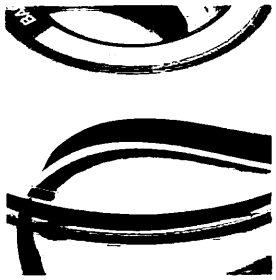
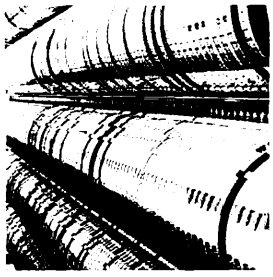
Prime Computer, Inc.

DOC10083-1LA
Subroutines Reference Guide
Volume IV












Subroutines Reference IV
Libraries and I/O

First Edition

by

Dick Frost

Updated for Rev. 22.0

by

John Breithaupt and Glenn Morrow

This guide documents the software operation of the Prime Computer and its supporting systems and utilities as implemented at Master Disk Revision Level 22.0 (Rev. 22.0).

Prime Computer, Inc.
Prime Park
Natick, Massachusetts 01760

The information in this document is subject to change without notice and should not be construed as a commitment by Prime Computer, Inc. Prime Computer, Inc., assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

PRINTING HISTORY

First Edition
Volume I (DOC10080-1LA) August 1986 for Revision 20.2
Volume II (DOC10081-1LA) August 1986 for Revision 20.2
Volume III (DOC10082-1LA) August 1986 for Revision 20.2
Volume IV (DOC10083-1LA) August 1986 for Revision 20.2
Volume V (DOC10213-1LA) August 1988 for Revision 22.0
Update 1
Volume II (UPD10081-11A) July 1987 for Revision 21.0
Volume III (UPD10082-11A) July 1987 for Revision 21.0
Volume IV (UPD10083-11A) July 1987 for Revision 21.0
Update 2
Volume II (UPD10081-12A) August 1988 for Revision 22.0
Volume III (UPD10082-12A) August 1988 for Revision 22.0
Volume IV (UPD10083-12A) August 1988 for Revision 22.0
Second Edition
Volume I (DOC10080-2LA) July 1987 for Revision 21.0

CREDITS

Project Support: Joan Karp
Editorial: Thelma Henner
Illustration: Mingling Chang
Production: Judy Gordon

HOW TO ORDER TECHNICAL DOCUMENTS

To order copies of documents, or to obtain a catalog and price list:

United States Customers	International
Call Prime Telemarketing, toll free, at 1-800-343-2533, Monday through Friday, 8:30 a.m. to 5:00 p.m. (EST).	Contact your local Prime subsidiary or distributor.

CUSTOMER SUPPORT

Prime provides the following toll-free numbers for customers in the United States needing service:

1-800-322-2838 (within Massachusetts)	1-800-541-8888 (within Alaska)
1-800-343-2320 (within other states)	1-800-651-1313 (within Hawaii)

For other locations, contact your Prime representative.

SURVEYS AND CORRESPONDENCE

Please comment on this manual using the Reader Response Form provided in the back of this book. Address any additional comments on this or other Prime documents to:

Technical Publications Department
Prime Computer, Inc.
500 Old Connecticut Path
Framingham, MA 01701

Contents

PART III -- SMLC/AMLC SUBROUTINES

PART IV -- APPLICATION LIBRARY

PART V -- SORT LIBRARIES AND
FORTRAN MATRIX LIBRARY

APPENDIXES

About This Book

The Subroutines Reference series gives a systematic description of the standard Prime subroutines and subroutine libraries. Each standard subroutine library is a file containing subroutines that perform a variety of related programming tasks. Whenever these tasks are to be performed, programmers can use the subroutines in the standard libraries instead of writing their own subroutines. Programmers need to write subroutines only to perform specialized tasks for which no standard subroutines exist.

OVERVIEW OF THIS SERIES

The Subroutines Reference consists of five volumes. A brief summary of the contents of each volume follows.

Volume I

Volume I is an introduction to the entire Subroutines Reference series. It describes the nature and functions of Prime's standard subroutines and subroutine libraries. It explains how subroutines can be called from programs written in Prime's programming languages: C, COBOL 74, FORTRAN IV, FORTRAN 77, Pascal, PL/I, BASIC V/M, and PMA.

Volume II

Volume II describes subroutines that deal with the access to and management of file system entities, the manipulation of EPFs in the execution environment, system search rules, and the use of a number of command environment functions. Three chapters describe subroutines related to the file system, one chapter describes system search rules, and one chapter each is devoted to subroutines related to EPF management and to the command environment.

Volume III

Volume III describes system subroutines. The subroutines covered in this volume are the general system calls to the operating system and standard system library. This excludes file and EPF manipulation, which are described in Volume II. Volume III also includes System Information and Metering (SIM) routines.

Volume IV

Volume IV presents several mature libraries: the Input/Output Control System (IOCS) libraries and other I/O-related subroutines, the Application libraries, the SORT libraries, and MATHLB.

IOCS provides device-independent I/O. The chapters on IOCS provide descriptions of the device-independent subroutines plus those device-dependent subroutines simplified by IOCS. Another section provides descriptions of the synchronous and asynchronous device-driver subroutines.

Sections on the Application Library, the Sort Libraries, and the FORTRAN Matrix library provide descriptions of other program development subroutines especially useful for FORTRAN programs.

Volume V

Volume V describes the event synchronization feature of PRIMOS ® and its use by two PRIMOS facilities: timers and the InterServer Communications (ISC) facility for message exchange between processes. Volume V is divided into three parts.

Part 1 provides a general overview of event synchronization.

Part 2 describes in detail how to create, destroy, and retrieve information about event synchronizers and event groups. It also describes how timers and the ISC facility use event synchronizers and event groups to synchronize user processes.

Part 3 describes in detail how to create, use, destroy, and retrieve information about timers. Timers make time-dependent process synchronization possible.

Part 4 describes in detail the ISC facility, which makes it possible for processes that are running simultaneously to exchange messages. These processes may be running on the same system or on two different systems connected by PRIMENET. Message exchange is coordinated by using event synchronizers.

SPECIFICS OF THIS VOLUME

Volume IV contains descriptions of low-level libraries that provide many useful routines. It has five parts:

I Introduction
II IOCS Library
III SMLC/AMLC Subroutines
IV Application Library
V Sort Libraries and FORTRAN Matrix Library

Part I consists of a single chapter that gives an overview of subroutines and libraries.

Chapter 1 summarizes the calling conventions for Prime subroutines and explains the format of the subroutine descriptions in this volume. It summarizes the parameter and returned-value data types used in the descriptions. It explains how to set bits in arguments, how to use keys, and how to interpret error codes. It discusses subroutine libraries and addressing modes as an introduction to the Loading and Linking Information section for each subroutine description.

Part II presents the IOCS (Input/Output Control System) Library, along with other subroutines that perform I/O. Among these libraries, for example, is a basic I/O procedure for accessing any peripheral device (terminal, printer, tape, disk, card-reader). Most of the

chores done by these subroutines are also done by the friendlier File Management subroutines described in Volume II. Nevertheless, these subroutines allow the programmer to be device-independent in performing I/O. They also allow the System Administrator to change device assignments by altering utility tables on the master disk.

Part III presents the SMLC/AMLC subroutines -- subroutines used for making assignments to the Synchronous Multiline Controller(s) and the Asynchronous Multiline Controller(s). Accompanying control block configurations are also described in this section.

Part IV presents the Application Library -- both the R-mode APPLIB and the V-mode VAPPLB. This mature user-oriented library provides a set of service routines that are designed to perform as functions but may also be called directly as subroutines. The seven categories of functions each receive a chapter for description. For example, in the chapter on String Manipulation Routines, you will find a function that converts an ASCII string to a binary string and then returns a Boolean value to indicate success or failure. Many of these functions were created to overcome the limited string manipulation capabilities of FTN; more recent languages often have these functions embedded in their instruction set.

Part V presents the Sort Libraries and the FORTRAN Matrix Libraries. Among these libraries, for example, is a subroutine to perform a bubble sort, and another for a shell sort. Many of these subroutines are better suited for system use. The programmer will often find the same functions embedded in the high level programming language being used.

The appendixes provide information about error handling, error handling for I/O subroutines, SVC, obsolete subroutines, and data type equivalents.

Three indexes enable the reader to find information quickly. These are:

- The Index of Subroutines by Function, a list of subroutines grouped by the general types of function that they perform. Use this index to find out which subroutines perform a particular function, such as controlling access to the file system.

- The Index of Subroutines, an alphabetical list of subroutines giving the volume, chapter, and page number of each subroutine. Use this index to locate the description of a particular subroutine in the Subroutines Reference.

- The Volume Index, a list of the topics treated in this volume. Use this index to find out where in this volume a particular topic, process, or term is described.

SUGGESTED REFERENCES

The PRIMOS User's Guide (DOC4130-5LA) contains information on system use, directory structure, the condition mechanism, CPL files, ACLs, global variables, and how to load and execute files with external subroutines.

The Programmer's Guide to BIND and EPFs (DOC8691-1LA) shows application programmers how to use the executable program format environment.

The Advanced Programmer's Guide, the companion to the Subroutines Reference series, consists of four volumes:

Advanced Programmer's Guide, Volume 0: Introduction and Error Codes (DOC10066-1LA)

Advanced Programmer's Guide, Volume I: BIND and EPFs (DOC10055-1LA)

Advanced Programmer's Guide, Volume II: File System (DOC10056-2LA)

Advanced Programmer's Guide, Volume III: Command Environment (DOC10057-1LA)

These volumes provide strategies for the use of subroutines by system programmers and application programmers. In addition to explanations for each error code message, the manual provides the most complete information on the use of EPFs, of file system subroutines, and of command environments.

The following related Prime publications are also available.

Operator's Guide to System Commands (DOC9304-3LA)

System Administrator's Guide, Volume I: System Configuration (DOC10131-1LA)

System Administrator's Guide, Volume II: Communication Lines and Controllers (DOC10132-1LA)

System Administrator's Guide, Volume III: System Access and Security (DOC10133-1LA)

System Architecture Reference Guide (DOC9473-2LA)

PRIME DOCUMENTATION CONVENTIONS

Subroutine descriptions use the conventions shown below. Examples illustrate use of these conventions.

Convention	Explanation	Example
UPPERCASE	In subroutine descriptions, words in uppercase indicate actual names of commands, options, statements, data types, and keywords.	FIXED BIN
lowercase	In subroutine descriptions, words in lowercase indicate variables for which you must substitute a suitable value.	key, filename
Parentheses ( )	In call statements, parentheses must be entered exactly as shown.	CALL TIMDAT(array, n)

Note

FORTRAN requires a colon (:) to indicate that octal notation follows.

Changes made to these pages since the last printing are identified by vertical bars in the margins. Each new routine in this package is marked with a bar beside the routine name, at the description's heading.

PART I

INTRODUCTION

1
Overview of Subroutines

A subroutine is a module of code that can be called from another module. It is useful for performing operations that cannot be performed by the calling language, or for performing standard operations faster. Users can write their own subroutines to supply customized or repetitive operations. However, this guide discusses only standard subroutines provided with the PRIMOS® operating system or in standard libraries.

This chapter summarizes the calling conventions for Prime subroutines and explains the format of the subroutine descriptions in this volume. It assumes that readers know a high-level language or Prime Macro Assembler (PMA), and that they are familiar with the concept of external subroutines. For more information on calling subroutines from Prime languages, see the chapter on your own language in Volume I.

FUNCTIONS AND SUBROUTINES

In this guide, a function is a call that returns a value. You call a function by using it in an expression; the function's returned value can then be assigned to a variable or used in other operations within the expression. Here, the value returned by DELE$A is assigned to the variable VALUE1:

```
VALUE1 = DELE$A(arg1, arg2);
```

A subroutine returns values only through its arguments. It is called this way:

```
CALL GV$GET(arg1, arg2, arg3, arg4);
```

However, the word subroutine is also used as the collective term for both of these modules.

SUBROUTINE DESCRIPTIONS

In this guide, each description of a subroutine contains the following sections:

- Purpose. A brief description of what the subroutine does.
- Usage. The format of a subroutine declaration and a subroutine call, using either PL/I language elements (for subroutines in V-mode, etc.) or FORTRAN language elements (for subroutines solely in R-mode). For more information, see the section SUBROUTINE USAGE later in this chapter.

  Note

  Certain subroutines in this volume are designed to aid the FORTRAN programmer in particular. Even though they exist in modes other than R-mode, they receive FORTRAN language elements in their Usage.

- Parameters. Information about the arguments the subroutine expects and the values it returns. For further information, see the section SUBROUTINE PARAMETERS later in this chapter.
- Discussion. Additional information about the subroutine and examples of its use. Not all subroutine descriptions have this section.
- Loading and Linking Information. Information about what libraries must be loaded during the linking and loading process. See the section Libraries and Addressing Modes later in this chapter for a brief discussion of modes.

Figure 1-1 shows an example of a subroutine description. The subroutine CTIM$A is in chapter 12 of this volume. Like the other subroutines in the Application libraries, CTIM$A is most suitably used in its function form.

CTIM$A

Purpose

CTIM$A is a double precision function that returns CPU time elapsed since login, in seconds as the function value, and as centiseconds in the cputim argument.

Usage

```
INTEGER*4  cputim
REAL*8     rt_val

rt_val = CTIM$A(cputim)
         (or)
CALL  CTIM$A(cputim)
```

Parameters

cputim

OUTPUT. CPU time in centiseconds.

