the TSD RTS. The name is the one specified in answer to the last question in CTSGEN. See Chapter 6. Regardless of the file name you choose, TSD will always identify itself as TSD. |

Once started, the TSD RTS identifies itself by displaying the following message at each of the terminals that it recognizes:

```
    TSD VERSION VBnn-nn
    *
```

where:

| | |
|---|---|
| nn-nn | is the version number. |
| (*) | the prompting character, indicates readiness to accept a command from the keyboard. At this point, any program linked for time-shared operation can be run. |

## 5.4.3 TSD Memory Allocation

The allocation of memory for a TSD System is shown in Figure 5-2.

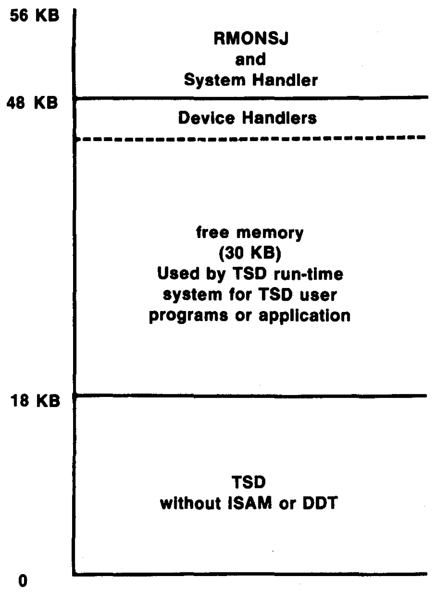

56 KB
RMONSJ and System Handler
48 KB
Device Handlers
free memory (30 KB) Used by TSD run-time system for TSD user programs or application
18 KB
TSD without ISAM or DDT
0

Figure 5-2 TSD Memory Allocation

## 5.5 XMTSD CHARACTERISTICS

This section applies only to the extended memory time-sharing program identified as XMTSD.

The use of virtual overlays supported in RT-11 V4 makes it possible to run a Save image program using virtual overlays as either a background or as a foreground job. Because only XMTSD of the two time-shared system programs uses virtual overlays, only XMTSD can be run as either a foreground or as a background program. As will be seen, this results in some unique capabilities.

As explained in Section 5.3.9, the RUN command can be used in any time-shared RTS to execute a program. It is also possible under XMTSD to execute a program simply by specifying the name of the program. For example:

```
*PROG <CR>
```

will cause program PROG.TSD to be executed. Because of this, the command to attach a program or to display detached programs must be issued as A and not as ATTACH under XMTSD because ATTACH would be interpreted as a program.

### 5.5.1 System Requirements for XMTSD

XMTSD requires that the XM monitor be selected during SYSGEN. If you intend to run XMTSD as a foreground program and plan to communicate with the background (see Section 5.5.2), implicit job startup must be selected during CTSGEN. There is no other special requirement, nor are special questions asked, either for building the XMTSD system or for the foreground / background run capability.

### 5.5.2 XMTSD in the Foreground

If you choose to run XMTSD as a foreground job, the time-shared programs are restricted to running in the extended memory region. This frees a part of lower memory for use by other programs. These other programs could be Save image programs such as PIP, DUP, DIR, DICOMP, LINK, or SUD programs.

When XMTSD is running as a foreground program, it is possible for XMTSD to communicate with a special program (supplied with CTS-300) running in the background. XMTSD has, as part of its system, a queuing routine for directing requests to this special background program. The next several subsections explain the communication process.

### 5.5.2.1 Foreground Queue Program

Foreground communication with the background is initiated via a special series of commands which are similar to DCL commands. It is important to note that these commands are, in reality, TSD programs constructed to implement the communications capability. The function of each is covered in detail in Section 5.5.2.3. A time-shared queue manager program, BGMAN.TSD, a part of XMTSD, is automatically invoked whenever one of these commands is issued. If BGMAN.TSD is not already running, it is started up in a detached state to respond to the command request (hence the need for implicit job startup). The request is queued (16 requests maximum) by BGMAN.TSD and if the request requires it, the information is sent to the special background program.

BGMAN.TSD will also accept input from a user-written program. The same rules and limitations, covered in detail in Section 5.5.2.3 for the SUBMIT, CANCEL, and SHOW commands apply to input from a user-written program. See Section 5.5.2.5 for more information on user-written programs.

BGMAN.TSD has, as part of its code, a timing facility to allocate processor time between the foreground and the background. When BGMAN.TSD is started up, it automatically assigns a value of one to this timer. A value of one allocates 1/60th (or 1/50th - depending on the system clock) of a second to background operation for each XMTSD scheduler cycle. This allocation can be changed when the job is submitted to BGMAN (see SUBMIT, Section 5.5.2.3). The value is returned to one once the submitted request has been carried out. The advantage of increasing this value is to allocate a greater proportion of time to the submitted background job. This allows the job to execute faster; however, foreground programs will slow down proportionately.

If BGMAN.TSD has no pending requests in its queue, it will terminate itself. It is started (implicitly) later if there is a need for it to be involved.

### 5.5.2.2 Background Listener Program

The background program supplied by CTS-300 to receive the requests (indirect files) from the foreground is called LISTNR.SAV. This program is run at the console terminal by the user once XMTSD is running in the foreground. The command and response are:

```
.R LISTNR
CTS300 V6 LISTENER
```

When the indirect file from the foreground is received by the listener program in the background, the listener program displays the following message:

```
BYE
```

and then builds a one-block system indirect file called SYSTEM.BKG. which contains:

```
     @FILE.COM            ;USER PASSED INDIRECT FILE
                          ;RECONSTRUCTED FOR THE RT-11 SYSTEM
     .R LISTNR            ;RESTART LISTENER
```

The listener submits the user request to RT-11 by chaining to SYSTEM.BKG. After the commands within the indirect file are processed, the listener is again started up. The listener displays the version message and informs XMTSD (BGMAN.TSD) that the job is done and that it is ready to receive the next command with its associated instructions in the form of another indirect file. At this time BGMAN.TSD also resets the background time allocation value to one.

### 5.5.2.3 Communications Commands

The special DCL-like communication commands provide the interface between the keyboard and the foreground queue manager (BGMAN). These commands are described in this section. The function of both the foreground queue manager and the background listener programs in response to each command is illustrated in Figure 5-3. Reference to this figure may help clarify the commands which follow.

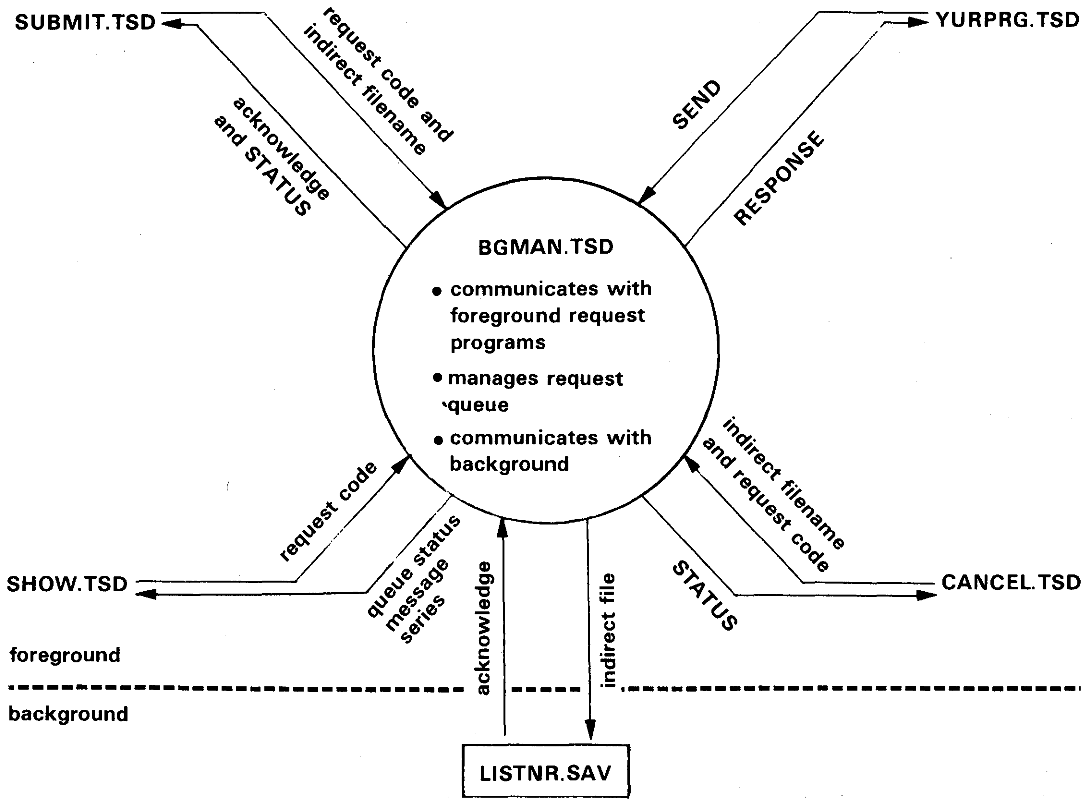


Figure 5-3 BGMAN

Error messages associated with the foreground / background communications commands are listed in Appendix B, Table B-4.

## SUBMIT

This command is used to present a previously constructed indirect file to the background listener program. The response to the command is a request to the user for an indirect file name. The indirect file previously constructed by the user is typically a list of instructions to be performed in the background. After receiving the indirect file name, the command-processing program (SUBMIT.TSD) issues a SEND to BGMAN.TSD. The SUBMIT program receives a response indicating the status of the request and passes it to the user. The command processing program then terminates. To run SUBMIT, the DCL-like command is entered in response to XMTSD's asterisk prompt:

```
    *SUBMIT <RET>
```

SUBMIT prompts for identification of the indirect file:

```
    _FILE/[n]:filspec/n
```

where:

| | |
|---|---|
| filspec | is the file specification of the desired indirect file. The default file extension is .COM. More than one file may be specified, if so, they are separated by commas. |
| n | is an optional number specified by the user to allocate the time available for background processing. It allocates 1/60th (or 1/50th) of a second to background operation for each XMTSD scheduler cycle (see Section 5.5.2.1). The usable range is from 1 to an upper limit determined by the particular application parameters. Two digits are allowed but anything greater than a single number usually has an unacceptable effect on the foreground. The default value of n is 3. |

SUBMIT.TSD returns a message indicating the status of your request:

Assume that jobs A.COM and C.COM, requested from terminal 1, are already in queue with the default time allocation of three and that a submission request is made for a job B.COM from terminal 2 as follows:

```
    _FILE/[n]:B/5
```

The response is:

```
     BACKGROUND QUEUE STATUS

     ENTRIES:  3
     CURRENT JOB:  DK:X.COM

     TERMINAL      FILENAME     TIME ALLOC
     _________________
        1          DK:A.COM          3
        1          DK:C.COM          3
        2          DK:B.COM          5

     *** END ***
```

The current job listed above is the file being processed at the time the command was issued.

## SHOW

The SHOW command allows you to determine the status of your request at some time after a request was submitted. SHOW does not communicate with the background. It simply receives a response from BGMAN.TSD regarding the contents of the queue file. The command is entered in response to XMTSD's asterisk prompt:

```
     *SHOW <RET>
```

The response format is identical to that illustrated in the SUBMIT command above.

The SHOW command can be used to determine if the background listener program is running. If the listener is not running, an error message appears informing you of this fact.

## CANCEL

The CANCEL command is issued whenever you want to retract a request while it is still in the BGMAN processing queue. The command is entered in response to XMTSD's asterisk prompt:

```
     *CANCEL <RET>
```

CANCEL prompts for identification of the indirect file(s):

```
     _FILES:filspec
```

where:

| | |
|---|---|
| filespec | is the file specification of a previously submitted indirect file. All files with the same file specification are canceled. The default extension is .COM. More than one file may be specified, if so, they are separated by commas. |

CANCEL returns the status of the queue thereby showing the file deleted. For example, a CANCEL request sequence for job A.COM (assuming A.COM, B.COM, and C.COM are in queue), would be:

```
     _FILES:A
```

The response is:

```
     BACKGROUND QUEUE STATUS

     ENTRIES:  2
     CURRENT JOB:  DK:X.COM

     TERMINAL      FILENAME     TIME ALLOC
     ___________________
        1          DK:C.COM         3
        2          DK:B.COM         5

     *** END ***
```

The current job listed above is the file being processed at the time the command was issued.

### 5.5.2.4 BGMAN.TSD Operation

An understanding of the operation of BGMAN.TSD in relation to both the requests submitted to it and the responses it returns is essential to anyone writing a program to be used as a special DCL-like command in the same way that SUBMIT and CANCEL are used. BGMAN.TSD interfaces with such a program via a message record. The details of this record and its use in communicating with BGMAN are the subject of this section.

The message record is structured identically for both requests and responses; however, some fields are used differently. For this reason the record is shown and discussed from both viewpoints.

The format of the message record for requests is:

```
         RECORD CDMESS

 F1,     D2               ;REQUEST CODE DEPENDENT (SEE
                          ;REQUEST CODE 01 BELOW)
 F2,     A2               ;REQUEST CODE (SEE BELOW)
 F3,     D2               ;TERMINAL NUMBER
 F4,     A10              ;ORIGINATING PROGRAM NAME
 F5,     A14              ;INDIRECT FILE SPECIFICATION
```

**Request Codes**

There are five request codes (field F2) recognized as input by BGMAN.TSD. They are listed with their function below:

| Request Code | Function |
|---|---|
| 01 | Enters the information specified in fields F3, F4, and F5 into the BGMAN queue. If a background time allocation other than three is desired, it is specified in F1. |
| 02 | Requests a response from BGMAN showing the queue status. The other fields are unused. |
| 03 | Requests that the file (if still in the queue) specified in F3, F4, and F5 not be processed when it is encountered in the queue. |
| 04 | Same as 01 except for the expected response (see responses for request code 04). |
| 05 | Same as 01 except for the expected response (see responses for request code 05). |

The format of the message record for receiving responses is:

```
          RECORD CDMESS

 F1,      D2                 ;RESPONSE CODE DEPENDENT (SEE
                             ;RESPONSE CODE 12, BELOW)
 F2,      A2                 ;RESPONSE CODE (SEE BELOW)
 F3,      D2                 ;UNUSED
 F4,      A10                ;UNUSED
 F5,      A14                ;RESPONSE CODE DEPENDENT (SEE
                             ;RESPONSE CODES 13 AND 14, BELOW)
```

## Response Codes

The possible response code(s) (field F2) received by a program from BGMAN.TSD depends on the original request.

| Request Code | Response Code | Meaning |
|---|---|---|
| 01 | 10 | The entry has been accepted, |
| | or one of the following: | |
| | 20 | The queue is full and the request has been rejected. |
| | 22 | The indirect file was not found. |
| | 23 | A device handler required by an element of the indirect file is not available. |
| | 30 | The special background listener program is not running. |
| 02 | 50 | The queue is empty, |
| | or the following series: | |

The following responses to 02 occur as a series of messages with the codes appearing sequentially in the order shown. Therefore, you will have to do successive receives and process each message as it is received. The codes received will be 12, 13, 14 (possibly multiple), and finally 19.

| | | |
|---|---|---|
| | 12 | The number of jobs in queue is contained in F1. |
| | 13 | The current job name is contained in F5. |
| | 14 | A queue entry is contained in F5. Repeated messages with this code will probably be sent by BGMAN until all queue entries are sent. |
| | 19 | Signals the end of the report from BGMAN. All queue entries have been passed. |
| 03 | 50 | The queue is empty, |
| | or the following series: | |
| | 12, 13, 14, 19 | A cancel request (03) automatically includes a request for a queue status report (see responses to request code 02). |

| | | |
|---|---|---|
| 04 | 10 | Entry accepted, |
| | followed by: | |
| | 40 | Job completed, |
| | or the response if there is an error is one of the following codes: | |
| | 20, 22, 23, 30 | Same as for request 01. |
| 05 | 10 | Entry Accepted, |
| | followed by the series: | |
| | 12, 13, 14, 19 | An 05 submission request automatically includes a request for a queue status report (see responses to request code 02). The SUBMIT.TSD program supplied with CTS-300 uses this code. |
| | or the response if there is an error is one of the following codes: | |
| | 20, 22, 23, 30 | Same as for request 01. |

### 5.5.2.5 User-Created Commands

The ability to execute a program under XMTSD by simply entering its name, the capabilities of XMTSD running in the foreground, and the special background program (LISTNR), enable you, the user, to create your own programs to be executed as commands.

Several unsupported utilities have been included in the V6 distribution of CTS-300. These are designed as examples to illustrate how utilities can be created to perform various functions. They are furnished in both DIBOL source code and as XMTSD executable programs. These utilities present messages based on the following codes:

| **Code** | **Message** |
|---|---|
| 10 | ENTRY ACCEPTED |
| 20 | QUEUE FULL-ENTRY REJECTED |
| 22 | FILE NOT FOUND |
| 23 | HANDLER NOT FOUND |
| 24 | I/O ERROR |
| 25 | DEVICE IN USE |
| 26 | DIRECTORY I/O ERROR |
| 30 | BACKGROUND LISTENER NOT RUNNING REQUEST DISCARDED |
| 40 | JOB COMPLETED |
| 90 | ILLEGAL BACKGROUND OPERATION |

A short description of each utility follows:

CREATE.TSD

This program asks for an output file name and transfers keyboard input into that field. It is suitable for creating short indirect command files for submission to the background. It produces the following prompt line:

```
-FILE:
```

The default extension is .COM. Errors are reported for: 22 and 23. CREATE.TSD is terminated with a CTRL/Z.

COPY.TSD

This program ask for the standard command string input without wild card features. It copies an ASCII text file from the input to output. The prompt character is an asterisk. It is of the form:

```
DEV:FILEOUT.EXT=DEV:FILEIN.EXT
```

Errors are reported for: 22, 23, 24, 25, and 26.

DELETE.TSD

This program is used to delete a file from a TSD keyboard. It accepts multiple file specifications separated by commas. There may be up to eight files. The prompt is:

```
-FILES:
```

Errors are reported for: 22,23, and 26.

RENAME.TSD

This program is used to rename a file from a TSD keyboard. The prompt character is an asterisk and it expects a standard command string input without wild card features. The output is the new filename and the input is the old filename.

```
DEV:NEWNAME.EXT=DEV:OLDNAME.EXT
```

Errors are reported for: 22, 23, and 26.

TYPE.TSD

This program asks for a file name without wildcard features, reads the file, and writes it to the terminal. It reads ASCII text files only. The default extension is .DBL. It is suitable for examining DIBOL program source code without having to use an editor. The prompt is:

```
-FILE:
```

Errors are reported for: 22, 23, 24, and 26.

PRINT.TSD

This program accepts multiple file specifications without the wild card feature and submits them to the line printer spooler. The prompt is:

```
-FILES:
```

Errors are reported for: 22, 23, 24, and 26.

| | |
|---|---|
| VIEW.TSD | This program is a modified version of RTEXIT and is used for system monitoring, or for getting job reports without having to run STATUS. There is no input. |
| SUBMIW.TSD | This program is an example of how to communicate with the background manager. It is a submit-and-wait-for-completion command. It takes multiple file specifications for each request submitted. The prompt is: `-FILES:` Codes received are: 10, 20, 22, 23, 24, 26, 30, 40, and 90. |

### 5.5.2.6 Running XMTSD in the Foreground

Once the XMTSD program has been properly built via CTSGEN, time sharing can begin. Assume you have four terminals, including the console terminal, and that four terminals were selected in both the RT-11 SYSGEN and the CTS-300 CTSGEN. It is recommended that you use the following procedure to execute the time-sharing program and the background listener program.

Load and execute the time-sharing program (XMTSD RTS) by using the foreground RUN command in the form:

```
.FRUN filnam.SAV
```

where:

filnam    is the name and extension of the XMTSD RTS as specified in answer to the last question in CTSGEN (see Chapter 6). Regardless of the file name you choose for your XMTSD RTS, XMTSD will always identify itself as XM-TSD.

- The extension .SAV must be specified.
- The program must be run from the system device.

Once started, the XMTSD RTS identifies itself by displaying the following message at each of the terminals that it recognizes (in this case, terminals 1, 2, and 3):

```
XM-TSD VERSION VCnn-nn
*
```

where:

nn-nn    is the version number.

(*)        the prompting character, indicates readiness to accept a command from the keyboard. At this point, you may set terminal characteristics but do not run any program unless you do not intend to run the listener.

If you want XMTSD to have access to the background, run the listener program at terminal 0, the background console. The command and response are:

```
.R LISTNR
CTS300 V6 LISTENER
```

Enter the following to return to the foreground to run time-sharing programs:

```
CTRL/F
```

#### 5.5.2.7 Memory Allocation

The allocation of memory for an XMTSD RTS in which XMTSD is run as a foreground program is shown in Figure 5-4.

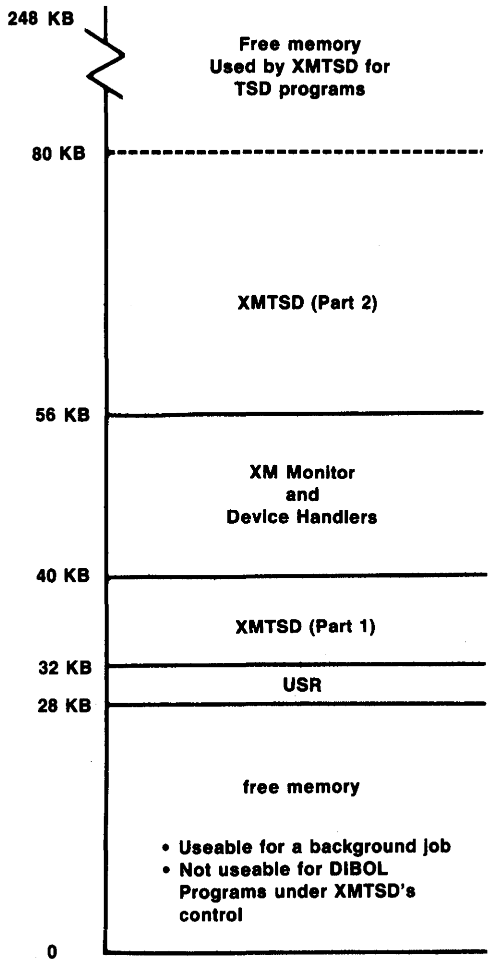


Figure 5-4  Memory Allocation for XMTSD as a Foreground Program

### 5.5.2.8 Applications

When XMTSD is run in the foreground, the system can be thought of as running in what is primarily a development mode, because in this mode many activities normally associated with development are efficiently handled. Among these are concurrent development and remote patching. Possible operation as a production system is also discussed.

#### Concurrent Development

The entire process of application program development can be undertaken by several users of a time-shared DIBOL system. Any user at a foreground terminal can edit and debug under XMTSD. The DIBOL editor DKED.TSD (see Chapter 3 in this manual) can be used to create and edit source files. These files can be DIBOL programs or indirect files consisting of commands to be sent to the background. Furthermore, by communicating with the background program, LISTNR, programs can be compiled and linked.

Note that when compiling programs in this mode you must use the /G option to create an error log file which will contain possible errors. See Chapter 4, DICOMP, for more information.

#### Remote Patching

Since it is possible for a remote terminal to communicate with the foreground program XMTSD and, through it, with the background, any remote terminal can be used to perform system maintenance, including patching.

#### Production Mode

While running XMTSD in the background is considered the conventional production mode (see Section 5.5.3), it is possible that running XMTSD in the foreground could be useful in some production environments. This would be an application that could make use of the background processing capability.

In such a situation, you would probably want all terminals, including the console terminal, to be available for production use. To accomplish this you must:

- Suppress the CTS300 V6 LISTENER and BYE commands generated by the listener. This is done by running RT-11 PATCH and setting location 1000 in LISTNR.SAV to a one.
- Set TT:QUIET for the console terminal.
- Ensure that the background operation you are performing does not output to the terminal.
- Be sure not to enter a CTRL/B which would attach the terminal to the background.

### 5.5.3 XMTSD in the Background

This is the conventional mode of operation in which XMTSD has the same capabilities and limitations (except, of course, for the greater memory capacity) as TSD running with the non-extended memory monitor.

#### 5.5.3.1 Running XMTSD in the Background

Once the XMTSD program has been properly built via CTSGEN, time sharing can begin. Load and execute the time-sharing program (XMTSD RTS) which must be on the system device by using the monitor's RUN command in the form:

```
.R filnam.ext
```

where:

filnam.ext
: is the name and extension of the XMTSD RTS as specified in answer to the last question in CTSGEN. See Chapter 6.

Once started, the XMTSD RTS identifies itself by displaying the following message at each of the terminals that it recognizes:

```
XM-TSD VERSION VCnn-nn
*
```

where:

| | |
|---|---|
| nn-nn | is the version number. |
| (*) | the prompting character, indicates readiness to accept a command from the keyboard. At this point, any program linked for time-shared operation can be run. |

### 5.5.3.2 Memory Allocation

The allocation of memory for an XMTSD System in which XMTSD is running as a background program is shown in Figure 5-5.

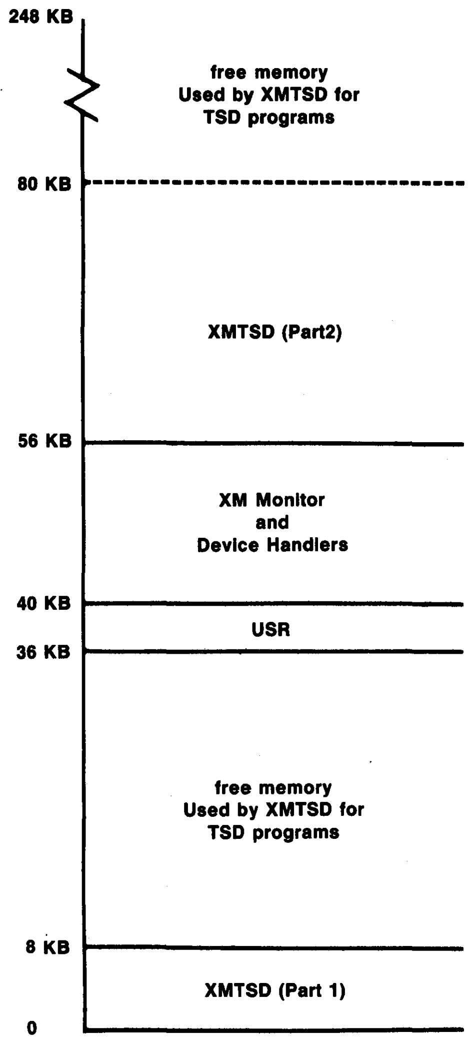


**Figure 5-5 Memory Allocation for XMTSD as a Background Program**

### 5.5.3.3 Applications

When XMTSD is run in the background (as contrasted with running in the foreground), the system can be thought of as running in a production mode. All the system resources, including lower memory and console terminal, are available for the day-to-day needs of application programs under the control of the run-time system.

## 5.6 TERMINATING TIME-SHARED OPERATION (RTEXIT)

### 5.6.1 Running RTEXIT

The RTEXIT program is used to terminate the operation of the time-shared system and the programs operating under its control. The procedure is the same for both TSD and for XMTSD when run in the background. For XMTSD in the foreground, see Section 5.6.3. For TSD and XMTSD (background) the following command is entered while the time-shared system is operating:

```
     * R RTEXIT
```

If no jobs are running, control returns immediately to the RT-11 monitor. If other jobs are running, the response is:

```
     DO YOU WISH AN ABSOLUTE SHUTDOWN?  (Y-N) (N)
```

An answer of Y causes any job currently running to be unconditionally terminated with unpredictable results for that job. If you answer N (or CR), RTEXIT will display the current status of jobs every 20 seconds until such time as all jobs are done. At this time, control returns to the RT-11 monitor. You may cancel the current status display by entering a CTRL/C at any time. This will allow you to run RTEXIT again and to select the absolute shutdown.

### 5.6.2 Chaining to RTEXIT

RTEXIT may be automated by having the preceding program chain to it with the SEND statement. This operation is essentially the same as the situation where RTEXIT is run manually; it has the same restrictions as XMTSD when it is run in the foreground. The sequence for a program running under a time-shared operating system program (except XMTSD in the foreground) could be:

```
     .
     .
     .
     SEND ('MSG','RTEXIT.TSD')
     .
     .
     .
     STOP 'RTEXIT.TSD'
```

The message 'MSG' is optional. When used with RTEXIT, it contains a file name and, separated by a space, an optional alpha character. The file name is that of a file to be executed once the time-shared program is terminated and could be a Save image program (.SAV) or an indirect file. The alpha character is the response to the absolute shutdown question. An absolute shutdown occurs if the character is absent. If the program preceding the running of RTEXIT is detached, the shutdown is always absolute regardless of the presence of the N alpha character in the message.

### 5.6.3 RTEXIT for XMTSD in the Foreground

When XMTSD is operating in the foreground, the selection of RTEXIT is somewhat more involved, because of possible interaction between foreground and background at the time RTEXIT is run. See below for an example of the steps that may be required.

To start time-sharing XMTSD in the foreground:

```
    .FRUN XMTSD.SAV
```

at the background terminal:

```
    .R LISTNR
```

To terminate the listener which is running in the background, enter:

```
    CTRL/C
```

to terminate XMTSD which is running in the foreground, enter:

```
    *R RTEXIT
    DO YOU WISH AN ABSOLUTE SHUTDOWN?  (Y-N) (N) <CR>
```

When all the jobs are complete and XMTSD terminates, at the console terminal, enter:

```
    UN F
```

This unloads the foreground job which is XMTSD.

## 5.7 UTILIZING RESOURCES ON A SMALL SYSTEM

The relatively limited capacity and access speed of a diskette, especially when used as the system device, make careful planning necessary if you want to make the best use of your system. The following guidelines are presented to make you aware of what may be done:

- Include only the essential programs, data files, utilities, and options.
- Use the SEGMENT:1 option when initializing diskettes prior to building your system. The result is a two-block directory on each diskette. This gives you a directory with a 72-file capacity and saves six blocks (two blocks per segment) of diskette space.

- If you have an application program that uses overlays, consider relinking your overlayed routines, where possible, to make them a part of the root section of your program. This reduces disk access time. Programs that use overlays heavily are slower on a diskette system than when run in a system with faster disks. The elimination of overlays will, therefore, increase performance.
- If you set USR to NOSWAP, you reduce the time to open and close files by 90 percent. This is recommended as a standard procedure if you plan to be manipulating files extensively and if you have the space in memory (the USR requires 4 KB).
- Depending on the size of your application and the functions you wish to support, you might segregate programs to be used together and place them on a single diskette. That is, not all the capabilities need be placed on the system disk but could be grouped by logical use onto individual diskettes. If you swap disks in this way to perform various functions, you also have to set USR to SWAP so the directory lookups for program name will work correctly.

## 5.8 ERROR MESSAGES

The operating system error messages are listed in Appendix A. Special error messages generated by the time-shared system program (TSD or XMTSD) are listed in Appendix B, Table B-3. The error messages for the foreground / background communication commands are listed in Appendix B, Table B-4.

# CHAPTER 6

# SYSTEM DEVELOPMENT

## 6.1 INTRODUCTION

A CTS-300 System is comprised of two major software packages: RT-11 and CTS-300. Both of these are furnished in a form that has the potential for many capabilities but which is probably not immediately or directly usable by any CTS-300 user. Each software package should be tailored to match the hardware and to provide the specific software capabilities needed for a given user. In this manual, this process is called system development. Both RT-11 and CTS-300 have their own programs and procedures to facilitate developing an optimal system with minimal memory requirement.

### 6.1.1 System Generation Programs

RT-11 software is adapted to the hardware and user requirements using the RT-11 SYSGEN program documented in the **RT-11 System Generation User's Guide**. The RT-11 SYSGEN program is modified slightly for CTS-300 users, and these modifications are explained in this chapter. CTS-300 software (both single-user and multi-user) is adapted to the user's requirements with a program called CTSGEN. CTSGEN is described, in detail, in this chapter.

### 6.1.2 Chapter Organization

The remainder of this chapter is comprised of three major sections. The first, Section 6.2, CTS-300/RT-11 SYSGEN, describes RT-11 SYSGEN as it applies to CTS-300. The second, Section 6.3, CTSGEN, provides detailed information on the CTSGEN program. This includes the actual CTSGEN dialog and information to help you make choices. The third, and last, section, Section 6.4, Error Messages, is a discussion of errors that may be encountered during a CTSGEN.

## 6.2 CTS-300/RT-11 SYSGEN

The first step in developing a working system is to build an appropriate RT-11 Operating System. You must run SYSGEN to select the RT-11 monitor services and device handlers you require. Since the RT-11 dialog is long and repetitive, an optional, shorter list is provided for CTS-300 users. This shorter version is known as CTS-300/RT-11 SYSGEN.

The first question asked in the CTS-300/RT-11 SYSGEN allows you to choose either the CTS-300/RT-11 version or the full RT-11 SYSGEN as described in the RT-11 documentation. The CTS-300/RT-11 version is different in that it rephrases some questions and provides a programmed selection of certain routinely chosen answers to selected questions. Users for whom the programmed answers are valid (most CTS-300 users) can save time and prevent errors by selecting CTS-300/RT-11 SYSGEN.

The questions that have been rephrased are:

| Question | Change |
|---|---|
| Q5 | Asks: Do you want the extended memory (XM) monitor? Default answer has been changed to a Y. |
| Q12 | Was: Do you want multiterminal support?<br>Now: Are you going to use TSD with this monitor? |
| RT-11's Q89 to Q93: | Line printer support questions have been removed and replaced in CTS-300/RT-11 by Q152 through Q164. The TSD multiterminal support handles a maximum of four line printers in any combination of parallel and/or serial. |
| Q152 Asks: | How many line printers in all? |
| Q153 Asks: | Is the first line printer (LP) parallel? (the default is a Y; answer with an N if serial) |
| Q154 and Q155: | ask for CSR and vector addresses. |
| Q156 through Q158 Ask: | as in Q153, 154, and 155, the same questions for the second line printer (LQ). |
| Q159 through Q161 Ask: | as in Q153, 154, and 155, the same questions for the third line printer (LR). |
| Q162 through Q164 Ask: | as in Q153, 154, and 155, the same questions for the fourth line printer (LS). |
| RT-11's Q152 through Q155 | have been renumbered in CTS-300/RT-11 to Q165, Q166, Q167, and Q168. |

The answers automatically provided with CTS-300/RT-11 SYSGEN are:

| | Question | Answer |
|---|---|---|
| Q2 | Baseline single-job monitor | No |
| Q6 | SJ timer support | Yes |
| Q8 | Error message to replace system halt upon receipt of a system I/O error for the single-job monitor | Yes |
| Q10 | .SPCPS request | No |
| Q11 | Idle loop light pattern | No |
| Q13 | Asynchronous terminal status | Yes * |
| Q23 | KW11-P clock as the system clock | No |
| Q25 | Floating point support | No |
| Q51 | TC11 DECtape support | No |
| Q52 | RF-11 fixed head disk support | No |
| Q54 | RJS03 or RJS04 disk support | No |
| Q88 | TA-11 cassette support | No |
| Q96 | PC-11 high speed paper tape reader/punch | No |
| Q97 | PR-11 high speed paper tape reader | No |
| Q100 | VT11 or VS60 graphics support | No |
| Q168 | Retention of system OBJ's | No |

* Answered as a YES only if you have answered YES to Q12 for TSD; otherwise, it is not preanswered.

If you wish to change any of the automatic selections, answer NO to the first question in CTS-300/RT-11 SYSGEN. By responding NO, you will have entered the full RT-11 SYSGEN, and you will then be asked the first question in the RT-11 SYSGEN. If you choose RT-11 SYSGEN, and you plan to run a time-shared DIBOL program, you must choose multiterminal support (Q12) and answer YES to asynchronous terminal status (Q13). Whether you choose CTS-300/RT-11 SYSGEN or RT-11 SYSGEN, you should refer to the **RT-11 System Generation Manual** before you start.