Discussion

The function value will be CPU time elapsed since login, in seconds. This value may be received as REAL*8.

Loading and Linking Information

```
APPLIB   --   R-mode
NVAPPLB  --   V-mode
VAPPLB   --   V-mode (unshared)
```

A Subroutine Description
Figure 1-1

SUBROUTINE USAGE

The Usage section of each subroutine description includes two items of information:

1. How to declare the subroutine in a program.

2. How to invoke it in a program.

The notation used is that of either the PL/I language or the FORTRAN language. If you do not use these languages, the explanation of the relevant syntax and data types descriptions in this section and the SUBROUTINE PARAMETERS section should enable you to call these subroutines from other languages. For further information see the chapter in Volume I that describes your language interface.

Subroutine Declarations with PL/I Elements

The following example shows a subroutine declaration in PL/I:

```
DCL CNIN$ ENTRY(CHARACTER(*), FIXED BIN, FIXED BIN);
```

DCL is the short form of DECLARE. The DECLARE statement is used to declare all data types, including subroutines and functions. CNIN$ is the subroutine name. ENTRY specifies that the item being declared is a subprogram.

The items in parentheses are the parameters of the subroutine.

Subroutine Calls with PL/I Elements

The following example shows a call to the subroutine declared above:

```
CALL CNIN$(buffer, char_count, actual_count);
```

PL/I does not distinguish between uppercase and lowercase characters. In the Usage section of a subroutine description, lower case letters indicate the items that must be supplied by the user, both arguments (actual parameters, as opposed to formal parameters) and data items. These are described more fully in the Parameters section.

The CALL statement above invokes the subroutine CNIN$. The arguments in parentheses correspond to the parameters in the subroutine declaration. The variables or constants used as arguments in a call to the subroutine must match the data types of the parameters in the declaration. Here, the variable buffer must be a character string, while char_count and actual_count must be integers. A subroutine that has no parameters is invoked simply by giving the CALL keyword and the name of the subroutine:

```
CALL TONL;
```

Subroutine Declarations and Calls with FORTRAN Elements

The FORTRAN language requires uppercase. It does not use a DCL line for subroutines. It requires declaration of the data types for those variables to be passed in a subroutine, without any additional sizing of parameters as for PL/I. Variable declarations are given at the beginning of the program, indented eight spaces -- as are all statements (except comment lines, which are marked by a "C", flush left). Variables given here are formal variables; you may choose your own names. The example for the subroutine CNIN$ now takes this form for FORTRAN:

```
      INTEGER*2 CH_CNT
C     FTN requires uppercase and expects variables of 6 characters
C     or less; thus CH_CNT instead of char_count as given in PL/I
      INTEGER*2 BUFFER(1)
C     The (1) suggests an array; you must substitute a value size
C     practical for your application, for example BUFFER(80)
      INTEGER*2 AC_CNT
C     (Later, after you assign CH_CNT a value):...
      CALL CNIN$(BUFFER, CH_CNT, AC_CNT)
```

Function Declarations with PL/I Elements

The following example shows a function declaration in PL/I:

```
DCL PWCHK$ ENTRY(FIXED BIN, CHAR(*) VAR) RETURNS(BIT(1));
```

The only difference between a function and a subroutine declaration is at the end of the DECLARE statement. The function declaration contains the keyword RETURNS, followed by a returns descriptor specifying the data type of the value returned by the function. In this case, it is a logical (Boolean) value.

Function Calls with PL/I Elements

A function is invoked when its name is used as an expression on the right-hand side of an assignment statement. The following example shows an invocation of the function declared above:

```
password_ok = PWCHK$(key, password);
```

The equal sign = is the assignment operator. password_ok is a logical (Boolean) variable that is assigned the value returned by the call to PWCHK$. key and password represent integer and character-string values, respectively.

Functions Without Parameters: A function that has no parameters is invoked with an empty argument list. The DATE$ subroutine is declared as follows:

```
DCL DATE$ ENTRY RETURNS(FIXED BIN(31));
```

Its invocation looks like this:

```
fs_date = DATE$();
```

Note

Functions called from FTN programs require parameters.

Function Declarations/Calls with FORTRAN Elements

FORTRAN has no DCL line to declare a function. An extra variable must be declared for a function -- the variable that is to hold the value returned by the function. Otherwise, preparation for a function call is exactly as for a subroutine call.

A note of warning: certain functions in this volume return FORTRAN logical values, sized to INTEGER*2 (a 16-bit halfword) instead of a single bit, as for PL/I. These functions are designed mainly as tools for FORTRAN programs. See the Application Library (chapters 9 - 16) for examples. Other languages may use them, but they must adjust the size of their returned logical value to the size of a halfword, instead of a (more-typical) single bit.

The subroutine call in FORTRAN uses the same form as in PL/I. The following example shows the declaration and call of RSTR$A, a FORTRAN logical function in chapter 10 of this volume:

```
      INTEGER*2 STRING(1)
C   (a string of characters to be rotated)
      INTEGER*2 LENGTH, COUNT
C   (LENGTH of string and COUNT of positions to rotate)
      LOGICAL   LOG
C   (To hold the LOGical value returned by the function)
      LOG = RSTR$A(STRING, LENGTH, COUNT)
```

SUBROUTINE PARAMETERS

Subroutines usually expect one or more arguments from the calling program. These arguments must be of the data type specified in the parameter list of the DECLARE statement, and must be passed in the order expected. All standard Prime subroutines are written in FORTRAN, PMA, or a system version of PL/I. Volume I discusses how to translate the data types expected by these languages into other Prime languages. A chart summarizing data type equivalents for all Prime languages is in Appendix F.

You must provide the number of arguments expected by the subroutine. If too few arguments are passed, execution causes an error message such as POINTER FAULT or ILLEGAL SEGNO. If too many arguments are passed, the subroutine ignores the extra arguments, but will probably perform incorrectly. A small number of subroutines, such as IOA$, accept varying numbers of arguments.

The Usage section of a subroutine description gives the data types of the parameters. The Parameters section explains what information these parameters contain and what they are used for. Each parameter description in this section begins with a word in uppercase that indicates whether the parameter is used for input or output:

- INPUT means that the parameter is used only for input, and that its value is not changed by the subroutine.
- OUTPUT means that the parameter is used only for output. You do not have to initialize it before you call the subroutine.
- INPUT/OUTPUT means that the parameter is used for both input and output. The argument you pass to it is changed by the subroutine.

You will note that a returned value from a function call receives no description in the Parameters section, since it is not truly a parameter of the subroutine. However, the Usage section defines the data type of the returned value.

Parameter and Returned-Value Data Types with PL/I

A PL/I parameter specification consists simply of a list of the data types of the parameters. The data types you will encounter, both in the parameter list and in the RETURNS part of a function declaration, are the following:

CHAR(n)	Also specified as CHARACTER(n), CHARACTER(n) NONVARYING. Specifies a character string or array of length n. A CHAR(n) string is stored as a byte-aligned string, one character per byte. (A byte is 8 bits.)
CHAR(*)	Also CHARACTER(*), CHARACTER(*) NONVARYING. Specifies a character string or array whose length is unknown at the time of declaration. A CHAR(n) string is stored as a byte-aligned string, one character per byte.
CHAR(n) VAR	Also CHARACTER(n) VARYING. Specifies a character string or array whose length can be a maximum of n characters. The first 2 bytes (one halfword) of storage for a CHAR(n) VAR string contain an integer that specifies the string length; these are followed by the string, one character per byte.
CHAR(*) VAR	Also CHARACTER(*) VARYING. Specifies a character string or array whose length is unknown at the time of declaration. The first 2 bytes (one halfword) of storage for a CHAR(*) VAR string contain an integer that specifies the string length; these are followed by the string, one character per byte.
FIXED BIN	Also FIXED BINARY, BIN, FIXED BIN(15). Specifies a 16-bit (halfword) signed integer.
FIXED BIN(31)	Specifies a 32-bit signed integer.
(n) FIXED BIN	An integer array of n elements. See below for more information about arrays.
FLOAT BIN	Also FLOAT BIN(23), FLOAT. Specifies a 32-bit (one-word) floating-point number.
FLOAT BIN(47)	Specifies a 64-bit (double-word) floating-point number.
BIT(1)	Specifies a logical (Boolean) value. A bit value of 1 means TRUE; a value of 0 means FALSE.
BIT(n)	Specifies a bit string of length n. BIT(n) ALIGNED means that the bit string is to be aligned on a halfword boundary.

POINTER — Also PTR. Specifies a POINTER data type. A pointer is stored in three halfwords (48 bits). If the pointer will point only to halfword-aligned data, it may occupy two halfwords (32 bits). The item to which the pointer points is declared with the BASED attribute (for instance, BASED FIXED BIN).

POINTER OPTIONS (SHORT) — Same as POINTER except that it always occupies only two halfwords and can only point to halfword-aligned data.

Note

When used as a parameter, POINTER can be used interchangeably with POINTER OPTIONS (SHORT).

When used as a returned function value, POINTER OPTIONS (SHORT) can be used in any high-level language except Pascal or 64V mode C, which require returned pointers to be three halfwords; in these cases, POINTER must be used. C in 32IX mode accepts only halfword-aligned, two-halfword pointers, and therefore requires the use of POINTER OPTIONS (SHORT).

Sometimes an argument is defined as an array or a structure. An array declaration looks like this:

```
DCL ITEMS(10) FIXED BIN;
```

Here, ITEMS is a ten-element array of integers. The keywords FIXED BIN, however, can be replaced by any data type. By default, arrays are indexed starting with the subscript 1; the first integer in this array is ITEMS(1).

An array with a starting subscript other than 1 is declared with a range specification:

```
DCL WORD(0:1023) BASED FIXED BIN;
```

WORD is an array indexed from 0 to 1023, and its elements are referenced by POINTER variables.

A structure is equivalent to a record in COBOL or Pascal. A structure declaration looks like this:

```
DCL 1 FS_DATE,
        2 YEAR BIT(7),
        2 MONTH BIT(4),
        2 DAY BIT(5),
        2 QUADSECONDS FIXED BIN(15);
```

The numbers 1 and 2 indicate the relative level numbers of the items in the structure. The name of the structure itself is always declared at level 1. The level number is followed by the name of the data item and its data type. In this example, the structure occupies a total of 32 bits. (Remember that a FIXED BIN(15) value occupies 16 bits of storage.)

Since no names are given to data items in parameter lists, the array declared above as ITEMS would be declared simply as (10) FIXED BIN. Similarly, the structure FS_DATE would be listed as

```
(..., 1, 2 BIT(7), 2 BIT(4), 2 BIT(5), 2 FIXED BIN(15), ...)
```

Data Types For FORTRAN

The Usage sections with FORTRAN employ FTN (not F77) elements. The data types you will encounter there are the following:

COMPLEX	Specifies a 64-bit element to hold a complex number, defined as two 32-bit (REAL*4) entities, the first for its real and the second for its imaginary part.
INTEGER*2	Also INTEGER. Specifies a 16-bit (halfword) signed integer. Bit 1 = sign bit.
INTEGER*4	Specifies a 32-bit signed integer. Bit 1 = sign bit.
LOGICAL	Specifies a logical (Boolean) value. Within a 16-bit halfword: the first 15 bits must be 0, the 16th bit indicates .FALSE. with 0 and .TRUE. with 1.
REAL*4	Also REAL. Specifies a 32-bit signed floating-point number. Bit 1 = sign bit. Bits 2-24 = mantissa. Bits 25-32 = exponent.
REAL*8	Also DOUBLE PRECISION. Specifies a 64-bit signed floating-point number. Bit 1 = sign bit. Bits 2-48 = mantissa. Bits 49-64 = exponent.

Data Type Variants For FORTRAN

Other declarations in the Usage section suggest the elements for which FTN has no data type:

BUFFER(1)	Given the data type of INTEGER*2, this shorthand declaration for an array suggests a character string or array whose length is unknown at the time of declaration (an equivalent to CHAR(*) in PL/I). The user must DIMENSION the array with an adequate size. If the size is known to be (n), then the variable declaration is given as BUFFER(n).
LOC(variable)	Specifies the equivalent of a POINTER data type. This built-in FORTRAN function automatically provides the prerequisite three halfwords (48 bits) for the pointer.

Optional Parameters

On Prime computers, some subroutines and functions are designed so that one or more of their parameters, input or output, can be omitted. Candidates for omission are always the last n parameters. Thus, if a subroutine has a full complement of three parameters, it may be designed so that the last one or the last two can be omitted; the subroutine cannot be designed so that only the second parameter can be omitted. The first parameter can never be omitted.

In the Usage section of a subroutine description, any optional parameters are enclosed in square brackets, as in the following declaration and CALL statement:

```
DCL CH$FX1 ENTRY (CHAR (*) VAR, FIXED BIN (15)
                                  [, FIXED BIN (15)]);

CALL CH$FX1 (string_to_convert, result
                                  [, nonstandard_code]);
```

In some cases, parameters can be omitted because they are not needed under the circumstances of the particular call. In other cases, when the parameter is of type INPUT, the subroutine will detect the missing parameter and will assume some value for it. For example, C1IN$, described in Volume III, Chapter 3, can be called with one, two or three arguments:

```
CALL C1IN$ (char);
CALL C1IN$ (char, echo_flag);
CALL C1IN$ (char, echo_flag, term_flag);
```

If echo_flag is missing, the subroutine acts as if it had been supplied with a value of "true". If term_flag is missing, the subroutine acts as if it had been supplied with a value of "false".

In still other cases, the subroutine changes its behavior depending on the presence of the parameter. For example, the subroutine CH$FX1 (described in Volume III, Chapter 6), whose Usage section is shown above, uses its third argument to return an error code. If the code argument is omitted and an error occurs, the routine signals a condition instead.

Most of the routines in the Subroutines Reference Guide have no optional parameters.

Optional Returned Values

In the architecture of Prime computers, a subroutine that was designed as a function can be called as a subroutine using the CALL statement. Frequently this makes no sense. The statement

```
CALL SIN(45);
```

does nothing useful; the value that the SIN function returns is lost. But, with functions that change some of their parameters as well as return a value, the returned value can be useful in some contexts and not of interest in other contexts. Consider CL$GET, described in Volume III, Chapter 3 of the Subroutines Reference Guide. It is a function that reads a line from the command device and, in addition, returns a flag that indicates whether a command input file is active. Most programs do not need to know whether a command input file is active. They would call CL$GET as a subroutine:

```
CALL CL$GET (BUFFER, 80, CODE);
```

A program that was interested in command input files, however, would call CL$GET as a function:

```
COMISW = CL$GET (BUFFER, 80, CODE);
```

Note

In PL/I and Pascal, a given subroutine may not be used both as a subroutine and as a function within a single source module.

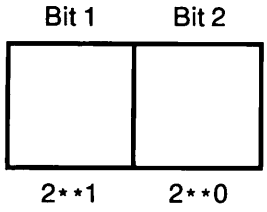
The Usage section of the subroutine descriptions gives both the

function invocation and the subroutine invocation for subroutines that are likely to be called in both ways.

How to Set Bits in Arguments

Sometimes a subroutine expects an argument that consists of a number of bits that must be set on or off.

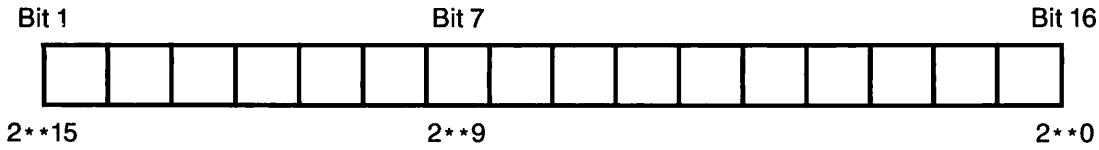
A data item is stored in a computer as a collection of bits, which can each have one of two values, off or on. On Prime computers, off is arbitrarily equated to 0 or false, and on is equated to 1 or true. (This is not the same as the FORTRAN values .FALSE. and .TRUE., which are the logical data type.) When bits are stored as part of a group, the position of the bit gives it another value in addition to 1 or 0. Its position equates it to a power of 2. Consider an argument that contained only two bits, represented in Figure 1-2.



Values of Bit Positions -- Two Bits
Figure 1-2

The low-order bit would be in the position of 2 to the 0 power, and its value, if ON, would be 1. The high-order bit would be in the position of 2 to the first power, and its value, if ON, would be 2. (If OFF, the value of a bit is always 0.) By convention, the low-order bit is called the rightmost bit and the high-order bit is called the leftmost bit.

In an argument containing 16 bits, choose the bits that you want to set ON, compute their value by position, and add these values. The resulting decimal value is what you should assign to the subroutine argument for the options you want. You can pass an integer as an argument that is declared as BIT(n) ALIGNED. The subroutine interprets the integer as a bit string. For example, if you want to set the sixteenth and the seventh bit, compute 2 to the 0 power plus 2 to the ninth power, which amounts to 1 plus 512, or 513. Figure 1-3 illustrates values of bit positions in a 16-bit argument.



Values of Bits in a 16-bit Argument
Figure 1-3

Key Names as Arguments

In calls to many subroutines, data names known as keys can be used to represent numeric arguments. The subroutine description explains which key to use. Numeric values are associated with these keys in the UFD named SYSCOM. The keys in SYSCOM are listed in Volume I. Each language has its own files of keys. The chapters on individual languages in Volume I explain how to insert these files into your program.

Keys are of the form x$yyyy, where x is either K or A and yyyy is any combination of letters. Keys that begin with K concern the file system; those that begin with A concern applications library routines; those that begin with E are error codes. Examples are:

```
K$CURR
A$DEC
```

For example, in the subroutine call

```
CALL GPATH$ (K$UNIT.....other arguments...);
```

the key K$UNIT represents the value 1.

For more information about keys, see Volume I.

Standard Error Codes

Many subroutines include as an argument a standard error code, which is similar to a key. The error code corresponds to an error message that the subroutine can return to indicate that the call to the subroutine succeeded or failed, or to report some other condition worth noting.

Standard error codes are of the form E$xxxx, where xxxx is any combination of letters. For example, the error code

```
E$DVIU
```

corresponds to the error message Device in Use.

The standard error codes are defined in the UFD named SYSCOM. Like a key file, the error code file for a particular language must be inserted in the program that calls the subroutine. Appendix A in this volume gives an overview of the standard error codes plus a pathname to the online list. A copy of the listing, current for Revision 20.2 of Primos, is given in Volume I. For an explanation of each standard error code, see Volume 0 of the Advanced Programmer's Guide.

Libraries and Addressing Modes

The Subroutines Reference Guide is organized to give a systematic description of subroutine libraries -- sets of routines, all broadly dealing with the same subject, grouped together into one file. There is a separate library for each of these subjects.

Prime computers offer several addressing modes to provide software compatibility to the user. (For a discussion of addressing modes, see the System Architecture Reference Guide.) To maintain this compatibility, a given subroutine library will normally exist in three general versions: R-mode, V-mode, and V-mode (unshared). (See Volume I for a discussion of shared and unshared libraries.)

A program is compiled in one of the segmented modes (V-mode or I-mode) or in the older R-mode. If the program is compiled in one of the segmented modes, it may call library routines written in any of the segmented modes. A single set of libraries is provided for all three modes. If the program is compiled in either V-mode or I-mode, it requires a V-mode version of a library (which services both V-mode and I-mode programs). If the program is compiled in R-mode, the program must use the R-mode version of that library.

Every routine description contains a section entitled Loading and Linking Information. It specifies the name of the library to use for that subroutine, depending upon the compilation mode of your program. During your BIND, SEG, or LOAD Session, you satisfy the subroutine references by providing a LI (for Library) command followed by the name of the library (in the appropriate mode) holding your subroutine(s). Several LI commands may be necessary. A final "LI", without a library specified thereafter, provides the system libraries that complete the linking or loading session. See Volume I for further information.

PART II

IOCS LIBRARY

2
Introduction to IOCS

ORGANIZATION OF PART II

IOCS (the Input/Output Control System) is a group of subroutines that perform input/output between the Prime computer and the disks, terminals, and other peripheral devices on the system.

These subroutines are very powerful, but you are urged to use the file system subroutines described in Volume II for your file I/O operations.

While these IOCS routines certainly can do the job, IOCS allows a maximum of 127 default PRIMOS file units per user, and each of these is assigned to a logical unit from 1 to 141. To make file unit assignments greater than 127, you must map each to a logical unit within the range 1-141, using a call to ATTDEV. But that is your limit. If you are using EPFs that allow suspended processes at multiple command levels, you may accidentally run out of Primos file units or FORTRAN logical units. On the other hand, the file system subroutines in Volume II do not require logical units. Furthermore, they draw on a pool of 32761 PRIMOS file units per user, and the system dynamically allocates and frees them for you.

If you need to exercise the control that IOCS gives you, then have a careful strategy for file unit allocation, knowing that only the first 127 file units are handled by default. Use calls to IOCS$F and IOCS$G to obtain a logical unit (within the range 1-141). Finally, use a call to ATTDEV to map that logical unit to an available file unit.

Generally, IOCS subroutines can be grouped into three levels:

Level 1	Device-independent drivers are routines that read and write ASCII or binary data and perform control functions such as opening a file.
Level 2	Device-dependent drivers issue the correct format for a particular device, but allow the data to be read later by device-independent drivers.
Level 3	The lowest level of IOCS functions are routines that perform raw data transfers.

The chapters in Part II are organized in the following manner:

Chapter 2	Device, unit, and argument definitions and tables for use with following chapters
Chapter 3	Changing device assignments
Chapter 4	Device-independent driver subroutines (which call the device-dependent routines in the following chapters, depending on the device specified)
Chapter 5	Disk (non-file system) subroutines
Chapter 6	Subroutines for the user terminal and paper tape (Many subroutines may be used for both peripherals.)
Chapter 7	Subroutines for other peripheral devices (printers, plotters, card processors, and magnetic tape)

The level-1 device drivers are presented in Chapter 4. Routines of levels 2 and 3 are grouped in the following chapters by device type rather than by level of the subroutine.

Table 2-1 shows the majority of IOCS routines discussed in Chapter 4 through Chapter 7. It shows the relationship of level-1 (device-independent) drivers to the others. Each column of this table represents an I/O function, and each row a certain physical device. All drivers in a single column are designed to be compatible in internal data format.

Tables 2-2 and 2-3 show the physical and logical device assignments, for use in changing device assignments as discussed in Chapter 3.

Figure 2-1 shows all the device-dependent drivers supported by Prime.

Table 2-1

Device-dependent Driver Selected by
Each Independent Driver According to Device

	Device-Independent Drivers				
	RDASC	WRASC	RDBIN	WRBIN	CONTRL
Device	Dependent Drivers				
User terminal	I$AA01(6)*	O$AA01(1)	I$BA01(2)	O$BA01(2)	C$A01(2)
Input command stream	I$AA12(1)				
Paper-tape reader	I$AP02(5)		I$BP02(2)		C$P02(5)
Paper-tape punch		O$AP02(5)		O$BP02(2)	
MPC card reader	I$AC03(3)	O$AC03(3)			
Serial line printer		O$AL04(3)			
9-track mag. tape	I$AM05(4)	O$AM05(4)	I$BM05(7)	O$BM05(7)	C$M05(4)
MPC line printer		O$AL06(4)			
PRIMOS file system (compressed)	I$AD07(1)	O$AD07(1)	I$BD07(1)	O$BD07(1)	SEARCH(1)
PRIMOS file system (uncompressed)	I$AD07(1)	O$AD08(1)	I$BD07(1)	O$BD07(1)	SEARCH(1)
Serial card reader	I$AC09(3)				
7-track mag. tape	I$AM10(4)	O$AM10(4)	I$BM10(7)	O$BM10(7)	C$M10(4)
7-track mag. tape (BCD)	I$AM11(7)	O$AM11(7)			C$M11(7)
9-track mag. tape (EBCDIC)	I$AM13(7)	O$AM13(7)			C$M13(7)
Versatec printer/ plotter		O$AL14(3)			
MPC card processor	I$AC15(3)	O$AC15(3)			

* Numbers in parentheses refer to the following notes.

Notes to Table 2-1

1. Available in R-mode and V-mode. Listed in CONIOC (Chapter 3) and may be called directly or via the device-independent drivers.

2. Available in R-mode only. Listed in CONIOC (Chapter 3) and may be called directly or via the device-independent drivers.

3. Available in R-mode only. Listed in FULCON but not CONIOC (Chapter 3). May not be called via the device-independent drivers, unless FULCON is assembled and loaded before the library is loaded.

4. Available in R-mode and V-mode. Listed in FULCON (Chapter 3). In V-mode programs, these routines may be called directly or via the device-independent drivers if the default FORTRAN library (PFTNLB) is loaded. If the R-mode or the nonshared V-mode library (NPFTNLB) is loaded, the routine may not be called via the device-independent drivers unless FULCON is assembled and loaded before the library is loaded. See Chapter 3 for a more complete discussion of IOCS table usage. Routine may be called by name without specific procedures.

5. Available in R-mode and V-mode. For R-mode, routine is listed in CONIOC (Chapter 3) and may be called directly or via the device-independent drivers. For V-mode, routine is listed in FULCON (Chapter 3) and may be used in same manner as R-mode as long as the default FORTRAN library (PFTNLB) is loaded. In R-mode, or V-mode when the nonshared FORTRAN library (NPFTNLB) is loaded, the routine may not be called via the device-independent drivers unless FULCON is assembled and loaded before the library is loaded. See Chapter 3 for a more complete discussion of IOCS table usage.

6. Available in R-mode and V-mode, but is not in CONIOC (Chapter 3) or FULCON. To call the routines via the device-independent drivers, the appropriate table must be modified, assembled, and loaded before the library is loaded. (See Chapter 3.) The routine may be called specifically without any special procedures.

7. Available in R-mode and V-mode. V-mode is listed in FULCON but not in CONIOC (Chapter 3). R-mode is not in CONIOC or FULCON. In V-mode, if the nonshared FORTRAN library (NPFTNLB) is loaded, the routine may not be called via the device- independent drivers unless FULCON is assembled and loaded before the library is loaded. In R-mode, the appropriate table must be modified, assembled, and loaded before the library is loaded. In both modes, the routine may be called specifically without any special procedures.