When you have completed your CTS-300/RT-11 SYSGEN (or RT-11 SYSGEN), you will have to run CTSGEN.

## 6.3 CTSGEN

The CTSGEN program is the CTS-300 Operating System Program generator. It is an interactive program that is used to select the parameters associated with a CTS-300 system. It can create a run-time code interpreter for a Single-User DIBOL (SUD) program, or it can create both a code interpretor and run-time system executive for a time-shared (TSD or XMTSD). As with the RT-11 SYSGEN, it is wise to plan ahead exactly what your requirements are before building the CTS-300 system. It may help if you read Chapter 5, CTS-300 Operating Systems, before initiating any CTSGEN activity.

### 6.3.1 Characteristics

#### 6.3.1.1 Choices

CTSGEN allows you to build either a Single-User DIBOL RTS program or a Time-Shared DIBOL RTS program.

**Single-User System**

If you are building a SUD system, the choices are limited to the selection of two system services. You may select:

support for ISAM files
support for DDT

**Time-Shared System**

If you are building a time-shared RTS, there are two basic types of support services you may select:

Run-time system services you may select are:

number of programs
number of messages stored
number of channels opened per program
support for ISAM files
support for the XM monitor (and residency of USR)
support for DDT
support for implicit or forced-job startup
auto job startup upon completion of the time-shared RTS load

Peripheral device support you may select:

standard (or non-standard) terminals
local or remote terminal use
DL11 or DZ11 terminal interface
mechanical operating characteristics of each terminal
total number of different peripheral devices

### 6.3.1.2 Preliminary Requirements

As you did with SYSGEN, you must plan ahead for your CTSGEN session. Acquaint yourself with the flow of CTSGEN by using the chart, Figure 6-1, in Section 6.3.2 and by reading the dialog and responses in the following sections for the system you intend to build. It is important to remember your SYSGEN responses dealing with terminals, because they affect your answers in CTSGEN.

#### Single-User System

The following modules are necessary for building a single-user RTS. They must be on the system disk.

|  |  |  |
|---|---|---|
| CTSGEN.SAV | ELONG.OBJ | ISAM.OBJ |
| LINK.SAV | DDT.OBJ | ISAMX.OBJ |
| MACRO.SAV | DDTX.OBJ | MATH.OBJ |
| SUD.RTS | SDIRT.OBJ | IO.OBJ |
|  |  | JOB.OBJ |

#### Time-Shared System

Use the lists in Section 6.3.1.1 and note the peripheral devices on your system and the run-time system services you require for your application.

The following modules are necessary for building a time-shared RTS (TSD and XMTSD). They must be on the system disk.

|  |  |  |
|---|---|---|
| CTSGEN.SAV | DMESS.OBJ | KDMESS.OBJ |
| LINK.SAV | DJOB.OBJ | KDJOB.OBJ |
| MACRO.SAV | DISAM.OBJ | KISAM.OBJ |
| SUD.RTS | DISAMX.OBJ | KISAMX.OBJ |
| SYSMAC.SML | DESHRT.OBJ | KESHRT.OBJ |
| QC.MAC | DELONG.OBJ | KELONG.OBJ |
| ST.MAC | FRUNIT.OBJ | KFRUN.OBJ |
| TD.MAC | FRUNXX.OBJ | KFRUNX.OBJ |
| TSDTBL.MAC | DDDT.OBJ | KDDT.OBJ |
| DEFS.MAC | DDDTX.OBJ | KDDTX.OBJ |
| TSDDFN.MAC | DMATH.OBJ | KMATH.OBJ |
|  | DDIRT.OBJ | KDIRT.OBJ |
|  | DTO.OBJ | KDTO.OBJ |
|  | DIO.OBJ | KDIO.OBJ |
|  | DTOINI.OBJ | KTOINI.OBJ |
|  |  | KCORE.OBJ |

#### 6.3.1.3 Question Types

Each CTSGEN question has a default answer. That default is noted immediately after the question and is placed within parentheses. After each question is displayed, you respond by typing the desired entry or by typing a carriage return (CR) to select the default.

There are two kinds of questions in CTSGEN:

- Questions that require a Y or an N response.
- Questions that require a numeric response.

Unlike SYSGEN, there is no need to specify numbers in octal. Questions that require a numeric response accept only decimal numbers. The default value is contained in parentheses immediately following the question.

If you need to change your response to a previous question, you may return to that question by entering Q followed by the question number. Take special care when responding to questions that require other than Y, N, or a default value.

### 6.3.2 CTSGEN Dialog

This section describes the dialog in detail. All text, questions, and responses displayed during the dialog are described. However, during any run of CTSGEN some of these questions and comments will not appear, since their occurrence may depend on the answers to previous questions.

Like SYSGEN, CTSGEN dialog has two forms. There is a short form with the question and the default answer only and a longer form which presents the question and explanatory comment. The short form is the default; however, when you enter a question mark (or an invalid response) in response to a CTSGEN question, the program repeats the question (and default) along with explanatory comments on the question displayed. Those comments can guide you through a CTSGEN session, and if you are not experienced with CTSGEN, this longer session will help you to be sure of the implications of your answers and to avoid inaccurate responses. If you are familiar with CTSGEN, use the short form.

Figure 6-1 illustrates the flow of questions in CTSGEN.

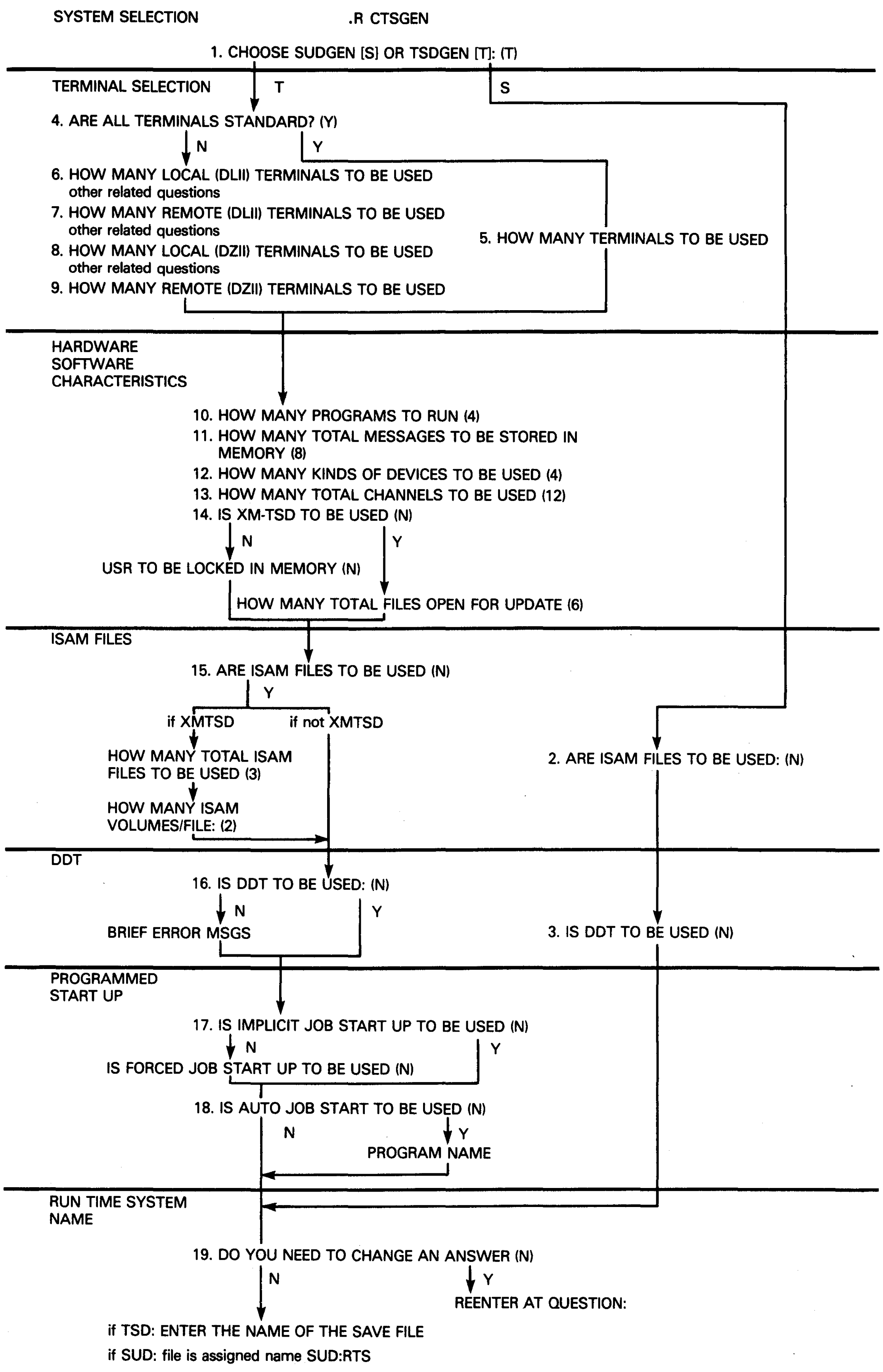

SYSTEM SELECTION
.R CTSGEN
1. CHOOSE SUDGEN [S] OR TSDGEN [T]: (T)
TERMINAL SELECTION
T
S
4. ARE ALL TERMINALS STANDARD? (Y)
N
Y
6. HOW MANY LOCAL (DLII) TERMINALS TO BE USED
other related questions
7. HOW MANY REMOTE (DLII) TERMINALS TO BE USED
other related questions
8. HOW MANY LOCAL (DZII) TERMINALS TO BE USED
other related questions
9. HOW MANY REMOTE (DZII) TERMINALS TO BE USED
5. HOW MANY TERMINALS TO BE USED
HARDWARE
SOFTWARE
CHARACTERISTICS
10. HOW MANY PROGRAMS TO RUN (4)
11. HOW MANY TOTAL MESSAGES TO BE STORED IN MEMORY (8)
12. HOW MANY KINDS OF DEVICES TO BE USED (4)
13. HOW MANY TOTAL CHANNELS TO BE USED (12)
14. IS XM-TSD TO BE USED (N)
N
Y
USR TO BE LOCKED IN MEMORY (N)
HOW MANY TOTAL FILES OPEN FOR UPDATE (6)
ISAM FILES
15. ARE ISAM FILES TO BE USED (N)
Y
if XMTSD
if not XMTSD
HOW MANY TOTAL ISAM FILES TO BE USED (3)
HOW MANY ISAM VOLUMES/FILE: (2)
2. ARE ISAM FILES TO BE USED: (N)
DDT
16. IS DDT TO BE USED: (N)
N
Y
BRIEF ERROR MSGS
3. IS DDT TO BE USED (N)
PROGRAMMED
START UP
17. IS IMPLICIT JOB START UP TO BE USED (N)
N
Y
IS FORCED JOB START UP TO BE USED (N)
18. IS AUTO JOB START TO BE USED (N)
N
Y
PROGRAM NAME
RUN TIME SYSTEM
NAME
19. DO YOU NEED TO CHANGE AN ANSWER (N)
N
Y
REENTER AT QUESTION:
if TSD: ENTER THE NAME OF THE SAVE FILE
if SUD: file is assigned name SUD:RTS

**Figure 6-1 CTS-300 Operating System Generator (CTSGEN)**

The following is the actual CTSGEN dialog and explanatory text. It is broken into sections in this manual to identify more clearly the activity taking place. The dialog is shown in upper case only. The explanatory material is shown in both uppercase and lowercase.

CTSGEN is executed with the following command; it responds as shown:

```
     .R CTSGEN

     CTSGEN Vnn-nn

     EACH OF THE FOLLOWING QUESTIONS IS FOLLOWED BY A DEFAULT
     RESPONSE IN PARENTHESES. THIS RESPONSE WILL BE USED
     IF A <CR> IS TYPED IN ANSWER TO THE QUESTION.   IF A
     QUESTION MARK OR ANY ILLEGAL RESPONSE IS TYPED, FURTHER
     INFORMATION CONCERNING THE CURRENT QUESTION WILL BE PRINTED
     AT THE TERMINAL. IN GENERAL YOU MAY RETURN TO ANY OF THE
     QUESTIONS THAT ARE MARKED WITH A LINE NUMBER. SIMPLY TYPE
     THE LETTER Q FOLLOWED BY THE LINE NUMBER. (I.E. Q1, Q3, Q10)
```

#### 6.3.2.1 Single-User or Time-Shared System

The first question in CTSGEN:

```
     1.  CHOOSE - SUDGEN [S] OR TSDGEN [T]:  (T)
```

Respond with S if you are going to build a single-user system or with T (or CR) if you are going to build a time-shared system. If you answer T, Q4 will appear. See Section 6.3.2.3.

#### 6.3.2.2 Single-User System

There are only two questions asked for a single-user system:

```
     2.  ARE ISAM FILES TO BE USED:  (N)
```

Choosing ISAM costs approximately 4.2 K bytes.

Respond with N (or CR) if you do not plan to use ISAM files. Respond with Y if you do plan to use ISAM files. The next question:

```
     3.  IS DDT TO BE USED:  (N)
```

Choosing DDT costs approximately 1100 bytes.

Respond with N (or CR) if you do not plan to use the DIBOL debugging program, DDT. Respond with Y if you do plan to debug.

With your answer to this last question, CTSGEN proceeds to automatically build a run-time code interpreter for Single-User DIBOL (SUD) programs. The name of the interpreter is assigned by CTSGEN to be SUD.RTS. This file must always be on the system disk for your SUD system. In addition, a link map (SUD.MAP) is generated which may help identify problems with the interpreter.

It is advised that you rename CTSGEN.COM to SUDGEN.COM using the command string:

```
.RENAME CTSGEN.COM SUDGEN.COM
```

You can not run any Single-User DIBOL program, not even CTSGEN, without SUD.RTS. The SUDGEN.COM that you just made is a backup you can use to recreate the same SUD.RTS selected by questions 2 and 3 above. All you have to do is type @SUDGEN. This can be very helpful if you should accidently destroy your SUD.RTS. Every time you run a CTSGEN, a new CTSGEN.COM is generated. For this reason, you must rename the CTSGEN.COM to SUDGEN.COM after this CTSGEN and before another.

It is permissible, and often useful, to create several versions of SUD.RTS and store them for later use. There are four possibilities by choosing to support (or not support) ISAM and/or DDT. The distribution version is without ISAM or DDT support. However, other versions must be stored with names other than SUD.RTS, since SUD programs automatically look for SUD.RTS when the RUN command is issued. Whenever a particular version is needed, you simply rename it to SUD.RTS (after storing its predecessor under its storage name) and run your SUD programs.

### 6.3.2.3 Time-Shared System

The first question related to a time-shared system:

```
4. ARE ALL THE TERMINALS STANDARD: (Y)
```

"Standard" is a term used here solely in regard to the CTSGEN dialog. A standard terminal is defined as being either a VT100, VT52, or a VT50H, each of which has the following attributes:

- Local DL11 interface only
- Master terminal

  You cannot initiate a time-shared program at a slave terminal.
- Line width of 80 characters
- No CTRL/C trap support

  CTRL/C trap prevents a CTRL/C from terminating a time-shared program. Such abnormal termination could result in loss or corruption of data files.

If your answer is N, Q6 will be displayed. Q6 is discussed in Section 6.3.2.5; but, first, see Section 6.3.2.4.

If your answer is Y or <CR>:

5. HOW MANY TERMINALS TO BE USED: (2)

You may select up to the number chosen in the RT-11 SYSGEN. A limit of 8 is recommended for systems in which terminal input will be heavy; CTSGEN will permit up to 12. A valid response prompts Q10 to be displayed. See Section 6.3.2.9 to continue selection of other system parameters.

### 6.3.2.4 Terminal Specification

Since your response to question 4 was N (your terminals are not all standard), more information is needed. Terminal information to be specified in the following sections of this chapter (CTSGEN questions 6 through 9) depends on the following factors.

Your answers in SYSGEN set the limits for your answers in CTSGEN. SYSGEN used your answers to assign hardware interfaces to support the terminals. CTSGEN asks you which of these terminals you want your time-shared program to recognize.

CTSGEN terminal questions are presented in the same four categories as in SYSGEN: DZ11 local, DL11 remote, DZ11 local, and DZ11 remote. In each category you are asked how many terminals are to be used and how many are to be unused. The total (used and unused) must equal the SYSGEN assignment for that category.

Additional information pertaining to these terminals is also asked in each of the four categories.

### 6.3.2.5 Local DL11 Terminals

This series of questions pertains to local terminals on the DL11 interface.

6. HOW MANY LOCAL (DL11) TERMINALS TO BE USED: (4)

Respond with the number of terminals that are connected to DL11 hardware and that are to be used locally. This question was asked in SYSGEN; your response may be the same or less than your response in SYSGEN. You must specify at least one for your console terminal.

HOW MANY LOCAL (DL11) TERMINALS LEFT UNUSED: (0)

Respond with the number of terminals that are not to be used. Your response to the preceding question and the number that you specified in SYSGEN for local use are important here. Your answer must be the difference between those two responses.

You have specified how many local DL11 terminals are to be recognized by the time-shared RTS. Now you must enter the operating characteristics of each one. A series of five questions follows for each terminal noted in the first part of Q6.

```
            TERMINAL n
     S TERMINAL n A SCOPE:  (Y)
```

Respond Y or <CR> if this terminal has a display (video) screen. Respond N if this terminal is a teleprinter (prints out hard copy). From within a DIBOL program, terminal 1 would be referred to as terminal 0.

```
     IS TERMINAL n A SLAVE:  (N)
```

Respond N or <CR> if this terminal is to be a master terminal. Respond Y if you do not want this terminal to be able to start jobs with keyboard commands. A Y response means that jobs at this terminal must be started by forced-job startup. Remember, if this is terminal 1 (the console terminal), it must be a master terminal.

```
     IS CTRL/C TRAP TO BE USED:  (N)
```

Respond N or <CR> if you want a CTRL/C to be recognized by the time-shared RTS. Respond Y if you want a CTRL/C to be ignored by the time-shared RTS. This CTRL/C is not passed to the DIBOL program.

```
     HOW MANY FILL CHARACTERS TO BE USED:  (0)
```

Respond with a decimal number according to your terminal model as noted in the following chart. The chart notes the number of fill characters required for each terminal model set for the baud rate shown. This number provides a delay time to allow completion of the carriage return/line feed.

| | | |
|---|---|---|
| VT100 | (set for 9600 baud) | 0 |
| VT50H | (set for 9600 baud) | 0 |
| VT52 | (set for 9600 baud) | 0 |
| LA36 | (set for 300 baud) | 0 |
| LA30 | (set for 300 baud) | 10 |
| LA30 | (set for 150 baud) | 4 |
| LA30 | (set for 110 baud) | 2 |
| VT05 | (set for 2400 baud) | 2 |
| VT05 | (set for 1200 baud) | 2 |
| VT05 | (set for 600 baud) | 1 |

The next question:

```
     WHAT IS THE LINE WIDTH TO BE USED:  (80)
```

Respond with a decimal number in the range of 1 through 132 according to your terminal model. This number is usually 80.

These preceding five questions appear for each local DL11 terminal you specified in question Q6 until operating characteristics are detailed for all local DL11 terminals to be supported at run time. Now you must consider DL11 terminals for remote use.

### 6.3.2.6 Remote DL11 Terminals

This series of questions pertains to remote terminals on the DL11 interface.

```
     7.  HOW MANY REMOTE (DL11) TERMINALS TO BE USED:  (0)
```

Respond with the number of terminals to be used that are connected to DL11 hardware and that will be used in a remote mode. This question was asked in SYSGEN; your response may be the same or less than your response in SYSGEN.

```
     HOW MANY REMOTE (DL11) TERMINALS LEFT UNUSED:  (0)
```

Respond with the number of terminals that will not be used. Your response to the preceding question and the number that you specified in SYSGEN for remote use are important here. Your answer must be the difference between those two responses.

You have specified how many remote DL11 terminals are to be recognized by the time-shared RTS. Now you must enter the operating characteristics of each one. A series of five questions follows for each terminal noted in the first part of Q7.

```
             TERMINAL n
     IS TERMINAL n A SCOPE:  (Y)
```

Respond Y or <CR> if this terminal has a display screen (a video screen). Respond N if this terminal is a teleprinter (prints out hard copy).

```
     IS TERMINAL n A SLAVE:  (N)
```

Respond N or <CR> if this terminal is to be a master terminal. Respond Y if you do not want this terminal to be able to start jobs with keyboard commands. A Y response means that jobs at this terminal must be started by forced-job startup.

```
     IS CTRL/C TRAP TO BE USED:  (N)
```

Respond N or <CR> if you want a CTRL/C to be recognized by the time-shared RTS. Respond Y if you want a CTRL/C to be ignored by the time-shared RTS. This CTRL/C is not passed to the DIBOL program.

```
     HOW MANY FILL CHARACTERS TO BE USED:  (0)
```

Respond with a decimal number according to your terminal model. The chart notes the number of fill characters required for each terminal model at the baud rate shown. This number provides a delay time to allow completion of the carriage return/line feed.

| | | |
|---|---|---|
| VT100 | (set for 9600 baud) | 0 |
| VT50H | (set for 9600 baud) | 0 |
| VT52 | (set for 9600 baud) | 0 |
| LA36 | (set for 300 baud) | 0 |
| LA30 | (set for 300 baud) | 10 |
| LA30 | (set for 150 baud) | 4 |
| LA30 | (set for 110 baud) | 2 |
| VT05 | (set for 2400 baud) | 2 |
| VT05 | (set for 1200 baud) | 2 |
| VT05 | (set for 600 baud) | 1 |

The next question:

```
     WHAT IS THE LINE WIDTH TO BE USED:  (80)
```

Respond with a decimal number in the range of 1 through 132, according to your terminal model. This number is usually 80 and may be thought of as setting the margins.

The preceding five questions appear for each remote DL11 terminal you specified in Q7 until operating characteristics are detailed for all remote DL11 terminals to be supported at run time.

Now you must consider terminals that interface with DZ11 hardware.

### 6.3.2.7 Local DZ11 Terminals

This series of questions pertains to local terminals on the DZ11 interface.

```
     8.  HOW MANY LOCAL (DZ11) TERMINALS TO BE USED:  (0)
```

Respond with the number of terminals to be used that are connected to DZ11 hardware and used locally. This question was asked in SYSGEN; your response may be the same or less than your response in SYSGEN.

```
     HOW MANY LOCAL (DZ11) TERMINALS LEFT UNUSED:  (0)
```

Respond with the number of terminals that are not to be used. Your response to the preceding question and the number that you specified in SYSGEN for remote use are important here. Your answer must be the difference between those two responses.

You have specified how many local DZ11 terminals are to be recognized by the time-shared RTS. Now you must enter the operating characteristics of each one. A series of six questions follows for each terminal noted in the first part of Q8.

```
            TERMINAL n
     IS TERMINAL n A SCOPE:  (Y)
```

Respond Y or <CR> if this terminal has a display (video) screen. Respond N if this terminal is a teleprinter (prints out hard copy).

```
     IS TERMINAL n A SLAVE:  (N)
```

Respond N or <CR> if this terminal is to be a master terminal. Respond Y if you do not want this terminal to be able to start jobs with keyboard commands. A Y response means that jobs at this terminal must be started by forced-job startup.

```
     IS CTRL/C TRAP TO BE USED:  (N)
```

Respond N or <CR> if you want a CTRL/C to be recognized by the time-shared RTS. Respond Y if you want a CTRL/C to be ignored by the time-shared RTS. This CTRL/C is not passed to the DIBOL program.

```
     HOW MANY FILL CHARACTERS TO BE USED:  (0)
```

Respond with a decimal number according to your terminal model. The chart notes the number of fill characters required for each terminal model at the baud rate shown. This number provides a delay time to allow completion of the carriage return/line feed.

| | | |
|---|---|---|
| VT100 | (set for 9600 baud) | 0 |
| VT50H | (set for 9600 baud) | 0 |
| VT52 | (set for 9600 baud) | 0 |
| LA36 | (set for 300 baud) | 0 |
| LA30 | (set for 300 baud) | 10 |
| LA30 | (set for 150 baud) | 4 |
| LA30 | (set for 110 baud) | 2 |
| VT05 | (set for 2400 baud) | 2 |
| VT05 | (set for 1200 baud) | 2 |
| VT05 | (set for 600 baud) | 1 |

The next question:

```
     WHAT IS THE LINE WIDTH TO BE USED:  (80)
```

Respond with a decimal number in the range of 1 through 132 according to your terminal model. This number is usually 80 and may be thought of as setting the margins.

```
     WHAT IS THE LOCAL TERMINAL BAUD RATE:  (9600)
```

Respond with a decimal number according to your terminal model type and the speed that is appropriate for it. Baud rates are:

```
      110,   150,    300,    600,   1200,
     1800,  2000,   2400,   3600,   4800,
     7200,  9600.
```

These preceding six questions appear for each local DZ11 terminal you specified in Q8 until operating characteristics are detailed for all local DZ11 terminals to be supported at run time.

Now it is time to consider your DZ11 terminals for remote use.

### 6.3.2.8 Remote DZ11 Terminals

These questions pertain to remote terminals on the DZ11 interface.

```
     9.  HOW MANY REMOTE (DZ11) TERMINALS TO BE USED:  (0)
```

Respond with the number of terminals to be used that are connected to DZ11 hardware and that will be used in a remote mode. This question was asked in SYSGEN; your response may be the same or less than your response in SYSGEN.

You have specified how many remote DZ11 terminals are to be recognized by the time-shared RTS. There is no need to specify remote DZ11 terminals left unused. Now you must enter the operating characteristics of each terminal noted in Q9. A series of six questions follows for each terminal noted in the first part of Q9.

```
            TERMINAL n
     IS TERMINAL n A SCOPE:  (Y)
```

Respond Y or <CR> if this terminal has a display screen (a video screen). Respond N if this terminal is a teleprinter.

```
     IS TERMINAL n A SLAVE:  (N)
```

Respond Y or <CR> if this terminal is to be a master terminal. Respond Y if you do not want this terminal to be able to start jobs with keyboard commands. A Y response means that jobs at this terminal must be started by forced-job startup.

```
     IS CTRL/C TRAP TO BE USED:  (N)
```

Respond N or <CR> if you want a CTRL/C to be recognized by the time-shared RTS. Respond Y if you want a CTRL/C to be ignored by the time-shared RTS. This CTRL/C is not passed to the DIBOL program.

```
     HOW MANY FILL CHARACTERS TO BE USED:  (0)
```

Respond with a decimal number according to your terminal model. The chart notes the number of fill characters required for each terminal model at the baud rate shown. This number provides a delay time to allow completion of the carriage return/line feed.

| | | |
|---|---|---|
| VT100 | (set for 9600 baud) | 0 |
| VT50H | (set for 9600 baud) | 0 |
| VT52 | (set for 9600 baud) | 0 |
| LA36 | (set for 300 baud) | 0 |
| LA30 | (set for 300 baud) | 10 |
| LA30 | (set for 150 baud) | 4 |
| LA30 | (set for 110 baud) | 2 |
| VT05 | (set for 2400 baud) | 2 |
| VT05 | (set for 1200 baud) | 2 |
| VT05 | (set for 600 baud) | 1 |

The next question:

```
     WHAT IS THE LINE WIDTH TO BE USED:  (80)
```

Respond with a decimal number in the range of 1 through 132, according to your terminal model. This number is usually 80 and may be thought of as setting the margins.

WHAT IS THE REMOTE TERMINAL BAUD RATE: (300)

Respond with a decimal number according to your terminal model type and the speed that is appropriate for it. The baud rate must be the same as that chosen in the RT-11 SYSGEN question for DZ11 support. Baud rates are:

```
 110,   150,    300,    600,   1200,
1800,  2000,   2400,   3600,   4800,
7200,  9600.
```

These preceding six questions appear for each remote DZ11 terminal you specified in Q9 until operating characteristics are detailed for all remote DZ11 terminals to be supported at run time.

You have completed the entry of details for all your terminals; Q10 now appears (see the next section).

#### 6.3.2.9 System Hardware/Software Configuration

The first of the system-related questions:

10. HOW MANY PROGRAMS TO BE RUN: (4)

The cost is approximately 60 bytes per program selected.

Respond with the decimal number of programs or jobs that you plan to run at any one time. The range is from 1 through 16.

11. HOW MANY TOTAL MESSAGES TO BE STORED IN MEMORY: (8)

The cost is approximately 14 bytes per message.

Respond with the decimal number of messages that can be stored in memory at any one time. This number depends on your programming needs. Consider which of your programs use SEND or RECEIVE statements and how many messages are generated (LPTSPL.TSD and BGMAN.TSD use this facility). The range is from 1 through 50.

12. HOW MANY KINDS OF DEVICES TO BE USED: (4)

The cost is approximately 10 bytes per device selected.

Respond with the decimal number of devices. This number notes categories of devices with the exception of printers. Each line printer requires a separate device handler. The range is from 1 through 10.

13. HOW MANY TOTAL CHANNELS TO BE USED: (12)

The cost is approximately 55 bytes per channel selected.

Respond with a decimal number in the range of 1 through 50. This refers to I/O channels (referenced by the DIBOL OPEN statement) that are needed for devices or files at any one time across all programs. In addition, your number sets the limit for questions prompted in Q14 (files opened for update) and in Q15 (total ISAM files).

    14. IS EXTENDED MEMORY TSD TO BE USED: (N)

Respond Y if you intend to run XMTSD. An answer of Y automatically implies that the User Service Routine (USR) will be locked in memory so the USR question, below, does not appear. Respond N or <CR> if you do not need DIBOL support for extended memory. If you answer N or <CR> the following question is asked:

    IS USR TO BE LOCKED IN MEMORY: (N)

Choosing USR to be locked in memory costs approximately 4.2 K bytes.

Respond N or <CR> if you want the USR to be swapped out during program execution. You will want the USR to be swapped out if memory space is too small to permit program execution. Respond with Y if you want the USR to remain in memory permanently. With the USR locked in memory, files OPEN and CLOSE much faster.

Any response to the previous question, or an answer of Y to Q14, will prompt:

    HOW MANY TOTAL FILES OPENED FOR UPDATE: (6)

The cost (in bytes) is approximately:

4 X [number of jobs] X [number of files opened for update].

Respond with a decimal number in the range of 0 through 25. This number, which is the total number open at any one time, cannot exceed your response to Q13 (total channels to be used). Your response prompts:

    15. ARE ISAM FILES TO BE USED: (N)

Choosing ISAM costs approximately 4.2 K bytes.

Respond N or <CR> if ISAM files are not to be accessed in this version of the time-shared RTS. Respond Y if ISAM files are to be created or accessed in this version of the time-shared RTS. If you answer Y now, and you have answered Y to Q14 concerning extended-memory time sharing, two additional questions appear:

    HOW MANY TOTAL ISAM FILES TO BE USED: (3)

The cost is a function of both this question and the next; therefore, see the discussion of the next question before answering this.

This is the maximum number of ISAM files to be open at any one time across all jobs. Respond with a decimal number in the range of 1 up to the number you specified in answer to Q13 (but not to exceed 25). A second question appears:

```
     HOW MANY ISAM VOLUMES/FILE:  (2)
```

The cost (in bytes) is approximately:

the number of files X [20 + [10 X the number of volumes per file]].

Respond with a decimal number in the range of 2 through 8. The ISAM file that occupies the largest number of volumes will determine your response. If you have prompted CTSGEN for explanatory remarks with either of the two preceding questions, or if you have exceeded limits set for these questions, the questions will be asked again; otherwise, your response prompts:

```
     16.  IS DDT TO BE USED:  (N)
```

The cost is approximately 600 bytes.

Respond N or <CR> if you will not require the use of DDT in this time-shared RTS. Respond Y if you require the use of DDT in this time-shared RTS. The Y response indicates that you also will get brief error messages that are listed by number only. If you answer N:

```
     DO YOU WANT BRIEF ERROR MESSAGES:  (Y)
```

The cost of selecting expanded error messages is approximately 1300 bytes.

Respond N if you want the run-time error message text to be displayed when an error is detected. Respond Y or <CR> if you want only the number of a run-time error message to be printed when an error is detected.

```
     17.  IS IMPLICIT JOB STARTUP TO BE USED:  (N)
```

Implicit job startup costs approximately 100 bytes.

Respond Y if you want the capability of implicit job startup. This will allow a job to start in response to a SEND statement. Selecting implicit job startup automatically selects forced job startup, below; and that question is therefore not asked. Implicit job startup must be selected if you intend to run XMTSD as a foreground program. If you do not want the capability of implicit job startup, respond N or <CR>.

If you answer N to the preceding question:

```
     IS FORCED JOB STARTUP TO BE USED:  (N)
```

Forced job startup costs approximately 280 bytes.

Respond Y if you want the capability for forced job startup. This enables you to start a job on another terminal with an XCALL to RUNJB. Respond with N or <CR> if you do not want to use this capability. You must respond with Y if the line printer spooler LPTSPL.TSD is to be used.

```
     18.  IS AUTO JOB STARTUP TO BE USED:  (N)
```

Auto job startup costs approximately 30 bytes.

Respond N or <CR> if you do not want to specify a DIBOL program to be started after the time-shared RTS is loaded. Respond with Y if you want to automatically start a DIBOL program upon completion of the time-shared RTS load. If you answer Y, you are prompted for the name of the program to be automatically started:

```
     ENTER PROGRAM NAME [DEV:FILNAM.EXT]:
```

At this point you have selected all the system parameters, and you are asked whether you wish to change any of your answers.

```
     19.  DO YOU NEED TO CHANGE AN ANSWER:  (N)
```

If you are satisfied that your answers reflect your hardware and software needs, respond with N or (CR); Q19 will appear. See the next section.

If you wish to change an answer, respond with a Y:

```
     REENTER AT QUESTION:
```

Respond with Q followed by the question number. Questioning will begin again at the question you select.

#### 6.3.2.10 Naming the Time-Shared Program

The final question:

```
     ENTER THE NAME OF THE SAVE FILE:
```

The file name entered represents your customized version of a time-shared RTS. The default extension is .SAV.

Your final carriage return begins a process of assembling and linking files that will produce your customized time-shared RTS. Your answers create the file TSDPAR.MAC and the indirect file CTSGEN.COM which contains the commands for assembling and linking the time-shared RTS. After the final carriage return is entered, CTSGEN chains to CTSGEN.COM to complete the creation of your customized run-time system. The system is identified by the file name you specified in the last question of the CTSGEN dialogue. In addition, the linker creates a link map with this same file name and a default extension of .MAP. This link map may help identify problems with a time-shared RTS.

## 6.4 ERROR MESSAGES

CTSGEN checks your responses to verify that they are within the range of permissible answers. If an error is detected, you are advised that your answer exceeds the possibilities and you are prompted with the question again. CTSGEN does not detect logical errors (such as specifying more terminals than exist). However, in the assembling process after CTSGEN execution, MACRO assembly errors can occur. If a message indicating that a file was not found appears during the assembling or linking process, one of the files specified in Section 6.3.1.2 is not present on the system device.

If you continue to receive errors, consult your DIGITAL software specialist.

# INTRODUCTION TO SECTION III

Section III contains the system utilities provided with CTS-300. These utilities assist you in maintaining your system, in monitoring its operation, and in providing a convenient way to present data. Each utility is discussed in its own chapter. There is no particular order of presentation. Some of the utilities are usable in all systems and some are applicable to TSD systems only.

The utilities and their major functions are:

**DDT** (Chapter 7)

A run-time debugging program for DIBOL programs.

**Spoolers** (Chapter 8)

There are two line printer spoolers. One for SUD systems and one for TSD systems. These programs implement the DIBOL LPQUE statement to print data.

**PRINTU** (Chapter 9)

A program that allows you to quickly develop and organize simple reports from a known data base structure.

**ISMUTL** (Chapter 10)

A program that permits you to develop an ISAM file that is tailored to your specific needs and to monitor this file and reorganize it as it grows.

**SORT/MERGE** (Chapter 11)

A program that allows you to sort or merge files using up to eight keys.

**STATUS** (Chapter 12)

A TSD utility that shows the current system operation parameters such as numbers and names of jobs and terminals.

**REDUCE** (Chapter 13)

A TSD utility used to eliminate unused blocks of a DIBOL program file resulting from the TSD linking process.

# CHAPTER 7

# DIBOL DEBUGGING UTILITY (DDT)

## 7.1 INTRODUCTION

DDT (DIBOL Debugging Technique) is a system utility that allows you to interact with your DIBOL program while it is executing.

### 7.1.1 Features

The features of DDT are intended to aid the DIBOL programmer in locating problems; correcting data values; and testing programming errors directly without having to edit, compile, and link again. Specifically, you may:

- Set predetermined stopping points.
- Examine and/or alter the contents of variables.
- Single step through lines of a DIBOL program.
- Trace through sequences of XCALL nestings.

### 7.1.2 Chapter Organization

The remainder of this chapter is arranged in two major sections. Section 7.2, Preparing for DDT, discusses the procedures required to prepare your system and program for DDT operation. Section 7.3, DDT Commands, describes the various DDT commands and their use.

## 7.2 PREPARING FOR DDT

This section details the procedures required to compile, link, and run with DDT.

### 7.2.1 CTSGEN

You must first request DDT during the CTSGEN in addition to compiling and linking individual routines for use with DDT.

### 7.2.2 Compiling

The main program, as well as all subroutines which are to be debugged, must first be compiled for use with DDT by specifying the DDT option in the DIBOL compiler command. This option generates a symbol table used by DDT. A typical command line might be:

```
.DIBOL/ONDEBUG PROG,SUB1,SUB2,SUB3
```

which generates PROG.OBJ, SUB1.OBJ, SUB2.OBJ, and SUB3.OBJ and the necessary symbol tables for DDT.

If certain subroutines are known to be already debugged, you may compile your program specifying only those modules you intend to further debug. The following command line illustrates this:

```
.DIBOL PROG/ONDEBUG,SUB1/ONDEBUG,SUB2,SUB3
```

With the above compiler command structure, only the main program and SUB1 are compiled for DDT.

### 7.2.3 Linking

The compiled main program and all subroutines must be linked with a special DDT module in order for it to be available at run time. Notice that both the SUD and TSD are linked to the same DDT module (TSDDT). The basic command for the main program and subroutines used in compiling would be:

For SUD operation:

```
.LINK PROG,SUB1,SUB2,SUB3,TSDDT,DIBOL
```

which would create PROG.SAV.

For TSD operation:

```
.LINK/EXE:PROG.TSD PROG,SUB1,SUB2,SUB3,TSDDT,TDIBOL/BOT:100000
```

which would create PROG.TSD.

### 7.2.4 DDT Operation

#### 7.2.4.1 Running DDT

DDT is initialized whenever a program compiled and linked for DDT operation is run under a run-time system which includes DDT. DDT outputs its version number and, on the next line, the hyphen prompt. In the following program, PROG which has been compiled and linked for DDT, is executed for manipulation with DDT commands:

```
.R PROG or .RU dev:PROG
DIBOL DDT VAnn-nn
-
```

where the A in the version identifier indicates a SUD system. It would be B for a TSD system and C for an XMTSD system.

At this point any valid command discussed in Section 7.3 can be entered.

#### 7.2.4.2 Failure to Properly Prepare for DDT

If you forget to perform one of the required steps in sections 7.2.1 through 7.2.3, the program will exhibit the following characteristics:

- If no DDT were requested at CTSGEN time:

  The program will run as though no DDT were requested.

- If no DDT were requested during compilation:

  The program will respond to DDT except for those commands that examine and/or alter the contents of variables.

- If no DDT were requested during linking:

  The program will run as though no DDT were requested.

#### 7.2.4.3 Error Messages

See Appendix B, Table B-5, for a list of DDT error messages.

## 7.3 DDT COMMANDS

This section discusses the commands that are valid for DDT. In the following text, when the term routine is used it refers to a specific program module; either the main program or an external DIBOL subroutine.

For future reference, the DDT commands and command formats are listed below in the order they appear in this section.

| **Command** | **Format** |
|---|---|
| Start or resume execution | CTRL/Z |
| Single step | CTRL/G |
| Setting breakpoints | $[name:]nnn |
| Clearing breakpoints | $[name] |
| Iteration of breakpoints | >n |
| Examining variables | vvv= |
| Setting variables | vvv=nnn |
| Extended variable manipulation | ++vvv= or ++vvv=nnn |
| Subroutine traceback | ^ |

DDT commands, like other commands, require a carriage return termination to enter the command.

### 7.3.1 Program Execution Control

Program execution control has two functions: It allows you to resume execution after a breakpoint has been encountered; and it allows you to single step through individual DIBOL statements to see if they are being properly executed.

#### 7.3.1.1 Program Execution

To start or resume execution of the DIBOL routine from a DDT breakpoint, enter the following command in response to the DDT prompt:

```
CTRL/Z
```

There are no arguments. The current routine simply starts or resumes execution.

Example:

```
-CTRL/Z
```

resumes execution of the current routine.

#### 7.3.1.2 Single Step

It is frequently desirable to know which branch of a computed GOTO, or of a complicated IF statement the program will take. The single step command executes the next instruction in the routine and halts. To single step, enter the following command in response to the DDT prompt:

```
     -CTRL/G
```

There are no arguments. The routine executes the present instruction and returns the following message:

```
     AT LINE xxxx IN ROUTINE yyyy
```

where:

| | |
|---|---|
| xxxx | is the line number of the instruction after execution of the single step. |
| yyyy | is the name of the routine in which line xxxx resides. |

You may now enter any DDT command you wish, as indicated by the DDT prompt.

Example:

Assume there is a conditional jump statement in routine SUB3 at line 47, and you wish to find the next instruction to be executed. First, while in subroutine SUB3, set a breakpoint (see Section 7.3.2.1) at line 47:

```
     $47
```

Start execution of the routine by issuing a CTRL/Z, and when (and if) line 47 is reached the display will be:

```
     DDT BREAK AT LINE 47 IN ROUTINE SUB3
     -
```

Respond to the prompt with CTRL/G which will, in turn, display:

```
     AT LINE _NN IN ROUTINE ABC
     -
```

which is the next instruction location. Routine ABC could be any subroutine including SUB3.

### 7.3.2 Breakpoint Control

A breakpoint is a user-determined stopping point within a routine. Breakpoints are used to position yourself in a routine so you can exercise other DDT capabilities.

#### 7.3.2.1 Setting Breakpoints

You set a breakpoint in a routine by typing the following command in response to the DDT prompt:

```
$[name:]nnn
```

where:

| | |
|---|---|
| name | is the name of the routine in which the breakpoint is to be set. If a breakpoint is to be set in the main program, the name of the first routine specified in the link command (by convention, the root segment) should be used. Otherwise, the name of the routine should match the name given in the subroutine statement. If the name argument is omitted, the current routine is assumed. |
| nnn | is the line number at which the routine is to halt. |

- The line at which the routine is halted has not yet been executed.
- A maximum of eight breakpoints may be set at any one time.
- Only one breakpoint is allowed in any main program or subroutine.
- A breakpoint in the data section has no meaning.

Example:

```
-$SUB1:50
```

sets a breakpoint at line 50 in subroutine SUB1.

```
-$21
```

sets a breakpoint at line 21 in the current routine.

### 7.3.2.2 Clearing Breakpoints

Previously set breakpoints may be cleared by typing the following command in response to the DDT prompt:

```
$[name]
```

where:

name — is the name of the routine in which the breakpoint is to be deleted. If name is omitted, the breakpoint in the current routine is cleared.

- The breakpoint in a routine need not be deleted before a breakpoint is set at a new line in that routine.
- Setting a new breakpoint automatically deletes any other breakpoint in that routine.

Example:

```
-$SUB2
```

clears the breakpoint in subroutine SUB2.

```
-$56
```

sets a breakpoint at line 56 in the current routine and clears any prior breakpoint in that routine.

### 7.3.2.3 Iteration of Breakpoints

To test the effects resulting from iterative procedures, it is sometimes useful to set a breakpoint in a loop and pass through it several times before allowing execution to halt. This is accomplished with the following command in response to the DDT prompt:

```
>n
```

where:

n — is the iteration count. This is the number of times the breakpoint is to be encountered before execution is halted.

- The iteration count can be set only in the current routine.
- You must be at the breakpoint before issuing the iteration command.
- Execution is halted the nth time the breakpoint is encountered.

Example:

Assuming that a breakpoint is set in a loop at line 25 of the current routine, and the program executes until reaching this point, the response will be:

```
DDT BREAK AT LINE 25 IN ROUTINE XXX
-
```

where:

XXX is the name of the current routine.

You might respond with an iteration count and execution command:

```
>8
-CTRL/Z
```

The routine then loops through this location; stopping the eighth time it reaches line 25 without executing the instruction there. The response is:

```
DDT BREAK AT LINE 25 IN ROUTINE XXX
-
```

### 7.3.3 Variable Manipulation

Variable manipulation allows you to examine or change variables in a routine to determine whether or not they are being correctly handled.

#### 7.3.3.1 Examining Variables

Variables may be examined to verify their contents with the following command in response to the DDT prompt:

```
vvv=
```

where:

| | |
|---|---|
| vvv | is the variable name. Subscripts, either single or double, may be used with the variable name to access a part of a field or data in an unlabeled field. |

Example:

Assume you have stopped at a breakpoint; then:

```
-VAR1=
```

results in a display of the present contents of this variable.

#### 7.3.3.2 Setting Variables

Variables may be set (loaded) with any desired value by using the following command in response to the DDT prompt:

```
     vvv=nnn
```

where:

| | |
|---|---|
| vvv | is the variable name. |
| nnn | is the value you wish to assign to the variable. |

- If the length of nnn is too long to store in vvv, the data is left justified in the field and the excess right-hand characters are truncated. This is true for both alpha and decimal fields.
- Do not use single quotes when specifying alpha data.
- A field, alpha or decimal, can be cleared by entering a space for an assigned value.
- Subscripts, single or double, can also be used to set variables.

Examples:

```
     -VAR1=ABCD
```

Assigns the value of ABCD to VAR1.

```
     -VAR1='ABCD'
```

Assigns the value of 'ABC to VAR1.

#### 7.3.3.3 Extended Variable Manipulation

It is possible under the specific circumstances explained here to examine, or to set, a variable used outside the current routine. This may be done only when the variable is defined in the routine which called the current routine or is defined in one of the routines in the chain of calls which led to the current routine. For example:

```
     -++VAR2=
```

will return the current value of VAR2 located in the chain of routines which called the current routine. The two plus signs indicate that the variable was defined in a routine located two calls back (two levels of nesting) in the chain which led to the current routine. Also:

```
     -++VAR2=EFGH
```

will set VAR2 to the value EFGH.

## 7.3.4 Subroutine Traceback

The subroutine traceback feature allows you to determine whether or not the calling sequences (XCALL statements) are executing in the expected manner. The output is a list of the routines and the line numbers in those routines of all the related preceding XCALL statements back to the main program. To obtain this list, enter the following command (a caret, up arrow, or circumflex) in response to the DDT prompt:

```
     ^
```

There are no arguments. The circumflex (^) causes the list to be generated.

Example:

Assume you have halted in a subroutine at a DDT breakpoint, or you have single stepped to the current position, and you need to know how you arrived at this point from the main program. The command and traceback list might look like the following:

```
     _^
     AT LINE 37 IN ROUTINE SUB3     (current location)
     AT LINE 192 IN ROUTINE SUB2    (SUB3 called from SUB2, line 192)
     AT LINE 21 IN ROUTINE MAIN     (SUB2 called from MAIN, line 21)
     _
```

You are still in routine SUB3 and may enter any DDT command.

# CHAPTER 8

# LINE PRINTER SPOOLERS

## 8.1 INTRODUCTION

The two line printer spoolers, LPTSP1.REL and LPTSPL.TSD, are utility programs used to print data and program source files. LPTSP1.REL is used with SUD systems, and LPTSPL.TSD is used with TSD and XMTSD systems.

### 8.1.1 Common Characteristics

Both spoolers operate in response to the DIBOL LPQUE statement. The function of the spooler, regardless of run-time system, is to queue files to be printed, and to issue print commands to the printer. The DIBOL OPEN statement is not used by a DIBOL program that uses a line printer spooler.

### 8.1.2 Chapter Organization

The remainder of this chapter is comprised of three sections: Section 8.2, SUD; Section 8.3, TSD; Section 8.4, XMTSD. Each section is organized the same way: first, the characteristics of the particular spooler are covered; then the details associated with using the spooler are presented.

## 8.2 LPTSP1.REL (SUD OPERATION)

### 8.2.1 Characteristics

#### 8.2.1.1 Features

The SUD spooler has the following features:

- The SUD spooler is a queue manager program written in assembly language.
- Shared operation of the line printer (between the spooler and a DIBOL program) is not supported. See Section 8.2.2.1 for further information.
- The SUD spooler operates as a foreground job. Other programs run concurrently in the background. See Section 8.2.2.2 for further information.
- Output is to one line printer only.
- LPTSP1.REL can queue up to ten print jobs.

#### 8.2.1.2 Requirements

The SUD spooler requires the following system supports:

- LPTSP1.REL requires 4 K bytes of memory.
- A handler, LP.SYS, must be resident (via the RT-11 LOAD command) in memory.
- LPTSP1.REL must be on the system device.
- LPTSP1.REL requires either the FB or the XM monitor.

### 8.2.2 Using LPTSP1.REL

#### 8.2.2.1 Shared Line Printer Operation

The line printer can not be shared between the line printer spooler and a DIBOL program or a CTS-300 system program. An attempt to use the line printer while LPTSP1 is running will result in a device-in-use error message.

### 8.2.2.2 Shared Terminal Operation

Both the line printer spooler and a DIBOL program running concurrently under the FB monitor can make use of the same terminal. As stated above, the spooler runs in the foreground, and the DIBOL program runs in the background.

I/O operations at the terminal are independent functions for the two programs. The line printer can display a message at any time by assuming control of the terminal. Data that is input at the terminal while the spooler is using the terminal is stored in a 40-character terminal buffer. When the spooler releases control, the stored input data is displayed. A symbol is displayed with each message to the terminal to identify the message source; likewise, an indicator must be provided by the operator to identify the program being addressed. These symbols are shown below:

| **Program** | Input Indicator | Output Indicator |
|---|---|---|
| Spooler program (foreground job) | CTRL/F | F> |
| DIBOL program (background job) | CTRL/B | B> |

If both the spooler and the DIBOL program require the terminal simultaneously, the spooler has priority because it is a foreground job and the foreground always has priority. Under these conditions, data output from the spooler continues to the end of the display line, and then control passes to the DIBOL program which, in turn, outputs a line.

### 8.2.3 Starting

Before running the spooler, the printer handler must be loaded. The handler is loaded for the exclusive use of the foreground program (the spooler). This avoids mixing output from the spooler with output from a DIBOL program. The command is:

```
.LOAD LP=F
```

If you want to print a file that is not on the system device, you must also load the handler for that device, to make it available for the spooler. Use the following command:

```
.LOAD dev=F
```

where:

| | |
|---|---|
| dev | is the name of the device where the file to be printed resides. |

The spooler is started by entering the following command:

```
     .FRUN LPTSP1
```

The response is:

```
     F>
     SINGLE-USER DIBOL LINE PRINTER SPOOLER VAnn-nn
     B>
     .
```

where:

nn-nn is the version number for the spooler program.

When the monitor's prompt (.) appears, any runnable DIBOL program or CTS-300 program can be executed. An LPQUE statement within that program will then cause the spooler to queue and print the specified file.

### 8.2.4 Run-Time Dialog

If either the FORM or ALIGN argument were included in the LPQUE statement, the spooler will output an appropriate message to the terminal. The message is preceded by F> to indicate that it has been issued by the foreground job (LPTSP1). The messages and appropriate responses are listed below:

| **Message** | **Response** |
|---|---|
| F> LOAD xxxxxx IN LP | The operator must load the specified form (xxxxxx), that is, invoices, payroll checks, etc., into the line printer. Operation is continued by typing CTRL/F <CR>. |
| F> IS ALIGNMENT OK ? | The operator must ensure that the two rows of alignment characters are properly positioned on the form in the line printer. If the form is correctly positioned, the response is CTRL/F Y <CR>. If the alignment is not correct, the operator must realign the form and request another group of alignment characters by typing CTRL/F <CR>. |

### 8.2.5 Error Messages

When an error condition occurs, LPTSP1.REL displays one of the error messages listed in Appendix B, Table B-6. All errors for the spooler are fatal errors. Any queued file(s) is lost, and the operator must take the following steps:

1. Enter at the terminal:

   ```
   .UNLOAD F
   ```

2. Correct the condition that caused the error.

3. Restart the spooler, as described in Section 8.2.3.

4. Request a print again.

## 8.3 LPTSPL.TSD (TSD OPERATION)

### 8.3.1 Characteristics

#### 8.3.1.1 Features

The TSD spooler has the following features:

- LPTSPL.TSD is a DIBOL program that operates under the control of the TSD RTS.
- LPTSPL.TSD consists of a queue manager and four satellite programs: LPSAT.TSD, LQSAT.TSD, LRSAT.TSD, and LSSAT.TSD. Each satellite program is active only as long as there are files to be printed on its associated printer.
- Each satellite automatically opens its line printer and loads its handler if not already loaded.
- A maximum of four line printers is supported.
- LPTSPL.TSD supports assignment of default line printers. See Section 8.3.2.1 for more information.
- If spooler execution is terminated before completion of all print requests, the remaining requests are stored for later recovery. The current print job is not lost. See Section 8.3.2.2 for more information.
- Spooler operation is suspended for printers being used by another program. See Section 8.3.2.3 for more information.
- LPTSPL.TSD can be run detached. See Section 8.3.2.4 for more information.
- LPTSPL.TSD can queue up to fifty files per printer. See Section 8.3.2.5 for more information.

#### 8.3.1.2 Requirements

The TSD spooler requires the following system supports:

- The LPTSPL.TSD queue manager requires 4 K bytes of memory; each satellite requires an additional 2 K bytes.
- The number of line printers must be specified during RT-11 SYSGEN.
- Forced job startup must be requested during CTSGEN in order to utilize LPTSPL.TSD.

### 8.3.2 Using LPTSPL.TSD (TSD Operation)

#### 8.3.2.1 Default Line Printers

Ordinarily, a specific line printer is assigned by line printer number to a print job from within a DIBOL program. For example, the following LPQUE statement assigns the second line printer as the printer to print the file named XXX.YYY:

```
LPQUE('RK1:XXX.YYY',LPNUM:2)
```

This however, is not a requirement, since the spooler accepts the assignment of a default line printer. (See Section 8.3.3.1 for assignment.) If more than one default printer is assigned, the default line printer for a given job is defined by LPTSPL.TSD as being the printer having the least number of print requests queued.

#### 8.3.2.2 File Recovery

If spooler execution is terminated before the completion of all print requests, the remaining requests are stored in file LPTSPL.LPQ. After reinitialization, the spooler checks this file and prints the remaining requests. Partially completed print jobs can be reprinted in full. See Section 8.3.4.

#### 8.3.2.3 Suspension of Spooling

When the spooler is started, it opens an I/O channel to the line printer(s). If the printer(s) is being used by another program, an error message indicating the printer is not free is displayed and spooler operation is suspended until that program releases the busy printer.

### 8.3.2.4 Detached Mode Operation

You may select detached program operation for LPTSPL. (See Section 8.3.3). LPTSPL runs independently of the terminal, as long as it is not required by the spooler. During this time the terminal is available for other DIBOL-11 programs. When LPTSPL requires the terminal for information or error messages, the execution of LPTSPL is suspended until an ATTACH command is issued. Then, LPTSPL displays the message; waits for user response, if any; and resumes operation in detached mode. Once detached operation is selected, it remains in effect until LPTSPL.TSD is attached by the user and terminated with a CTRL/C, an S, or killed via STATUS.

### 8.3.2.5 The Queue File

LPTSPL maintains a list of files to be printed in file LPQFIL.LPQ on the system device. LPQFIL has four sections, one for each possible line printer. Each section has fifty slots. If the queue section is full when a print request is received from LPQUE, the spooler attempts to display a message to a terminal indicating that the queue is full, and that the file was not accepted for printing. If the spooler is running detached, see Section 8.3.2.4.

## 8.3.3 Starting

Line printer spooler, LPTSPL.TSD, may be started by any one of four methods. In methods 2, 3, and 4, '?YNNY' refers to answers to the default printer selection. The questions are discussed in Section 8.3.3.1 and the SEND statement is discussed in Section 8.3.3.2 and in the DIBOL-11 Language Reference Manual.

**Method 1, Direct startup:**

```
*R LPTSPL
```

**Method 2, Forced job startup:**

```
XCALL RUNJB ('LPTSPL.TSD',term#)
```

or, if a -1 is specified as the terminal number (indicating a detached job), the spooler must be supplied with the default printer selection information as follows:

```
SEND ('?YNNY','LPTSPL.TSD')
XCALL RUNJB ('LPTSPL.TSD',-1)
```

**Method 3, Chain mode startup:**

```
     STOP 'LPTSPL.TSD'
```

or, if the chaining job is detached, the spooler must be supplied with the default printer selection information as follows:

```
     SEND ('?YNNY','LPTSPL.TSD')

     STOP 'LPTSPL.TSD'
```

**Method 4, Implicit job startup:**

```
     SEND ('?YNNY','LPTSPL',-2)
```

#### 8.3.3.1 Response with an Attached Terminal

When LPTSPL.TSD is started, the following message is displayed:

```
     TIME-SHARED DIBOL LINE PRINTER SPOOLER - VBnn-nn
```

where:

nn-nn    is the version number.

The spooler then asks for default printer identification:

```
     DEFAULT PRINTERS?
     LINEPRINTER 1
```

If line printer 1 is to be a default printer, enter Y. If this printer is not to be a default printer, enter N. The remaining printers are then similarly presented for default selection.

One last question is asked:

```
     DO YOU WANT TO RUN DETACHED?
```

If detached operation is desired, enter Y. The spooler then proceeds to print or wait for a file to be queued. Terminal control returns to the run-time system which displays the asterisk prompt.

If detached operation is not desired, enter an N. No other program can then be executed from that terminal.

#### 8.3.3.2 Response with a Detached Program

If LPTSPL.TSD were started as a detached program, the version number message would not be displayed, so default printer selection could not be asked. The default printer selection information must be supplied via a DIBOL SEND statement from the DIBOL-11 program which is initiating startup of LPTSPL.TSD. This is illustrated above, in Section 8.3.3, in examples 2 through 4. In these examples, the question mark in the SEND statement message SEND('?YNNY',...) indicates to the spooler that the information defines default line printers. Each character after the question mark is a response to a default printer selection question (yes, no, no, yes; in the examples).

### 8.3.4 Stopping

LPTSPL.TSD can be stopped, and the memory it used made available to the system, only if it is attached to a terminal. The spooler is stopped with either a CTRL/C or an S entered at the keyboard, or it can be stopped with the kill option in STATUS; see Chapter 12.

### 8.3.5 Run-Time Dialog

During runtime, LPTSPL.TSD may display messages as a result of interrupted or terminated printing, or as a result of the FORM or ALIGN arguments in the DIBOL LPQUE statement.

**Interrupted or Terminated Print**

If printing is interrupted or terminated, the print job can be resumed. When LPTSPL.TSD is started again, the LPQFIL.LPQ file is checked for incomplete jobs. If such a job is found, the following message is displayed:

| | |
|---|---|
| CONTINUE PRINTING filenam? | If you wish to continue printing from the beginning of the file, respond with Y. If you wish to ignore the file, respond with a <CR>. |

**FORM or ALIGN Arguments**

If the FORM or ALIGN argument(s) were included in the LPQUE statement, the spooler will output the message(s) these arguments imply. If FORMS or ALIGN are used, you should run LPTSPL as an attached job since, without a terminal, LPTSPL will be suspended until it is attached whenever the terminal message is output. The messages with appropriate responses are listed below:

| Message | Response |
|---|---|
| LOAD xxxxxx IN LINE PRINTER n<br>where: n is 1,2,3, or 4 | The operator must load the specified form (xxxxxx), that is, invoices, payroll checks, etc., into the printer. Operation is continued by typing <CR>. |
| ALIGNMENT OKAY FOR PRINTER n?<br>where: n is 1,2,3, or 4 | The operator must ensure that the two rows of alignment characters are properly positioned on the form in the printer. If the form is correctly positioned, the response is Y. If the alignment is not correct, the operator must realign the form and must request another group of alignment characters by typing <CR>. |

### 8.3.6 Error Messages

Error messages for LPTSPL.TSD are listed in Appendix B, Table B-7.

If an error occurs while the spooler is running detached from the terminal, spooler operation is suspended until the operator issues an ATTACH LPTSPL command, at which time the error message will be displayed.

## 8.4 LPTSPL.TSD (XMTSD OPERATION)

### 8.4.1 Characteristics

The operation of LPTSPL.TSD in an XMTSD system is the same as in a TSD system, except that the handlers for the printers must be loaded before the spooler can be run. See Sections 8.4.1.1 and 8.4.3.

#### 8.4.1.1 Features

The XMTSD spooler has the following features:

- LPTSPL.TSD is a DIBOL program that operates under the control of the XMTSD RTS.
- LPTSPL.TSD consists of a queue manager and four satellite programs: LPSAT.TSD, LQSAT.TSD, LRSAT.TSD, and LSSAT.TSD. The satellite programs are running only as long as there is a file to be printed.
- A maximum of four line printers is supported.
- LPTSPL.TSD supports assignment of default line printers. See Section 8.3.2.1 for more information.
- If spooler execution is terminated before completion of all print requests, the remaining requests are stored for later recovery. See Section 8.3.2.2 for more information.
- Spooler operation is suspended for printers being used by another program. See Section 8.3.2.3 for more information.
- Detached terminal operation is supported. See Section 8.3.2.4 for more information.
- LPTSPL.TSD can queue up to fifty files per printer. See Section 8.3.2.5 for more information.

#### 8.4.1.2 Requirements

The XMTSD spooler requires the following system support:

- LPTSPL.TSD requires 4 K bytes of memory; each satellite requires an additional 2 K bytes.
- Appropriate printer handlers must be resident in memory. They are: LPX.SYS, LQX.SYS, LRX.SYS, or LSX.SYS.
- The number of line printers must be specified during RT-11 SYSGEN.
- Forced job startup must be requested during CTSGEN in order to utilize LPTSPL.TSD.

### 8.4.2 Using LPTSPL.TSD (XMTSD Operation)

See Section 8.3.2. The information for TSD operation is valid for XMTSD operation except for the ATTACH command which becomes A under XMTSD (see Chapter 5, Section 5.5).

### 8.4.3 Starting

Before execution of LPTSPL.TSD in the XMTSD environment, the appropriate handler(s) must be loaded. If all four handlers are required, the procedure is:

```
.LOAD LP
.LOAD LQ
.LOAD LR
.LOAD LS
```

or you may issue a single line command:

```
.LOAD LP,LQ,LR,LS
```

Handlers LP, LQ, LR, and LS correspond to line printers 1, 2, 3, and 4, respectively.

Under the XMTSD monitor, the correct handler is loaded with the above commands. This is done despite the fact that the XM handler name has a third character (X). After the handlers are loaded, proceed as in Section 8.3.3 to run the spooler.

### 8.4.4 Stopping

See Section 8.3.4

### 8.4.5 Run-Time Dialog

See Section 8.3.5.

### 8.4.6 Error Messages

Error messages for LPTSPL.TSD are listed in Appendix B, Table B-7.

If an error occurs while the spooler is running detached from the terminal, spooler operation is suspended until the operator issues an ATTACH LPTSPL command, at which time the error message will be displayed.

# CHAPTER 9

# PRINTU

## 9.1 INTRODUCTION

PRINTU is a utility program for the creation of simple report programs using data from either a sorted sequential file or from an ISAM file. PRINTU utilizes a user-written control file that describes the parameters of the desired report. Given the control file, PRINTU generates a DIBOL program that is compiled and linked like any other DIBOL program. The resulting program, when run, produces the report.

### 9.1.1 Features

PRINTU has a number of features that makes it particularly useful:

- PRINTU has the ability to process data files without physically reordering them by using a sorted tag file generated by the SORT utility.
- Once a report program is created, it can be stored for later use.
- Since PRINTU generates a DIBOL program, extra features or capabilities can be added by simply modifying the program to achieve your needs. These needs could be special output print lines, more header lines, output based on logical test results, operation with packed records as input, or any other feature you wish to develop.

### 9.1.2 Limitations

There are limitations that require foresight when you are using PRINTU. These concern the way you handle file control records (explained in the next subsection) and the DIBOL requirement for an end-of-file (EOF) marker (a CTRL/Z) in all data files. Potential problems in both of these areas are easily avoided, however, as you will see in the next two subsections.

### 9.1.3 File Control Records

Some application system files are designed to contain a control record as the first record. More than likely this will produce meaningless output on the first report page printed. There are two ways to prevent this from happening: the first way is to do a tag sort on the file and ensure that the control record sorts out as the first record in the tag sort file. You can then use an editor to delete this record. PRINTU will then begin with the second record in the file. The second way to prevent a control record from interfering with a print report is to place the control record at the end of the file. To prevent PRINTU from accessing this record you can precede this record with another record containing an EOF marker as the first character. These two records can be inserted using direct access. The EOF character is obtained by using the decimal equivalent of CTRL/Z (026) and an XCALL to ASCII. The control record is then not accessible by PRINTU.

### 9.1.4 Chapter Organization

The remainder of this chapter is comprised of two sections. Section 9.2, The Control File, is a detailed discussion of the PRINTU control file and how the individual control statements relate to each other and to the desired printed report. Section 9.3, Using PRINTU, illustrates the manner in which the control file is used with the PRINTU program to produce a DIBOL file, and how this DIBOL file is compiled and linked.

## 9.2 THE CONTROL FILE

PRINTU depends on information supplied by the user to describe the input file(s), output file, and other parameters of the report. This information is given to PRINTU via a control file containing control statements. The control file is created using an editor. The control file is the heart of PRINTU. It is here that you provide the parameters that determine the appearance of the report.

The control file has its own terminology and, at times, this can be confusing. For this reason, the following essential terms are introduced. Many of them are further defined in context.

| | |
|---|---|
| accumulation field | A user-identified field whose value from record to record is accumulated to produce a total. |
| break field | When the data in a break field changes, a print occurs. The break field is covered in detail in Section 9.2.5. |

| | |
|---|---|
| detail print | The normal print mode (as opposed to a summary print - see below). Individual values are printed, as well as the totals. |
| input data file | The file containing the data which is to be used as the source for your report. |
| report file | The output of the report program when that program is run. |
| report program | The program generated by running PRINTU and then compiling and linking the resulting DIBOL source file. |
| summary print | A printout that includes only the totals from each accumulation field whenever a new value occurs in a break field. |
| tag file | The file produced by the SORT utility when its TAG option is selected. Used by PRINTU to access the input data file. |

There are nine possible control statements in the control file, of which only four are required. The other five control statements supply optional capabilities that you may or may not need. The control file structure, format, content, individual control statement descriptions, and other characteristics are explained below and in subsequent sections.

Before we discuss the control file, briefly consider what your goals for the report are. You begin with a file whose record structure is known to you, and your major goal is to produce a printed report that shows the data in this file in some desired order in relationship to the data in each record field or fields. Additionally, there may be some data in each record that you wish to ignore, and there are probably one or more fields in related records which contain numeric data that you would like to have totaled.

How these and other requirements are achieved is covered in detail in the individual control statement descriptions. However, the following gives you an idea where, in the context of a simple control file, these requirements are specified.

You will find PRINTU easy to understand and use if for the first few times you use it you confine yourself to the four required control statements. A useful report can be generated with these four alone. The four control statements and their characteristics are:

| | |
|---|---|
| IDENT | This statement provides a means of assigning a name to the control file. It has no effect on input data selection, format, or content of the printed report. |
| OUTPUT | This statement serves only to assign a unique name to the resulting print report. This statement is optional in the sense that, if not included, a prompt will request the name at program run time. |

INPUT — INPUT is one of the two most important field definition statements. It is here that you:

- Select the input data file.

  If this statement is omitted, this information will be requested via a prompt at run time.

- Describe the fields that you need from within the input data record.
- Select break fields.
- Request a new printed page concurrent with a break field change.
- Select whether you print only totals or intermediate values, as well.
- Request to read an ISAM input file sequentially.

PRINT — PRINT is the other field definition statement that is of prime importance. It is here that you:

- Make the logical connection between fields identified in the input statement and fields to be printed.
- Assign column headings for each field printed.
- Select numeric fields that are to be accumulated.
- Format the numeric data to be printed.
- Assign column spacing.

All nine control file statements are discussed in the following sub-sections. Optional input for each statement is shown in brackets. Comment lines are allowed only as shown for each entry. An example is given for each statement as it is presented. Each example is part of an overall exercise to build a usable control file, so, therefore, some examples build on previous concepts. The completed control file is shown as part of the example in Section 9.3.1.

### 9.2.1 IDENT

IDENT is the first entry in the control file. It provides the control file title and appears on the first line of the DIBOL listing of the report program and at the top of each printed report page. Its most important use is to identify your control file when you run PRINTU.

It is of the form:

```
     IDENT program, author                                   [;comment]
```

where:

| | |
|---|---|
| program | is the identifier you wish to assign to the report. It can be up to 22 characters long with a slash (/) used as a word separator. |
| author | is any text up to 24 characters long. |

Example:

```
     IDENT PAY/ANALYSIS, Your name
```

The START line of the DIBOL listing would look like the following:

```
 START    ;PAY ANALYSIS      -YOUR NAME
```

## 9.2.2 HEAD1 and HEAD2 (Optional)

HEAD1 defines the first heading line appearing on each page of the report and HEAD2 defines the second. Both are optional, and headings are centered on the page. The form is:

```
     HEAD1 'text'                                            [;comment]

     HEAD2 'text'                                            [;comment]
```

where:

| | |
|---|---|
| text | is a string of characters from the DIBOL set, exclusive of quotes, up to 132 characters long. |

There may be more than one HEAD1 or HEAD2 line. If so, the individual text lines for given HEADn statements are concatenated. This is necessary for long titles. If HEAD1 is less than 67 characters, the line will be expanded by inserting a space between each character. The total number of characters for either line is limited to 132.

Example:

```
     HEAD1 'PAY '
     HEAD1 'ACCOUNTABILITY'
     HEAD2 'BY EMPLOYEE AND TASK CODE'
```

The heading on each page would then appear as follows:

```
                    P A Y   A C C O U N T A B I L I T Y

                         BY EMPLOYEE AND TASK CODE
```

### 9.2.3 EXECUTE (Optional)

EXECUTE creates the DIBOL stop and chain exit in the report program. The format is:

```
     EXECUTE filespec                                     [;comment]
```

where:

filespec is a CTS-300 DIBOL file specification.

Example:

```
     EXECUTE DL1:STATUS.TSD
```

Upon completion of the report, the report program would chain to a program called STATUS.TSD on device DL1.

**NOTE**

The EXECUTE statement must appear after IDENT and before the INPUT statements.

### 9.2.4 OUTPUT

OUTPUT identifies, by nature of your response, either a printer or a report file that is the destination of the output that results from running the report program. Its form is:

```
     OUTPUT filespec
```

where:

filespec is a CTS-300 DIBOL file specification.