Table 2-2
Physical Device Numbers

Physical Device	Device Description
1	User terminal
2	Paper-tape reader or punch
3	MPC card reader
4	Serial line printer
5	9-track magnetic tape ASCII/BINARY
6	MPC line printer
7	PRIMOS file system (compressed ASCII)
8	PRIMOS file system (uncompressed ASCII)
9	Serial card reader
10	7-track magnetic tape ASCII/BINARY
11	7-track magnetic tape BCD
12	(User terminal/command file) command input
13	9-track magnetic tape EBCDIC
14	Versatec Printer/Plotter

Table 2-3
Logical Devices, Physical Devices, and File Units

FORTRAN Default Logical Unit Number	Physical Device or Unit
1	User terminal
2	Paper-tape reader or punch
3	MPC card reader
4	Serial line printer (system option controller or SOC)
5	PRIMOS file unit 1
6	PRIMOS file unit 2
7	PRIMOS file unit 3
8	PRIMOS file unit 4
9	PRIMOS file unit 5
10	PRIMOS file unit 6
11	PRIMOS file unit 7
12	PRIMOS file unit 8
13	PRIMOS file unit 9
14	PRIMOS file unit 10
15	PRIMOS file unit 11
16	PRIMOS file unit 12
17	PRIMOS file unit 13
18	PRIMOS file unit 14
19	PRIMOS file unit 15
20	PRIMOS file unit 16
21	9-track magnetic tape unit 0
22	9-track magnetic tape unit 1
23	9-track magnetic tape unit 2
24	9-track magnetic tape unit 3
25	7-track magnetic tape unit 0
26	7-track magnetic tape unit 1
27	7-track magnetic tape unit 2
28	7-track magnetic tape unit 3
29	PRIMOS file unit 17
30	PRIMOS file unit 18
31	PRIMOS file unit 19
.	.
.	.
.	.
139	PRIMOS file unit 127
140	MPC printer 0
141	MPC printer 1

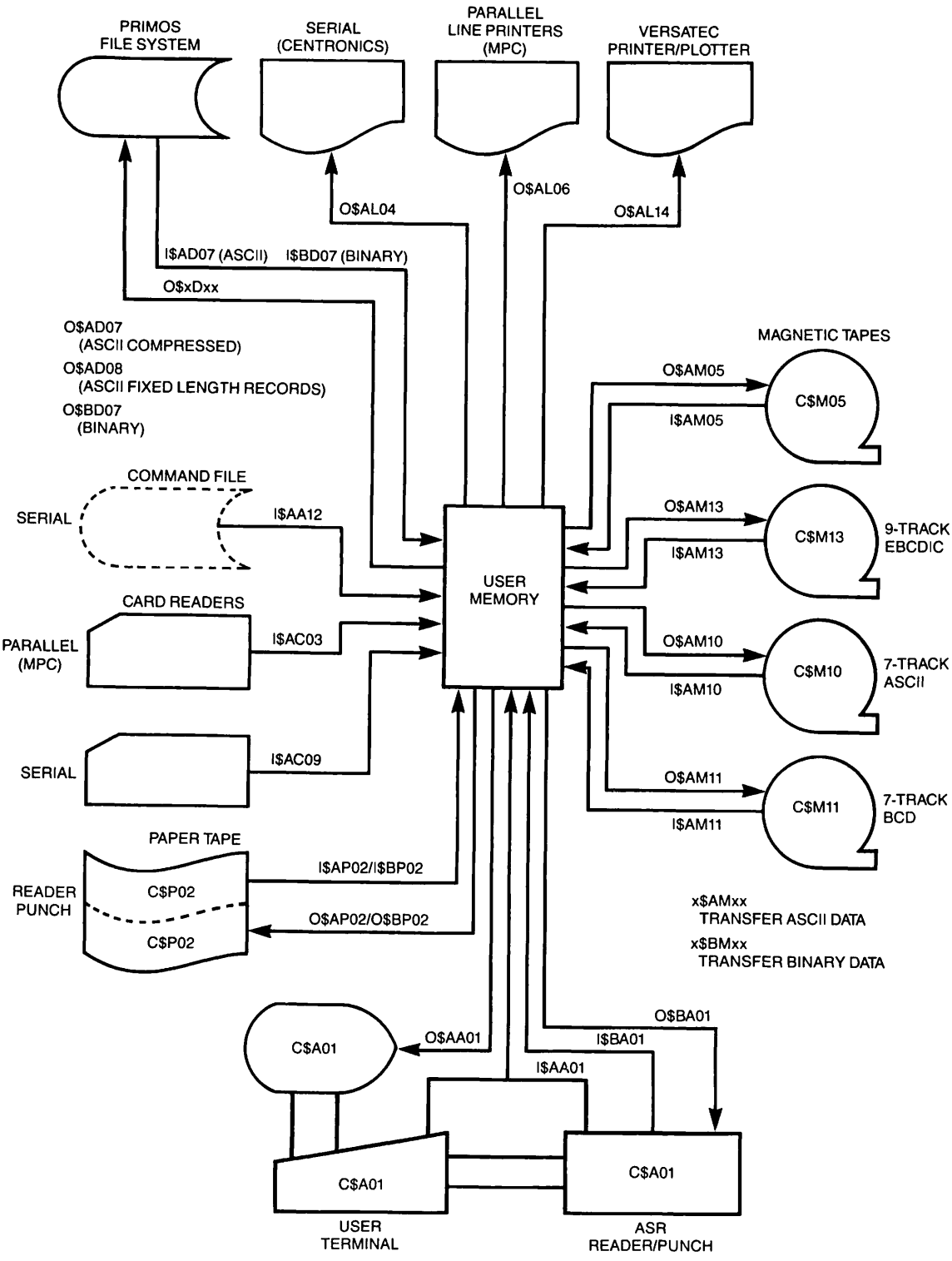
Notes to Table 2-3

All IOCS routines use the Logical Unit Number Table. All Logical Unit Numbers therefore must fall within the range 1-141. Note also that these Logical Unit Numbers supply default mapping values for Primos Physical File Units in the range 1-127.

With Revision 19.4 the full range for Primos Physical Unit numbers (1-127 per user) was expanded to cover 1-32761. However, there has been no change in default mapping of Physical Units to Logical Units.

There is no default mapping of Primos Units to Logical Units where the Primos Unit is 128 or larger.

If a Primos Unit greater than 127 is used, and the user wishes to use IOCS subroutines, a call to ATTDEV must be made to explicitly map this Physical Unit to an available Logical Unit (within the range of 1-141).



Transfer of Data to and from High-speed User Memory
Figure 2-1

PARAMETERS USED FOR IOCS SUBROUTINES

The following parameter names are used throughout Part III. The IOCS subroutines were first developed with FTN programmers in mind; therefore data types here receive FORTRAN descriptions. However, other languages (especially PL/I) may also call these subroutines, with certain restrictions. For example, refer to altrtn below. Therefore, the individual subroutines are given data descriptions in a PL/I format.

altrtn — An INTEGER*2 assigned the value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. Programs in PL/I may also use the (fixed bin(15)) altrtn; since such programs are in V-Mode, users must consider the caution below. Other calling languages should omit the argument (not use 0).

Note

If in V-Mode, the altrtn label must be in the same stack frame as the code that made the call.

buffer — The name of a data area to or from which data is moved (INTEGER*2 array in FTN or char array in PL/I).

count — The number of halfwords to be transferred, or the length of a buffer or filename (INTEGER*2 or fixed bin(15)).

buffer_size — The record size associated with the logical unit. Must be as large as the maximum record size, measured in 16-bit halfwords (INTEGER*2 or fixed bin(15)).

logical_device — Same as logical_unit below. (INTEGER*2 or fixed bin(15)).

logical_unit — The FORTRAN logical unit (see Table 2-3). Must be between 1-141 inclusive. (INTEGER*2 or fixed bin(15)).

name — A filename, also called name(1) to suggest a FTN array. (INTEGER*2 or char(*)).

physical_device	The position in the device-type table (see Table 2-2). A physical device is a device type such as magnetic tape or a user terminal. (INTEGER*2 or fixed bin(15)).
physical_unit	The sub-unit number of a physical device having more than one unit (see Table 2-3). A physical unit designation distinguishes among the units of a physical device that has multiple units, such as a magnetic tape controller. For disk (the file system), the physical unit corresponds to the file unit (below). If the device has only one unit, its sub-unit number is 1. If it is a multiple-unit device such as disk or tape, sub-units 1 through 8 may be specified. (On disk, a sub-unit is actually processed as file 1-8.) (INTEGER*2 or fixed bin(15)).
file_unit	The PRIMOS file-unit (funit) number from 0 through 32761. (Users may assign 2 and above; the system makes assignments for 0 and 1.) File units are discussed in Vol. II as well as in the Advanced Programmer's Guide. (INTEGER*2 or fixed bin(15)).
sub_unit	The unit for multi-unit devices (for disk, file unit number). This is the same as the physical unit (see Table 2-3). (INTEGER*2 or fixed bin(15)).

3
Device Assignment

TEMPORARY DEVICE ASSIGNMENT

The user may assign any device by calling the ATTDEV subroutine. ATTDEV controls mapping of logical units into physical devices and controls the record size associated with the logical unit. Nonsharable devices may also be assigned on command level with the PRIMOS command ASSIGN. If you wish to make a permanent device assignment, go to that section after the descriptions of IOCS$G, IOCS$F, and ATTDEV.

As discussed in Chapter 2, IOCS is limited to the use of 127 default file units per user, whereas the file system subroutines in Volume II use PRIMOS to dynamically allocate up to 32761 file units per user.

IOCS is also limited to a maximum of 141 FORTRAN logical units that it does not dynamically allocate and free. Therefore, if you plan to extensively use IOCS subroutines, you must make a strategic use of the logical unit handlers IOCS$G and IOCS$F to obtain an available logical unit. You must then map that logical unit to an available file unit, using a call to ATTDEV. You may assign file units greater than the default 127, but you are still limited to 141 logical units.

> **Caution**
>
> R-Mode subroutines can be called from FTN and PMA in R-Mode only. If you call an R-Mode routine from a program in a different mode, the results are unpredictable. Refer to the FORTRAN and PMA chapters in Volume I for information on declaring parameters in FTN and PMA, respectively.

IOCS$_GET_LOGICAL_UNIT

Alternate Name

Calls from FTN programs require the six-character name: IOCS$G. Other languages may use IOCS$G as an optional calling form.

Purpose

This routine is used to perform two tasks: 1) to provide an available logical file unit number to a calling program; 2) to set aside as "in use" a particular logical file unit number already found available.

Usage

```
DCL IOCS$_GET_LOGICAL_UNIT ENTRY(FIXED BIN(15), FIXED BIN(15),
                                 FIXED BIN(15));

CALL IOCS$_GET_LOGICAL_UNIT (key, logical_unit,
                             code);
```

Parameters

key

INPUT. Indicates the desired function to be performed. Values may be:

1 get an available logical unit

2 set the specified logical unit to "in use"

logical_unit

INPUT/OUTPUT. If key=1, logical_unit returns as output the number of an available logical unit. If key=2, you must input in logical_unit the number of that logical unit whose bit is to be set in the logical unit table (LUTBL). Valid logical unit values are in the range 1-141.

code

OUTPUT. Indicates the result of the subroutine request. Aside from the usual code of (0) for success, possible values are:

E$NSUC no available logical unit numbers

E$UIUS logical unit already in use

E$BUNT logical unit is not a valid number

Discussion

When a program calls this subroutine with key=1 (get an available logical file unit), the routine returns that number in logical_unit. If there are no available logical unit numbers, the routine returns E$NSUC in code.

When a program calls this subroutine with key=2 (set a bit in the logical unit table), the routine will attempt to set the bit corresponding to that logical unit number input in logical_unit. If the unit is already in use, code returns E$UIUS. If the unit is not valid, code returns E$BUNT. Otherwise code returns the usual 0 to indicate completion with no errors.

Loading and Linking Information

NPFTNLB -- V-Mode (unshared)
PFTNLB -- V-Mode *

IOCS$_FREE_LOGICAL_UNIT

Alternate Name

Calls from FTN programs require the six-character name: IOCS$F. Other languages may use IOCS$F as an optional calling form.

Purpose

This routine allows a calling program to free a logical file unit number so that it is made available in the Logical Unit Table (LUTBL) to another calling program.

Usage

```
DCL IOCS$_FREE_LOGICAL_UNIT ENTRY(FIXED BIN(15), FIXED BIN(15));

CALL IOCS$_FREE_LOGICAL_UNIT (logical_unit, code);
```

Parameters

logical_unit

INPUT. This must contain the logical file unit number that is being freed to the logical unit table. Valid logical unit values are in the range 1-141.

code

OUTPUT. Indicates the result of the subroutine request. The possible values are:

E$OK	The call to IOCS$F was completed without error.
E$BUNT	The logical unit is not a valid number.
E$UNOP	The logical unit is not open.

Discussion

IOCS$_FREE_LOGICAL_UNIT frees to the Logical Unit Table the number specified in logical_unit. The routine returns a success code of E$OK if the unit was freed, or E$UNOP if the unit is not open.

If the unit number passed is not a valid unit number, then an error code E$BUNT is returned in code.

Loading and Linking Information

NPFTNLB -- V-Mode (unshared)
PFTNLB -- V-Mode

ATTDEV

Purpose

ATTDEV attaches specified devices by associating logical_device with physical_device and associating the logical_device with a specific physical unit or file unit for the device.

Usage

```
DCL ATTDEV ENTRY(FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15));

CALL ATTDEV (logical_device, physical_device, physical_unit,
             buffer_size);
```

Parameters

logical_device

INPUT. The device-independent logical I/O unit (Table 2-3), synonymous with the FORTRAN logical unit. This number cannot be changed.

physical_device

INPUT. The number corresponding to the relevant device-type in (Table 2-2).

physical_unit

INPUT. The sub-unit number of a physical device having more than one unit (refer to Table 2-3). A physical unit designation distinguishes among the units of a physical device that has multiple units, such as a magnetic tape controller. For disk (the file system), the physical unit corresponds to the file unit. If the device has only one unit, its sub-unit number is 1. If it is a multiple-unit device such as disk or tape, sub-units 1 through 8 may be specified. (On disk, a sub-unit is actually processed as file 1-8.)

buffer_size

INPUT. The record size associated with the logical unit. It must be as large as maximum record size, expressed in 16-bit halfwords.

Discussion

For the given logical_device, set the physical_device, physical_unit, and buffer_size so that the logical unit has a current mapping. Note that buffer_size is measured in halfwords (holding two characters in each).

Example

To reassign:

- a card reader (logical unit 3)
- to physical device 2 (which has no sub-units)
- with the ability to read 80-column cards (i.e., 80/2)

enter the following:

```
CALL ATTDEV(3, 2, 0, 40)
```

Errors

If device is incorrect, ATTDEV returns the message:

ATTDEV BAD UNIT (physical_unit)

Loading and Linking Information

FTNLIB	--	R-Mode
PFTNLB	--	V-Mode
NPFTNLB	--	V-Mode (unshared)

PERMANENT DEVICE ASSIGNMENT

Users whose programs need to use devices other than the user terminal, the disks, or paper-tape reader or punch, or who wish to change the assignment of logical to physical devices must consult their System Administrator. The following discussion is an overview of the System Administrator's work.

To facilitate changes to device assignments, the tables used by IOCS (such as LUTBL and PUTBL) are in the following files on the master disk.

V-Mode	SYSTEM_LIBRARYSRC>INSERT>CONIOC.INS.PMA
R-Mode	RFTNLIB>IOCS>CONIOC.PMA

Ask your System Administrator how to locate the master disk on a multidisk system.

Note that the R-Mode CONIOC.PMA in the RFTNLIB supports only the user terminal, the paper-tape reader, paper-tape punch, and the PRIMOS file system. An attempt to perform I/O to a physical device not supported by CONIOC will fail. The default CONIOC for V-Mode supports the user terminal and PRIMOS file system only.

IOCS Tables

If a computer installation requires that user programs use devices not supported by CONIOC, the System Administrator must modify the CONIOC tables RATBL, RBTBL, WATBL, and WBTBL, and then rebuild the FORTRAN library. There is a version of CONIOC that contains all the available IOCS drivers set up in the appropriate tables. This file is INSERT>FULCON.INS.PMA in SYSTEM_LIBRARYSRC, or IOCS>FULCON.PMA in RFTNLIB. The System Administrator can use FULCON as an example of how to set up CONIOC. The table entries that are not required can be set to 0.

The System Administrator may also change the default logical-to-physical-device association as given in Tables 2-2 and 2-3 by changing the IOCS tables RATBL, TBTBL, WATBL, and CNTBL in CONIOC. For example, the fifth entry of LUTBL (indicating logical device 5) contains 7. Entry 7, the RATBL, contains I$AD07, which is a driver for the PRIMOS file system. Other numbers indicate physical devices, as shown in Table 2-2. PUTBL is the sub-unit table. The sub-unit table contains the individual unit or file numbers as required for multifile devices. For example, LUTBL contains the same number of logical devices 21, 22, 23, and 24, indicating 9-track magnetic tape. PUTBL contains 0, 1, 2, and 3 for logical devices 21, 22, 23, and 24 indicating unit 0, 1, 2, and 3 of 9-track magnetic tapes.

Modifying CONIOC to Change Device Assignment

Changing a device assignment is a System Administrator's responsibility and not a user function. The System Administrator may add or delete a device to any of the following tables.

RATBL	Read ASCII table.
RBTBL	Read binary table.
WATBL	Write ASCII table.
WBTBL	Write binary table.
CNTBL	Perform control function (endfile, rewind, etc.).

Input-only Devices: Input-only devices such as the card reader do not need WATBL and WBTBL entries. Furthermore, an ASCII-only device (such as a line printer) does not need RBTBL and WBTBL entries.

Order of Entries: The order of entries in the above-mentioned tables corresponds to physical-device numbers defined in Table 2-2.

R-Mode Procedures:

1. Attach to RFTNLIB>IOCS.
2. Edit the appropriate tables within CONIOC.PMA.
3. Replace the 0 with the corresponding subroutine name for the desired device.
4. Rebuild the RFTNLIB library. (See below.)

V-Mode Procedures:

1 Attach to SYSTEM_LIBRARYSRC>INSERT.

2 Edit the appropriate tables within the CONIOC.INS.PMA.

3 Replace the word NULLDEVICE with the appropriate device subroutine name.

4 Rebuild the SYSTEM_LIBRARYSRC Library. (See below.)

How to Rebuild the FORTRAN Library after Modifying CONIOC

After you have made changes in CONIOC for either the R-mode or V-mode version of the FORTRAN library (see the previous procedures), you must rebuild the library before the changes will take effect.

R-Mode Procedures: Rebuild the R-Mode FORTRAN library as follows:

1 Attach to RFTNLIB.

2 Run RFTNLIB.BUILD.CPL.

3 Run INSTALL_FTNLIB.CPL.

4 Share the new library (a System Administrator procedure).

V-Mode Procedures: Rebuild the V-Mode FORTRAN library as follows:

1 Attach to SYSTEM_LIBARYSRC

2 Run SYSTEM_LIBRARY.BUILD.CPL.

3 Share the new library (a System Administrator procedure).

4
Device-Independent Drivers

This chapter presents the subroutines listed in the top (horizontal) row of Table 2-1. The subroutines have the following functions:

Routine	Function
WRASC	Write ASCII data
RDASC	Read ASCII data
WRBIN	Write binary data
RDBIN	Read binary data
CONTRL	Other control functions

Maintain device independence in your data transfers through the use of these IOCS drivers. These device-independent or first-level drivers route the I/O request to one of the device-dependent drivers, as shown in Table 2-1 and Figure 2-1. The device-dependent drivers are presented in the following chapters (5 through 7). Each column of Table 2-1 represents an I/O function, and each row a specific physical device. All drivers in a single column are designed to be compatible in terms of internal data format.

DATA FORMATS

All first-level and second-level device drivers are uniform in the internal representation of data. All ASCII data, for example, has the same internal format regardless of the physical device.

ASCII Data

Data associated with logical I/O functions RDASC (Read ASCII) and WRASC (Write ASCII) are represented internally as an ASCII string in card image format. This string is of length N halfwords with each halfword containing ASCII-coded characters. (N is defined in the calling sequence to the driver.)

Notes

1. The new line character ('212) must not be used as data because it is the end-of-record indicator.

2. ASCII drivers should be used only to transfer printable ASCII characters.

Binary Data

Use RDBIN and WRBIN to transfer binary data. The external format varies considerably from device to device, but the internal format remains the same. Binary data can consist of anything and is not interpreted by the driver in any way.

The parameter buffer (buffer address) in a call to RDBIN (Read Binary) or WRBIN (Write Binary) defines the first halfword of the binary data. The user must define the halfword count on output.

> **Caution**
>
> R-mode subroutines can be called from FTN and PMA in R-mode only. If you call an R-mode routine from a program in a different mode, the results are unpredictable. Refer to the FORTRAN and PMA chapters in Volume I for information on declaring parameters in FTN and PMA, respectively.

WRASC

Purpose

WRASC writes ASCII characters to any output device.

Usage

```
DCL WRASC ENTRY(FIXED BIN(15), CHAR(*), FIXED BIN(15),
                FIXED BIN(15));

CALL WRASC (logical_device, buffer, count, altrtn);
```

Parameters

logical_device

INPUT. The device-independent logical I/O unit (Table 2-3), synonymous with the FORTRAN logical unit. This number cannot be changed.

buffer

INPUT. The name of a data area from which data in memory is moved to the output device.

count

INPUT. The number of halfwords to be transferred, or the length in halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Discussion

The contents of buffer are moved from memory to the output device. The format of the data on the output medium is device-specific. Memory is assumed to consist of ASCII, two characters per halfword.

Loading and Linking Information

FTNLIB -- R-mode
PFTNLB -- V-mode
NPFTNLB -- V-mode (unshared)
SVCLIB -- R-mode (maintained for PRIMOS-II)

RDASC

Purpose

RDASC reads ASCII characters from any input device.

Usage

```
DCL RDASC ENTRY(FIXED BIN(15), CHAR(*), FIXED BIN(15),
                FIXED BIN(15));

CALL RDASC (logical_device, buffer, count, altrtn);
```

Parameters

logical_device

INPUT. The device-independent logical I/O unit (Table 2-3), synonymous with the FORTRAN logical unit.

buffer

OUTPUT. The name of a data area to which data is moved from the input device.

count

INPUT. The number of halfwords to be transferred, or the length halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument but not use 0.

Discussion

One record is brought into memory. Buffer is always filled with count ASCII characters, two per halfword. If the record is longer than count halfwords, buffer contains the first count halfwords in the record and the next successive read will give the first count halfwords of the next record, not the remaining halfwords of the long record. If the record is less than count halfwords, the remainder of the buffer will be blank-filled.

Loading and Linking Information

FTNLIB	--	R-mode
NPFTNLB	--	V-mode (unshared)
PFTNLB	--	V-mode
SVCLIB	--	R-mode (maintained for PRIMOS-II)

WRBIN

Purpose

WRBIN writes binary data to any output device.

Usage

```
DCL WRBIN ENTRY(FIXED BIN(15), CHAR(*), FIXED BIN(15),
                FIXED BIN(15));

CALL WRBIN (logical_device,buffer,count,altrtn);
```

Parameters

logical_device

INPUT. The device-independent logical I/O unit (Table 2-3), synonymous with the FORTRAN logical unit. This number cannot be changed.

buffer

INPUT. The name of a data area from which data in memory is moved to the output device.

count

INPUT. The number of halfwords to be transferred, or the length in halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument but not use 0.

Discussion

The number of halfwords specified by count are written from buffer to the specific output device. The format of the data is device-dependent.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

RDBIN

Purpose

RDBIN reads binary input from any input device.

Usage

```
DCL RDBIN ENTRY(FIXED BIN(15), CHAR(*), FIXED BIN(15),
                FIXED BIN(15));

CALL RDBIN (logical_device,buffer,count,
            altrtn);
```

Parameters

logical_device

INPUT. The device-independent logical I/O unit (Table 2-3), synonymous with the FORTRAN logical unit. This number cannot be changed.

buffer

OUTPUT. The name of a data area in memory to which data is moved from the input device.

count

INPUT. The number of halfwords to be transferred, or the length in halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument but not use 0.

Discussion

A record is read into memory. Count is the maximum number of halfwords that will be read into buffer. If the record is less than count long, then count will be set to the number of halfwords actually read. If the record is longer than count, only the first count halfwords will be read.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

CONTRL

Purpose

Use of CONTRL provides certain nondata transfer functions, such as opening a PRIMOS file for reading. CONTRL has generally been replaced with SRCH$$, but is maintained here for certain IOCS applications.

Usage

```
DCL CONTRL ENTRY(FIXED BIN(15), CHAR(*) VARYING, FIXED BIN(15),
                FIXED BIN(15));

CALL CONTRL (key, name, logical_device, altrtn);
```

Parameters

key

INPUT. A numeric option code that may have the following values:

1	Open for reading.
2	Open for writing.
3	Open for read/write.
4	Close.
5	Delete file.
6	Move forward one file mark (MT only).
7	Rewind to beginning of file.
8	Select device and read status (MT only). Status is returned in the A-register, and must be read by a user-written PMA subroutine.
-1	Write file mark (MT only).
-2	Backspace one record (MT only).
-3	Backspace one file mark (MT only).
-4	Rewind to beginning of tape (MT only).

Note

For calls to disk files, key may have many other values. See SRCH$$. Keys other than 1-4 are not device-independent.

name

INPUT. Filename (0 if none).

logical_device

INPUT. The device-independent logical I/O unit (Table 2-3), synonymous with the FORTRAN logical unit.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument but not use 0.

Discussion

Functions not applicable to a particular device are ignored; therefore, functions can be requested in a device-independent way. See Table 4-1 for operation effects.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

Table 4-1
List of Keys and Operating Effects for CONTRL

Key	Terminal (C$A01)	Paper-Tape Reader/Punch (C$P02)	Magtape (C$Mxx)	Disk (SEARCH)
1	a	a	a	a
2	q	q	b	b
3	q	q	c	c
4	r	r	d	p
5	--	--	h	e
6	q	q	i	z
7	s	s	n	f
8	--	--	k	g
-1	--	--	l	z
-2	--	--	m	z
-3	--	--	n	z
-4	--	--	o	z

a	Open for read.
b	Open for write.
c	Open to read and write.
d	Rewind and close file.
e	Delete file.
f	Position to beginning of file.
g	Truncate file.
h	Move forward one record.
i	Move forward one file mark.
k	Select device and read status.
l	Write file mark.
m	Backspace one record.
n	Backspace one file mark.
o	Rewind to BOT (beginning of tape).
p	Close file.
q	Turn on punch and punch reader.
r	If device was open for output, punch trailer and turn off paper-tape punch and reader.
s	Halts allowing operator to rewind tape. Type 'START' to continue.
z	Abort (BAD KEY error).