No comment line is allowed with this statement.

Example:

```
     OUTPUT RK1:PAYACC.DDF
```

When the report program is run, the resulting report is placed on RK1 and is named PAYACC.DDF.

**NOTE**

The OUTPUT statement must appear after IDENT and before the INPUT statement.

If the OUTPUT statement is omitted from the control file, the report program will request the file specification at run time. The message will be:

```
OUTFILE NAME:
```

The ability to supply the output file specification at run time permits you to use the same report program on the same data file to produce a file with a different name and/or device destination.

## 9.2.5 INPUT

INPUT is a multiline control statement. It specifies the data source for the report program (Input Statement Line) and describes the input data file's record structure (Field Description Lines).

**NOTE**

> The input file must be sorted in relation to the break field(s) before it is supplied to the report program. This requirement becomes obvious when you become familiar with the function of a break field.

The multiline format is:

```
INPUT [filespec][/SU][/IS]
[;comment]

        |A|
[name], | |n [.m] [,Lr [P]]
        |D|

[;comment]
```

The Input Statement Line and Field Description Lines are discussed separately.

Input Statement Line

Taking the INPUT statement by itself, the form is:

```
     INPUT [filespec][/SU][/IS]
     [;comment]
```

where:

| | |
|---|---|
| filespec | is the CTS-300 DIBOL file specification for the input data file from which the report is generated. |
| /SU | is the summary switch. When included, it causes suppression (during print) of individual values being accumulated. See PRINT, Section 9.2.8. |
| /IS | indicates that the input data file is an ISAM file to be read sequentially. IS must be used when you wish to generate a report directly from an ISAM file without using the INDEX statement. See Section 9.2.6. |

**NOTE**

The default condition assumes that the input data file is a sequential file. If the input data file is an ISAM file, either the IS option or the INDEX statement must be used.

Comments are on a separate line as shown.

If the file specification alone is omitted, the report program will request it at run time. The message will be:

```
     INFILE:
```

The options SU and IS may not be supplied at run time, but can be selected in the control file, with a file specification to be supplied at run time, by using the form:

```
     INPUT /SU
```

The ability to supply the input data file specification at run time, combined with the same ability for the output file, makes it possible to use a given report program with any number of input data files and to produce a separate output for each. Of course, the record structure must be the same in all such input files.

Field Description Lines

Before analyzing a field description line, it is necessary to understand exactly what a field description line does.

One function of a field description line is to describe the input data file's record structure. The data record is partitioned into fields, and the nature of each field is described in detail. These details include name, data type, size, and, for decimal fields, number of decimal places.

The field description line is also used to identify break fields. The break field contains data that is constant over a series of records. However, for each break field there are corresponding fields which may contain data which can vary from record to record. It is this relationship that is the basis for reports. And, in PRINTU, it also controls when data is printed.

An example may help to clarify this concept:

A record might contain an employee's name, the date, a task code, and the hours worked for this task code. For a given employee, there will exist at least one record for every task code for which time was charged. In the simplest case, you may want a report showing the hours worked by each employee. Here the field containing the employee's name would be identified as the break field. The report would then print all the hours worked by each employee. Note that it may also be desirable to show, for a given employee, the total hours worked in a given day or on a given task.

More than one break field can be identified. For example, it may also be desirable to report hours worked for each task code, as well as for each employee. In this case, task code would also have to be identified as a break field. With more than one break field, a way to show this relative importance is necessary. PRINTU allows you to identify both break fields within the control file INPUT statement and to show their relative importance. In this second case, task code hours would appear as subtotals every time the task code changed, and a total of hours by each employee's name would appear as each name changed.

From the preceding paragraphs, it can be seen that a break field controls the printing of data. Whether it is a summary print or a detail print is also determined by the INPUT statement. Since the file is read sequentially, the input file must, in the first case above, be sorted by employee name. If other break fields of lesser importance were identified, they also would have had to have been identified as minor keys (in the correct order) in the sort. Task code as a break field would be identified as a minor key in the sort.

Again, the format for a field description line is:

```
           |A|
    [name],| | n [.m] [,Lr [P]]
           |D|
    [;comment]
```

where:

| | |
|---|---|
| name | is the symbolic name of the field to be used for reference by the COMPUTE and PRINT statements. See Sections 9.2.7 and 9.2.8. It can contain up to six alphanumeric characters. |
| \|A\|<br>\|D\| | indicates alpha or decimal field type. |

| | |
|---|---|
| n | is the size of the field expressed in decimal characters. |
| m | is the number of decimal places in the field and is valid only for decimal fields. |
| Lr | indicates a break field in which the r is a single decimal digit, which expresses the relative importance of the break field compared to other break fields (1 is the least important and 9 is the most important). When a "break" occurs, the total is printed for fields being accumulated. This total also includes all other fields of lower break level. See PRINT, Section 9.2.8. All accumulated values are then cleared to zero. |
| P | indicates that the report will start a new page after the totals for this break are printed. |

Comments are on a separate line as shown.

Twenty is the maximum number of labeled fields allowable within the INPUT control statement.

Example (INPUT statement and field definition lines):

```
INPUT EXAMPl
;comment line
NAME,      A15,L2P
,            A7
DATE,      D6
TCODE,     A4,L1
CRATE,     D3.2
HOURS,     D2.1
```

Five fields have been identified for the data contained in each record of file EXAMPl. Data areas within the record that are not of interest can be ignored by using an unnamed alpha field of appropriate size. In this example, there is one such area seven characters long. The two fields, NAME and TCODE, are identified as break fields and NAME is of greater importance. Whenever the data in TCODE changes, the output lists specified totals (and individual values, unless a summary is requested). The exact format of the data to be printed will be defined in the PRINT statement (see Section 9.2.8). When NAME changes, the totals printed include the totals from intermediate TCODE lines. A new page is started after each NAME change. HOURS is a two-character field with one decimal place. CRATE is a three-character field with two decimal places.

## 9.2.6 INDEX/LIST (Optional)

INDEX and LIST are statements that enable PRINTU to utilize a tag file generated by the SORT utility. See Chapter 11, SORT/MERGE, and the TAGS:LIST and TAGS:INDEX options. The usefulness of such a tag file becomes apparent when you consider that it is unlikely that the input data file will be arranged in the order that you wish for your report.

There are two solutions to this problem. You can either reorder the input data file, which is especially time-consuming if you wish several kinds of reports from the same data, or you can use the SORT-generated tag file. The tag file allows access to the input data file in its original form. Consequently, it is not necessary to reorder the entire input file when a different sort sequence is desired. All you have to do is generate a new tag file.

A tag sort is first carried out on the input data file. The output is a file of records containing only the relative record number, or if it is an ISAM input file, the file contains the SORT keys also.

The tag file is specified for use in PRINTU by either the INDEX or LIST statement. When INDEX is used, it implies that the input data file (as defined by the INPUT statement) is an ISAM file. When LIST is used, it implies that the input data file (as defined by the INPUT statement) is a sequential file.

The format, if the input file is a sequential file, is:

```
     LIST filespec                                      [;comment]
```

where:

| | |
|---|---|
| filespec | is a CTS-300 DIBOL file specification of the tag file generated by the SORT utility. |

There are no qualifiers with LIST. The tag file is assumed to consist of records which are seven-digit decimal numbers.

Example:

```
     LIST DK1:SORTO.DDF
```

This statement line specifies tag file SORTO.DDF on DK1. SORTO.DDF is a result of previously having run SORT with the TAGS:LIST option on the sequential file defined in the PRINTU control file INPUT statement. This tag file is a file of seven-digit numbers.

The format, if the input file is an ISAM file, is:

```
     INDEX filespec/recl,key,length                     [;comment]
```

where:

| | |
|---|---|
| filespec | is a CTS-300 DIBOL file specification of the tag file generated by the SORT utility. |
| recl | is an integer defining the record length of the tag file as generated by the SORT utility. |
| key | is an integer defining the starting position of the ISAM key within a tag file record. |
| length | is an integer defining the length of the ISAM key within a tag file record. |

Example:

```
     INDEX RK1:SRTOUT.DDF/12,8,5
```

This statement line specifies a secondary input file SRTOUT.DDF on RK1. SRTOUT.DDF is the result of previously having run SORT with the TAGS:INDEX option on the ISAM file defined in the PRINTU control file INPUT statement. This example indicates that the secondary input file record length is twelve characters long, and that the ISAM key is five characters long, beginning at character position eight. The seven-character relative record number is not used.

If neither the INDEX nor LIST statement is included in the control file, the input file is processed as a sequential file.

## 9.2.7 COMPUTE (Optional)

COMPUTE is a multiline control statement that allows you to include data in your report that does not exist in the input data file. This new data is a result of a mathematical process performed on the input data. The COMPUTE line is followed by one or more additional lines describing the mathematical process. There can be up to eight of these additional lines. The multiline format is:

```
     COMPUTE
     [;comment]
     name=expression[#n]
```

where:

| | |
|---|---|
| name | is the symbolic name given to the computation result and is used for reference by the PRINT statement (see Section 9.2.8). It must neither duplicate any input field name nor any previous computation result name. |
| expression | is any valid DIBOL expression. It may be an alpha or decimal field or a literal. See SPECIAL RULES below. |
| #n | is an integer specifying the number of decimal places the resulting computation will have. See SPECIAL RULES below. |

The comment line is a separate line as shown.

**SPECIAL RULES FOR COMPUTE:**

These special rules concern the number of decimal places for operands within a mathmatical operation. The PRINT control statement, unlike the DIBOL-11 language, maintains the number of decimal places. In all arithmetic operations, the operands must have an equal number of decimal places. Operands can be either constants or variables. The following rules determine the resulting decimal places:

| | |
|---|---|
| addition (+) | no change in decimal places |
| subtraction (-) | no change in decimal places |
| multiplication (*) | the sum of decimal places |
| division (/) | the difference in decimal places |

To adjust the number of decimal places of any operand, multiply by a suitable decimal representation for the constant 1. For example, if operand 1 has two places, and operand 2 has no places; then operand 1 * operand 2 * 1.00 will be acceptable and will result in a product which will have four decimal places. For example:

```
COMPUTE
;compute total pay
TPAY=CRATE*HOURS*1.00#2
```

Total pay (TPAY) is equal to compensation rate (CRATE) multiplied by hours worked (HOURS). TPAY is to have two decimal places.

### 9.2.8 PRINT

PRINT is a multiline control statement that defines the format of the final print report and identifies the variable fields that are to be accumulated (added) and printed as totals for a given break field. The PRINT control statement is followed by one or more additional statements describing the overall print format and accumulation fields. The multiline format is:

```
PRINT
[;comment]
name,'text' [,A] [,picture]
    or
,An
```

where:

| | |
|---|---|
| name | is the name of a field from an INPUT field description line (see Section 9.2.5) or the name assigned to a mathmatical result within a COMPUTE statement (see Section 9.2.7). |
| text | is a string of characters from the DIBOL character set (excluding single quotes). This text is used as the heading for the individual columns for the data fields identified by name as well as in the line printed for totals for each break field. An asterisk in the text string will cause the remaining text to be printed on the next line. Text is automatically centered. An asterisk is treated as a space when the text is printed. |

| | |
|---|---|
| A | indicates that the field is to be accumulated and the sum is to be printed as a total for a break field. This field must be decimal. If the /SU option were selected in the INPUT statement, only the total would be printed (summary print), not the intermediate values. |
| picture | is a string of text in the form of a DIBOL data format field. Here it is valid only for decimal fields. If a picture is not included, PRINT will create one using the description of the field. If it is an accumulated field, two extra places will be assumed. The form is: XX,XXX.XX- |
| ,An | is an alternate format line that will produce blank column separators of n characters in width between printed output columns (where n is an integer from 1 to 9). If not included, two blank columns will be assumed. |

The comment line is a separate line as shown.

Example:

```
PRINT
;comment
NAME,      'EMPLOYEE'
,A4
TCODE,     'TASK*CODE'
,A4
HOURS,     'HOURS*WORKED',A,ZZ.Z
,A3
TPAY,      'TOTAL*PAY',A,ZZZZ.XX
```

Assuming that INPUT and COMPUTE statements remain as previously illustrated, in this example the print output would be:

```
EMPLOYEE    TASK       HOURS      TOTAL
            CODE       WORKED      PAY

 name1      code1       ZZ.Z      ZZZZ.XX
                        ZZ.Z      ZZZZ.XX

    TASK CODE TOTAL   ZZZZ.Z    ZZZZZZ.XX

 name1      code2       ZZ.Z      ZZZZ.XX
                        ZZ.Z      ZZZZ.XX
                        ZZ.Z      ZZZZ.XX

    TASK CODE TOTAL   ZZZZ.Z    ZZZZZZ.XX
     EMPLOYEE TOTAL   ZZZZ.Z    ZZZZZZ.XX

 name2      code1       ZZ.Z      ZZZZ.XX    (New Page)
      (and so forth)
```

### 9.2.9 END (Optional)

The END statement is optional and consists of a line containing only the word:

```
     END
```

## 9.3 USING PRINTU

### 9.3.1 Producing the Report Program

Once the control file is written, using PRINTU is a straightforward process. The control file is created manually by the user with the aid of an edit program. The file simply consists of the required PRINTU statements from Section 9.2 plus any optional statements desired. Regardless of which editor you use to create your PRINTU control file, you will be required to specify a name for the file resulting from your edit session. Specify this name as input to the PRINTU program. The resulting DIBOL program source file is compiled and linked just like any other such program. The following dialog illustrates in detail the steps required. The same control file statements developed for the examples in Section 9.2 are used.

There are five steps:

1. **Create the control file:**

```
.R DKED
*PTSAMP.TXT=
IDENT PAY/ANALYSIS, YOUR NAME
HEAD1'PAY ACCOUNTABILITY'
HEAD2'BY EMPLOYEE AND TASK CODE'
EXECUTE URPROG.TSD
 ;return to your
 ;program when
 ;finished
OUTPUT RK1:PAYACC.DDF
INPUT EXAMP1.DDF
;input file is a sequential file on the default device, no
;summary wanted
NAME, A15,L2P
,   A7
DATE, D6
TCODE, A4,L1
CRATE, D3.2
HOURS, D2.1
COMPUTE
;compute total pay
TPAY=CRATE*HOURS*1.00 2
PRINT
;comment
```

```
NAME, 'EMPLOYEE'
,A4
TCODE, 'TASK*CODE'
,A4
HOURS, 'HOURS*WORKED',A,ZZ.Z
,A3
TPAY, 'TOTAL*PAY',A,$$$$.XX
END
<GOLD/COMMAND>
EXIT
```

**2. Run PRINTU:**

Enter the following command:

```
.R(RU) PRINTU
```

The general form for entry of arguments in PRINTU in response to the asterisk prompt is:

```
*outfil,lstfil=txtfil
```

where:

| | |
|---|---|
| outfil | is the CTS-300 DIBOL file specification for the name assigned to the output of PRINTU. It consists of the appropriate device, the name of the text file containing the PRINTU control file. Since the output of PRINTU is supplied to the compiler, an extension of .DBL should be chosen. |
| lstfil | is the CTS-300 DIBOL file specification for the PRINTU list file. |
| txtfil | is the name given to the PRINTU control file in the edit session. |

For the example being developed here the entry is:

```
*PTSAMP.DBL,LP:=PTSAMP.TXT
```

which produces the DIBOL source file PTSAMP.DBL on the default device and a listing of this program on the line printer.

It might be useful to first find the syntax errors, using the following command:

```
*PTSAMP.DBL,TT:=PTSAMP.TXT
```

edit out the errors, and (assuming you do not need a listing) enter:

```
*PTSAMP=PTSAMP.TXT
```

**3. Compile:**

```
.DIBOL PTSAMP
```

4. **Link:**

```
.LINK PTSAMP,DIBOL
```

5. **Run the report program:**

```
.R PTSAMP
```

**NOTE**

> Ensure that the input file is sorted in relation to the chosen break fields before the report program is run. See Section 9.2.5, INPUT.

PTSAMP is the report program, and the result of running it will be file PAYACC.DDF which is written to RK1 in the format specified in the PRINT statement.

## 9.4 ERROR MESSAGES

See Appendix B, Table B-8, for a list of error messages for PRINTU.

# CHAPTER 10

# ISAM (ISMUTL)

## 10.1 INTRODUCTION

ISMUTL is the DIBOL utility program used to build and reorganize CTS-300 ISAM (Indexed Sequential Access Method) files. This chapter introduces CTS-300 ISAM and its terminology and then explains the internal structure in the detail necessary for you to be able to build and use an ISAM file. Each of the various functions of ISMUTL are discussed separately, using flowcharts (where appropriate), actual program dialog, and examples.

### 10.1.1 Features

ISAM files offer several advantages over random or sequential files:

- Simplified record storage and retrieval
- Greater file access speed
- Ability to delete records
- More easily designed and modified to fit the growth requirements of the application.

### 10.1.2 Chapter Organization

The remainder of this chapter is comprised of three major sections. First is Section 10.2, ISAM Basics, for users who are not familiar with ISAM concepts and terminology. Section 10.3, ISAM Internals, discusses the internal structure of an ISAM file and explains the relationships that exist within the file as well as the factors that must be considered when designing an ISAM file. ISAM-related DIBOL statements are also covered, along with data storage and access. Section 10.4, Using ISMUTL, describes the use of the ISAM utility to create or reorganize your particular ISAM file.

## 10.2 ISAM BASICS

The basic concepts and terminology of ISAM are presented in this section. It is a brief, overall description of an ISAM file. The purpose is to provide background for detailed discussions in the following sections. The indexed sequential access method (ISAM) is a means of organizing data for storage and subsequent access. An ISAM file is composed of two major sections: the data section and the index section.

### 10.2.1 Data Section

Records are assigned, in ascending order, to a specific location in an ISAM file, called the data section, which is comprised of data files. A user-identified field within the record determines where that record is stored in a data file within the data section. This field is called the record key. In CTS-300 ISAM, there is only one such key field per record. Records are sequentially stored by this key in data groups of constant size. These data groups are organized to form as many as seven data files. These data files constitute the data section.

### 10.2.2 Index Section

The greatest record key within each data group is entered, along with the address of its associated data group, into an index. The index is contained in the index section, often called the index file. This index section (index file) is the second major division of an ISAM file. By searching this index for a key match, the group which contains the desired record may be quickly accessed. Each record in the group is then searched sequentially for a key match, and the desired record is obtained.

### 10.2.3 Handling Added Records

Suppose you wish to store a record that, because of its key value, belongs in a given group but which cannot be placed there because the group is full. One solution would be to leave a few empty record spaces in each group when the file is first constructed and the initial data is loaded. When this is done, the number of spaces left empty is called the load exclusion factor. But what happens when even these spaces in a given group are filled? A way to handle this is to set aside an area somewhere within the total ISAM file to place the new records. This is called the overflow area, and it physically precedes the index in the index section. This overflow area is organized in groups identical to data file groups but, as will be explained later, access is different.

There is one remaining question: How do you add records which have greater key values than those of any existing group? These new greater key records cannot go into overflow, because they do not logically belong in any existing group. There is no such group, because when an ISAM file is created from a given input file, the ISAM file size is determined by the initial space requirements of the input file. The solution is to provide space at the end of the data area, in addition to the known initial space requirement. This area is known as the append area.

### 10.2.4 Summary of ISAM Basics

In summary, there are two basic parts to an ISAM file: the index section and the data section. The index section is by far the smaller of the two and contains the overflow record groups and the index to the data portion of the file. A search for data is made via the index which points to a data group which is, upon access, searched sequentially for the desired record. Added records of intermediate key value go into open record spaces, load exclusion record spaces or, if the group is full, into the overflow area. Records with keys greater than the current key range are placed in the append area. The structure of an ISAM file is illustrated in Figure 10-1.

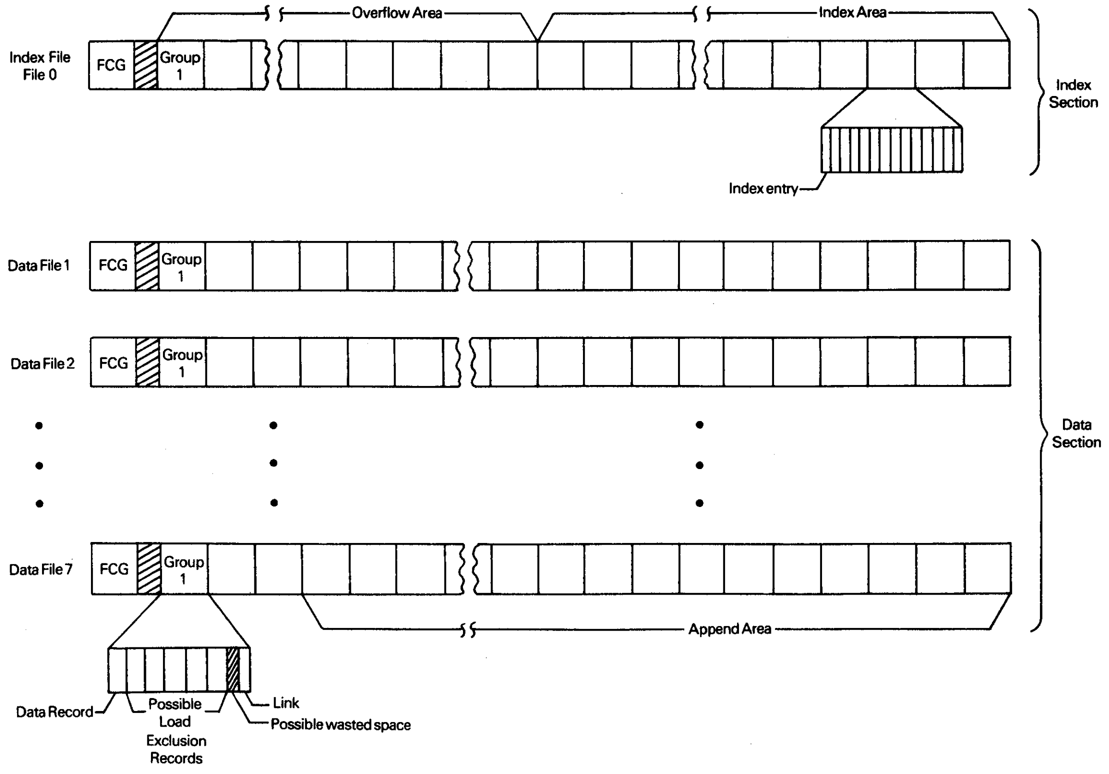


Figure 10-1 ISAM File Overview

## 10.3 ISAM INTERNALS

The previous section presented a simplified picture of an ISAM file. However, a more detailed understanding is necessary for the ISAM user in order to respond to questions asked by a utility (ISMUTL) that will direct that utility to construct the index, overflow, and append areas. To build a file that provides maximum efficiency in terms of access speed and storage requirements, requires a thorough and complete understanding of both the ISAM file, itself, and the particular requirements dictated by the data. This section, then, discusses the file structure in detail; the interrelationships between the various parts of the file; and, as an aid to more efficient use, the DIBOL statements used to access ISAM files. Section 10.3.4 illustrates how data is handled in an ISAM file.

### 10.3.1 Detailed Structure

#### 10.3.1.1 Data Files

Data files are discussed first, because the overflow area is constructed like a data file, and also because the index and overflow area are more meaningful after the data file is understood.

As stated before, there may be up to seven data files (or logical volumes) in an ISAM file. These are labeled 1 through 7. Logical volumes are RT-11 files which may be located on the same physical volume (a disk, for example) or on different physical volumes. All access to the file is done using the name assigned to the index file. The main reason for multiple data files is to allow division of the total ISAM data file over the system resources.

Each of the data files in an ISAM file may be a different size but all are constructed the same. Each file is segmented into sequentially numbered groups of equal size. The first group (group 0) of each data file is reserved for system use and contains only 132 bytes of data that are meaningful, regardless of group size. This first group is called the File Control Group (FCG). FCGs are created at the beginning of each data file and at the beginning of the index file when the ISAM file is created. As the ISAM file changes, only the FCG in the index file is updated. More information about the FCG is contained in the index file discussion. All remaining groups in the data file are data groups.

Each data group consists of a record area and, at the end of the group, an area containing linking information to the next logical data group. Within the record area are the data records which are of equal length. Data groups eventually become filled as records are stored.

When each group is filled, the records are ordered so that the last record in the group has the greatest key. This key is then inserted in a preassigned position in the index file. Thus each data group has a corresponding index entry. Index space is set aside when the ISAM file is originally built, with one index entry being allocated per data group.

The last four bytes of each group are reserved for linking and are not accessible to the user. The link information consists of one byte indicating the number of presently valid records in the group; one byte for the file number of the next logical data file; and two bytes for the number of the next logical data group within that next logical data file. Until overflow occurs, a link will always point to the next data group in the present file or to the first data group in the next data file. When overflow occurs, the link points to a group in the overflow area. The groups in overflow are identical in structure, including link structure, to those in the data file. The link in the overflow group points back to the original data file if the number of records requires only one overflow group; or if more than one overflow group were required to contain the overflowed records, the link in the last overflow group would point back to the original data file.

The append area is simply an extension of the data section, resulting from the need for more storage area than is indicated by the input file size. Since the append area is a part of the data section, added records in the append area may also cause overflow.

The choice of group size depends on many factors and these are discussed in Section 10.3.2.

Figure 10-2 illustrates the concepts of data group organization discussed above using a group that is 512 bytes in size with five 100-byte records, two of which are identified as load exclusion.

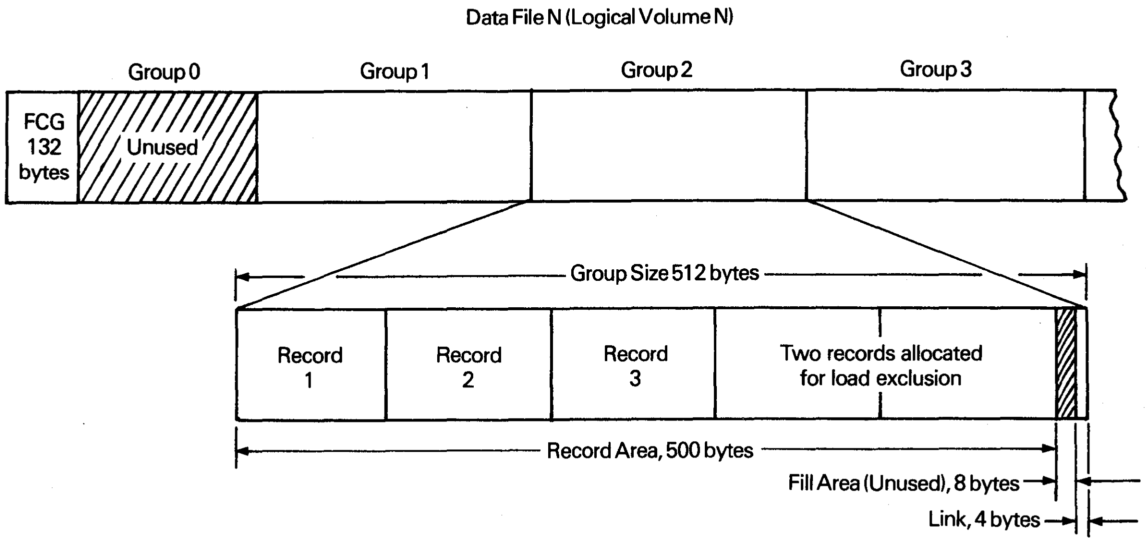


Figure 10-2 Data Group

### 10.3.1.2 Index File

An ISAM file contains one, and only one, index section (file). This is defined as file 0. File 0 is always examined first to determine the status of the entire ISAM file and also to determine the group location of a desired record, or the location of the group in which to store a record.

The entire index file is divided into groups which are the same size as the data file groups. The first group, group 0, as in the data file, is an FCG. The first 132 bytes of this group contain a description of the ISAM file structure, file status, and data file allocations in terms of number of groups. Unlike the data file FCGs, the FCG for the Index File is updated as the ISAM file grows and changes. Because CTS-300 ISAM files are designed to be compatible with CTS-500 ISAM files, the FCG contains some information not used by CTS-300. Table 10-1 shows the contents of the index file FCG.

Following the FCG group in the index file is the overflow area (if selected) of a size determined by the user. No index entries were set aside for the overflow groups, as there were for the data section groups. All records in overflow are accessed via the linking information in each data file group. Once a group is accessed, whether a data file group or an overflow group, the records in that group are accessed sequentially.

The last part of the index section is the index. Since the group size is the same as in the data file; and since the key is smaller than its corresponding record, there are many more index key entries per group than records per group. Each index entry consists of the record key and associated link to its data group or, as you will see, to another index entry. If the keys and their links do not fit exactly into the space available in the group, the unused space is ignored.

To speed access, the index, itself, can be indexed. The resulting levels of indexing are determined by an algorithm based on the total number of data file groups. In fact, all index entry spaces are preassigned, and links are preconstructed, based on this total number of data file groups. The highest (coarsest) level of indexing appears first in the file and the lowest (finest) level of the index (the largest part) is at the end of the file. The higher index entries point, via index entry links, to lower levels until, at the lowest level, the index entry points to a specific group in a data file.

The organization of the index area of the index section is shown in Figure 10-3. This is a hypothetical index for an ISAM file of approximately 8000 records with five data records per group and 20 index entries per group. There are seven data files with approximately 1200 records (240 groups) per data file. Keys are numeric, starting with one. The illustration is of a search for a record whose key is 5877. The search is started in the highest (coarsest) level of the index. In this example this level consists of four entries in group 101. The search is for a key of equal or greater value than 5877. The index entry link for index key 5998 points to group 104. The search of this group stops at key 5887, whose associated link points to group 157 in the lowest (finest) section of the index area. The search of the group ends at key 5879 which points, via the link, to data file five, group 257. Group 257 is searched sequentially for the desired record.

Table 10-1
File Control Group

| No. Bytes | Description | Comment |
|---|---|---|
| 2 | No. of data groups in this file | |
| 2 | No. of records per group | |
| 2 | Record length in bytes | |
| 2 | Key length (1-n) | |
| 2 | Key location in record (1-n) | |
| 2 | Allow duplicate keys | 0=no,-1=add at beginning,+1=add at end |
| 2 | No. retries for locked block | |
| 2 | No. levels of indexing | |
| 2 | Group No. of first group in primary index | |
| 2 | No. index entries per group | |
| 2 | No. data records per group | |
| 2 | Current No. groups in overflow | |
| 2 | Maximum No. groups in overflow | |
| 2 | Load exclusion factor (0-n) | |
| 2 | File protection code | Not used in CTS-300 ISAM (0) |
| 2 | Group length | |
| 2 | Last file in use (0-7) | |
| 2 | File 0 clustersize (2-256) | 1 |
| 2 | File 1 clustersize (2-256) | 1 |
| 2 | File 2 clustersize (2-256) | 1 |
| 2 | File 3 clustersize (2-256) | 1 Not used in CTS-300 |
| 2 | File 4 clustersize (2-256) | 1 ISAM |
| 2 | File 5 clustersize (2-256) | 1 |
| 2 | File 6 clustersize (2-256) | 1 |
| 2 | File 7 clustersize (2-256) | 1 |
| 2 | Data file 0 allocation | |
| 2 | Data file 1 allocation | |
| 2 | Data file 2 allocation | |
| 2 | Data file 3 allocation | |
| 2 | Data file 4 allocation | (No. groups per file, -65535) |
| 2 | Data file 5 allocation | |
| 2 | Data file 6 allocation | |
| 2 | Data file 7 allocation | |
| 18 | [proj,prog]filnam.ex | Not used in CTS-300 ISAM |
| 6 | Data file 1 device | |
| 6 | Data file 2 device | |
| 6 | Data file 3 device | No colon, right space fill |
| 6 | Data file 4 device | |
| 6 | Data file 5 device | |
| 6 | Data file 6 device | |
| 6 | Data file 7 device | |
| 2 | Total No. records (low order) | Valid only in index file |
| 2 | Total No. records (high order) | Valid only in index file |
| 2 | Multikey ISAM | Always 0 (no multikey for CTS-300) |

132 Total bytes

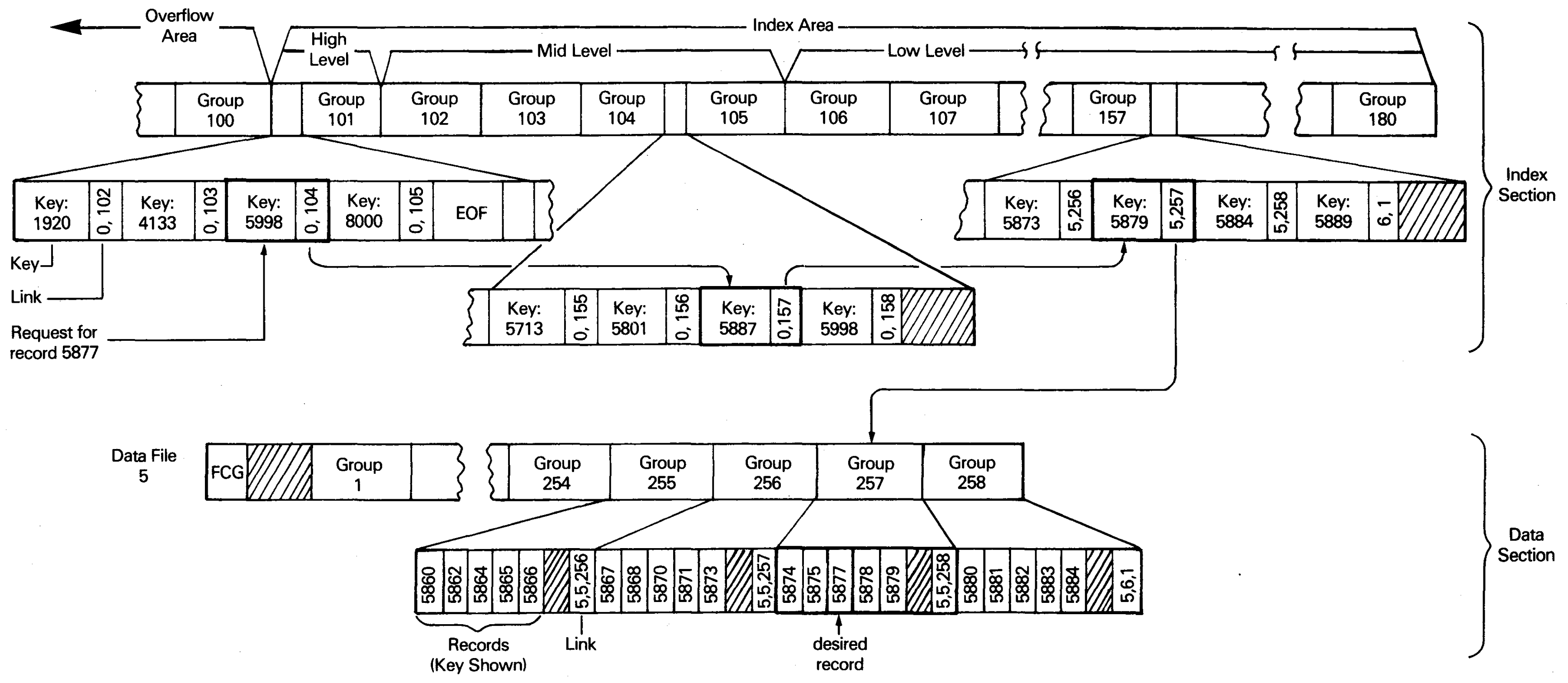


Figure 10-3 Index Area

### 10.3.2 Interrelationships and Tradeoffs

There is no file organization that is best for all applications, but there is one that is better suited to your application. This section discusses choices and decisions for each file component on an individual basis, starting with the key. Physical limits imposed by CTS-300 ISAM are also given. Some of these limits may seem impractically large and should certainly present no problem, but all limits must be observed.