Keys other than 1 through 4 are not device-independent.

5
Disk Subroutines

This chapter describes two groups of subroutines for disk I/O operations. It also describes the subroutine DKGEO$.

The first group is a subset of the device-dependent drivers listed in Table 2-1. They are the drivers listed in the rows for the PRIMOS file system. Most users will find that other (file system) subroutines described in Volume II do the same function as these routines, but in a manner more accessible to the user.

The second group of subroutines are obsolete non-file-system disk subroutines: D$INIT, RRECL, and WRECL. The subroutines are maintained here for any remaining sites using such a disk system not based on files.

The last subroutine, DKGEO$, is used for registering the format of non-standard disks with a disk driver.

These are the subroutines presented or listed in this chapter:

Routine	Meaning
O$AD07	Write ASCII to disk (obsolete).
I$AD07	Read ASCII from disk.
O$BD07	Write binary to disk.
I$BD07	Read binary from disk.
O$AD08	Write ASCII to disk (fixed-length records).
D$INIT	Initialize disk (obsolete).
RRECL	Read one disk record (obsolete).
WRECL	Write one disk record (obsolete).
DKGEO$	Register disk format with driver.

Caution

R-mode subroutines can be called from FTN and PMA in R-mode only. If you call an R-mode routine from a program in a different mode, the results are unpredictable. Refer to the FORTRAN and PMA chapters in Volume I for information on declaring parameters in FTN and PMA, respectively.

DRIVER SUBROUTINES

These subroutines are the drivers listed in Table 2-1 as the device-dependent drivers for the PRIMOS file system, in both compressed and uncompressed formats. They are: O$AD07, I$AD07, O$BD07, I$BD07, and O$BD08.

O$AD07

Note

O$AD07 has been replaced by WTLIN$ (see Volume II). The description for O$AD07 has been relocated to Appendix E "Other Obsolete Subroutines."

I$AD07

Purpose

I$AD07 reads information from the disk file open on file_unit, in compressed ASCII format.

Usage

```
DCL I$AD07 ENTRY(FIXED BIN(15), CHAR(*),
                 FIXED BIN(15), FIXED BIN(15));

CALL I$AD07 (file_unit, buffer, count, altrn);
```

Parameters

file_unit

INPUT. The PRIMOS file unit (funit) number from 0 through 32761. (Users may assign 2 through 32761.) Since a file unit has a position and access method, a user program need not keep track of a file's position and access. Examples of file unit strategy are given with SRCH$$ in Volume II.

buffer

OUTPUT. The name of a data area in memory to which data is moved from the disk file.

count

INPUT. The number of halfwords to be transferred, or the length in halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

O$BD07

Purpose

O$BD07 writes binary information to the file open on file_unit.

Usage

```
DCL O$BD07 ENTRY(FIXED BIN(15), CHAR(*),
                 FIXED BIN(15), FIXED BIN(15));

CALL O$BD07 (file_unit, buffer, count, altrtn);
```

Parameters

file_unit

INPUT. The PRIMOS file unit (funit) number from 0 through 32761. (Users may assign 2 through 32761.) Since a file unit has a position and access method, a user program need not keep track of a file's position and access. Examples of file unit strategy are given with SRCH$$ in Volume II.

buffer

INPUT. The name of a data area in memory from which data is moved to the disk file.

count

INPUT. The number of halfwords to be transferred, or the length in halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

I$BD07

Purpose

I$BD07 reads binary information from the file open on file_unit.

Usage

```
DCL I$BD07 ENTRY(FIXED BIN(15), CHAR(*),
                 FIXED BIN(15), FIXED BIN(15));

CALL I$BD07 (file_unit, buffer, count, altrtn);
```

Parameters

file_unit

INPUT. The PRIMOS file unit (funit) number from 0 throuhg 32761. (Users may assign 2 through 32761.) Since a file unit has a position and access method, a user program need not keep track of a file's position and access. Examples of file unit strategy are given with SRCH$$ in Volume II.

buffer

OUTPUT. The name of a data area in memory to which data is moved from the disk file.

count

INPUT. The number of halfwords to be transferred, or the length in halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

O$AD08

Purpose

O$AD08 writes ASCII from buffer onto the disk file open on file_unit.

Usage

```
DCL O$AD08 ENTRY(FIXED BIN(15), CHAR(*),
                 FIXED BIN(15), FIXED BIN(15));

CALL O$AD08 (file_unit, buffer, count, altrtn);
```

Parameters

file_unit

INPUT. The PRIMOS file unit (funit) number from 0 through 32761. (Users may assign 2 through 32761.) Since a file unit has a position and access method, a user program need not keep track of a file's position and access. Examples of file unit strategy are given with SRCH$$ in Volume II.

buffer

INPUT. The name of a data area in memory from which data is moved to the disk file.

count

INPUT. The number of halfwords to be transferred, or the length in halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Discussion

Information is written on the disk in fixed-length records. Each record consists of count halfwords followed by a halfword containing NL and NULL ('105000). This driver is not in the standard CONIOC supplied by Prime.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

OBSOLETE DISK SUBROUTINES

The subroutines D$INIT, RRECL, and WRECL are not in FTNLIB. They are intended for use by the System Administrator.

D$INIT

Purpose

The D$INIT routine is called to initialize disk devices.

Usage

```
DCL D$INIT ENTRY(FIXED BIN(15));

CALL D$INIT (pdisk);
```

Parameters

pdisk

INPUT. The physical disk number to be initialized. (See RRECL, following the Discussion below.)

Discussion

D$INIT initializes the disk controller and performs a seek to cylinder 0 on pdisk. D$INIT must be called prior to any RRECL or WRECL calls. pdisk must be assigned by the PRIMOS ASSIGN command before calling this routine. D$INIT was intended by use only by outdated system utilities.

RRECL

Purpose

Subroutine RRECL reads one disk record from a disk into buffer in memory. Before RRECL is called, the disk must be assigned by the PRIMOS ASSIGN command and D$INIT must be called to initialize the disk.

The RRECL routine was intended for use only by now-outdated system utilities such as FIXRAT, MAKE, and the old disk COPY.

Usage

```
DCL RRECL ENTRY(PTR, FIXED BIN(15), FIXED BIN(15),
                FIXED BIN(31), BITU(16) ALIGNED, FIXED BIN(15));

CALL RRECL (addr(buffer), length, option_word
            ra, pdisk, altrtn);
```

Parameters

addr(buffer)

INPUT. Pointer to an array into which length halfwords from record ra is to be transferred.

length

INPUT. The number of halfwords to be transferred.

option_word

INPUT. A 16-bit halfword with the following options:

Bit 1 set — Perform current record address check.

Bit 2 set — Ignore checksum error.

Bit 3 set — Read an entire track (beginning at ra) into a buffer 3520 halfwords long, beginning at the buffer pointed to by ra. (This feature may be used only if RRECL is running under PRIMOS II, is reading a disk connected to the 4001/4002 controller, and is a 32-sector pack.)

Bit 4 set Format the track. This bit is only significant for storage module disks.

Bits 5-8 Reserved.

Bits 9-16 Must be set on (1).

ra

INPUT. A 32-bit integer (INTEGER*4) specifying a disk record address. Legal addresses depend on the size of the disk.

Size	ra Range
Floppy disk	0-303
1.5M disk pack	0-3247
3.0M disk pack	0-6495
30M disk pack	0-64959
128K fixed-head disk	0-255
256K fixed-head disk	0-511
512K fixed-head disk	0-1023
1024K fixed-head disk	0-2047

pdisk

INPUT. The physical disk number of the disk to be read. pdisk numbers are the same numbers available for use in the ASSIGN and STARTUP commands of PRIMOS.

altrtn

INPUT. An integer variable in the user's FORTRAN program to be used as an alternate return in case of uncorrectable disk errors. If this argument is 0 or omitted, an error message is printed.

Discussion

If an error is encountered and control goes to altrtn, ERRVEC (see Appendix B in this Volume) is set as follows:

Code	Message	Meaning
ERRVEC(1) = WB	On supervisor terminal: 10 times	Disk hardware
ERRVEC(2) = 0	DISK RD ERROR pdisk ra status On user terminal: UNRECOVERED ERROR	WRITE PROTECT error
ERRVEC(1) = WB	On user terminal: 10 times	Current record
ERRVEC(2) = CR	DISK RD ERROR pdisk ra status followed by UNRECOVERED ERROR	Address error

See the System Administrator's Guide for a description of status error codes.

Notes

Length must be between 0 and 448 unless pdisk is a storage module, in which case length must be between 0 and 1040. If this number is not 448 and pdisk is 20-27 (diskette), a checksum error is always generated; bypassing can be accomplished by setting the option-word's bit 2 to 1. No check is made for legality of ra.

On a DISK NOT READY, RRECL does not wait for the disk to become ready under PRIMOS III or PRIMOS. Under PRIMOS II, RRECL prints a single error message and waits for the disk to become ready.

On any other read error, an error message is printed at the system terminal, followed by a seek to cylinder 0 and a reread of the record. If 10 errors occur, the message UNRECOVERED ERROR is typed to the user or altrtn is taken.

WRECL

Purpose

Subroutine WRECL writes the disk record to a disk from buffer in memory. The arguments and rules of the WRECL call are identical to those of RRECL except for bits 1 and 2 of option-word, which have no meaning on write. For a call to write a record on the diskette, the buffer length must be 448 words.

D$INIT must be called before a call to WRECL.

Usage

```
DCL WRECL ENTRY(PTR, FIXED BIN(15), FIXED BIN(15),
                BIT(16) ALIGNED, FIXED BIN(15));

CALL WRECL (addr(buffer), length, option_word,
            ra, altrtn);
```

Parameters

Same as for RRECL. See below for clarification.

Discussion

The meaning of the parameters is the same as previously described in RRECL, except that the function of the command is to write to, rather than read from, the specified record address. The user of WRECL is responsible for being careful to write only on areas of the disk that do not contain significant user or operating system information. An attempt to write on a write-protected disk generates the message:

```
DISK   WT  ERROR    pdisk    option-word    status
WRITE  PROTECT
```

on the supervisor terminal and the message:

```
UNRECOVERED ERROR
```

at the user terminal. ERRVEC(1) will contain error code WB, unless altrtn is taken. Other write errors are retried ten times in a manner similar to read errors. (Refer to RRECL.)

DKGEO$

Purpose

This subroutine supplies the disk driver with the sector count for non-standard disk formatting. You may have programs that read and write assigned disks directly, without using the PRIMOS file system. The default formatting for disk files is 9 sectors per track. If your disk is formatted with a different number of sectors, DKGEO$ must be called to register the number of sectors with the disk driver.

Usage

```
DCL DKGEO$ ENTRY (FIXED BIN(15), POINTER, FIXED BIN(15));

CALL DKGEO$ (pdev, structp, code);
```

Parameters

pdev

INPUT. Physical device number.

structp

INPUT. Pointer to the address of an input structure, with the following format:

```
dcl 1 struc,
      2 version_number bin(15),    /* Must be 0 (input)*/
      2 spt bin(15);               /* (input)*/
```

code

OUTPUT. Returns either 0 for success or one of the standard error codes, as given in Appendix A.

Discussion

This subroutine is necessary only if the number of sectors per track is not 9. The disk driver's record reverts to 9 when the disk is unassigned.

Loading and Linking Information

NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode

6
Terminal Drivers and Terminal/Paper-Tape Subroutines

OVERVIEW

This chapter defines certain terminal driver subroutines.

This chapter also defines subroutines used to transfer data to and from a user terminal or card reader/punch (ASR). Some of these are a subset of the device-dependent IOCS drivers shown in Table 2-1, in the rows for the user terminal and for paper tape.

The subroutines described in this chapter are listed in Table 6-1.

For the continuity and completeness of this chapter on user-terminal and paper-tape subroutines, a second table, (Table 6-2) also lists other subroutines for general terminal use. However, these subroutines are described in Volume III.

Note

These subroutines expect data to be halfword-aligned. Be aware that F77 calls with a substring argument do not always pass the argument left-justified in a halfword.

Table 6-1
Subroutines for User Terminal and Paper Tape

Device	Routine	Function
User terminal	C$A01	Controls functions for user terminal.
User terminal or ASR punch	O$AA01	Outputs ASCII to the user terminal or ASR punch.
Keyboard or ASR reader	I$AA01	Inputs ASCII from terminal or ASR reader.
	I$AA12	Performs the same function as I$AA01 but also allows the input to be from a cominput file.
Paper Tape	C$P02	Controls functions for paper tape.
	I$AP02	Inputs ASCII from the high-speed paper-tape reader.
	O$BP02	Outputs binary data to the high-speed paper-tape punch.
	P1IB	Inputs one character from the high-speed paper-tape reader to Register A.
	P1OB	Outputs one character to the high-speed paper-tape punch from Register A.
	P1IN	Inputs one character from paper tape, sets high-order bit, ignores line feeds, sends a line feed when carriage return is read.
	P1OU	Outputs one character to the high-speed paper-tape punch.