#### 10.3.2.1 Key

Size: 1 to 100 characters (to change limits, see Section 10.3.2.3).

The key is an ordinary data field, defined within a DIBOL record statement, which contains alphabetic or numeric information.

CTS-300 ISAM can, optionally, accept duplicate keys. That is, more than one record with the same key may be stored. Within the CREATE dialog, you are asked whether the more recent record is to be stored before or after other such records with this key. It is the storage placement that determines the access order. If placed before, it will be accessed first with a READS (sequential read) statement; if after, it will be accessed last after all other records with that key.

If you elect to allow duplicate keys, it is wise to provide sufficient overflow area, because duplicate keys have the potential for unlimited overflow.

#### 10.3.2.2 Record

Size: 2 to 16,383 bytes (to change limit, see Section 10.3.2.3).

The most important consideration concerning record size is its relation to group size; therefore, record size cannot be selected without also considering group size. A major design goal for an ISAM file is to fit as many records as possible in a group with little or no wasted space. If the record can be designed so that a multiple will fit the chosen group size, the file will operate at maximum efficiency.

Too many records in a group could cause a performance degradation, because records are always searched sequentially after the group is accessed.

Too few records per group could limit load exclusion availability, and the addition of many records in any such group can result in heavy overflow usage.

### 10.3.2.3 Changing Record and Key Sizes

The standard record length buffer is 1500 characters, and the standard key length buffer is 100 characters. However, it is possible to create a file with a record or key length that exceeds these limits by changing the limits in the UTL2.DBL, RORG3.DBL, and CRET1.DBL subroutines used by the ISMUTL program. Insert the following decimal values as shown below:

```
@@@@ = NEW RECORD LENGTH
%%%% = NEW RECORD LENGTH MINUS 132
+++  = NEW KEY LENGTH

Using EDIT:

.R EDIT
*EBUTL2.DBL$$
*FBUFLN,$5J$4C@@@@$$
*FKEYLN,$5J$3C+++$$
*EX$$

.R EDIT
*EBRORG3.DBL$$
*FRECBUF$1A$FA$4C@@@@$$
*6A$FA$4C%%%%$$
*5A$FA$4C%%%%$$
*3A$FA$3C+++$$
*EX$$

.R EDIT
*EBCRET1.DBL$$
*FRECBUF$1A$FA$4C@@@@$$
*6A$FA$4C%%%%$$
*4A$FA$4C%%%%$$
*EX$$
```

After you have modified these routines, you must recompile and relink them as shown below for ISMUTL.SAV (SUD system):

```
.DIBOL/NOLINE UTL2,FCGFX,RORG1,RORG2
no-error response

.DIBOL/NOLINE RORG3,RORG4,STAT,CRET1
no-error response

.DIBOL/NOLINE CRET2,CRET3,NUMQ
no-error response

.LINK/PROMPT/EXE:ISMUTL.SAV UTL2,FCGFX,DATE,DIBOL
RORG1/O:1
RORG2/O:1
RORG3/O:1
RORG4/O:1
STAT/O:1
CRET1/O:1
CRET2,NUMQ/O:1
CRET3/O:1//
```

### 10.3.2.4 Group

Size: 132 to 16,383 bytes (additionally the size can not exceed 127 records).

The first consideration in determining group size is related to machine I/O. For efficiency, it is best if the ISAM group size is equal to the machine I/O block size. Data is normally read by the hardware in blocks of 512 bytes. Therefore, a group size smaller than the hardware block size would not use part of the data obtained with each read. If the group size is made larger, that is, if a group crosses a block boundary, the cost may be extra I/O for a given record access. Another consideration with groups that cross block boundaries is that it requires additional processing by the run-time system.

In TSD operation, when a record is opened in update mode, the record is locked. Its entire group, and all blocks associated with that group, are also locked and can not be accessed by another user.

However, I/O byte block size can be changed in whole multiples of 512 bytes with the DIBOL PROC(n) statement. When using the PROC statement you must remember that it affects all files opened on the system. Regardless of group size, however, the link area still requires only 4 bytes. Therefore, the record space available would be 508, 1020, 1532, etc. bytes for values of n (in PROC(n)) of 1, 2, 3, etc.

The second consideration is to design records to be placed in this group, or to make the group fit a multiple of existing records. If you are free to design the record, this fit can usually be achieved with little waste. On the other hand, if for example you are creating an ISAM file from an existing sequential file, this can not be done. In either case, you will likely have to make a decision whether or not to artificially fill any remaining space in the group. This is done at the expense of losing some storage space. However, it avoids the performance degradation caused by mismatching the group and block size. The amount of space wasted by filling has to be balanced on an individual basis by evaluating the increased access efficiency. Ideally, the amount of fill is small, and the decision to fill is then the logical one.

The number of load exclusion records chosen for the group has no effect on the total number of records. The purpose of load exclusion records is to reserve one or more open record spaces within each group. It is selected when you expect to add a random distribution of relatively few records, thus lowering the likelihood of group overflow.

### 10.3.2.5 Overflow Area

Size: 0 to 16,383 groups

Allocation of overflow area depends on the nature of expected file growth. Overflow is required when new records are likely to be distributed unevenly throughout the file. That is, where the likelihood is great that there will be many records that logically fall under one, or a few, keys. The problem with overflow, as mentioned before, is that access within the overflow area must be accomplished via links and sequential search which involves multiple reads. Within overflow, the advantages of ISAM are lost, except that storage is still sequential. Overflow should be used only as a temporary solution to accommodate a particular class of added records. As the overflow area becomes filled, the need to reorganize the file to increase access efficiency becomes greater. See Section 10.4.5. The manipulation of records by ISAM when overflow occurs is illustrated in Section 10.3.4.

### 10.3.2.6 Append Area

Size: 0 to number of records determined by system storage limitations

The append area, since it is indexed, does not have the drawbacks associated with the overflow area. Index entries are created for each append area group as it is filled. Efficient use of an append area is achieved whenever records are added in ascending order by key value. If a record with a high key value is prematurely stored, this could eventually cause records with lower key values to be placed in overflow.

If it is the nature of the application that new data records being added to an existing ISAM file will always or usually have a key with a greater value than those records already in the ISAM file, then a large append area would be required. In this case, little or no load exclusion area or overflow area would be necessary since we are not expecting new data records to have to be inserted between existing records.

Append area is assigned in whole groups, even though it is specified in records. Thus, if you had previously chosen four records per group for group size, and specified six records for append, the append area would be eight records (two groups) in size. Also, ISMUTL always appends a minimum of one group. So, even if you requested no append area, in this example there would be space for four records in the append area.

If no input file is specified when creating the ISAM file, the amount of space allocated in the append area becomes the entire data area of the file.

## 10.3.3 DIBOL Statements

ISAM file access and data manipulation is accomplished with both standard DIBOL statements and special ISAM DIBOL statements. All bookkeeping and updating involved are handled by the run-time system. A complete description of these statements is found in the DIBOL-11 **Language Reference Manual**. This section discusses their specific relationship to ISAM files.

**NOTE:** All media containing any part of the ISAM file must be on-line for any ISAM DIBOL statement operation.

The following are DIBOL statements used with ISAM files:

**OPEN**

This statement for an ISAM file is in the form:

```
              |SI|
    OPEN (ch, |  | ,filespec)
              |SU|
```

where:

| | |
|---|---|
| ch | is the channel to be opened. |
| SI | indicates input from an ISAM file; no change to the file is allowed, and no records are locked. The following statements may be used with a file opened in SI mode: READ, READS, and CLOSE. |
| SU | indicates update of an ISAM file. When specified in a TSD environment, the record accessed is locked (no other user may access the record). The ISAM group, of which the ISAM record is a part, is also locked, as is any block which is a part of the group. Any ISAM statement may be used with a file opened in SU mode. |
| filespec | is an alphanumeric literal, field, or record that contains the file specification of the file to be opened in the form dev:filnam.ext. |

- The ISAM file must be opened before any operation is performed.
- The two modes SI and SU are exclusively for ISAM use.
- The OPEN statement opens all volumes of an ISAM file on a single RT-11 channel.
- The following statements will unlock a record locked in update mode: UNLOCK, DELETE, WRITE, STORE, CLOSE, or a READ of a record in another group.

## READ

This statement for an ISAM file is in the form:

```
     READ (ch,record,keyfld)
```

where:

| | |
|---|---|
| ch | is the channel opened for the file. |
| record | is the destination of the retrieved data. |
| keyfld | is the ISAM key for the desired record. |

- The first record with a key match is the record returned.
- If there are duplicate keys, successive records must be accessed with the READS statement.
- If the key is not found, the record with the next greater key is returned, along with an error message indicating that the requested key was not found. If the key size is less than the size of the data file key, it is interpreted as a partial key. In this case the first record retrieved is the one whose initial characters match the characters of the specified key field. If the characters do not match, the record with the next greater partial key is returned along with the error message.
- If a READ of a locked record is attempted, the key remains intact.

## READS

This statement for an ISAM file is in the form:

```
     READS (ch,record[,label])
```

where:

| | |
|---|---|
| ch | is the channel opened for the file. |
| record | is the destination of the retrieved data. |
| label | is the label of the statement where control is transferred when the ISAM end-of-file is detected. |

- The READS statement is normally issued after a READ; in which case the next logical block is returned.
- If the statement first issued after an OPEN is a READS, the first logical record is returned.

- If a READS of a locked block is attempted, you will receive an error message indicating the locked block condition.

- A READS is accomplished using a key value of zero and there is no error message because of the failure to find a key match.

## WRITE

This statement for an ISAM file is in the form:

```
WRITE (ch,record,keyfld)
```

where:

| | |
|---|---|
| ch | is the channel opened for the file. |
| record | is the source of the output data. |
| keyfld | is the ISAM key for the desired record. |

A WRITE may be performed only if the record is the one most recently retrieved by a READ or READS. It is a read-modify-write operation.

## STORE

This is an ISAM statement of the form;

```
STORE (ch,record,keyfld)
```

where:

| | |
|---|---|
| ch | is the channel opened for the file. |
| record | is the record to be written. |
| keyfld | is the field containing the key of the record to be stored. |

- This statement places the record in the ISAM file according to its key value.

- If a record with the same key already exists, and duplicate keys are not allowed, an error is generated.

**NOTE:** Do not use the STATUS program (see STATUS utility) kill option or a CTRL/C to terminate a program doing STORES to an ISAM file opened in SU mode. Such a termination of the program (or the last program, if more than one) will prevent updating of the index file FCG. An attempt to REORG this file will then result in an error message indicating that more input records were found than were specified. See the error message table for further information.

**DELETE**

This is an ISAM statement of the form:

DELETE (ch,keyfld)

where:

| | |
|---|---|
| ch | is the channel opened for the file. |
| keyfld | is the field containing a value equal to the key of the record to be deleted.<br><br>The keyfield must contain a value equal to the key of the last record read. |

**UNLOCK**

This is an ISAM statement of the form:

UNLOCK (ch)

where:

| | |
|---|---|
| ch | is the channel opened for the file. |

The UNLOCK statement clears any existing lock condition on the specified channel.

CLOSE

The form is:

CLOSE (ch)

where:

| | |
|---|---|
| ch | is the channel opened for the file. |

The CLOSE statement terminates the use of a channel by closing the file and releasing the channel. Any record in the device data buffer is written to the file. The index file FCG is updated with the number of records current upon the last close (in a multiuser system) by any update user (SU).

## 10.3.4 Data Storage

As will be seen in Section 10.4 ISAM files may be created without any data as input; with a sequential file as input; or with another ISAM file as input. In this section, files will be shown as data is added to illustrate the characteristics of an ISAM file. Figure 10-4 shows an empty file to which records are added. Figure 10-5 shows an ISAM file created with a sequential file as input, followed by later addition of records.

**Records Added to an Empty ISAM File**

Consider a data file with group number "N" as shown in Figure 10-4. This file was created without any specified input. This is data file two in an ISAM file whose group size is 512 bytes, each of which contains five records of 100 bytes each. Each data group in this file contains, in addition to the 5 records, 8 bytes of unused space, because the fill option was chosen. Figure 10-4 consists of six parts used to illustrate changes in a data group as records are added and deleted. The parts, as identified in the figure, are:

a) Initially the group has no data. The only entry is an EOF in the first record position. The index entry corresponding to the data group also has an EOF as the key entry. The link shows one valid record (the EOF) and points to the next group in the same data file. No search of the data file (or the ISAM file) will proceed beyond this EOF record, since it is the greatest possible key value.

b) A record is added (with the STORE statement) which has a key value of 13. The EOF is now the second record in the group. The link shows two valid records, and the index entry remains unchanged.

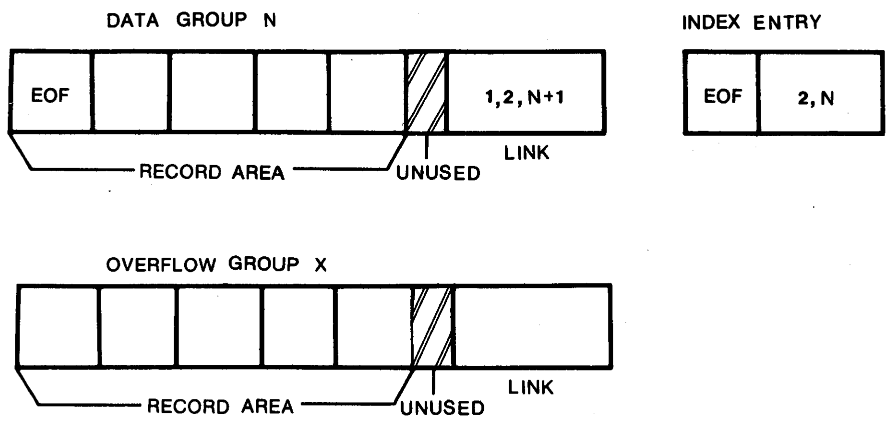
c) Records with key values of 15, 17, 18, and 19 are added. The records are stored in ascending order. The EOF record is replaced by the record with the key of 19. The key of 19 is now placed in the index for this group. No record with a key greater than 19 can hereafter be placed in this group number N. The link for the group indicates 5 valid records.

d) The addition of one more record (key of 16) now exceeds the capacity of the group and results in the split to the overflow area as shown. The index entry for all records in this group is still 19 but now access to records 17, 18, and 19 is achieved via the link to overflow. Data group N contains copies of records 18 and 19, but the link identifies only three valid records in this group. The overflow group link points back to the next logical group (N+1) in the data file.

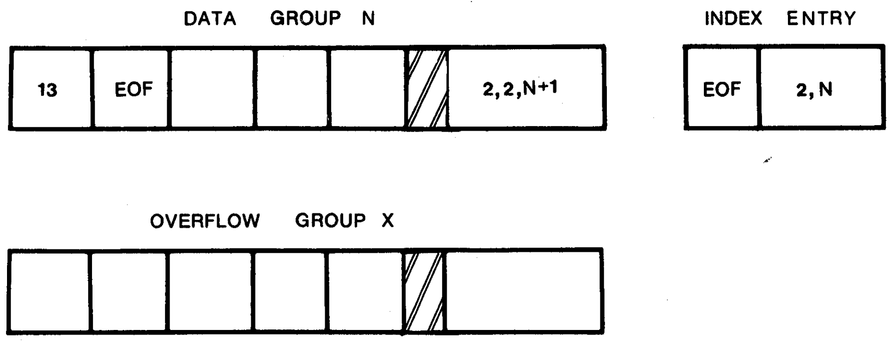
e) If record 15 were now deleted, the group would be as shown. The group N link is unchanged except there are now only the two valid records indicated. Note there are now two copies of record 18 and three of record 19 shown. Only the single entries in the overflow group are accessible.

   Note that the figure is not to any scale and the information shown consists only of key values, EOF entries, and link information. The link information is: number of valid records in the group, number of the next data file, and number of the next logical record.

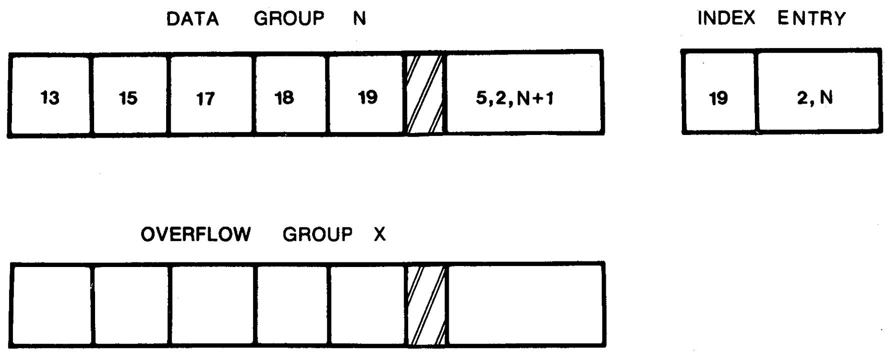
a) no data



b) one record (key 13)



c) four records (keys 15, 17, 18, and 19 added) index entry of 19



d) record with key of 16 added

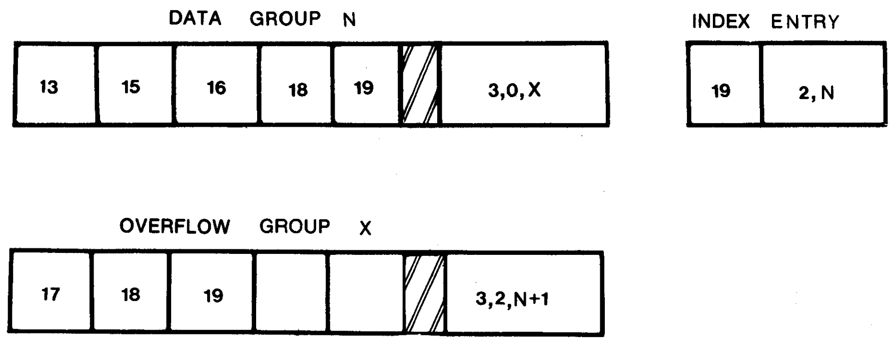
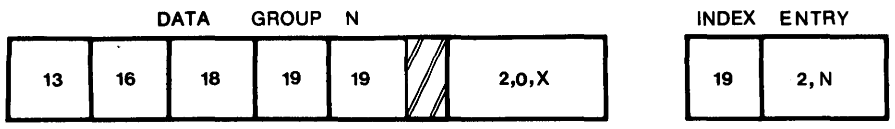


Figure 10-4 Data Storage (File Initially Empty)

e) record with key of 15 deleted



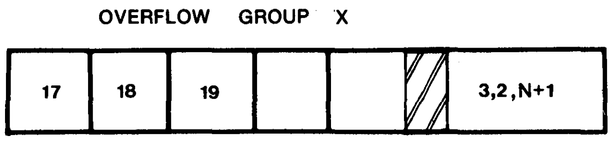


**Figure 10-4 Data Storage (File Initially Empty) (Cont.)**

## Records Input from a Sequential File during CREATE

When an ISAM file is created with a sequential file specified as the initial input, the user has the option of selecting the load exclusion option. This option is illustrated here in Figure 10-5. This is data file number 3 in an ISAM file whose group size, as in Figure 10-4, is 512 bytes, each of which contains 5 records of 100 bytes each. Each data group in this file contains, in addition to the 5 records, 8 bytes of unused space, because the fill option was chosen. A load exclusion factor of 2 record spaces has been chosen. The parts of this figure, as identified, are:

a) When this ISAM file is created, the data from the sequential input file is immediately entered. In this example this group happens to receive records with key values of 13, 17, and 20. Records with greater key values go into the next group because the load exclusion of two records prevents further entries during the creation phase. The key of 20 is placed in the index entry for the group. No record with a key greater than 20 can hereafter be placed in this group N. The link for the group indicates three valid records.

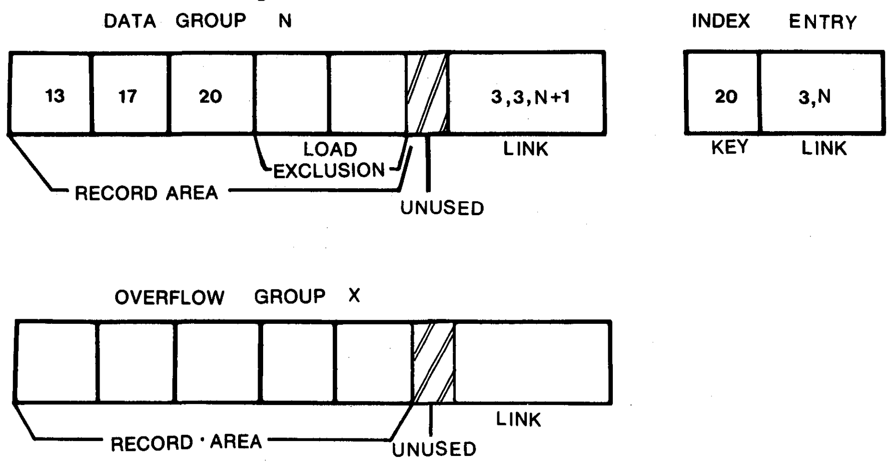
b) At some time after the file is created and all the sequential file input data is stored, records are added (one at a time, with the STORE statement) which have key values of 15 and 18. The link now shows 5 valid records, and the index entry remains unchanged.

c) A record with a key value of 14 is now added. The number of records now exceeds the capacity of the group and a split to overflow occurs, as with the example in Figure 10-4.

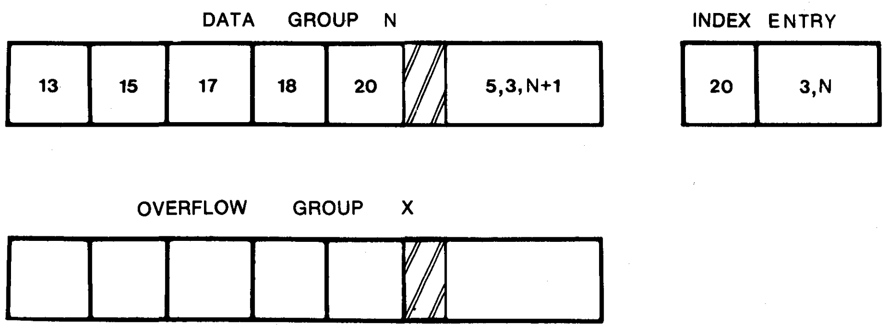
d) The record with key value 13 is now deleted. The only change is a reduction in the number of valid records in group N and a shift of the other records to the left. This creates a total of 3 copies of the record with key value 20. Only the one in overflow is accessible.

Note that this figure, like Figure 10-4, is not to any scale and the information shown consists only of key values, EOF entries, and link information.

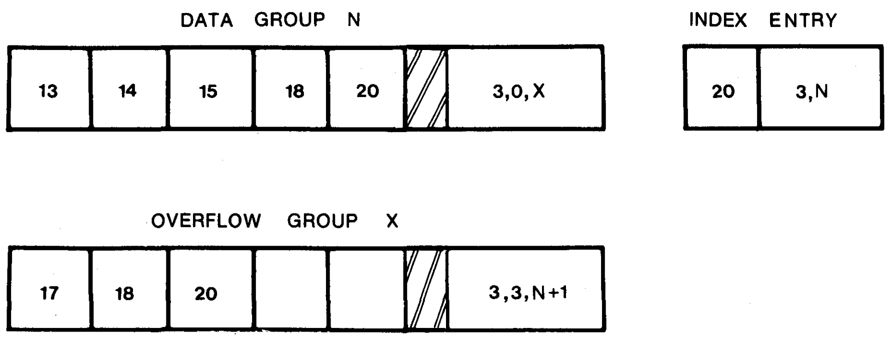
a) data from sequential file as input



b) records 15 and 18 added



c) record with key of 14 added causing overflow



d) record with key of 13 deleted

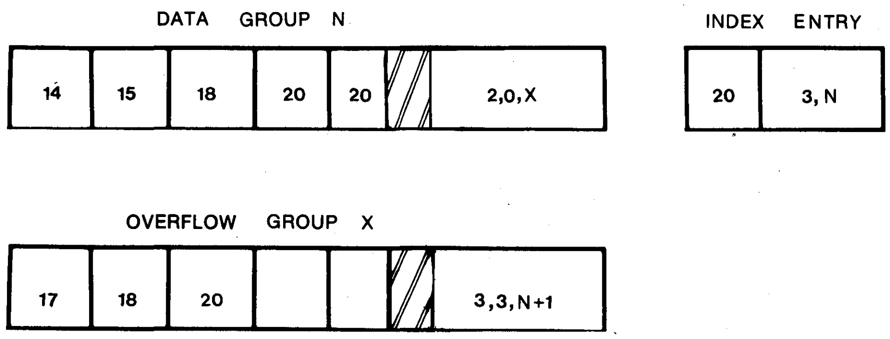


Figure 10-5 Data Storage (Sequential File as Input)

## 10.4 USING ISMUTL

ISMUTL is the DIBOL utility that allows you to select the parameters to define, build, and support the desired ISAM file. ISMUTL performs four selectable operations:

- An ISAM file is constructed by using the create (CREATE) feature of ISMUTL.
- The status (STATUS) feature of ISMUTL allows the operator to check file structure, file status, utilization of overflow, and (in chain mode) append areas. When one or both of the latter become full, or nearly so, it is usually necessary to rebuild your ISAM file. This may be done with either the create or the reorganization feature of ISMUTL.
- Assuming the basic parameters are still valid, the reorganization (REORG) feature of ISMUTL can automatically create a new ISAM file which restores file expansion areas to their original values and consolidates all the data. The result is then a larger (or perhaps smaller) addressable (indexed) area.
- The exit (EXIT) feature terminates ISMUTL and returns control to the run-time system (TSD or XMTSD) or the operating system.

### 10.4.1 ISMUTL Requirements

Before you actually use any of the features of ISMUTL, there are several requirements and characteristics of the utility that must be understood.

#### 10.4.1.1 SUD Operation

In order to execute a DIBOL program that contains ISAM access statements, ISAM must first be selected during CTSGEN.

#### 10.4.1.2 TSD Operation

TSD operation with ISAM and ISMUTL requires special attention during CTSGEN. The CTSGEN must include ISAM, and a specific number of channels must be specified in CTSGEN, depending on the number of logical volumes to be assigned in ISMUTL CREATE. It is also possible to add a data file during REORG, and this would affect channel requirements.

Twelve is the maximum number of channels required if you are creating or reorganizing an ISAM file of seven data volumes. TSD channel requirements, as a function of data volumes, are listed below:

| NUMBER OF LOGICAL DATA VOLUMES | NUMBER OF CHANNELS REQUIRED |
|---|---|
| 1 | 6 |
| 2 | 7 |
| 3 | 8 |
| 4 | 9 |
| 5 | 10 |
| 6 | 11 |
| 7 | 12 |

NOTE

Under TSD, no other user should access the ISAM file while it is being created or reorganized.

### 10.4.2 Running ISMUTL

Assuming proper linking and a CTSGEN for ISAM, the command for file ISMUTL.SAV (SUD) is then:

```
.R ISMUTL
```

or, assuming proper linking, a CTSGEN for ISAM, and sufficient channels, the command for file ISMUTL.TSD (TSD) is:

```
*R ISMUTL
```

The response is:

```
CTS300 ISAM UTILITY PROGRAM, Vnn-00n
SELECT FUNCTION (CREATE,STATUS,REORGANIZE, OR EXIT):
```

Your response to this selection message is the first letter of the chosen function. But first see the appropriate section below for the operating details of that function.

### 10.4.3 Creating a File (CREATE)

The previous sections of this chapter have covered the ISAM file and its structure. This section contains a discussion of the special characteristics of CREATE and details of the CREATE process.

### 10.4.3.1 Special CREATE Characteristics

#### Input File

Input for ISAM file creation may be a sequential file; another ISAM file; or an ISAM file may be created without an input file. If the input is a sequential file, there is a special requirement: the file must first be sorted in ascending order by the same key that will become the ISAM key.

After naming the input file, you will be asked if you want to delete the input file after CREATE. If you chose to delete the input file, CREATE opens two checkpoint files which, in an alternating manner, keep track of the cleanup process associated with the deletion of a same-name file.

If a machine malfunction occurs during CREATE with the input file deleted, the checkpoint files allow resumption at the correct point. See Section 10.4.3.5 for details on restarting the process.

**NOTE:** If you do decide to delete your input file, you may want to make a back-up copy to eliminate loss of data due to a hardware malfunction.

**NOTE**

Certain ASCII characters must be avoided in ISAM files. These characters (listed below) interact with intermediate work files during ISMUTL CREATE or later REORG. Any of these characters will cause truncation of the record in which it is found.

| Character | Decimal Code |
|---|---|
| Null | 0 |
| HT | 9 |
| LF | 10 |
| VT | 11 |
| FF | 12 |
| CR | 13 |
| ^Z | 26 |
| Esc | 27 |

#### Output File

You must name the resulting ISAM file at the beginning of the CREATE dialog. Allocation of the index and data files to the system resources (devices) takes place at the end of the dialog.

The output file name can be any six-character file name. The default extension is .ISM, which is assigned to the index file. The data files are assigned default extensions of .IS1 through .IS7. If an extension is specified, it must be at least two characters long; if it is three characters, the last character can not include the numbers 1 through 7. These numbers are assigned by ISMUTL and will override any third character position. The extension .ROG can not be used either, since that is a temporary extension used by the system when the file is reorganized. If the output file name is the same as the input file name, temporary files, .TMP and .TM1 through .TM7, are built by CREATE and there should be no other files of that name on the system.

Device identification takes place at the end of the CREATE dialog when files are allocated. There must be enough contiguous space on each device to open its data file and, for one device, to open the index file as well. If any difficulty arises, see Section 10.4.3.5.

Ordinarily volumes containing data files must always be mounted for use on the same devices and units as those on which they were created. However, by using the RT-11 ASSIGN command for device names, you are not restricted to the same physical devices.

Total File Requirements

In addition to files already mentioned, in all CREATE operations there are also two work files that are opened to build the output index file. CREATE will try to find space for the work files on the device you specify for your output files. If enough space is not found, CREATE will ask for additional devices. These are temporary files that will be deleted automatically after CREATE has built the index file.

Below is a summary of the files that will be automatically created, depending on deletion and same-name choices:

If you wish to delete the input file and both input and output files have the same name:

```
  filnam.CK1  (checkpoint file on system device)
  filnam.CK2  (checkpoint file on system device)
  filnam.TMP  (temporary file)
  filnam.TM1 to .TM7 (depending on the number of input data files)
  filnam.ROG  (temporary REORG file)
  filnam.R01 to .R07 (depending on the number of input data files)
  filnam.WK0  (work file)
  filnam.WK1  (work file)
```

If you wish only to delete the input file:

```
  filnam.CK1  (checkpoint file on system device)
  filnam.CK2  (checkpoint file on system device)
  filnam.WK0  (work file)
  filnam.WK1  (work file)
```

If you do not wish to delete the input file:

```
     filnam.WK0 (work file)
     filnam.WK1 (work file)
```

**Auto-Create**

The CREATE function includes the optional capability of operating from an external data file of input responses. This is a special function for experienced users and should not, in any case, be invoked to create a file with unproven parameters. For this reason, a complete discussion of this capability is provided under Section 10.4.7.2, rather than here.

### 10.4.3.2 Design/CREATE Process

Creating an ISAM file is a five-step process. The first four of these steps are preliminary to running CREATE:

1. Determine the key/record/group/block size relationship.
2. Determine record addition characteristics for load exclusion, overflow, and append area requirements.
3. Determine the output names.
4. To satisfy requirements which apply to your proposed ISAM file, determine the following in advance of running CREATE:

    - Whether to do an auto-create
    - Input file (name, ext .DDF assumed)
    - Whether to delete your input file (ISAM or sequential only)
    - Output file name
    - Approximate number of input records (sequential only)
    - Record length (only if no input file)
    - Number of records per group
    - Load exclusion value
    - Whether to fill out the group to end of block boundary
    - Append area size
    - Overflow area size

    The following two factors are automatically supplied if the input file is an ISAM file:

    - Key length
    - Key location

- Whether to allow duplicate keys, and, if so, whether to place at beginning or end (if ISAM input file, question appears only if not already allowed)

- Output allocation:

  Index File
  Data Files

  The device is specified without the colon.

5. Select CREATE by typing C in response to the ISMUTL selection message and enter the values you have just determined. The following section will help you during the CREATE process.

### 10.4.3.3 CREATE Dialog

This section contains the actual text for the create process. All create questions are included here, even though no given create could include every question. Allowable ranges are shown as a final check on your answer. Figure 10-6 illustrates the dialog flow for the possible input file types and subsequent choices.

The following is the CREATE dialog:

```
DO YOU WANT TO DO AN AUTO-CREATE?  (YES/NO)
```

Answer NO, or carriage return, unless you are familiar with the auto-create process. The special procedures for this function are detailed in Section 10.4.7.2.

```
INPUT FILE (DEV:NAME.EXT):
```

If not specified, assumed device is the default device and assumed extension is .DDF. A carriage return implies no input file, in which case the next question does not appear.

```
DELETE INPUT FILE AFTER CREATE (YES/NO):
```

If yes, your input file will be automatically deleted after the output file is created.

```
OUTPUT FILE (NAME.EXT, DEFAULT EXT=ISM):
```

Just enter the file name; device names are supplied later.

```
APPROXIMATE NUMBER OF INPUT RECORDS (0-16,777,215):
```

Appears only if sequential input is specified. You can specify up to the system limit. Unneeded space is not used. Your answer determines the size of the work files.