Table 6-2
Subroutines for General Terminal Use

Device	Routine	Function
User terminal	BREAK$	Inhibits or enables CONTROL-P.
	C1IN	Gets next character from terminal or command file.
	C1IN$	Gets next character from command line until carriage return.
	CNIN$	Moves characters from terminal or command file to memory.
	COMANL	Reads a line of text from the terminal or from a command file.
	ERKL$$	Reads or sets erase and kill characters.
	TNOU	Outputs count characters to the user terminal followed by a line feed and carriage return.
	TNOUA	Outputs count characters to the user terminal.
	TOVFD$	Outputs the 16-bit integer num to the terminal.
	T1IB	Reads one character from the user terminal into Register A.
	T1IN	Reads one character from the user terminal.
	T1OB	Writes one character from Register A to the user terminal.
	T1OU	Outputs char to the user terminal. The data type must be a 16-bit integer in F77.

Table 6-2
Subroutines for General Terminal Use
(Continued)

Device	Routine	Function
User terminal	TIDEC	Inhibits or enables CONTROL-P.
	TIDEC	Inputs decimal number.
	TIOCT	Inputs an octal number.
	TIHEX	Inputs a hexadecimal number.
	TODEC	Outputs a six-character signed decimal number.
	TOOCT	Outputs a six-character unsigned octal number.
	TOHEX	Outputs a four-character unsigned hexadecimal number.
	TONL	Outputs Carriage return and Line feed.

Caution

R-mode subroutines can be called from FTN and PMA in R-mode only. If you call an R-mode routine from a program in a different mode, the results are unpredictable. Refer to the FORTRAN and PMA chapters in Volume I for information on declaring parameters in FTN and PMA, respectively.

C$A01

Purpose

C$A01 provides control functions for the user terminal. Because it is written in R-mode only, the calling program must be written in either FTN (as described below) or PMA.

Usage

```
INTEGER*2 key
INTEGER*2 name(1)
INTEGER*2 unit
INTEGER*2 altrtn

CALL C$A01 (key, name, unit[, altrtn])
```

Parameters

key

INPUT. Valid keys for C$A01 are 1 through 4 and 6 and 7. Refer to Table 4-1 for the operating effects for each key.

name(1)

INPUT. The filename of the array for which the key declares a control function. Rules for PRIMOS filenames apply.

unit

INPUT. Indicates the sub-unit number for this user terminal.

altrtn

INPUT. A parameter not used by this routine, but maintained for coding purposes.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

O$AA01

Purpose

O$AA01 outputs ASCII to the user terminal or ASR punch.

Usage

```
DCL O$AA01 ENTRY(FIXED BIN(15), CHAR(*),
                 FIXED BIN(15)[, FIXED BIN(15)]);

CALL O$AA01 (sub_unit, buffer,
             count[, altrtn]);
```

Parameters

sub_unit

INPUT. The sub-unit number of a physical device having more than one unit. If the multi-unit device is an ASR card reader, the possible choices are:

0	CR0, first controller
1	CR1, second controller

buffer

INPUT. Name of data area holding data for output to the device.

count

INPUT. Number of halfwords to be moved, two characters per halfword.

altrtn

INPUT. A parameter not used by this routine, but maintained for coding purposes.

Discussion

This subroutine itself calls the driver TNOU to perform the output from buffer to the size of count.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

I$AA01

Purpose

I$AA01 reads ASCII from the terminal or ASR reader.

Usage

```
DCL I$AA01 ENTRY(FIXED BIN(15), CHAR(*) VARYING,
                 FIXED BIN(15)[, FIXED BIN(15)]);

CALL I$AA01 (sub_unit, buffer,
             count [, altrtn]);
```

Parameters

sub_unit

INPUT. The sub-unit number of a physical device having more than one unit. If the multi-unit device is an ASR card reader, the possible choices are:

0	CR0, first controller
1	CR1, second controller

buffer

OUTPUT. Name of data area that holds the data output from the device.

count

INPUT. Number of halfwords to be moved, two characters per halfword.

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error. It is a numeric label in the user's program; the number must be preceded by a $. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Discussion

The kill and erase characters (question mark and quote mark by default) may modify the input line, as with the PRIMOS III command line. The characters NUL, DEL, DLE, DC2, DC3, and DC4 are ignored. The character EXT ('203) indicates the end of file and is used for reading tapes through the user terminal.

Note that I$AA01 is not the entry for the user terminal in the Prime-supplied CONIOC (Chapter 3). Ask your System Administrator to put I$AA01 in the RATBL, as explained in Chapter 3, to read paper tapes with user programs. The editor should be used to read in the tape, and then the user may read the file from disk.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

I$AA12

Purpose

I$AA12 performs the same function as I$AA01 (it reads ASCII from the terminal or ASR reader) but also allows the input from a cominput file.

Usage

```
DCL I$AA12 ENTRY(FIXED BIN(15), CHAR(*) VARYING,
                 FIXED BIN(15)[, FIXED BIN(15)]);

CALL I$AA12 (sub_unit, buffer,
             count[, altrtn]);
```

Parameters

sub_unit

INPUT. The sub-unit number of a physical device having more than one unit. If the multi-unit device is an ASR card reader, the possible choices are:

0	CR0, first controller
1	CR1, second controller

buffer

OUTPUT. Name of data area that holds the data output from the device.

count

INPUT. Number of halfwords to be moved, two characters per halfword.

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error. It is a numeric label in the user's program; the number must be preceded by a $. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Discussion

Refer to the discussion for I$AA01 for details on how I$AA12 handles command characters. However, note that I$AA12 is the subroutine referenced in the RATBL already.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

C$P02

Purpose

C$P02 provides control functions for paper tape.

Usage

```
DCL C$P02 ENTRY(FIXED BIN(15), CHAR(*) VARYING,
                FIXED BIN(15)[, FIXED BIN(15)]);

CALL C$P02 (key, name,
            physical_unit [, altrtn]);
```

Parameters

key

INPUT. Valid keys for C$P02 are 1 through 4 and 6 and 7. Refer to Table 4-1 for the operating effects for each key.

name

INPUT. The filename for which the key declares its control function. Rules for PRIMOS filenames apply.

physical_unit

INPUT. Indicates the sub-unit number for this paper-tape reader/punch.

altrtn

OUTPUT. A parameter not used by this routine, but maintained for coding purposes.

Loading and Linking Information

```
FTNLIB   --   R-mode
NPFTNLB  --   V-mode (unshared)
PFTNLB   --   V-mode
SVCLIB   --   R-mode (maintained for PRIMOS-II)
```

I$AP02

Purpose

I$AP02 reads ASCII from the high-speed paper-tape reader.

Usage

```
DCL I$AP02 ENTRY(FIXED BIN(15), CHAR(*) VARYING, FIXED BIN(15)[,
                 FIXED BIN(15)]);

CALL I$AP02 (sub_unit, buffer, count[,
             altrtn]);
```

Parameters

sub_unit

INPUT. The sub-unit number of a physical device having more than one unit. If the multi-unit device is an ASR card reader, the possible choices are:

0	CR0, first controller
1	CR1, second controller

buffer

OUTPUT. Name of data area that holds the data output from the device.

count

INPUT. Number of halfwords to be moved, two characters per halfword.

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error. It is a numeric label in the user's program; the number must be preceded by a $. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Discussion

The KILL and ERASE characters (question mark and double quote by default) modify the input. NUL, DEL, DLE, DC2, DC3, and DC4 are ignored. The character ETX ('203) indicates end of file.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

O$BP02

Purpose

O$BP02 writes binary data to the high-speed paper-tape punch. Because this subroutine is written in R-mode only, so must the calling program. The Usage description below is given in FTN to suggest a calling program in R-mode, compiled without the -64V option.

Usage

```
INTEGER*2 unit
INTEGER*2 buffer(1)
INTEGER*2 hwcnt
INTEGER*2 altrtn

CALL O$BP02 (unit, buffer, hwcnt[, altrtn])
```

Parameters

UNIT

INPUT. A sub-unit of the physical device, in this case a high-speed paper-tape reader. Refer to Table 2-2 (paper-tape punch is assigned to physical device #2).

buffer

INPUT. Data area name for the array that receives data from the device.

hwcnt

INPUT. A count of the number of halfwords to be moved, two characters per halfword.

altrtn

INPUT. A parameter not used by this routine, but maintained for coding purposes.

Discussion

The format of the paper-tape output can be found in a listing of this driver. Ask your System Administrator for a copy of the listing.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

P1IB

Purpose

P1IB reads one character from the high-speed paper-tape reader to Register A.

Usage

```
DCL P1IB;

CALL P1IB;
```

Discussion

This subroutine has no arguments; the calling program (for example, a program written in Prime Macro Assembly language) must have access to Register A.

Note

Data items used by the routines CNIN$, TNOU, TNOUA, TOVFD$, T1IB, T1OB, T1IN, T1OU, TIDEC, TIOCT, TIHEX, TODEC, TOOCT, TOHEX, TONL, P1IB, P1OU, AND P1IN must be halfword-aligned. Thus, for example, a FORTRAN statement such as

```
CALL TNOUA (A(I:I),INTS(3))
```

always outputs a halfword-aligned byte of data item A, though the user may be expecting the second byte of a halfword to be displayed.

Loading and Linking Information

```
FTNLIB   --   R-mode
NPFTNLB  --   V-mode (unshared)
PFTNLB   --   V-mode
SVCLIB   --   R-mode (maintained for PRIMOS-II)
```

P1OB

Purpose

P1OB writes one character to the high-speed paper-tape punch from Register A.

Usage

```
DCL P1OB;

CALL P1OB;
```

Discussion

This subroutine has no arguments; the calling program must have access to Register A.

Loading and Linking Information

```
FTNLIB   --   R-mode
NPFTNLB  --   V-mode (unshared)
PFTNLB   --   V-mode
SVCLIB   --   R-mode (maintained for PRIMOS-II)
```

P1IN

Purpose

P1IN reads one character from paper tape.

Usage

```
DCL P1IN ENTRY(FIXED BIN(15));

CALL P1IN (char);
```

Parameters

char

OUTPUT. The character being loaded into memory from paper tape.

Discussion

The subroutine sets the high-order bit, ignores line feeds, and sends a Line feed when a Carriage return is read.

Note

Data items used by the routines CNIN$, TNOU, TNOUA, TOVFD$, T1IB, T1OB, T1IN, T1OU, TIDEC, TIOCT, TIHEX, TODEC, TOOCT, TOHEX, TONL, P1IB, P1OU, AND P1IN must be halfword-aligned. Refer to P1IB for a FORTRAN example.

Loading and Linking Information

FTNLIB	--	R-mode
NPFTNLB	--	V-mode (unshared)
PFTNLB	--	V-mode
SVCLIB	--	R-mode (maintained for PRIMOS-II)

P1OU

Purpose

P1OU writes one character to the high-speed paper-tape punch.

Usage

```
DCL P1OU entry(fixed bin(15));

CALL P1OU (char);
```

Parameters

char

INPUT. The character being written to paper tape.

Discussion

Zero the high-order bit before punching. No special action is taken on Carriage returns or Line feeds.

Note

Data items used by the routines CNIN$, TNOU, TNOUA, TOVFD$, T1IB, T1OB, T1IN, T1OU, TIDEC, TIOCT, TIHEX, TODEC, TOOCT, TOHEX, TONL, P1IB, P1OU, AND P1IN must be halfword-aligned. Refer to P1IB for a FORTRAN example.

Loading and Linking Information

FTNLIB	--	R-mode
NPFTNLB	--	V-mode (unshared)
PFTNLB	--	V-mode
SVCLIB	--	R-mode (maintained for PRIMOS-II)

7
Other Peripheral Devices

This chapter describes subroutines that control line printers, printers/plotters, card readers, and magnetic tapes. These subroutines are used for both formatted and raw data. Not all are in IOCS. Table 7-1 gives a list of the subroutines in this chapter.

LINE PRINTER SUBROUTINES

IOCS contains subroutines to control three types of line printers:

- O$AL04 to print on a Centronics line printer connected to the System Option Controller (SOC);
- O$AL06 to print on a parallel-interface line printer connected to the MPC Line Printer Controller;
- O$AL14 to print on a Versatec Printer/Plotter connected to a Versatec-SOC.

This section also includes SPOOL$ and SP$REQ for queuing files to be printed, and T$LMPC to move data to the MPC line printer.

Table 7-1
Peripheral-handling Subroutines

Line Printers

O$AL04	Centronics LP.
O$AL06	Parallel interface to line printer (MPC).
O$AL14	Versatec printer.
T$LMPC	Move data to LPC line printer.
SPOOL$	Insert a file in spooler queue.
SP$REQ	Insert a file in spooler queue.

Printer/Plotter

O$AL14	Versatec.
T$VG	Versatec.

Card Reader/Punch

I$AC03	Input from parallel card reader.
I$AC09	Input from serial card reader.
I$AC15	Read and print card from parallel interface reader.
T$CMPC	Input from MPC card reader.
O$AC03	Parallel interface to card punch.
O$AC15	Parallel interface to card punch and print on card.
T$PMPC	Raw data mover.

Magnetic Tape

C$M05	Control functions for 9-track ASCII/binary (obsolete).
C$M10	Control functions for 7-track ASCII/binary (obsolete).
C$M11	Control functions for 7-track EBCDIC (obsolete).
C$M13	Control functions for 9-track EBCDIC (obsolete).
O$AM05	Write ASCII to 9-track (obsolete).
O$AM10	Write ASCII to 7-track (obsolete).
I$AM05	Read ASCII to 9-track (obsolete).
I$AM10	Read ASCII from 7-track (obsolete).
O$BM05	Write binary to 9-track (obsolete).
O$BM10	Write binary to 7-track (obsolete).
I$BM05	Read binary from 9-track (obsolete).
I$BM10	Read binary from 7-track (obsolete).
O$AM11	Write BCD to 7-track (obsolete).
O$AM13	Write EBCDIC to 9-track (obsolete).
I$AM11	Read BCD from 7-track (obsolete).
I$AM13	Read EBCDIC from 9-track (obsolete).
T$MT	Raw data mover.