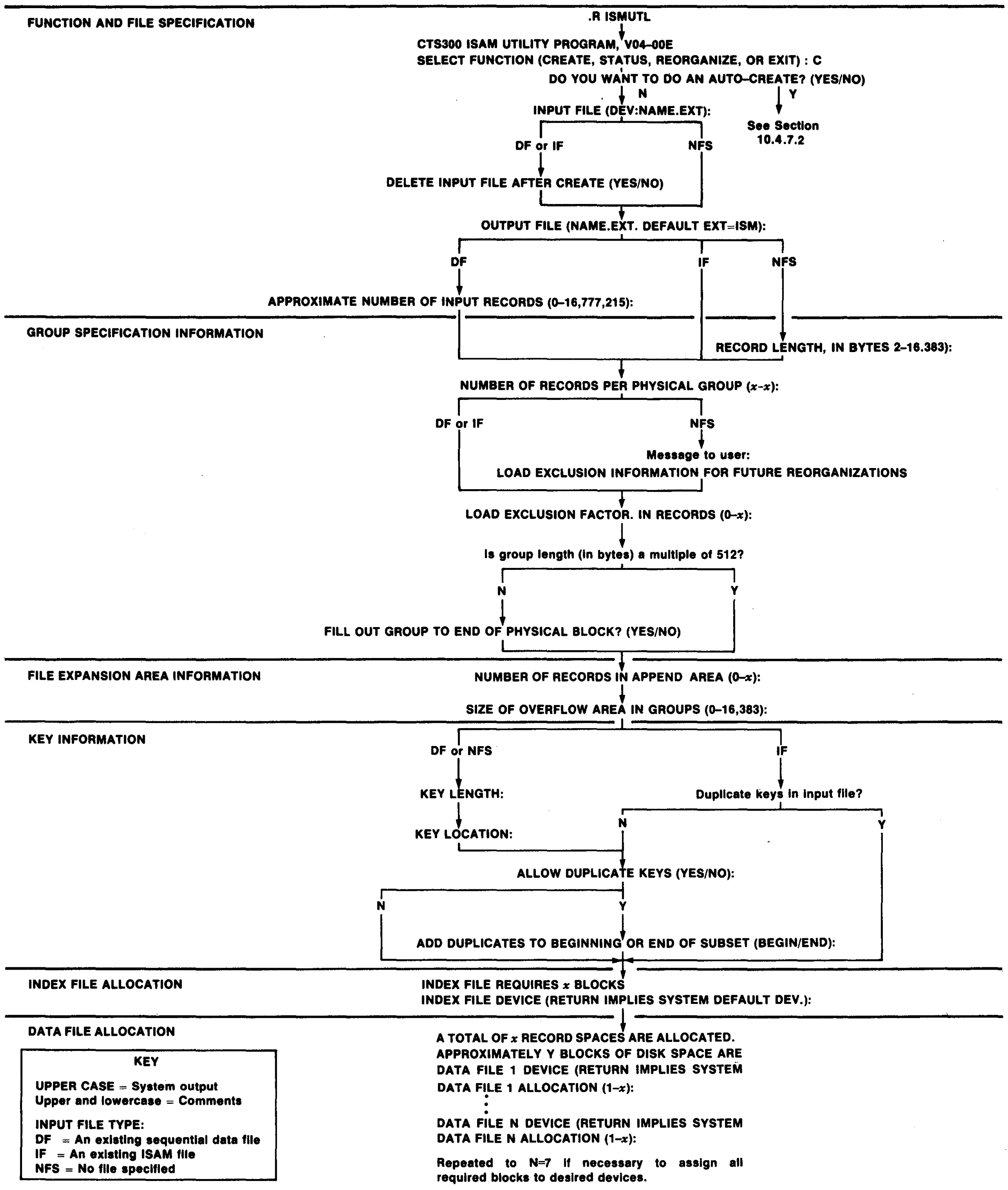

FUNCTION AND FILE SPECIFICATION
.R ISMUTL
CTS300 ISAM UTILITY PROGRAM, V04-00E
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT) : C
DO YOU WANT TO DO AN AUTO-CREATE? (YES/NO)
N
Y
See Section 10.4.7.2
INPUT FILE (DEV:NAME.EXT):
DF or IF
NFS
DELETE INPUT FILE AFTER CREATE (YES/NO)
OUTPUT FILE (NAME.EXT. DEFAULT EXT=ISM):
DF
IF
NFS
APPROXIMATE NUMBER OF INPUT RECORDS (0-16,777,215):
GROUP SPECIFICATION INFORMATION
RECORD LENGTH, IN BYTES 2-16.383):
NUMBER OF RECORDS PER PHYSICAL GROUP (x-x):
DF or IF
NFS
Message to user:
LOAD EXCLUSION INFORMATION FOR FUTURE REORGANIZATIONS
LOAD EXCLUSION FACTOR. IN RECORDS (0-x):
Is group length (in bytes) a multiple of 512?
N
Y
FILL OUT GROUP TO END OF PHYSICAL BLOCK? (YES/NO)
FILE EXPANSION AREA INFORMATION
NUMBER OF RECORDS IN APPEND AREA (0-x):
SIZE OF OVERFLOW AREA IN GROUPS (0-16,383):
KEY INFORMATION
DF or NFS
IF
KEY LENGTH:
Duplicate keys in input file?
N
Y
KEY LOCATION:
ALLOW DUPLICATE KEYS (YES/NO):
N
Y
ADD DUPLICATES TO BEGINNING OR END OF SUBSET (BEGIN/END):
INDEX FILE ALLOCATION
INDEX FILE REQUIRES x BLOCKS
INDEX FILE DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEV.):
DATA FILE ALLOCATION
A TOTAL OF x RECORD SPACES ARE ALLOCATED.
APPROXIMATELY Y BLOCKS OF DISK SPACE ARE
DATA FILE 1 DEVICE (RETURN IMPLIES SYSTEM
DATA FILE 1 ALLOCATION (1-x):
DATA FILE N DEVICE (RETURN IMPLIES SYSTEM
DATA FILE N ALLOCATION (1-x):
Repeated to N=7 if necessary to assign all required blocks to desired devices.
KEY
UPPER CASE = System output
Upper and lowercase = Comments
INPUT FILE TYPE:
DF = An existing sequential data file
IF = An existing ISAM file
NFS = No file specified

Figure 10-6 ISAM CREATE Flowchart

RECORD LENGTH, IN BYTES (2-16,383):

Appears only if no input is specified.

NUMBER OF RECORDS PER PHYSICAL GROUP (x-x):

The range is determined by a minimum group size of 132 bytes and by a maximum group size of 127 records or 16,383 bytes, whichever is larger.

LOAD EXCLUSION INFORMATION FOR FUTURE REORGANIZATION

Appears only if there is no specified input file. Since there is no initial data, there can be no exclusion. However, when the file is reorganized, the amount of exclusion selected here will be provided.

LOAD EXCLUSION FACTOR, IN RECORDS (0-x):

The maximum number of records that can be allocated for exclusion is always one less than the total number of records in a group. If there were no input file specified, this message would be preceded by the following:

FILL OUT GROUP TO END OF PHYSICAL BLOCK? (YES/NO)

This answer depends on the record/group/block size relationships and is a matter of judgement. You trade efficiency for access speed.

NUMBER OF RECORDS IN APPEND AREA (0-x):

The append area is for the addition of records with key values greater than the initial input. Although the low range of your choice is labeled 0, you actually get at least one group. The maximum number is limited by system storage capacity.

SIZE OF OVERFLOW AREA IN GROUPS (0-16,383):

The overflow area is for addition of records of intermediate key value to groups that may eventually be filled.

KEY LENGTH

Key length is the length of the key in bytes. It appears only if your input is not an ISAM file.

KEY LOCATION

Key location is the beginning byte position of the key within the record. The first byte position of a record is number one. It appears only if your input is not an ISAM file.

ALLOW DUPLICATE KEYS (YES/NO):

Do you wish to allow more than one record with the same key value? It appears only if your input is not an ISAM file, or if your input ISAM file did not allow duplicate keys.

ADD DUPLICATE KEYS TO BEGINNING OR END OF SUBSET (BEGIN/END):

To place added records before or after existing records with the same key, respond with B or E. Storage at the beginning means that the last record will be stored at the beginning of the duplicate set and will be the first record accessed with a READ. Storage at the end means that the last record will be stored at the end of the duplicate set and will be the last record accessed with a READS. This question is not asked if duplicates are not allowed.

```
INDEX FILE ALLOCATION.
INDEX FILE REQUIRES x BLOCKS.
INDEX FILE DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEV.):
```

The program has calculated the space requirement for the index file. The x indicates the blocks necessary to create the index file. Respond with the device you want without the colon. A carriage return indicates the default device. The entire index file must be on a single device.

```
DATA FILE ALLOCATION.
A TOTAL OF x RECORD SPACES ARE ALLOCATED.
APPROXIMATELY y BLOCKS OF DISK SPACE ARE REQUIRED.
DATA FILE 1 DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEVICE):
```

At this point you must finalize your decision on how many files (logical volumes) the ISAM file will have and upon what device(s) they will be placed. The values of x and y are approximations, but they are close enough to indicate space requirements. With a small file, it may be possible to place the entire data file on the same device as the index file. Select the device, without the colon, for the first data file.

```
DATA FILE 1 ALLOCATION (1-x):
```

Assign the blocks to the first device. The range is from one to the maximum required. Select a value in keeping with the disk capacity. If a choice is lower than the upper limit, unassigned blocks will have to be assigned to other data volumes. The above sequence of assignment of device and blocks will continue as follows:

**NOTE**

A choice higher than, or equal to, the upper limit will default to the upper limit, thereby terminating the dialog.

```
DATA FILE 2 (through 7) DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEVICE):
```

Assign device two (through seven).

```
DATA FILE 2 ALLOCATION (1-n):
```

Assign the blocks to the second (through seventh) device. There will be a diminishing number of blocks to be assigned, as indicated by the value of n.

If the end of the sequence is reached but not all the blocks have been assigned, the process is repeated, starting at the beginning of the data file assignment sequence.

It is recommended that you keep an ISAM file on as few physical and logical volumes as possible because nothing is gained unless you have a fragmented disk due to bad blocks.

If any difficulty is encountered, see Section 10.4.3.5.

#### 10.4.3.4 CREATE Example

The following is an example of the dialog for the creation of an ISAM file from a sequential input file. The input file is known to consist of 72-byte records with the key of eight bytes located at the beginning of the file. Seven records per group will be chosen to allow a group to exist within a 512 byte block. Since this leaves little wasted space, fill-out to the end of the block is chosen. There are somewhat less than 250 records in the input file.

New records to be stored are expected to be scattered throughout the file and, in addition, there will be new records whose key values are greater than any presently existing in the input file. A load exclusion factor of two is chosen to handle the initial scattered records of intermediate key value. There are most likely less than 700 new intermediate records to be added that cannot be accommodated by load exclusion; however, an overflow area of 100 groups is chosen. There are estimated to be somewhat less than 200 records that will be added to this file that have greater than present key values. A value of 200 records is therefore chosen for the append area size. Duplicate keys will not be allowed.

After file allocation is complete, ISMUTL automatically gives you the status of the new file. Notice the values. Later, in Section 10.4.4.1, STATUS Example, this same file with added records will be shown.

The CREATE example is on the next page.

The CREATE Example:

```
.R ISMUTL

CTS300 ISAM UTILITY PROGRAM, V06-00

SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT):  C
DO YOU WISH TO DO AN AUTO - CREATE? N
INPUT FILE (DEV:NAME.EXT):  XDATA1.DDF

DELETE INPUT FILE AFTER CREATE (YES/NO): N
OUTPUT FILE (NAME.EXT, DEFAULT EXT=ISM):  XDATA1.ISM
APPROXIMATE NUMBER OF INPUT RECORDS (0-16,777,215):  250

GROUP SPECIFICATION INFORMATION.
        NUMBER OF RECORDS PER PHYSICAL GROUP (2-127):   7
        LOAD EXCLUSION FACTOR, IN RECORDS (0-6):   2
        FILL OUT GROUP TO END OF PHYSICAL BLOCK? (YES/NO)  Y

FILE EXPANSION AREA INFORMATION.
        NUMBER OF RECORDS IN APPEND AREA (0-16,776,863):   200
        SIZE OF OVERFLOW AREA IN GROUPS (0-16383):  100

KEY INFORMATION.
        KEY LENGTH:  8
        KEY LOCATION:  1
        ALLOW DUPLICATE KEYS (YES/NO):  N

INDEX FILE ALLOCATION.
        INDEX FILE REQUIRES 107 BLOCKS.
        INDEX FILE DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEV.)

DATA FILE ALLOCATION.
        A TOTAL OF 553 RECORD SPACES ARE ALLOCATED.
        APPROXIMATELY 80 BLOCKS OF DISK SPACE ARE REQUIRED.
        DATA FILE 1 DEVICE (RETURN IMPLIES SYSTEM DEFAULT DEVICE)
        DATA FILE 1 ALLOCATION (2-80):  80
14-APR-80                           FILE LOCATION    GROUPS ALLOCATED
KEY LENGTH IS- 8                    INDEX FILE    DK             104
RECORD LENGTH IS- 72                DATA FILE #1  DK              78
LEVELS OF INDEXING IS- 1
LOAD EXCLUSION FACTOR- 2
KEY STARTS AT LOCATION- 1
DUPLICATE KEYS ALLOWED- NONE
CURRENT NUMBER OF RECORDS- 238
GROUPS ALLOCATED TO OVERFLOW- 100
GROUPS REMAINING IN OVERFLOW- 100
NUMBER OF RECORDS PER GROUP- 7      GROUP LENGTH IS- 512
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT):  E

END ISAM UTILITY.
```

Note that the ISAM file created in the example could become an inefficient file because such a large overflow area was chosen. Because of this large overflow area, the index file is larger than the one data file.

### 10.4.3.5 Handling CREATE Problems

**Failure During Clean-up**

If a machine malfunction occurs during the cleanup routine of a CREATE with input file deleted, it is possible to complete the operation. To do so, you must have at least one of the checkpoint files on the system device. Start the CREATE again and specify the same file previously named in response to the input file question. The program searches for a file with the CK1 or CK2 extension and, when found, restarts and completes the function automatically.

NOTE: Remember that if CREATE does not terminate normally, do not delete temporary files, work files, or checkpoint files. Doing so may result in the loss of your input ISAM file.

**More Space Required for Output Files**

If, at the end of the CREATE dialog, you have failed to allocate all the space required by the ISAM file, you will receive a message:

```
ALL FILES ALLOCATED AND MORE
SPACE IS REQUIRED.  PLEASE TRY AGAIN.
```

The allocation sequence starts again, and you must allocate all of the required file space before you reach the end.

**Insufficient Work File Space**

It is possible that there is not enough space for the CREATE function to open all the files necessary to operate. If this is the case, the following message will appear:

```
WORK FILES REQUIRE x BLOCKS OF DISK SPACE
WORK FILE DEVICE:
```

Respond with a device name.

## 10.4.4 Determining the Status of an ISAM File (STATUS)

### 10.4.4.1 STATUS Selection and Characteristics

To determine the status of an ISAM file, type S in response to the ISMUTL selection message. The program responds with a description of group structure, of current number of records, and of how much overflow area has been used. Status selected in this manner does not provide any information on the append area space which remains. To obtain information on the free append space, see Section 10.4.7.1.

### 10.4.4.2 STATUS Example

The ISAM file created in the example in Section 10.4.3.4 is used to illustrate the changed STATUS information after the addition of some records.

The number of records has increased by 110 records to a total of 348 records. There are now 12 overflow groups in use for a maximum of 84 records in overflow (12 x 7 records/group); therefore, the remaining added records are probably accounted for by load exclusion record spaces becoming filled and by records going into the append area.

This same file is reorganized in Section 10.4.5.3.

The STATUS Example:

```
.R ISMUTL

CTS300 ISAM UTILITY PROGRAM, V06-00

SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT):  S
INPUT FILE (DEV:NAME.EXT):  XDATA1.ISM
14-APR-80                              FILE LOCATION     GROUPS ALLOCATED
KEY LENGTH IS- 8                       INDEX FILE    DK              104
RECORD LENGTH IS- 72                   DATA FILE #1  DK               78
LEVELS OF INDEXING IS- 1
LOAD EXCLUSION FACTOR- 2
KEY STARTS AT LOCATION- 1
DUPLICATE KEYS ALLOWED- NONE
CURRENT NUMBER OF RECORDS- 348
GROUPS ALLOCATED TO OVERFLOW- 100
GROUPS REMAINING IN OVERFLOW- 88
NUMBER OF RECORDS PER GROUP- 7         GROUP LENGTH IS- 512
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT):  E

END ISAM UTILITY.
```

### 10.4.5 Reorganizing a File (REORG)

As you add and delete records in an ISAM file, the sequential order and index are maintained, except, of course, there are no index entries for overflow records. When load exclusion, append, or overflow areas become filled, or performance deteriorates, you will want to restore these areas and place all records into indexed data groups. There are two ways to reorganize an ISAM file. You can rerun CREATE or run REORG. If you do not wish to change the original specifications, REORG is the logical choice. REORG allows continued growth in the manner originally specified, and it does not permit you to change any of these specifications; however, if the resulting file is larger, you must specify how to allocate more space. Except for dialog, the process is the same as a CREATE.

#### 10.4.5.1 Special REORG Characteristics

**Input File**

The input for a REORG is your present ISAM file. This file will be deleted, and the new file will be created.

**Output File**

The output file has the same name as the input file. There are temporary extensions assigned to the output files of .ROG and .R01 through .R07.

Note that the volumes containing data files must always be mounted for use on the same devices and units as those on which they were created (by CREATE or REORG). By using the RT-11 ASSIGN command for device names, you are not restricted to the same physical devices.

**File Growth**

The amount of growth is determined by a comparison of the number of records in the present file with the number of records in the file just after the last REORG or CREATE. Since the overflow area size does not change, the index expands or decreases by the space required for key entries. Since append and load exclusion are restored to their original values, the data area increases by a combination of records added from overflow, append, and load exclusion; or decreases as a result of records deleted.

**Total File Requirements**

Because the input file is deleted and because the output file has the same name as the input, the REORG function is similar to the CREATE function with deleted input and same input/output name. The files created are the same as the CREATE function under these conditions, namely:

```
filnam.CK1  (checkpoint file on system device)
filnam.CK2  (checkpoint file on system device)
filnam.TMP  (temporary file)
filnam.TM1 through .TM7
            (depending on the number of input data files)
filnam.ROG  (temporary REORG file)
filnam.R01 through .R07
            (depending on the number of input data files)
filnam.WK0  (work file)
filnam.WK1  (work file)
```

Until enough space is found to open all the required output files, the actual reorganization cannot start. After the output files are built, REORG enters the cleanup routine. The cleanup routine:

1. Writes a dummy checkpoint file.
2. Renames input files to .TMP and .TM1 through .TM7.
3. Renames output files to correct extensions.
4. Rewrites the output FCGs.
5. Deletes input files.
6. Checks to see if all output files are on the correct drives. (If yes, go to step 8.)
7. Moves files, if possible, to the correct device. After every successful move, it rewrites the FCG. If there is not enough room on a device, it terminates the REORG and tells you which device did not have enough room.
8. Rewrites a correct and complete FCG and gives you a status.

If REORG fails during this process, see Section 10.4.5.4.

### 10.4.5.2 REORG Process and Dialog

To run REORG, type R in response to the ISMUTL selection message. The program asks you for the name of the ISAM input file, and then proceeds with the reorganization. During the process, you may be asked to provide more work file space, or be notified of a problem during the cleanup procedure associated with the deletion of the input file. If any of these occur, see Section 10.4.5.4. The last response you will have to make is in regard to allocation of space for the added records in the indexed data files. The following is the text that will appear for a reorganization:

```
INPUT FILE (DEV:NAME.EXT):
```

This is your ISAM file to be reorganized.

```
APPROXIMATELY XX ADDITIONAL BLOCKS OF DISK SPACE REQUIRED
A=ADD EQUALLY TO EACH DATA FILE - -
B=ADD TO LAST DATA FILE
C=ADD EXTRA DATA FILE
PLEASE RESPOND WITH A,B OR C:
```

If you respond with C, the following additional question is asked:

```
ASSIGN DATA FILE DEVICE:
```

Respond with a device to accommodate the added file requirements.

### 10.4.5.3 REORG Example

The ISAM file illustrated in Sections 10.4.3.4 and 10.4.4.1 (CREATE and STATUS examples), is used to show the effect doing a reorganization has on a file with records in overflow, load exclusion, and the append area.

The reorganization dialog consists essentially of specifying the input file and of allocating additional storage space. The automatic status response shows the file after the reorganization.

The new file has the same 348 records as indicated in Section 10.4.4.1; but now the overflow area has been restored to a full 100 groups. Although not shown, the load exclusion record spaces and append area have also been restored. All the records are now in indexed data groups. Consequently the data file has increased by 22 groups and the index file by one group.

The REORG Example:

```
.R ISMUTL

CTS300 ISAM UTILITY PROGRAM, V06-00

SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT):  R
INPUT FILE (DEV:NAME.EXT):  XDATA1.ISM

APPROXIMATELY 23 ADDITIONAL BLOCKS OF DISK SPACE ARE REQUIRED
A=ADD EQUALLY TO EACH DATA FILE - - B=ADD TO LAST DATA FILE
C=ADD EXTRA DATA FILE
PLEASE RESPOND WITH A,B OR C:          A
14-APR-80                           FILE LOCATION    GROUPS ALLOCATED
KEY LENGTH IS- 8                    INDEX FILE     DK             105
RECORD LENGTH IS- 72                DATA FILE #1   DK             100
LEVELS OF INDEXING IS- 1
LOAD EXCLUSION FACTOR- 2
KEY STARTS AT LOCATION- 1
DUPLICATE KEYS ALLOWED- NONE
CURRENT NUMBER OF RECORDS- 348
GROUPS ALLOCATED TO OVERFLOW- 100
GROUPS REMAINING IN OVERFLOW- 100
NUMBER OF RECORDS PER GROUP- 7      GROUP LENGTH IS- 512
SELECT FUNCTION (CREATE, STATUS, REORGANIZE, OR EXIT):  E

END ISAM UTILITY.
```

### 10.4.5.4 Handling REORG Problems

#### Failure During Cleanup

If a machine malfunction or a forced termination occurs during REORG, it is still possible to complete the operation. Do not delete any temporary files, work files, or checkpoint files and just start the REORG again and specify the input file name. The program searches for a file with the CK1 or CK2 extension and, when found, restarts and completes the function.

#### Insufficient Space to Open Output Files

There are three ways you can deal with this situation. The method you use depends on your file and your system. The possible solutions are:

1. Eliminate all unnecessary files, squeeze the disk(s), and retry ISMUTL. This process makes use of the checkpoint files.

2. Eliminate all the unnecessary files, including the input and checkpoint files, squeeze the disk(s), and start the entire process over. To do this, you must have a backup copy of the input.

3. Write a small program to convert your ISAM input file into a sequential file and, using this file, use CREATE to produce the desired ISAM file.

#### Insufficient Work File Space

REORG attempts to open output files on the same devices as the input file counterparts. If this is not possible, due to space limitations, a work device is requested via the message:

```
WORK FILES REQUIRE x BLOCKS OF DISK SPACE
WORK FILE DEVICE:
```

Respond to this with a device name. If the specified device does not have enough space, the question will be repeated. Be sure that none of the devices made available to REORG have files with the same name as the input file; files with .ROG extension (or .R01 through .R07); or files with .CKn extensions. Any existing file of this same name will be deleted.

Until enough space is found to open all the required files (including the index build work files), reorganization cannot start.

### 10.4.6 Exiting from ISMUTL (EXIT)

When you no longer need ISMUTL, you may return to the monitor or the operating system by responding to the ISMUTL selection message with an E. The response is:

```
     END ISAM UTILITY
```

In chain mode, the utility automatically chains to the specified user program.

### 10.4.7 Miscellaneous ISMUTL Capabilities

#### 10.4.7.1 STATUS and REORG in Chain Mode

STATUS and REORG can be automated. If you end your program with ISMUTL as the optional argument with the STOP statement, control is automatically passed to the ISAM utility program. Within your user program, you must previously send, with the SEND statement, a record of directions to ISMUTL. Depending on the function to be performed, the record must contain specific field sizes in a certain order. An example for REORG and for STATUS is shown below. It illustrates the required format for the record.

Example of a record sent to do a REORG:

```
 USER PROGRAM (USPROG)

 RECORD REORG
FNCT,     A1,'R'                ;DO A REORG
CHNFLG,   D1                    ;0=DO NOT CHAIN TO USER PROGRAM
                                ;SET USRPG TO SPACES
                                ;1=CHAIN TO USRPG
FILIN,    A17,'DK:ISAM.ISM'     ;YOUR ISAM FILE
USRPG,    A17                   ;PROGRAM TO CHAIN TO
ADBLR,    A1                    ;ADDITIONAL SPACE (A, B, OR C)
EXPDV,    A6                    ;DEV IF C (SPACES IF A OR B)
WORKNM,   D1                    ;NUMBER OF WORK FILES
WORKDV,   8A6                   ;(1-8 DEVICES) (3 CHAR DEVICE NAME,
                                ;NO COLON !)
                                ;IF THE NUMBER IS LESS THAN 8,
                                ;REMAINDER IS FILLED IN WITH SPACES
 PROC
  .
  .
  .
 SEND (REORG,'ISMUTL',TERM)      ;TERM=NUMBER OF TERMINAL
  .                              ;YOU ARE CURRENTLY
  .                              ;RUNNING ON
  .

 STOP 'ISMUTL'                   ;STOP THIS PROGRAM START
                                 ;UP ISMUTL
```

Example of a record sent to do a STATUS:

In the following example, the first time the user program (USPROG.SAV) is run, there is no message to receive, so the record STATUS is therefore sent to ISMUTL with instructions to chain (fields CHNFLG and USRPG) to USPROG.SAV.

At the end of the ISMUTL STATUS function, a record is sent to the user program (specified by USRPG), and ISMUTL chains to USPROG. The record STAT must contain the fields shown in order to receive the STATUS information sent by ISMUTL.

```
 USER PROGRAM (USPROG)

 RECORD STATUS
FNCT,     A1,'S'                  ;DO A STATUS
CHNFLG,   D1,'1'                  ;0=DO NOT CHAIN TO USRPG (SET
                                  ;USRPG TO SPACES)
                                  ;1=CHAIN TO USRPG
FILIN,    A17,'DK:ISAM.ISM'       ;YOUR ISAM FILE
USRPG,    A17,'USPROG.SAV'        ;FILE TO CHAIN TO
,         A56                     ;FILLER
APDFLG,   D1                      ;0=DO NOT RETURN APPEND RECORDS,
                                  ;NON ZERO VALUE
                                  ;= RETURN # APPEND RECORDS


 RECORD STAT
CUROFL,   D6                      ;NO. OF UNUSED OVRFLOW GROUPS
TOTOFL,   D6                      ;NO. OF GROUPS ALLOCATED TO OVRFLOW
TOTREC,   D12                     ;CURRENT NO. OF RECORDS IN FILE
ORGREC,   D12                     ;ORIG NO OF RECORDS IN FILE
APDREC,   D8                      ;REMAINING NO. OF RECORD SPACES IN
                                  ;THE APPEND AREA

 PROC
  .
  .
  .

 RECV(STAT,LABEL)          ;CHECK FOR STATUS INFO. IF NONE,
                           ;SEND MESSAGE
  .                        ;OTHERWISE PROCESS AND OUTPUT
  .                        ;STATUS
  .

 STOP                      ;STOP WITHOUT CHAIN
  .
  .
  .

LABEL,    SEND(STATUS,'ISMUTL.SAV',TERM) ;TERM=TERMINAL YOU ARE ON
  .
  .
  .

 STOP 'ISMUTL'             ;STOP THIS PROGRAM START
                           ;UP ISMUTL
```

The STATUS function will perform considerably faster if the remaining number of record spaces in the append area is not requested. When APDFLG in record STATUS contains zero, the remaining number of record spaces will not be returned.

### 10.4.7.2 Auto-CREATE

The CREATE function requires considerable operator interaction to produce a file. The auto-create capability eliminates most, or all, of this interaction. There are many applications for this capability. One use would be the situation where the parameters of a file are known, and similar files are to be built. It could also be used when some predictable small changes are to be made to an ISAM file, and it would be convenient to avoid numerous sessions with the dialog. Another situation would be a "turnkey" system, in which the operator is not permitted to do a manual create. A final example of an application would be an auto-create to do a reorganization with different parameters than originally specified; because as a file matures the distribution of records to be added often changes.

The auto-create is selected with an answer of YES to the first question in the CREATE dialog. This is followed by a request for an auto-filename. The auto-filename is a data file constructed by the user which contains a list of answers for the expected ISAM CREATE questions. Upon specifying the auto-filename, the CREATE process continues to completion, or until an invalid answer is encountered.

In either SUD or TSD operation, access to the auto-filename or response file can be done manually with the YES answer to the first CREATE question. Access may also be made automatic (programmed). Programmed access to the auto-filename is a function of whether you are operating under SUD or TSD.

Under SUD the program selection (R ISMUTL) and initial responses (C, YES, auto-filename) are supplied from an indirect file. The responses to the CREATE dialog are then supplied by the auto-filename. Below is an illustration of this indirect file:

```
!INDIRECT FILE ISMCRT
R ISMUTL     !RUN ISMUTL
C            !CREATE FUNCTION
Y            !AUTO OPTION
filnam       !AUTO FILENAME (CREATE QUESTION RESPONSES)
```

TSD operation does not support indirect file operation, but a message can be sent to ISMUTL to supply the initial responses in the same manner as with REORG and STATUS when they are run in chain mode. The YES answer to the auto-create question is assumed by ISMUTL when this mode is chosen. The record sent must contain the specific field sizes in the order shown below:

```
 USER PROGRAM (USPROG)


 RECORD CREATE
FNCT,    A1,'C'                          ;DO A CREATE
CHNFLG,  D1                              ;0=DO NOT CHAIN TO USRPG (SET
                                         ;USRPG TO SPACES)
                                         ;1=CHAIN TO USRPG
FILIN,   A17                             ;AUTO-FILENAME (RESPONSE FILE)
USRPG,   A17                             ;CHAIN FILE


 PROC
   .
   .
   .
 SEND (CREATE,'ISMUTL',TERM)            ;TERM=NUMBER OF TERMINAL
                                        ;YOU ARE CURRENTLY
                                        ;RUNNING ON

 STOP 'ISMUTL'                          ;STOP THIS PROGRAM START
                                        ;UP ISMUTL
```

Following is an example of an auto-file. In the previous examples, this is filnam for SUD, or field FILIN in the message sent from the TSD program. This auto-file creates the same ISAM file as shown in the example in Section 10.4.3.4.

```
.R DKED
*AFILE=              ;AFILE is filnam or FILIN
XDATA1.DDF           ;input file
N                    ;do not delete input file
XDATA1.ISM           ;output file
250                  ;number of input records
7                    ;number of records per group
2                    ;load exclusion
Y                    ;fill out group
200                  ;records in append area
100                  ;overflow groups
8                    ;key length
1                    ;key location
N                    ;no duplicate keys
DK                   ;allocate index file to default dev
DK                   ;allocate data file to default dev
E                    ;exit ISMUTL
<GOLD/COMMAND>
EXIT
```

If ISMUTL is called from a detached job via chaining, the normal dialog which appears on the screen will be written to a file called XXX.LOG where XXX is the name of the auto-file. This file will show all display input and output, and will help reveal problems with execution of ISMUTL in detached mode.

If any answer to the response file is invalid, the CREATE will fail. In the manual mode, an invalid answer will result in the question being repeated; in auto mode, the program terminates. If the file is being created in the detached mode (TSD) when an error occurs, the program chain will be broken. A detailed study of the program queries and appropriate responses must be made. There are many conditions which will appear only at run time. Among the things that must be considered are disk space and disk fragmentation.

Table 10-2 summarizes the methods you can use to create an ISAM file.

Table 10-2
ISMUTL CREATE Modes

| | System | |
|---|---|---|
| **Mode** | SUD | TSD |
| Manual | R ISMUTL<br>C<br>N<br>Respond to the<br>CREATE dialog | |
| Semi-<br>Auto-<br>mated | R ISMUTL<br>C<br>Y<br>auto-filname | |
| Auto-<br>mated | @ISMCRT.COM<br>where ISMCRT.COM con-<br>tains:<br>R ISMUTL<br>C<br>Y<br>auto-filname | In the calling program:<br>RECORD CREATE<br>FNCT, A1,'C'<br>CHNFLG,D1<br>FILIN, A17,'auto-filname'<br>USRPG, A17<br><br>PROC<br><br>SEND (CREATE,'ISMUTL.TSD')<br>STOP 'ISMUTL' |

## 10.5 ERROR MESSAGES

See Appendix B, Table B-9, for a list of error messages for ISMUTL.

# CHAPTER 11

# SORT/MERGE

## 11.1 INTRODUCTION

At one time or another, you will probably be faced with a file whose records are not in the order you need for a particular use. Or you may find that you have two or more files containing information in identically structured records that you want to combine into one file to form a common data base. A sort program facilitates the reordering of a file, and a merge program is used to combine files having the same record structure.

CTS-300 includes tools that enable the user to develop a sort or merge program. The techniques for doing this are described in this chapter. The development of a sort or merge program starts with a user-written control file that describes the parameters of the operation. The SORTG program, with the control file as input, generates the variable data division of the desired sort or merge program. This file and SORTM.DBL, which is the procedure division of the sort or merge program, are then compiled to produce an object file. This object file is linked like any other such file to produce the usable sort or merge program. When the program is run, the specified files are sorted or merged, as requested.

### 11.1.1 Characteristics

Several features of CTS-300 Sort/Merge should be pointed out:

- Files may be sorted by multiple keys, and the sort for each key may be in ascending or descending order.
- An optional key sort (TAGS sort) is available which allows a sort of key fields only, without transporting entire records across work files.
- Optional statements are provided to allow the user to define such operating parameters as memory allocation, work files, and so forth.
- The sort or merge program developed by the CTS-300 user has the optional ability to receive a control record sent from another program. This control record facilitates changes in selected sort or merge parameters at run-time.

When you use a CTS-300 Sort/Merge program, you must be aware of some limitations:

- The caret symbol (^), which is 136 octal, is the internal control character used in Sort for the end-of-string marker. If used as the first character of a record, sort results will be unpredictable.
- The Sort program is not able to sort a multivolume sequential file, because Sort requires a CTRL/Z at the end of all input files.
- The Sort/Merge program is not able to sort or merge records with packed keys. Compressed records can be handled as long as the record lengths and displaced key(s) are correctly described, and the sort keys remain unpacked.
- The output of a sort or merge is a sequential file. Sorting an ISAM file will not produce another ISAM file as output.

### 11.1.2 Chapter Organization

The remainder of this chapter is structured in the general sequence required to develop a sort or merge program. It is comprised of three sections. Section 11.2, the Control File, covers control file characteristics and requirements. The individual statements within the control file are explained, and examples are given where necessary. The statements are then summarized. This is followed by Section 11.3, Sort/Merge Program Development and Use, which covers the development procedure. Development consists of using the routines supplied by CTS-300, SORTG and SORTM, to integrate the control file into the desired program. The last steps of the procedure are basically routine compilation and linking. Examples are used to illustrate both features and correct usage. Last of all, the use of a control record for chain mode operation is presented. Section 11.4, Error Messages, is a reference to the Sort/Merge error messages.

## 11.2 THE CONTROL FILE

The sort or merge that you want operates on information supplied by you, the user. The control file describes the input file(s), output file, sort key(s), record format and other necessary parameters and options. This control file is created by you with the aid of an editor.

The control file is composed of five required control statements and six optional control statements. A different control file must be built for each sort or merge. There are situations where another control file may involve just a simple change of input file name, or it may require a change of most or of all the parameters. Whenever any statement is changed, the control file must go through the procedure of being processed by SORTG, compiled along with SORTM, and linked to produce the usable program.

The characteristics and use of all eleven statements are discussed on an individual basis in the following subsections. Control statement options are shown in brackets. Comment lines are allowed on any statement line if preceded by a semicolon. Examples are given for each statement requiring clarification.

Your first Sort/Merge exercise will be easier if you avoid the optional statements. The five required statements are summarized below:

| | |
|---|---|
| INPUT | This statement is used to specify the input file(s) for the sort or merge. |
| OUTPUT | This statement is used to specify the name of the sorted or merged file that results from running the sort or merge program. |
| RECORD | The program must know the record structure of the input file(s) it is dealing with and the location of the key(s) within the record. This statement specifies these parameters. |
| KEYS | This statement defines the order in which the file is to be sorted, in relation to the chosen key(s). |
| END | This statement identifies the end of the control file. |

The required statements are presented in the above order in the following sections. Optional statements are grouped together and are listed in alphabetical order between the KEYS and the END statements. There is no required order for the statements; the order shown here is just a logical arrangement.

## 11.2.1 INPUT

The INPUT statement is normally the first statement in a control file. The file to be sorted or the files to be merged are specified here. The specification of a single file indicates that a sort is desired. The specification of two or more files indicates that a merge is to be performed.

The format is:

```
     IN[PUT]:filspec1[,filspec2...,filspec7][(count)]
```

where:

| | |
|---|---|
| filspec1 | is the first input file specification. If this is the only file specified, the operation is a sort of this file. |

filspec2 through filspec7
: is one or more additional file specification(s) up to a maximum of seven files. If included in the statement, the operation is a merge of the files specified here. Files specified in addition to the first file are separated by commas.

**NOTE**

> Files to be merged must be sorted prior to merging.

(count)
: is optional when used with sequential files. It specifies the number of records, starting with the first record, to be sorted or merged. If multiple files are specified, the count argument must either be specified for all of the files or for none of the files. The number must be enclosed by parenthesis.

**NOTE**

> The count argument must be used for an ISAM input file(s) and must, in this case, contain only the letters ISM.

- Mixed file types can not be used as input to a merge operation. That is, ISAM and sequential files can not both be specified in the same merge operation.
- The count argument can be overridden at run time by a program that chains to your Sort/Merge program. See Section 11.3.6.

Example:

```
INPUT:RK0:PAY.DDF(2400)
```

Sorts the first 2400 records of file RK0:PAY.DDF.

```
INPUT:LABR.DDF
```

Sorts file LABR.DDF on the default device. The entire file is sorted.

```
INPUT:RK1:LABM.DDF(3000),RK1:LABT.DDF(4000)
```

Merges the first 3000 records of file RK1:LABM.DDF and the first 4000 records of file RK1:LABT.DDF.

```
INPUT:RK0:PAY.ISM(ISM)
```

Sorts the ISAM file on RK0 called PAY.ISM.

## 11.2.2 OUTPUT

The output statement specifies the name you wish to give the file that is the output of the sort or merge operation.

The form is:

```
     OU[TPUT]:filspec
```

where:

filspec is the desired name of the output file.

- The output filename can not be the same as the input filename for ISAM files if a merge is performed, or if a TAGS:SORT, TAGS:LIST, or TAGS:INDEX sort is being done (see Section 11.2.9). An error message will result if any of these is attempted. In the case of the TAGS:SORT, the error message will appear only if there is not sufficient free space for a temporary copy of the input file.
- If a sequential file is being sorted, and if the output file name is the same as the input file, the input file is overwritten.

Example:

```
     OUTPUT:RK1:WEEKLY.DDF
```

The sorted or merged output is written as a file named WEEKLY.DDF on device RK1.

## 11.2.3 RECORD

The record statement defines the format of the records in the input file(s). The format for each file is identical to the DIBOL record format, except that no initial value may be specified.

The format is:

```
 RE[CORD]:

 field def 1
         .
         .
         .
 field def n
```

where:

field def 1 to field def n
: are field definitions for each key field to be used in the subsequent sort or merge; for fields to be used as filler to position the key fields within the record; and for fields to fill out the record to the total length of the input record.

  - Each field def is on a separate line.
  - Each field def is formed in accordance with the rules for a DIBOL data division statement.

Example:

```
RECORD:

FILLA,   A5      ;fill characters
NAME,    A20
FILLB,   A35     ;fill characters
BADGE,   A5
```

This record statement defines a record of 65 characters. Positions 6-25 contain a key field named NAME, and positions 61-65 contain a key field named BADGE. The other positions in the record may contain any number of other fields, but they are of no interest to the Sort/Merge program.

### 11.2.4 KEYS

The keys statement specifies which fields within the input file records are to be used to control the order of the output file. These fields are known as control keys.

The format is:

```
KE[YS]:fldnam1[-][,fldnam2[-]...,fldnam8[-]]
```

where:

fldnam1 through fldnam8
: indicates the field(s) to be used as the control field(s) (or control key(s)).

  - Each fldnam must be a name of a field defined in the RECORD statement.
  - The first fldnam specified becomes the major control key; the last fldnam specified becomes the minor control key; and the fldnams between become intermediate control keys in the order specified.

is an optional argument associated with a given fldnam that, if included, causes records to be ordered in descending sequence by that fldnam. If not included, the records are ordered in ascending sequence.

No more than eight control keys may be specified.

Example:

```
     KEYS:NAME,DATE-
```

Specifies a sort or merge using the field called NAME as the major control field, ordered in ascending order, and the field called DATE as the minor control field, ordered in descending order. Names appearing first in the alphabet would appear first in the file, and for each name category, the records would be arranged with the most recent date appearing first. Note that a blank is treated differently depending on whether it appears in an alpha or a decimal field. An alpha space is on octal 40 and a numeric zero is an octal 60.

### 11.2.5 DETACH (Optional)

The DETACH statement is an optional statement that causes the sort or merge program to detach from its associated terminal. This allows you to execute another job while a sort or merge is running.

The format is:

```
     DE[TACH]:
```

There are no arguments.

The characteristics of this statement are:

- DETACH is ignored under SUD operation.
- Any detached program that produces terminal output will hang until it is attached.
- If error messages are generated, they will not be displayed until the program is attached.
- DETACH occurs only once. If you reattach, your terminal is dedicated to the attached program until it terminates. See Chapter 5 of this manual for information on the ATTACH and DETACH commands.

### 11.2.6 EXECUTE (Optional)

EXECUTE is an optional statement that specifies the name of a program to be executed (chained to) after the Sort/Merge program terminates.

The format is:

```
     EX[ECUTE]:filspec
```

where:

filspec — specifies the name of the program to be executed.

- If EXECUTE is omitted, control returns to the system monitor.
- EXECUTE may be overridden. See Section 11.3.6.

Example:

```
     EXECUTE:MYFIL.SAV
```

Indicates that program MYFIL.SAV on the default device is to be executed upon completion of the Sort/Merge operation.

### 11.2.7 SPACE (Optional)

The SPACE statement is an optional statement that controls the amount of additional memory available for a Sort/Merge operation.

The format is:

```
     SP[ACE]:kbytes
```

where:

kbytes — is a decimal number used to specify, in thousand byte segments, the total amount of main memory to be available to a Sort/Merge program. This includes the DIBOL program and all buffers, but does not include the DIBOL interpreter, monitor, or any other requirement.

- The default memory size if the SPACE statement is not used is 16 K bytes.
- If the default of 16 K bytes is used, and the record is too large to allow a minimum of three records to be held in the work buffers, SORTG suggests, via a message, the amount of space required to support the number of work files requested. You then have the option of reducing the number of work files or of increasing the space.

- The Sort/Merge program is designed to work within artificial limits imposed by TSD. If the sort requirements exceed the memory space available, a message (as above) will appear, suggesting new values for space and work file allocation.

- For faster operation, add 1 K byte of memory for every work file assigned in addition to the default of three work files.

- Files with very large records may require more than the default of three work files (see Section 11.2.10). Specify 1 K byte of additional memory for each added file.

- Within TSD operation, the SPACE option should be used with consideration for the memory requirements of other users.

Example:

```
     SPACE:18
```

Indicates that a Sort/Merge program may use up to 18 K bytes of memory.

### 11.2.8 SU (Optional)

SU is an optional statement that specifies that the Sort/Merge program is going to be run as a Single User DIBOL program. SUD operation is faster than TSD operation, for several reasons, and should be used for large files.

The format is:

```
     SU:
```

There are no arguments.

The characteristics of SU are:

- Compared to TSD, SU operation is faster, because more memory can be used. There is no contention for memory or disk resources.

- I/O buffer allocation is automatically doubled with the SU option.

- The SU option forces six work files. More may be specified with the WORK statement, but it is not advised for sorts that utilize random access I/O (TAGS:SORT).

- The SU option overrides the SPACE statement and automatically optimizes the internal working areas.

- If you use the SU option with the XM monitor, you may receive a message indicating there is not enough memory. If this happens, use the SJ monitor.

### 11.2.9 TAGS (Optional)

The TAGS statement is an optional statement that allows a sort to be performed using only the keys instead of the entire record. A key sort offers substantial advantages in execution speed and minimized disk requirements, since the entire file need not be manipulated. The TAGS option should be used for most sorts.

The format is:

```
     TA[GS]:type
```

where:

type — is either SORT, LIST, or INDEX.

TAGS:SORT — specifies a sort that produces a sorted version of the input file. The key or keys are first sorted, and then a pass is made to write each complete record in the new output file.

- TAGS:SORT must not be used on an ISAM file.
- A TAGS:SORT requires much smaller work files than a normal sort.
- A TAGS:SORT can provide a significant performance advantage over a normal sort when working with a record of approximately 80 characters and a short key. With larger records, the advantage is greater.
- The final pass to write the output file consumes much of the time required for a TAGS:SORT. This pass is eliminated with either the TAGS:LIST or the TAGS:INDEX sort below.

TAGS:LIST — specifies a sort that produces an output file which consists of a seven-byte record number. This file can then be used to access the original file. The TAGS:LIST is basically like the TAGS:SORT but without the pass required for the post-sort write.

- Each of the records in the output file contains a record number identifying the record in the original file that corresponds to the relative position of this number in the output file. That is, the third record in the output file contains the record number for the record in the original file that has become third in the desired sort sequence.

- Leading zeros in the output file are encoded as spaces.

- The output file name should ~~be~~ different from the input file name.

TAGS:INDEX specifies an output file consisting of the key field(s) specified in the KEYS statement and a corresponding seven-byte relative record number, as in the TAGS:LIST statement. The record format consists of the major key, the intermediate key(s), the minor key, and the relative record number.

- If your original file is an ISAM file, use the TAGS:INDEX option and add the ISAM key as a minor sort key. It has little effect on sort speed, and you can then access the original ISAM data file by means of the sorted TAGS:INDEX output file. That is, you can select a record on the basis of any chosen key in the TAGS:INDEX output file, and then access the record in the ISAM file using the corresponding ISAM key.

- The output file name should be different from the input file name.

## 11.2.10 WORK (Optional)

The WORK statement is an optional statement that specifies both the number of work files to be used for a sort and the devices on which the work files are to reside.

The format is:

```
WO[RK]:[number][,devl:...,devn:]
```

where:

number is a decimal number between three and nine, inclusively, that specifies the number of work files to be allocated. If the number (or the WORK statement itself) is omitted, the default is three work files on the default device.

dev1: through devn:
specifies a list of one or more devices to which the work files are allocated. Dev: must be a valid device name and must include the colon. If dev: is omitted, all work files are allocated to device DK:.

- The first work file is assigned to the first device in the list. The second work file is assigned to the second device, and so forth. The same device may be specified more than once.
- If the number of devices specified is less than the number of work files, devices are allocated to files by starting over with the first device specified. File assignments cycle through the device list, until all the files have been assigned to devices.
- If the number of devices assigned exceeds the number of work files, the excess devices are ignored.
- If a merge is specified, the WORK statement is ignored.
- With a file of approximately 5000 to 10,000 records, four to six work files, rather than the default of three, will result in a measurable performance improvement.

Example:

```
WORK:4
```

Allocates four work files, all on device DK:.

```
WORK:3,RK2:
```

Allocates three work files, all on device RK2.

```
WORK:3,RK1:,RK2:,RK3:,RK4:
```

Allocates three work files, one each on devices RK1, RK2, and RK3. RK4 is ignored, since only three work files are specified.

```
WORK:3,RK1:,RK2:
```

Allocates three work files, the first on device RK1, the second on device RK2, and the third on device RK1.

### 11.2.11 END

The END statement indicates the end of the control file. The format is:

```
EN[D]:
```

there are no arguments. The only restriction on the statement is that it must be the last one in the control file.

### 11.2.12 Summary of Control File Statements

Below are the control file statements in the order presented previously, with optional statements indicated.

| Statement | Optional | Comments |
|---|---|---|
| IN[PUT]:filspecl[...,filspec7] [(count)] | No | First statement |
| OU[TPUT]:filspec | No | Name of the file resulting from the sort or merge operation |
| RE[CORD]: field def | No | Fields must be in specified order |
| KE[YS]:fldnaml[-] [...,fldnam8[-]] | No | Maximum of 8 |
| WO[RK]:[number] [,devl:...,devn:] | Yes | Ignored for merge |
| TA[GS]:SORT :LIST :INDEX | Yes | Fast sort |
| SP[ACE]:kbytes | Yes | |
| SU: | Yes | SU overrides SPACE allocation |
| DE[TACH]: | Yes | Ignored with SU option |
| EX[ECUTE]:filspec | Yes | |
| EN[D]: | No | Last statement in control file |

## 11.3 SORT/MERGE PROGRAM DEVELOPMENT AND USE

Now that you have developed a control file, it must be processed by SORTG. The output of SORTG is compiled together with SORTM, and the result linked to produce your Sort/Merge program. The program is then run to perform the sort or merge. The details of these steps are contained in the following sections.

### 11.3.1 SORTG

SORTG is a CTS-300 supplied file (SORTG.SAV) that generates the data division of your Sort/Merge program, using your control file as input. It is an executable program that asks you for the name of its input file and of its output file. The dialog sequence follows:

```
.R SORTG
SORT GENERATOR VAnn-nn
INFILE=
```

Respond with the name given to your control file when you created it with the editor. The response will be:

```
OUTFILE=
```

Respond with the desired file specification of your Sort/Merge program (SMPROG.DBL for the sake of the following discussion).

After you provide the output file name, SORTG generates the output file unless an error is detected. Errors for SORTG are listed in Appendix B, Table B-10. If an error is detected, a message will be displayed indicating that output has been suspended. This message will be followed by the error message. The error message is usually preceded by the line number of the control statement in error.

### 11.3.2 Compiling with SORTM

SORTM.DBL is another CTS-300 supplied file and is the procedure division of your Sort/Merge program. SORTM and the output resulting from running SORTG (SMPROG.DBL) must both be compiled to produce an OBJ file. That is:

```
.DIBOL SMPROG+SORTM
```

whose output is SMPROG.OBJ.

### 11.3.3 Linking

The .OBJ file produced by the compiler must be linked to run in either SUD or TSD mode.

for SUD:

```
     .LINK SMPROG.OBJ,DIBOL
```

whose output is:  SMPROG.SAV ,or

for TSD

```
     .LINK/EXE:SMPROG.TSD SMPROG.OBJ,TDIBOL/BOT:100000
```

whose output is:  SMPROG.TSD

You will probably want to run REDUCE on this TSD file.

### 11.3.4 Running the Sort or Merge Program

The Sort/Merge program is run like any other program:

```
     .R SMPROG or .RU dev:SMPROG
```

The input file(s) specified in the control file is sorted (or merged) to produce a file with the specified output name. Errors generated during execution are those listed for SORTM in Appendix B, Table B-11.

### 11.3.5 Example

The following is an example illustrating all the steps required to generate and run a sort program.

There are five steps:

1. Create the control file:

```
     .R DKED
     *FILE.CTL=
     IN:DATA1.DDF               ;the unsorted input data file
     OU:SDATA1.DDF              ;the desired sorted file
     RE:                        ;record definition follows
     ,    A5                    ;space filler
     NAME,A20                   ;a field to be used as a sort key
     ,    A35                   ;space filler
     BADGE,A5                   ;a field to be used as a sort key
     SU:                        ;the sort program will be
                                ;run in a single-user environment
     EN:                        ;end of control file
     <GOLD/COMMAND>
     EXIT
```

2. Run SORTG

```
.R SORTG
SORT GENERATOR VAn-nn
INFILE=FILE.CTL
OUTFILE=SORT.DBL
```

3. Compile:

```
.DIBOL SORT.DBL+SORTM
```

whose output is: SORT.OBJ

4. Link:

```
.LINK SORT.OBJ,DIBOL
```

whose output is: SORT.SAV

5. Run the sort program:

```
.R SORT.SAV
```

when you run SORT.SAV file DATA1.DDF will be sorted as specified into a new file SDATA1.DDF.

### 11.3.6 SORT/MERGE in Chain Mode

Certain parameters in the Sort/Merge Control File can be changed at run time. The parameters that can be changed are:

- The record count in the control file INPUT statement.
- The name of the program to be executed after the sort or merge is completed (originally specified in the control file EXECUTE statement).
- The EXECUTE statement, as originally specified, may be selected as the only statement to be executed. That is, no sort or merge is to be performed.
- A message can be sent by a preceding program to the program specified in the EXECUTE statement of the Sort/Merge Control File. The size of the message is 100 characters but it can be changed by editing SORTM.DBL.

The operations listed previously are accomplished via a Sort/Merge Control Record that is sent to the sort or merge program prior to execution. The required format of this control record is:

```
        RECORD SCR
OPCODE, A1,'x'  ;value determines the interpretation of
                ;the control record (see below)
COUNT,  D6      ;record count (overrides count in INPUT
                ;statement)
NEXT,   A14     ;filspec of program to follow sort
MSG,    A100    ;message to be passed to the program to
                ;which sort or merge is chained
```

The value of the opcode character determines how the control record is interpreted. This character may have one of three values. They are: %, @, or a character other than % or @ (including a space).

If the opcode value is a %:

- If COUNT (in the control record) is nonzero, its value is inserted as the control file INPUT statement record count. Any previous value is overwritten.

- If NEXT is nonblank, its contents replace the program specified in the control file EXECUTE statement. NEXT, therefore, forces sort or merge to chain to a program specified at run time.

  If the opcode value is a @:

- No sort or merge takes place. Instead the program specified by the control file EXECUTE statement is executed. This feature allows SUD-chained programs to dynamically bypass a sort without breaking the run chain.

  If the opcode value is anything other than a % or a @:

- The message in field MSG of the control record is sent to the program specified by the EXECUTE statement in the control file.

Thus a control record may consist of the following combinations of fields and functions:

| | |
|---|---|
| OPCODE(%) | sets COUNT and/or NEXT |
| OPCODE(@) | indicates no sort or merge; simply chain directly to program specified by the EXECUTE statement |
| OPCODE (other than % or @) | indicates a message is to be sent to the program specified by the EXECUTE statement. |

As an example of how this feature is used, consider a program (PROG1) running prior to a sort or merge. Within PROG1 a Sort/Merge Control Record is sent (with the DIBOL SEND statement) to the Sort/Merge program (SMPROG). PROG1 then chains to SMPROG by having SMPROG specified as the chain program name in the DIBOL STOP statement at the end of PROG1. SMPROG then runs with modifications determined by the control record contents. Upon completion of the sort or merge, the chain program (if so specified by the EXECUTE statement in the sort control file) is executed or control returns to the monitor.

## 11.4 ERROR MESSAGES

See Appendix B, Tables B-10 and B-11, for a list of error messages for SORTG and SORTM.

# CHAPTER 12

# STATUS

## 12.1 INTRODUCTION

STATUS is a TSD utility used to obtain information on your TSD Run-Time System.

### 12.1.1 Features

The information obtained using STATUS is determined by the option you choose. The specific kinds of information available are:

- List of STATUS options
- Available free memory and its relative location
- Total active jobs
- Information on a specified active job
- Total line printer active jobs
- Active job information for a specific line printer
- Pending messages
- Characteristics of the current version of a time-shared RTS.

Additionally, two options that are useful after information has been retrieved are:

- Termination of an active job
- Exit from STATUS

Some of the other characteristics of STATUS are:

- STATUS runs only on a time-shared system with an attached terminal.
- STATUS requires 6 K bytes of memory.
- STATUS outputs to either a terminal or a line printer.

### 12.1.2 Chapter Organization

The remainder of this chapter is comprised of two sections. Section 12.2, Options, details the individual options available. Section 12.3, Using STATUS, explains how to use the STATUS utility to gather information on your time-shared system.

## 12.2 OPTIONS

STATUS functions as a result of the options you select. These options are explained after a short discussion of the conventions used in STATUS.

### 12.2.1 STATUS Conventions

A list of options is initially displayed every time STATUS is run, and a brief option list is displayed each time STATUS prompts for input. Only one option can be selected at a time. STATUS recognizes a maximum of three characters to designate an option; however, most options require only one character.

The individual options are all shown with the STATUS prompt line followed by a typical output.

### 12.2.2 Option F

Option F retrieves the amount of free memory and identifies its location.

Typical Option F response:

```
ENTER OPTION (F,H,J,K,L,M,T OR X): F
FREE MEMORY                                     DD-MMM-YY HH:MM:SS
HIGH MEMORY:      0 BYTES
LOW MEMORY:  16824 BYTES
STATUS.TSD:   6304 BYTES  IN LOW MEMORY
```

The numbers shown indicate the amount of memory available in extended memory (HIGH) and in lower memory (LOW). The dividing line between LOW and HIGH memory is at 56 KB. The high memory value will always be zero unless you have an XMTSD system. The size and location of STATUS indicate additional memory that will be available when STATUS is terminated.

### 12.2.3 Option H

Option H presents a list of the STATUS options.

The response is:

```
ENTER OPTION (F,H,J,K,L,M,T OR X): H
OPTIONS  (F)     FREE MEMORY
 (H)     HELP
 (J)     ACTIVE JOB LIST
 (JX)    ACTIVE JOB NO. DESCRIPTION
 (KX)    KILL AN ACTIVE JOB
 (L)     LPQFIL JOBS PENDING - ALL LP'S
 (LX)    LPQFIL JOBS PENDING - BY PRINTER (L1,L2,L3,L4)
 (M)     MESSAGES PENDING
 (T)     TSD PARAMETERS
 (X)     EXIT
```

### 12.2.4 Option J

Option J retrieves the complete list of active jobs on the system.

Typical Option J response:

```
ENTER OPTION (F,H,J,K,L,M,T OR X): J
JOB NO. JOBNAME        TERM.NO.  SIZE  MEM.LOC. DD-MMM-YY HH:MM:SS
  0     DK :STATUS.TSD    0      6232    LOW
  1         KBDLDR        1       688    LOW
  3     DK :COUNT2.TSD   DET      920    LOW
        <MESSAGE AREA>           2048    LOW
```

JOB NO. is a numeral assigned by STATUS to identify a job. JOBNAME is displayed in usual filspec format (dev:filnam.ext). The remaining entries are self-explanatory, with the following exceptions: If there is no job running at a terminal, and if the terminal is able to receive a job request, that terminal will show a pseudo job indicated by the name KBDLDR. The last line in the example shows the message buffer allocation for XMTSD systems only. If there are no messages pending, the message area report does not appear. In this case, it shows there is 2 K bytes of buffer space used for this purpose.

### 12.2.5 Option Jx

Option Jx retrieves information on a specific job. The STATUS job number is specified by a numeric value for x where x may be up to two characters. Use the J option to determine the STATUS job number.

Typical Option Jx response:

```
ENTER OPTION (F,H,J,K,L,M,T OR X): J3
JOB NO.  3                                     DD-MMM-YY HH:MM:SS
JOBNAME  DK :COUNT2.TSD TERM.NO.    DET SIZE    920  LOW MEMORY
CHANNEL  DEV:FILNAM.EXT  FLSIZE     MODE   TOT.USERS   UPD.USERS
  1      DK :NUMBER.DDF    3049      0         1          255
```

This message repeats the job number (3, selected with J3) on the first line. The second line specifies the job filspec, the terminal number (indicated by DET because job 3 is detached), and the size of the job in bytes. The fourth line shows the channels open for the job; the device and files being accessed; the file size; the mode of access; the total users of that file; and the number of update users of that file. If the job number specified is valid, but no files are open, a message appears indicating that no files are open. If an invalid job number is specified, a message will appear indicating that no job by that number exists.

### 12.2.6 Option Kx

The option Kx kills the job indicated by the specified numeric value of x where x may be up to two characters. This option permits you to terminate any job, regardless of whether or not the job is attached or detatched, or on a master or a slave terminal. Use the J option to determine the number of the job you wish to terminate and again to verify that the termination took place.

When the job specified is running attached, the TSD version number is displayed on the attached terminal once the job is terminated.

When the job specified is running detached, there is no message.

**NOTE**

The Kx option operates like a double CTRL/C. Files opened in output mode are lost when Kx is entered; and files that were opened in update mode may not reflect the latest record updates, if Kx is entered.

## 12.2.7 Option L

Option L retrieves the contents of the print queue that was created with the print spooler utility, LPTSPL.TSD.

Typical Option L response:

```
ENTER OPTION (F,H,J,K,L,M,T OR X): L
NAME NO. DEV:FILENAM.EXT  COPIES  ALIGN  DELETE DD-MMM-YY HH:MM:SS
 LP   1  RK1:MIPROG        1        Y       N
 LP   1  RK1:URPROG        2        Y       N
 LR   3  RK1:BIGPRG        4
 LS   4  RK1:SMLPRG        1
```

Each print request is identified by a two-character line printer name, followed by the line printer number; the file to be printed; and the number of copies. The response to the ALIGN and DELETE arguments is as specified in the original DIBOL LPQUE statement. If the queue is empty, a message is displayed indicating this condition.

## 12.2.8 Option Lx

Option Lx retrieves the contents of the print queue for a specific printer. When entered, x is a decimal number in the range 1 to 4 which indicates the desired printer.

Typical Option Lx response:

```
ENTER OPTION (F,H,J,K,L,M,T OR X): L1
NAME NO.   DEV:FILNAM.EXT  COPIES  ALIGN  DELETE DD-MMM-YY HH:MM:SS
 LP   1    DK1:MIPROG        1        Y       N
 LP   1    DK1:URPROG        2        Y       N
```

Each file in the queue for the selected printer is on a separate line. The information is arranged in this order: printer name and number; file specification; and, as with Option L, the number of copies. The response to the ALIGN and DELETE arguments is, again, as specified in the original DIBOL LPQUE statement.

If there are no print jobs pending for this printer, a message is displayed indicating that there are none.

## 12.2.9 Option M

Option M retrieves the contents of the message queue.

Typical Option M response:

```
ENTER OPTION (F,H,J,K,L,M,T OR X): M
MESSAGE FOR PROGRAM   TERM. NO.  SIZE  DD-MMM-YY HH:MM:SS
DK1:MIPROG               1        704
```

Each pending message is on a separate line. Information is provided on the device; program name; size (in bytes); and the number of the terminal for which the message is queued.

If there are no messages pending, a message is displayed indicating there are none.

### 12.2.10 Option T

Option T retrieves the operating parameters of the current version of the TSD or XMTSD RTS.

Typical Option T response:

```
ENTER OPTION (F,H,J,K,L,M,T OR X): T
TSD PARAMETERS                                     DD-MMM-YY HH:MM:SS
MAX. NO. OF JOBS:          4    NO. OF TERMINALS:        2
MAX. NO. OF CHANNELS:     12    MAX. NO. OF MESSAGES     8
MAX. NO. OF DEVICES:       6    SLICE:                  64
SWAP USR:                YES    ISAM:                  YES
DDT:                     YES    MONITOR(SJ/FB/XM)       FB
FORCED JOB STARTUP:      YES
```

The values displayed are the limits for the current version of the TSD or XMTSD system. Most of these reflect decisions made during CTSGEN and can be changed only with another CTSGEN, the exceptions are SLICE and swap/noswap USR.

### 12.2.11 Option X

This is the exit option. It allows control to return to the run-time system. When option X is selected, the TSD or XMTSD RTS regains control of the terminal and displays the current TSD version number and the asterisk prompt. Any program linked for TSD operation can then be run.

## 12.3 USING STATUS

The STATUS program is run by entering the following response to the TSD asterisk prompt:

```
*R STATUS or *RU dev:STATUS
```

the response is:

```
OUTPUT TO LINE PRINTER?  ENTER Y/N:
```

Respond with Y or N. An N response indicates that you want the output device to be the terminal. A Y response indicates that you want the output device to be a line printer. Line printer selection produces a further question:

```
     ENTER PRINTER NUMBER.  (1-4):
```

Enter the number for the desired printer on your system. The printer must have been specified during the RT-11 SYSGEN. If the printer were not specified in SYSGEN, or if the printer is busy, an appropriate message is displayed. If an invalid printer number (not 1-4) is entered, the system prompts for the number again.

After selection of output device, a list of options is displayed or printed, as with Option H.

## 12.4 ERROR MESSAGES

Error messages for STATUS are contained in Appendix B, Table B-12.

# CHAPTER 13

# REDUCE

## 13.1 INTRODUCTION

REDUCE is a time-shared utility program that decreases the size of a file linked at a base address of 100000 (octal) (16 K bytes). A file so linked includes 63 unused blocks. In a small system, especially those using floppies, this wasted space can become excessive. Eighty TSD files, linked as described, contain more than enough wasted space to fill an entire RK05. REDUCE accepts an input file, reads it; then copies the file, minus the 63 unused blocks. It then deletes the input file. For overlaid files, the relative block number in the overlay handler tables is also modified.

### 13.1.1 Characteristics

- Requires 4 K bytes of memory.
- Recognizes file-structured devices only.
- Supported by RT-11 V3 (or later) only.

### 13.1.2 Chapter Organization

The remainder of this chapter is comprised of two sections. The first, Section 13.2, REDUCE Options, is a discussion of the options available with REDUCE. The second, Section 13.3, Using REDUCE, describes the use of REDUCE.

## 13.2 REDUCE OPTIONS

There are three modes in which REDUCE can operate: query, no query, and version number. These modes are a result of options selected when you specify the file(s) to be reduced. There are two options. Modes and options are discussed next. Because REDUCE is such a straightforward program, selection and format for mode and option are shown in Sections 13.3.2 and 13.3.3 rather than in this section.

### 13.2.1 Query Mode

This is the default mode; no option is specified. Each file that satisfies the stated file specification(s) is listed for you to decide whether to reduce it. You respond with Y if you wish the file reduced, or with N (or carriage return), if you do not wish the file reduced. If you attempt to reduce a file that was previously reduced, or a file that was not linked for a base of 100000, a message will be displayed indicating that REDUCE has ignored that file.

### 13.2.2 /N Option (No Query)

The /N option suppresses the individual file queries present in the query mode. Processing proceeds on all the files that meet the file specification(s). This option provides fast processing of your files. Messages are displayed, as in the query mode, indicating a previously reduced file or a file not linked for a base of 100000.

### 13.2.3 /V Option (Version Number)

The /V option causes the REDUCE utility to display its current version number.

## 13.3 USING REDUCE

### 13.3.1 Conventions

**Input files**

- REDUCE operates only on files that have been linked for a base address of 100000 (octal). Upon receipt of an input file specification, REDUCE searches the header block for the correct link address, and makes sure the file has not already been reduced.
- REDUCE will select only files with an extension of .TSD.
- More than one input file can be specified.

**Output files**

The output file resulting from reduction has the same name as the input file(s).

### 13.3.2 Running REDUCE

To start REDUCE, enter the following command:

```
.R REDUCE or .RU dev:REDUCE
```

or, for TSD

```
*R REDUCE or  *RU dev:REDUCE
```

REDUCE responds with an asterisk prompt for the input file(s).

The general form of file specification and option selection is:

```
filespec/n
```

where:

filespec is the file specification of the file(s) to be reduced.

- If more than one file is to be reduced, the file specifications are separated by commas.
- REDUCE assumes a default extension of .TSD.

/n is the option specifier with n being either an N for no query or a V for version number.

### 13.3.3 REDUCE Examples

The modes and options available with REDUCE are illustrated in the following subsections.

#### 13.3.3.1 Query Mode

To reduce the two files MYFIL and YURFIL, enter the following in response to the asterisk prompt:

```
*MYFIL,YURFIL
```

The result will be

```
DK:MYFIL.TSD?
```

The file is on the default device and exists with a .TSD extension. Respond with Y or N. The next line will be:

```
DK:YURFIL.TSD?
```

The same conditions and possible response exist as did for the first file.

#### 13.3.3.2 No Query Mode

To reduce the two files used in the query mode example without query, the response to the asterisk prompt is:

```
*MYFIL,YURFIL/N
```

to which the only response would be the prompt indicating that the files have been reduced.

#### 13.3.3.3 Version Number Mode

To obtain the version number of the REDUCE utility, the response to the asterisk prompt is:

```
*/V
```

The display would then be

```
REDUCE UTILITY VAnn-nn
```

where:

| | |
|---|---|
| nn-nn | is the current version number. |

### 13.4 ERROR MESSAGES

See Appendix B, Table B-13, for REDUCE error messages.

# APPENDIX A

# CTS-300 RUN-TIME ERROR MESSAGES

The following is a list of the CTS-300 Run-Time Error Messages and their meanings. They are listed in order by error number. Table B-3 in Appendix B of this manual contains a few special errors generated by the DIBOL Time-Shared RTS only. Additional errors are also listed in Appendix B of this manual for the CTS-300 Utility Programs and in the **RT-11 System Message Manual** for the RT-11 Operating System.

There are certain errors listed which suggest you notify your Digital representative. These errors indicate there is a serious logic error in the system software rather than the application software. You should inform DIGITAL in writing notifying us of the problem and include information on how to reproduce the problem.

Messages are identified by type: T for trappable errors and NT for nontrappable errors.

## CTS-300 Run-Time Error Messages

| Number/Message | Type | Meaning |
|---|---|---|
| 1 END OF FILE | T | Logical end-of-file (CTRL/Z) detected:<br>● When reading an input file (except via ACCEPT field).<br>● When operator typed CTRL/Z in response to the message requesting you to mount a successor device for input.<br>● When requesting an ISAM record whose key value exceeds the greatest key already in the file. |
| 2 RETURN WITHOUT CALL | NT | Program executed a RETURN statement but no CALL or XCALL statement was in effect. |
| 3 COMPILATION ERROR | NT | Attempt to execute a statement line that contains an error previously detected by the compiler. |
| 4 DIBOL STACK OVERFLOW | NT | Too many nested subroutine calls and/or subroutine calls with many arguments.<br><br>Subexpression nesting too deep. |
| 5 RECURSIVE EXTERNAL CALL | NT | An external subroutine attempted to call itself (XCALL statement) either directly or indirectly through another external subroutine. |
| 6 INCORRECT NUMBER OF ARGS | NT | The number of arguments passed to an external subroutine by an XCALL statement is not equal to the number of arguments specified under the subroutine's SUBROUTINE statement. |
| 7 SUBSCRIPT ERROR | NT | The subscript value of a singly subscripted variable specified a data element outside the program's data area, or was zero or negative.<br><br>The first subscript of a doubly subscripted variable specified a value greater than that of the second subscript; one of the subscripts was a negative value; or the subscript specified a data element outside the program's data area. |

## CTS-300 Run-Time Error Messages

| Number/Message | Type | Meaning |
|---|---|---|
| 8 WRITING INTO A LITERAL | NT | Attempt by an external subroutine to store data in a literal passed as an argument to the subroutine call. |
| 9 NOT ENOUGH MEMORY | T | Insufficient memory for the device buffer and/or the device handler being loaded:<br>● In response to an OPEN statement.<br>● In response to a call to the DELET external subroutine.<br>Insufficient memory for the message being sent (SEND statement). |
| 10 ILLEGAL CHANNEL NUMBER | NT | The channel number specified in an OPEN, ACCEPT, DISPLAY, FORMS, WRITE, WRITES, READ, or READS statement is out of the range for legal channel numbers (1-15). |
| 11 CHANNEL NOT OPEN | NT | Attempt to perform I/O to a channel prior to issuing an OPEN statement. |
| 12 INPUT FROM WRITE-ONLY DEVICE | T | Attempt to issue an OPEN statement in input (I) mode to an output-only device (for example, a line printer). |
| 13 CHANNEL DEFINITION ERROR | NT | The operating system detected a logic error in the run-time system software. Notify your DIGITAL representative. |
| 14 UNDEFINED OPCODE | NT | The run-time system detected a compiler-generated error in your program. Notify your DIGITAL representative. |
| 15 NUMBER TOO LONG | T | The program performed a calculation producing a result greater than 18 digits.<br>An INCR statement caused a number to exceed the specified size of its data field. |
| 16 DIBOL CHANNEL IN USE | NT | Attempt to issue an OPEN statement to a channel that is already open. |
| 17 BAD FILE SPECIFICATION | T | Incorrect syntax in the file specification for LPQUE, OPEN, or SEND statement, or in an external call to RENAME. |

## CTS-300 Run-Time Error Messages

| Number/Message | Type | Meaning |
|---|---|---|
| 18 FILE NOT FOUND | T | The file specified in an OPEN statement does not exist. |
| 19 HANDLER NOT AVAILABLE | T | The handler for the device specified in the file specification of an OPEN statement is not installed and/or loaded. |
| 20 BAD DIGIT | T | The alphanumeric value being converted to a decimal value consisted of characters other than: space, +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. |
| 21 BAD OPEN | T | Attempt to issue an ACCEPT, READ, or READS statement to an output-only device (for example, line printer, paper tape punch).<br><br>Attempt to issue a DISPLAY, WRITE, FORMS, or WRITES to an input-only device (for example, paper tape reader).<br><br>Attempt to open a file of length zero in I or U mode.<br><br>Attempt to perform other than a READ or a WRITE in U mode. |
| 22 I-O ERROR | T | The system detected bad data during an input or output operation. This may be either a software or a hardware error. If the operation being performed is sequential (READS or WRITES), attempts to retry may cause loss or duplication of data. |
| 23 LINE TOO LONG | T | The data record being input is larger than the space reserved for it in memory by the associated RECORD statement in the Data Division. |
| 24 NO SPACE FOR FILE | T | The number of blocks of storage requested in the file specification of an OPEN statement is not available on this device.<br><br>The device cannot contain more files, because its directory is full. On TSD systems, squeezing the disk may relieve the problem. |

| Number/Message | Type | Meaning |
|---|---|---|
| 25 OUTPUT FILE FULL | T | This message is output when the operator types a CTRL/Z at the terminal in response to the message MOUNT SUCCESSOR TO dev:filnam.ext FOR OUTPUT. It indicates that:<br><br>The blocks of storage requested in the file specification of the OPEN statement are used.<br><br>No further storage space is available on this device. |
| 26 FIELD OR RECORD TOO LONG | NT | The size of a record or field is greater than 16,383 characters. |
| 27 UPDATE OF NON-FILE DEVICE | T | Attempt to issue an OPEN statement in update mode to a nonrandom access device (that is: papertape, line printer, magnetic tape, or terminal). |
| 28 ILLEGAL RECORD NUMBER | T | A READ or WRITE statement specified a record number argument that is negative, zero, or greater than the number of records in the file. |
| 29 INCOMPATIBLE COMPILER | NT | The program was compiled using a version of the compiler that is incompatible with the run-time system. |
| 30 DIVIDE BY 0 | T | The program performed an arithmetic operation that resulted in division by zero. |
| 31 ARGUMENT WRONG SIZE | NT | Attempt to call one of the external utility subroutines supplied by DIGITAL using one or more arguments of the wrong length. |
| 32 SUPERSEDING EXISTING FILE | T | Indicates that the program attempted to create an output file having the same name as an existing file. This error will always occur with magnetic tape. With disks this condition is detected only if set up by the FLAGS external utility subroutine (see the DIBOL-11 Language Reference Manual). |

CTS-300 Run-Time Error Messages

| Number/Message | Type | Meaning |
|---|---|---|
| 33 TOO MANY CHANNELS OPEN | NT | A program attempted to open more I/O channels than the Time-Shared DIBOL supervisor allows. This limit is set during system generation. This error occurs while executing an OPEN statement. |
| 34 TOO MANY HANDLERS CALLED | NT | A program attempted to use more devices than the Time-Shared DIBOL supervisor allows. This limit is set during CTS-300 system generation. This error occurs while executing an OPEN statement. |
| 35 TOO MANY FILES OPENED | NT | The maximum number of files that Time-Shared DIBOL allows to be open simultaneously was exceeded. |
| 36 TOO MANY BUFFERS ALLOCATED | NT | The maximum number of buffers that Time-Shared DIBOL allows was exceeded. |
| 37 DEVICE IN USE | T | A program operating under Time-Shared DIBOL attempted to open a channel to a nonsharable I/O device that was being used by another program. This error is detected during execution of an OPEN statement. |
| 38 FILE IN USE | T | A program operating under Time-Shared DIBOL attempted to open an output file (O mode) that was already opened by another program. |
| 39 OUTPUT TO READ-ONLY DEVICE | T | Attempt to issue an OPEN statement in output (O mode) to an input-only device (that is: paper tape reader). |
| 40 RECORD LOCKED | T | A READ or WRITE statement specified a record that is currently being accessed by another program running under Time-Shared DIBOL. |
| 41 ?M-ILL USR | NT | Indicates an internal problem with the system software that is beyond your control. Notify your DIGITAL representative. |

| Number/Message | Type | Meaning |
|---|---|---|
| 42 ?M-NO DEV | NT | Attempt to use the line printer while the single-user line printer spooler is running.<br><br>Indicates an internal problem with the system software that is beyond your control. Notify your DIGITAL representative. |
| 43 ?M-DIR IO ERR | T | Hardware error:<br>● Device write-locked.<br>● Device either cannot read or cannot write in its directory. If condition persists, notify your DIGITAL representative. |
| 44 ?M-BAD FETCH | NT | Error while loading a device handler. The file cannot be read. If condition persists, notify your DIGITAL representative. |
| 45 ?M-OVLY ERR | NT | Error while reading DIBOL-11 program overlay into memory. This is a hardware error. If condition persists, notify your DIGITAL representative. |
| 46 ?M-DIR OVFLO | T | During execution of an OPEN statement (O mode) no entry space for the new file was available in the device directory. |
| 47 ?M-ILL ADDR | NT | Indicates an internal problem with the CTS-300 software. Notify your DIGITAL representative. |
| 48 ?M-ILL CHAN | NT | Indicates an internal problem with the CTS-300 software. Notify your DIGITAL representative. |
| 49 ?M-ILL EMT | NT | Indicates an internal problem with the CTS-300 software. Notify your DIGITAL representative. |
| 50 UNRECOVERABLE SYSTEM ERROR | NT | Indicates an internal problem with the system software that is beyond your control. Notify your DIGITAL representative. |

## CTS-300 Run-Time Error Messages

| Number/Message | Type | Meaning |
|---|---|---|
| 51 TOO MANY MESSAGES | T | The total size of all messages is more than 16 K bytes, there are too many messages, or the size of this message is too large. In all cases, the message is not stored. |
| 52 ILLEGAL KEY | T | An ISAM key was specified that was not defined in the data section of the program. That is, the key position was wrong. This message occurs with the READ, NEXT, STORE, and WRITE statements in conjuction with ISAM. |
| 53 KEY NOT SAME | T | This message can occur while doing a DELETE or a WRITE:<br>● This message occurs with a READ if the size of the key is equal to the size of the data file key but the system can not find the key specified. The next higher approximate key is returned along with this message.<br>● This message occurs while doing a DELETE or a WRITE if the key value identified within the record to be deleted or updated is not the same as the key of the last record read. This last record is still locked and the DELETE or WRITE is not performed. |
| 54 NO DUPLICATES | T | In an ISAM file, a record with a duplicate key was added to a file where duplicate keys are not allowed. |
| 55 NO ISAM CTSGENED | NT | During CTSGEN, a negative answer was specified to DO YOU WANT ISAM? Therefore, if SI or SU is used with the OPEN statement, this message will occur. |
| 56 NOT ISAM FILE | T | The specified file was not an ISAM file and was OPENed in SI or SU mode. The message occurs with DELETE and STORE statements. |
| 57 OVERFLOW FULL | T | In an ISAM file, there is not enough room for another record to be added (that is, there are no empty overflow groups or load exclusion spaces). |

## CTS-300 Run-Time Error Messages

| Number/Message | Type | Meaning |
|---|---|---|
| 58 JOB STARTUP ERROR | T | There is not enough memory, the terminal is busy, or an I/O error occured while loading the job. |
| 59 ILLEGAL TERMINAL NUMBER | T | You are either sending a message specifying a terminal number that does not exist or you are trying to start up a job on a terminal that does not exist. |
| 66 R6 STACK OVERFLOW | NT | An attempt was made to use imbedded CALLs, or a routine calls itself. |
| 68 NOT ENOUGH MEMORY TO RUN PROGRAM | NT | Program cannot be run, due to insufficient addressable memory space. In Time-Shared DIBOL, the size of the program is greater than available free memory. |
| 69 PROGRAM NOT LINKED WITH /B:100000 ? | NT | A program is linked at an octal address other than 100000. |
| 70 BAD BINARY FILE | NT | The data is corrupted, or this is not a DIBOL file. |
| 71 ILLEGAL DEVICE | NT | In XMTSD, the device handler is either not installed or is installed and not loaded. In nonextended memory TSD, a legal device name has been specified but the handler is not installed in the system. |
| 72 JOB TABLES FULL | T | You have exceeded the maximum number of jobs specified when you did your CTSGEN for your configuration. |
| 73 JOB EXCEEDS MAXIMUM SIZE | T | The job has exceeded the maximum permissible size. In TSD or XMTSD the maximum addressable amount of memory is 16 K words (32 K bytes). In SUD maximum size is limited by available free memory. |
| 74 TOO MANY ISAM VOLUMES PER FILE | NT | The maximum number of volumes per file that you specified in CTSGEN has been exceeded. |
| 75 TOO MANY ISAM FILES | NT | The number you specified in CTSGEN has just been exceeded. |
| 76 TOO MANY FILES OPENED IN U-MODE | NT | The number you specified in CTSGEN has just been exceeded. |

## CTS-300 Run-Time Error Messages

| Number/Message | Type | Meaning |
|---|---|---|
| 77 ARGUMENTS OUT OF ORDER | T | The order of the arguments in the call to the PAK/UNPAK subroutine does not match the record definition. |
| 78 ARGUMENT OUT OF RECORD LIMIT | T | An argument passed in the call is not part of the record being packed or unpacked. |
| 79 ARGUMENT COUNT | T | A minimum of three arguments is required by the PAK subroutine. |
| 80 FIELD NOT PACKED | T | The UNPAK subroutine is asked to unpack a record that was not previously packed. |
| 81 FB INIT ERROR | NT | A program has attempted to use a system communication feature that is reserved for exclusive use by XMTSD. |
| 82 ILLEGAL XCALL | NT | A call to the GLINE subroutine is invalid from a time-shared program. |

# APPENDIX B

# ERROR MESSAGES

The following tables B-1, B-2, and B-4 through B-11 present the error information for the CTS-300 Utility Programs. Table B-3 contains errors generated by a DIBOL Time-Shared RTS program.

All error messages are in blue print.

**Table B-1**
**DKED Error Messages**

Advance line finds end of page
The advance line function moved the cursor to the top of the page. Use any valid function or command.

Arrow command finds extremity of page
The meaning depends on the particular arrow function:

- The uparrow function has moved the cursor to the bottom of the page.
- The leftarrow function has moved the cursor to the beginning (first character position) of the page.
- The downarrow function has moved the cursor to the bottom of the page.
- The rightarrow has moved the cursor to the end (last character position) of the page.

Use any valid function or command.

Backup line finds beginning of page
The backup function moved the cursor to the beginning of the page. Use any valid function or command.

Bad digit in repeat count
A character other than a digit was entered as part of the repeat count.

Buffer is full
The text buffer is full. Keyboard data is no longer accepted. Use EXIT, REOPN, or PAGE to continue entering data.

Change case finds beginning of page
In the backup direction mode, the change case function changed the case of the first character in the page. Use any valid function or command.

Change case finds end of page
In the advance direction mode, the change case function changed the case of the last character in the page. Use any valid function or command.

CTRL/C ignored use quit
DKED ignores a single CTRL/C. Use any valid function or command.

Cursor not at target
DKED could not complete a REPLACE or SUBSTITUTE function because either the cursor is not at the search model or you have not specified a search model.

Table B-1 (Cont.)
DKED Error Messages

DELETE finds end of page
The DELETE function deleted the last character in the page. Use any valid function or command.

DELETE finds beginning of page
The DELETE function erased the first character in the page. Use any valid function or command.

Dot (.) cannot be 1st chr of target
When you are specifying wildcards in a search model, the first character can not be a dot.

Dot (.) cannot be last chr of target
When you are specifying wildcards in a search model, the last character can not be a dot.

EOL finds end of page
The end-of-line function moved the cursor to the end of the page. Use any valid function or command.

EOL finds beginning of page
The end-of-line function moved the cursor to the beginning of the page. Use any valid function or command.

Illegal command
DKED could not recognize the command you specified.

Illegal command sequence. Please start over.
This error occurs when you open a file that already exists; you elect to continue and then issue a QUIT command.

Illegal function for mode of use
This error occurs in inspection use mode. When you are inspecting a file, you can not enter commands or perform functions that modify the file.

Invalid terminating key
DKED could not complete a FIND function for one of the following reasons:

- You included an illegal character in the model.
- You did not terminate the model with the ADVANCE or BACKUP function.

More than 130 chars./record
You entered more than 130 characters since the previous carriage return / line feed.

**Table B-1 (Cont.)**
**DKED Error Messages**

Move to bottom when at bottom
You have attempted to move to the bottom of the page when you are already there.

Move to top when at top
You have attempted to move to the top of the page when you are already there.

No search model defined
DKED could not process the FIND-NEXT, REPLACE, or SUBSTITUTE function because you have not yet specified a search model.

NOT ENOUGH ROOM IN OUTPUT FILE - PLEASE START OVER
You have opened a file and have specified a value for number of blocks that results in a file smaller than the input, and then you exited or paged.

There are not enough blocks in the output file, or intermediate output file, you have specified.

There is no room on the physical device for the output file or intermediate output file.

Nothing to undelete
DKED could not complete the undelete function because the corresponding line or character buffer is empty.

No valid select range
There are three circumstances under which this message can occur:

- You tried to CUT or APPEND before using the SELECT function.
- You have modified the file after using the SELECT function.
- You used either a PAGE or a YANK after the SELECT RANGE function.

Repeat = 0 ??
Repeat count can not be a "0".

Section finds beginning of page
The section function moved the cursor to the beginning of the page. Use any valid function or command.

Section finds end of page
The section function moved the cursor to the end of the page. Use any valid function or command

Asterisk (*) cannot be 1st chr of target
When you are specifying wildcards in a search model, the first character can not be an asterisk.

Table B-1 (Cont.)
DKED Error Messages

| | |
|---|---|
| Asterisk (*) cannot be last chr of target | When you are specifying wildcards in a search model, the last character can not be an asterisk. |
| Target not found | A FIND or FINDNEXT function moved the cursor to the bottom or top of the page (or file if p option used in forward mode) without finding a string that matches the model specified. |
| Text buffer is filling up | DKED has detected that the text buffer is close to being filled. Use EXIT, REOPN, or PAGE to continue entering data. |
| Unable to open input file | The file specified was not found. |

Table B-2
DICOMP Error Messages

In the following table the errors are identified as to type:

E = error, compilation not stopped

F = fatal error, compilation stops

W = warning, program will run, but unpredictably

| Message | Type | Meaning |
|---|---|---|
| BAD DUMMY VARIABLE SIZE | W | The size of a dummy variable should be 0. To prevent the warning message, the dimension should be specified as 0 or omitted altogether. |
| BAD IF STMNT | E | A START, END, or illegal statement was detected within an IF statement, or no statement was found on the right side of the IF statement. |
| BAD LEFT SIDE OF '=' | E | No variable was found on the left side of "=", or it was not a legal variable. |
| BAD LPQUE STMNT | E | A colon was missing, or the order of arguments was incorrect in an LPQUE statement. |
| BAD OVERLAY | E | There was no previously defined record to be overlaid. |
| BAD PROC # | E | The left parenthesis was missing, or n was greater than 16. The statement must be of the form PROC (n). |
| BAD SWITCH | F | The compiler detected an unrecognized switch. The compiler is restarted. The switches are: W,B,S,D,O,A,C,P:N,G, and L |
| CCP ERROR | E | Every .IFDEF and .IFNDEF must have a corresponding .ENDC for each level of nesting. |
| COMMA MISSING | E | No comma was found where it was expected. |
| CREF I/O ERROR | F | A hardware error occurred while writing the CREF output file. The system returns to monitor command level. |

Table B-2 (Cont.)
DICOMP Error Messages

| Message | Type | Meaning |
|---|---|---|
| CREF OUT OF ROOM | F | The space allocated for the temporary CREF output file was completely filled. |
| DECIMAL FIELD >18 CHARS | W | The legal limit of a decimal field for run time is eighteen (18) characters. |
| EXPECTED LABEL MISSING | E | A label was not found where it was expected or required. |
| EXPRESSION NOT DECIMAL | E | The expression is not defined, or it is in alphabetic format. |
| EXTRA CHARS AT STMNT END | E | This is an error in the PROC section. |
| | W | This becomes a warning in the DATA section. |
| FIELD TOO LARGE OR ZERO | E | A field exceeded 16,383 bytes (characters). |
| FILE NOT FOUND | F | The compiler could not find the file(s) specified in the command line. The compiler is restarted. |
| ILLEGAL COMMAND | F | Only two output files may be specified in a compiler command string. The compiler is restarted. |
| ILLEGAL DEVICE | F | A device specified in the compiler command string was not recognized as a legal device. |
| ILLEGAL FIRST STMNT | E | The only allowable initial statements in a program segment are the RECORD, SUBROUTINE, START, and CCP statements. |
| ILLEGAL NAME | E | The first character in a name was not an alphabetic character. |
| ILLEGAL OR MISSING OPERATOR | E | No comma appeared in the data formatting statements, or an invalid operator was found within an expression. |
| ILLEGAL STMNT | E | The statement was not recognized as a legal DIBOL statement, or the PROC statement was used more than once, or an END or START was found within an IF statement. |

Table B-2 (Cont.)
DICOMP Error Messages

| Message | Type | Meaning |
|---|---|---|
| INITIAL VALUE NOT ALLOWED | W | No fields within overlay records can be given initial values. |
| INITIAL VALUE NOT EQUAL TO SIZE | W | The field size specifier and the size of its initial value do not agree. |
| INPUT FILES? | F | The compiler command string contained no input files. |
| INPUT I/O ERROR | F | A hardware error occurred while an input file was being read. The system returns to monitor command level. |
| LINE TOO LONG | E | The line exceeded 511 characters including carriage return / line feed pairs. |
| LISTING I/O ERROR | F | A hardware error occurred while writing the listing file. The system returns to monitor command level. |
| LISTING OUT OF ROOM | F | The space allocated for the listing file was completely filled. |
| MISSING CLOSE PAREN | E | Not every left parenthesis used within an expression has a corresponding right parenthesis. |
| MISSING OPERAND | E | An expected variable or literal was missing. |
| MISSING QUOTE | E | A quote (') was not found where it was expected. |
| NAME PREVIOUSLY DEFINED | E | Multiple definition of a field or record name is not allowed. |
| NAME TOO LONG | W | A name with more than six characters, or a subroutine name with more than five characters, was detected. The excess characters are ignored. |
| NO FIELD STMNTS IN PREVIOUS RECORD | E | A record size of zero was detected. The problem occurs because of the lack of, or the improper definition of, field statements. |

Table B-2 (Cont.)
DICOMP Error Messages

| Message | Type | Meaning |
|---|---|---|
| NO LISTING FILE FOR CREF | F | A CREF listing was specified in the compiler command string via the /C switch, but no listing file specification was included in the same command string. The compiler is restarted. |
| NOT A OR D | E | A field specification can be only A (alphabetic) or D (decimal). |
| NOT DECIMAL | E | The initial value of a decimal field was not specified in decimal format. |
| NOT ENOUGH MEMORY | F | There was not enough memory to compile the program. Reduce either the number of output files and different device handlers, or the number of symbols in the program. The system returns to monitor command level. |
| NOT I, O, OR U | E | The second argument of an OPEN statement does not begin with an I, O, U, or S. |
| OBJ I/O ERROR | F | A hardware error occurred while writing the object file. The system returns to monitor command level. |
| OBJ OUT OF ROOM | F | The space allocated for the object file was completely filled. |
| RECORD TOO BIG | E | The named record exceeded 16,383 bytes (characters), or an overlay exceeded the size of the record it was overlaying. |
| RELATIONAL NOT IN IF STMNT | W | A relational operator appears outside an IF statement. |
| STMNT TOO COMPLEX | E | Internal compiler storage was exceeded in an attempt to compile too complex an expression. Break the expression into two or more statements. |
| SUBSCRIPT ERROR | E | The specified subscripts are alphabetic, or are not separated by a comma, or are not closed by a right parenthesis, or total more than two subscripts. |
| SUBSCRIPT STACK OVERFLOW | E | Internal compiler storage was exceeded in an attempt to compile too many nested subscripts. |

Table B-2 (Cont.)
DICOMP Error Messages

| Message | Type | Meaning |
|---|---|---|
| SUBROUTINE NAME MISSING | E | The name of the subroutine in a SUBROUTINE or XCALL statement is missing. |
| TOO MANY SYMBOLS OR LABELS | F | More than 1023 symbols, or more than 1023 labels occurred in a program segment. |
| TOO MUCH DATA | E | More than 32,767 bytes of data and literals were found in the program. |
| UNDEFINED LABEL | E | An unidentified label was referenced in the PROC section. |
| UNDEFINED NAME | E | A variable not defined in the DATA section was encountered in the PROC section. |
| WRONG DATA TYPE | E | An expression contains incompatible data types: alphabetic and decimal. |

Table B-3
Time-Shared DIBOL Error Messages

PROGRAM NOT DETACHED
An attempt was made to issue an ATTACH command to a program that was currently attached to another terminal.

PROGRAM NOT FOUND
The program specified in an ATTACH or RUN command does not reside on the specified device.

TRAP TO 4 PC=
A time-out or bus error has occurred, there has been an attempt to access nonexistent memory, or a word instruction was attempted on an odd address. Consult your DIGITAL hardware/software specialist.

TRAP TO 10 PC=
An attempt to execute an undefined instruction has occurred. Consult your DIGITAL hardware/software specialist.

TSD INITIALIZATION ERROR CANNOT RUN IN FOREGROUND PARTITION
An attempt to run nonextended memory TSD in the foreground partition was made. Control returns to the RT-11 monitor.

XM-TSD INITIALIZATION ERROR CANNOT CREATE EXTENDED MEMORY REGION
An attempt was made to run XMTSD with the SJ or FB monitor.

Table B-4
Foreground/Background Communication Command
Error Messages

| Message | Meaning |
|---|---|
| BACKGROUND LISTENER NOT RUNNING REQUEST DISCARDED filespec | The special background listener program is not running, so the request from the foreground has to be rejected. The file discarded is identified by filespec. |
| FILE NOT FOUND | The indirect file specified in the request from the foreground was not found. |
| HANDLER NOT FOUND | In processing the indirect file from the foreground, the background has found a necessary device handler missing. |
| ILLEGAL BACKGROUND OPERATION | A communication command was issued when XMTSD is running in the background. |
| QUEUE FULL-ENTRY REJECTED filespec | The job queue is full, so the request has to be rejected. The file rejected is identified by filespec. |
| QUEUE IS EMPTY | There are no jobs in the queue for background processing. This is not considered an error message when in response to a SHOW command. |

Table B-5
DDT Error Messages

| Message | Meaning |
|---|---|
| LITERAL? | The variable name entered appears to be a literal. |
| NO SUCH VARIABLE | There is no variable by the name entered. |
| SUBSCRIPT ERROR | You have attempted to manipulate a variable using subscripts to access part of a field and you have not conformed to the rules for subscripts. See the **DIBOL-11 Language Reference Manual.** |
| TOO MANY BREAKPOINTS | There may be only one breakpoint in the main program and in each of the subscripts. A maximum of eight breakpoints total is allowed at any one time. |
| WHAT? | The number was not specified in an iteration command. |
| | You specified a routine name in which to set a breakpoint but did not follow it with a colon. |
| | A decimal value was expected but was not entered. |

Table B-6
Spooler Error Messages (LPTSP1.REL)

| **Message** | Meaning |
|---|---|
| FATAL LPQUE ERROR | The spooler detected an error in the file being read. |
| ?M-NO DEV | The device handler for the device containing the file to be printed was not loaded when the spooler was initialized. |
| ?PLEASE LOAD LINE PRINTER HANDLER | The spooler was run before the line printer handler was loaded. |

Table B-7
Spooler Error Messages (LPTSPL.TSD)

| **Message** | Meaning |
|---|---|
| BAD LPQFIL.LPQ | This message occurs under the following conditions:<br>● The spooler attempts to create this file and the OPEN fails.<br>● The file already exists and the spooler attempts to open it in update mode and the OPEN fails.<br>● The file exists and a failure occurs while doing I/O to the file. |
| ERROR OPENING dev:filnam.ext | The queue file entry specified a nonexistent file (dev:filenam.ext), or a nonexistent or illegal I/O device. Processing continues with the next queue file entry. |
| I/O ERROR dev:filnam.ext | An error occurred while the spooler was reading the file (dev:filnam.ext) currently being printed. A YES response to the resulting message CONTINUE PRINTING filename? will reprint the file. Otherwise, processing continues with the next queue entry. |
| LP HUNG-FIX & TYPE CR | The line printer is in an off-line state. This could be because it ran out of paper, was placed off line, or possibly because of a malfunction. The record that was being printed is lost. Correcting the problem and typing carriage return will resume printing with the next record. |

Table B-7 (Cont.)
Spooler Error Messages (LPTSPL.TSD)

| Message | Meaning |
|---|---|
| PRINTER n NOT FREE | The printer handler is not loaded or a job has just been submitted to a satellite which attempts to open a printer. The printer is found to be already in use. The satellite program terminates and the spooler searches for another job to print. When the spooler returns to this point in the queue, another attempt will be made to print the file. |
| SATELLITE L \| P/Q/R/S \| SAT CAN NOT RUN | Failure to CTSGEN for forced job startup can cause this error message. It may also indicate that a satellite program is already running although, it is unrecognized by the spooler, or that the spooler is attempting a forced job startup on an already active satellite. Kill that job via the STATUS option Kx and bring the TSD system down and restart it. |

Table B-8
PRINTU Error Messages

The number near the error message (indicated by a pound sign) indicates the character position nearest where the error occurred.

| Message | Meaning |
|---|---|
| ALPHA LITERAL REQUIRED # | Expected alpha literal is missing. |
| ALREADY DEFINED | An attempt was made to name a field in the INPUT or COMPUTE section with a name that was previously used. |
| HEADER IS TOO LONG | The header line exceeds 132 characters. |
| IMPROPER DEFINITION # | Filler item in the PRINT section is used incorrectly. |
| IMPROPER LITERAL # | Literal too large |
| IMPROPER USE OF DECIMAL PLACES # | The number of decimal places exceeds the size of the field being defined. |
| INTEGER FROM 1-15 REQUIRED # | The expected decimal field is out of range. |
| INTEGER FROM 1-132 REQUIRED # | The expected decimal field is out of range. |
| INTEGER REQUIRED # | Integer missing where expected. |
| LITERAL TOO LONG # | Field description exceeds 30 characters. |
| MUST BE IDENT # | The first section in the control file must be IDENT. |
| MUST BE NUMERIC ITEM # | An item expected to be numeric is defined incorrectly. |
| MUST BE SU OR IS | SU and IS are the only legal options in the INPUT statement that follow the comma. |
| NEED FILE NAME | File name missing from the IDENT statement. |
| NO ENDING QUOTE # | No closing quote for a HEAD1, HEAD2, or for the text portion in a PRINT statement. |
| NO INPUT DIRECTIVE | The INPUT statement is missing. |

Table B-8 (Cont.)
PRINTU Error Messages

| Message | Meaning |
|---|---|
| NO PRINT ITEMS | No fields are specified following the PRINT statement. |
| NOT DEFINED | An attempt was made to print a field that has not been defined. |
| NOT ENOUGH RIGHT PARENTHESES # | A statement in the COMPUTE section has too few right parentheses. |
| PICTURE TOO LONG | The picture, or edit mask, for printing exceeds 22 characters. |
| SYNTAX ERROR # | Statement contains illegal characters or options. |
| TOO MANY COLUMNS IN REPORT | More than 132 columns under HEAD1, HEAD2, or PRINT sections. |
| TOO MANY COMPUTE STATEMENTS | More than eight COMPUTE statements were specified. |
| TOO MANY DATA ITEMS | More than 20 data items in the INPUT section. |
| TOO MANY LEFT PARENTHESES | A statement in the COMPUTE section is too complicated to be handled by PRINTU. |
| TOO MANY LIST ITEMS | More than 20 data items in the INPUT section. |
| TOO MANY RIGHT PARENTHESES | A statement in the COMPUTE section has too many right parentheses. |
| UNKNOWN DIRECTIVE | An invalid section statement. Only IDENT, HEAD1, HEAD2, INPUT, COMPUTE, PRINT, and END are legal. |

Table B-9
ISMUTL Error Messages

| Message | Meaning |
|---|---|
| ALL FILES ALLOCATED AND MORE SPACE IS REQUIRED PLEASE TRY AGAIN | The required disk space was not allocated within the limit of seven data files. |
| BAD FILE SPECIFICATION | Something other than a specification of the form DEV:FILENA.EXT was given, or the specification contains an invalid element. (The extension is not necessary when specifying an output file.) |
| DUPLICATE KEYS FOUND | During CREATE, the input file was found to contain records with duplicate keys when the answer had been NO to ALLOW DUPLICATE KEYS. |
| FILE ALREADY EXISTS --- REPLACE? (YES/NO) | An existing file name was specified for an output file. |
| FUNCTION REQUIRES ISAM INPUT FILE | During a REORG or STATUS function, a file other than an ISAM file was specified as the input file. |
| MORE INPUT RECORDS THAN SPECIFIED - RESTART FUNCTION | The number of records in the input file is greater than the number specified during the ISAM file CREATE. Restart the CREATE and specify a number for input records equal to, or greater than, the number in the input file. This message may also occur during a REORG if the number of records specified by the Index File Control Group is less than the number of records in the ISAM input file. In this situation a restart will not help. You must build a sequential file from the corrupted ISAM file and use this as input for a new ISAM CREATE. |
| NO SPACE FOR FILE | While running REORG, the number of blocks of storage requested is not available on the device, or the device directory is full. |
| NOT ENOUGH WORKFILE SPACE ALLOCATED | While running REORG in CHAIN mode, there was not enough work space available to perform the function. Control is returned to the monitor. |
| OUT OF RANGE | A number larger than the limit was used during the CREATE or REORG dialog. |
| PLEASE TRY AGAIN | Neither a Y nor an N was given as response to a YES/NO question. |

Table B-10
SORTG Error Messages

| Message | Meaning |
|---|---|
| EXECUTE STATEMENT ERROR | The EXECUTE statement does not contain a valid file specification. |
| FILENAME MISSING | An INPUT or OUTPUT statement has no file name. |
| INPUT: STATEMENT MISSING | No INPUT statement exists. |
| INSUFFICIENT SPACE | The main memory allocation is too small to support the record size. You must either increase main memory allocation, decrease the number of work files, or change to a TAGSORT.<br><br>The SORT/MERGE internal buffer is slightly smaller than the maximum DIBOL record size permitted. This error message could be caused by record sizes which are larger than the limits of the work area. In this case, a TAGSORT could be used. |
| INVALID INPUT RECORD COUNT | The record count for an input file in the INPUT statement is not valid. |
| KEYS: STATEMENT MISSING | No KEYS statement exists. |
| MULTIPLE CONTROL STATEMENT | More than one control statement has the same identifier. |
| OUTPUT: STATEMENT MISSING | No OUTPUT statement exists. |
| RECORD HAS NO DATA ITEMS | The RECORD statement does not contain any field definitions. |
| RECORD: STATEMENT MISSING | No RECORD statement exists. |
| SIZE DEFINITION ERROR | A field definition in the RECORD statement describes the field as having an invalid length. |
| SORT KEY MISSING | The KEYS statement has no control key names, or ends with a comma. |

Table B-10 (Cont.)
SORTG Error Messages

| Message | Meaning |
|---|---|
| SORT MODE FILENAME CONFLICT | An attempt was made to write the file into itself when:<br>• The input file is an ISAM file.<br>• A file merge is being done.<br>• TAGSORTS that do not produce a re-ordered file (i.e., INDEX or LIST) are being done. |
| TAG SORT TYPE INVALID | A tag other than SORT, LIST, or INDEX has been used and is not recognized. |
| TOO MANY MERGE FILES | The INPUT statement has more than seven files specified. |
| TOO MANY SORT KEYS | The KEYS statement contains more than eight control key names. |
| TYPE DEFINITION ERROR | A field definition in the RECORD statement describes the field to be other than alphabetic or decimal. |
| UNRECOGNIZED STATEMENT | SORTG was unable to identify the meaning of the line. |

Table B-11
SORTM Error Messages

| Message | Meaning |
|---|---|
| END OF SORT RECIN-xxxxx RECOUT=yyyyy | Possibly due to a bad I/O transfer, the number of input records does not agree with the number of output records. In this case, xxxxx will be the count of input records, and yyyyy will be the count of output records. |
| HANDLER NOT AVAILABLE | An unassigned device name has been specified. You must assign the logical device name to a physical device. |
| OPEN ERROR FREE SPACE MARGINAL | SORT/MERGE cannot find enough disk space to allocate the files. |
| WRONG RECORD SIZE | An attempt was made to specify a file name with an improper record description. |

Table B-12
STATUS Error Messages (STATUS.TSD)

| **Message** | Meaning |
|---|---|
| BAD SPECIFICATION | A printer number for a nonexistant printer has been specified. |
| LP IN USE | A number between 1 and 4 has been entered; however, that line printer is busy. |
| NO JOB BY THAT NUMBER | A job number for a nonexistant job has been entered. |
| NO SUCH OPTION | A letter other than a valid option letter has been entered. |

Table B-13
REDUCE Error Messages

| Message | Meaning |
|---|---|
| ?REDUCE-F-ERROR READING DIRECTORY | The directory is nonexistent or damaged. This is often a hardware error. |
| ?REDUCE-F-ERROR READING FILE | The input file cannot be read. This is usually a hardware error. |
| ?REDUCE-F-ERROR WRITING FILE | The file cannot be written. This is usually a hardware error or a WRITE-LOCKED device. |
| ?REDUCE-I-FILE INCORRECTLY LINKED DEV:FILENA.EXT | An input file linked for a base address of other than 100000 has been specified. |
| ?REDUCE-I-IMPROPER BASE ADDRESS IN OVERLAID FILE DEV:FILENA.EXT | An input file linked for a base address other than 100000 has been specified. |
| ?REDUCE-I-INCORRECT RELATIVE BLOCK NUMBERS IN OVERLAID FILE DEV:FILENA.EXT | A library .OBJ file has been linked into an overlay region. |
| ?REDUCE-I-FILE-PREVIOUSLY REDUCED DEV:FILENA.EXT | An input file that has already been REDUCED has been specified. |
| ?REDUCE-W-FOREIGN HANDLER | A device handler unknown to REDUCE has been specified.<br><br>A nonfile-structured device has been assigned a file-structured name. |
| ?REDUCE-W-ILLEGAL COMMAND | This is a syntax error; usually a typographical error. |
| ?REDUCE-F-ILLEGAL DIRECTORY | A device with a non-RT-11 directory has been specified for input. |
| ?REDUCE-W-ILLEGAL OPTION "/OPT" | An option other than /N or /V has been entered. |
| ?REDUCE-F-INSUFFICIENT MEMORY | There is not enough memory on the system to support REDUCE and its associated data. |
| ?REDUCE-F-INSUFFICIENT ROOM IN DIRECTORY OR ON DEVICE | The output device specified cannot contain the file. |

Table B-13 (Cont.)
REDUCE Error Messages

| Message | Meaning |
|---|---|
| ?REDUCE-I-VERNUM | This message is displayed with the current version number (VERNUM) whenever the /V option is specified in the command line. |
| ?REDUCE-W-SPECIFIED INPUT FILE(S) NOT FOUND | The input file is not on the device specified. This can also be caused by a null line. |
| ?REDUCE-W-UNSUPPORTED DEVICE: "DEV" | A nonfile-structured device has been specified. |
| ?REDUCE-W-[+2K MEMORY] | This notes that, due to processing needs, the USR is forced to swap to obtain the necessary processing space for REDUCE. |

# INDEX

## READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Please cut along this line.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________

Organization ______________________

Street ______________________

City ______________ State ____________ Zip Code or Country ____________

— — — —Do Not Tear - Fold Here and Tape — — — — — — — — — — — — — — — — — — —

digital

No Postage
Necessary
if Mailed in the
United States

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Applied Commercial Engineering MK1-2/H32
Continental Boulevard
Merrimack N.H. 03054

ATTN: Documentation Supervisor

— — — — Do Not Tear - Fold Here and Tape — — — — — — — — — — — — — — — — — — —