Caution

R-mode subroutines can be called from FTN and PMA in R-mode only. If you call an R-mode routine from a program in a different mode, the results are unpredictable.

O$AL04 or O$AL06

Purpose

Both these subroutines provide an interface to the line printers. The two use identical subroutine calling formats, and therefore have a Usage description below as O$ALxx, where xx is replaced by the numbers of the subroutine required. However, O$AL04 is for the serial line printer, and must be called by a program using R-mode (either an FTN or PMA program) while O$AL06 is for the MPC line printer.

O$AL14 is discussed separately below.

Usage

```
DCL O$ALXX ENTRY (FIXED BIN(15), CHAR(*) VARYING, FIXED BIN(15),
                  FIXED BIN(15));

CALL O$ALXX (physical_unit, buffer, count, altrtn);
```

Parameters

physical_unit

INPUT. Indicates the line printer unit number, with the following possible values:

0	PR0, first controller
1	PR1, first controller
2	PR2, second controller
3	PR3, second controller

buffer

INPUT. The name of the buffer where the text to be printed resides. Print text is placed in the buffer, two characters per halfword.

count

INPUT. The number of 16-bit halfwords of data to be printed.

altrtn

INPUT. An optional parameter, used if O$ALxx encounters an error. If an error occurs, control passes to the area within the calling program named by this parameter.

Discussion

For more information on arguments, see Chapter 5.

Printer Control: The action taken by O$ALxx depends on the data in the buffer, and the current vertical control mode. Certain characters within the data control the manner in which the data is printed. These characters (codes) are described in the following paragraphs.

Vertical Control Modes: O$ALxx has three vertical control modes:

- Forms control
- Header line and pagination control
- No-control

O$ALxx checks the first character in the data buffer for a .SOM. or start-of-message character (ASCII '001). This character signifies a change in the control mode. If the first character in the buffer is not .SOM., the line is printed according to the current control mode. The default mode is forms control.

Forms Control Mode: The first character in the buffer is not printed; instead, it is used for forms control. Two different forms control modes exist, one for FORTRAN and one for COBOL, as described below.

FORTRAN Mode

FORTRAN mode allows the attaching of vertical format information to each line of the data file. The first character position of each line from the file does not appear in the printed output, and is interpreted as shown below.

Character	Meaning
1	Eject to top of next page.
+	Print over previous line.
space	Advance one line.
0	Advance two lines.
-	Advance three lines (skip two lines).

All other characters are interpreted as advance one line.

COBOL Mode

COBOL mode is identical to FORTRAN mode except that the format information occupies the first two character positions of the line. The first character is the same as for FORTRAN mode and the second character is ignored.

Header Line and Pagination Control Mode: In header line and pagination mode, O$ALxx causes a header line to be printed, followed by three blank lines, followed by 38 text lines. The header line consists of up to 43 characters followed by a page count that is generated by O$ALxx when printing in this mode.

For O$AL06 and O$AL14, enter pagination mode with a first halfword of '000001 in buffer. In pagination mode with O$AL04, a form feed (octal 14 or 214) may be anywhere in the buffer line. All characters preceding the form feed are printed, and all characters after it are ignored. With O$AL04, the form feed must be in column 1 or 3.

No-control Mode: In No-control mode, no actions are taken by O$ALxx. A line containing an ASCII Form feed character (FF, '214) causes the line preceding it to print, followed by a page eject. Carriage return (CR, '215) causes the line preceding it to print with no line spacing. Line feed (LF, '212) causes the line preceding it to print followed by a line spacing operation. Any characters following a CR, LF, or FF are ignored.

Change of Mode Commands: Any data buffer beginning with a .SOM. character causes O$ALxx to take some action to change control mode. The control mode change is determined by the character following the .SOM., and is activated when a file is printed. The character interpretations are:

000	Enter no-control mode.
001	Enter control mode.
036	New header line - DO NOT reset page count.
037	Enter new page size specified by the 16-bit number contained in the next computer halfword.
Any Other	Enter header control mode characters.

Early Buffer Termination: A line feed (LF, '212) character terminates the print line in the buffer, regardless of the count parameter.

Load Information: O$AL04 calls no other subroutines. O$AL06 calls T$LMPC.

Loading and Linking Information

For O$AL04:

FTNLIB	--	R-mode

For O$AL06:

FTNLIB	--	R-mode
NPFTNLB	--	V-mode (unshared)
PFTNLB	--	V-mode

T$LMPC

Purpose

T$LMPC is a raw data mover, moving information from the user to one line on the MPC line printer.

The user normally prints lines under program control using either FORTRAN WRITE statements or a call to O$AL06, which in turn calls T$LMPC. However, it is possible to call T$LMPC directly.

Usage

```
DCL T$LMPC ENTRY(FIXED BIN(15), PTR, FIXED BIN(15),
                 FIXED BIN(15), FIXED BIN(31));

CALL T$LMPC (logical_unit, addr(buffer), count,
             instr, status);
```

Parameters

logical_unit

INPUT. Line printer unit.

addr(buffer)

INPUT. A pointer to the buffer holding information to be printed on the line printer. Information is expected to be packed two characters per halfword.

count

INPUT. Number of halfwords to print on the current line.

instr

INPUT. The instruction required by the line printer. Valid instructions are:

Instruction (Octal)	Meaning
'100000	Read status.
'40000	Print a line.
'20012	Skip a line.

Instruction (Octal)	Meaning
'20014	Skip to top of page.
'20100-20113	Skip to tape channel 0-11.
'20120-20137	Skip from 1 to 15 lines.

status

OUTPUT. A three-halfword vector that contains device code, status of printer, and a space. Possible printer status is:

Octal Value	Condition
'200	Online
'100	Not busy

Discussion

Under PRIMOS, line printer output is buffered. If T$LMPC is called and the buffer is full, the user is placed in output-wait state. Later, when the buffer is no longer full, the user is rescheduled, and the T$LMPC call is retried. The user may issue a status-request call to check if the buffer is full. If the buffer is full, then the not-busy status is reset. Using this feature, a user program may check that the buffer is not full, then output one line, or do another computation if the buffer is full. Under PRIMOS II, output is not buffered, and control does not return to the user until printing is complete.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

SPOOL$

Purpose

SPOOL$ accesses the spool queue.

Usage

```
DCL SPOOL$ ENTRY(FIXED BIN(15), CHAR(*), FIXED BIN(15), CHAR(80),
                 CHAR(*), FIXED BIN(15), FIXED BIN(15));

CALL SPOOL$ (key, name, namlen, info, buffer, buflen, code);
```

Parameters

key

INPUT. Indicates a user option. Possible values are:

1 Copy named file into queue. The name argument holds the pathname of the file to be queued as a fixed-length string.

2 Make a queue entry and open a data file on the file unit given in info(2). This gives the program write access to the file for printing, but the program loses control of the file when it is closed.

3 Modify the spool queue entry identified by the request number in info(8-10). This queue entry must belong to the calling user or a user who has privileged queue access rights.

4 Close the file on unit info(2), update the queue entry for that request number, (in elements 8 -10 of info), to record the file size, and reactivate a local despooler that matches the request attributes.

name

INPUT. If key is 1, indicates the pathname of the file to be queued. If key is 2, indicates the name to appear on the header page.

namlen

INPUT. Length of name, in characters. If key is 1, name can be from 1 to 160 characters. If key is 2, name can be from 0 to 32 characters.

info

INPUT. Information array of 40 16-bit halfword elements, as follows:

1 Reserved.

2 If key is 2, open print file on this unit. A value of 0 implies that SPOOL$ should select a free file unit. This halfword returns the number of the file unit opened.

3 Print option halfword. Specifies printer and plotter information, as shown below. Corresponding PRIMOS command level options appear in parentheses.

Bit	Meaning
1	Use FORTRAN-format mode (-FTN). Column 1 of each data line holds a format control character; the printed data starts in column 2. Refer to the Forms Control Mode section of the O$AL04 or O$AL06 description earlier in this chapter for a list of these codes.
2	Reserved.
3	Generate line numbers at left margin (-LNU).
4	Suppress header page (-NOH).
5	Suppress final page eject after printing (-NOE).
6	Suppress format control mode (-NOF).
7	Use raster plot mode (-PLOT). info(7) is the raster size.
8	Defer printing to time specified in info(11) (-DEFER).
9	Reserved.
10	Use the logical destination name specified in info(13-20).

11 Replace name with info(21-28) (-AS).

12 Spool the number of copies specified in info(29) (-COPIES).

13 Use COBOL-format mode (-COBOL). This is identical to FORTRAN-format mode except that the data to print begins in column 3 of each line. Refer to the Forms Control Mode section of the O$AL04 or O$AL06 description earlier in this chapter for a description of these codes.

14 Suppress page header format. This is identical to the default print format (pagination mode) except that the normal header line on each page is omitted.

15 Inform user when printing is done (-NOTIFY).

16 Extended array used. This bit must be set if any of the control bits in info(30-40) are used.

4-6 Form type; 6 ASCII characters, blank filled (-FORM). This field is treated as a device attribute when adding or modifying a request in a Rev. 21 spool queue.

7 Plot raster scan size (plot only). This represents the number of halfword/raster scan.

8-10 If key is 3 or 4, input the request number to be modified or closed as a decimal string (use of the prefix PRT, while still supported at Rev. 22, is not recommended). For other keys, returns the request number of the new spool queue entry.

11 Deferred print time as a binary value of minutes past midnight (-DEFER). If the time given is earlier than the current time, the request is deferred to the given time on the following day. Significant only if bit 8 of info(3) is set.

12 Size of request, returned if key is 1. This is the size of the file, in records, multiplied by the number of copies to be printed; 32767 is the maximum size.

13-20 Logical destination name when key is 1, 2, or 3, blank filled (-ATT). Significant only if bit 10 of info(3) is set. This field is treated as a device attribute when adding or modifying a request in a Rev. 21 spool queue.

21-28 Substitute filename to be used in banner page, blank filled (-AS). Significant only if bit 11 of info(3) is set.

29 Number of copies to print when key is 1, 2, or 3 (-COPIES). Significant only if bit 12 of info(3) is set. If this bit is not set, only one copy is printed.

The remaining 11 elements are for the extended array. If the extended array is used, bit 16 of info(3) must be set.

30 Extended print option halfword, with bit descriptions as shown below.

Bit	Meaning
1	Use spool queue on disk identified by info(31) (-DISK).
2	If key is 1, 2, or 3, treat request as priority (-RUSH). Only privileged users can use this bit.
3	If key is 3, cancel priority of request. Only privileged users can use this bit.
4	Reserved.
5	Use spool queue on network node identified by info(35-37) (-ON).
6	Suppress file information on header page (-SFI).
7	Truncate lines longer than defined printer width (-TRU).
8	If key is 3, cancel defer time.
9	Inhibit overprinting (-NOP, -CRLF).
10	Use PostScript procedure named in info(32-34) (-PROC).
11	If key is 1, suppress copying of data file (-NOCOPY).
12-16	Reserved.

31 Logical disk number of disk holding spool queue to which request is to be added. Significant only if bit 1 of info(30) is set. If the disk is on the local system, the disk information is ignored and the request is added to the local queue. If the disk is on a remote system and holds a pre-Rev. 21 spool queue, the request will be added to that queue. If the disk is on a remote system but does not hold a pre-Rev. 21 spool queue, the spooler will attempt to add the request to a Rev. 21 queue on the remote system.

32-34 PostScript procedure name for laser printers supporting the PostScript language. Significant only if bit 10 of info(30) is set.

35-37 Name of network node on which the spool queue is to be accessed (-ON). Significant only if bit 5 of info(30) is set. A single call to SPOOL$ cannot specify both a disk and a node name.

38-40 Reserved.

buffer

SCRATCH. If key is 1, this is the data buffer area used to copy the file. It is used as both an input and output buffer, and must be at least 40 16-bit halfwords long. Copy time is inversely proportional to buflen size.

buflen

INPUT. Length of buffer in halfwords. This should be at least 300 halfwords. A multiple of 1024 gives the best performance.

code

OUTPUT. Standard error code.

Loading and Linking Information

V-mode and I-mode: Load VSP00$.BIN
Code in SP$LIB.RUN.

V-mode and I-mode with unshared libraries: Load VSP00$.BIN.
Code in SP$LIB.RUN.

R-mode: Not available.

SP$REQ

Purpose

SP$REQ places a file in a spool queue and handles requests issued with the SPOOL command.

Usage

```
DCL SP$REQ (CHAR(1024) VAR, FIXED BIN(15), ENTRY(CHAR(*), FIXED
            BIN(15)), FIXED BIN(31), FIXED BIN(15), FIXED BIN(15));

CALL SP$REQ (in_string, op_mode, entry_var, rqst_no, ret_info, code);
```

Parameters

in_string

INPUT. Command argument string composed of the same arguments as those applicable to the SPOOL command. FORTRAN programmers can build the string from a structure of fixed-length substrings. Any number of spaces is permitted between arguments in this string.

op_mode

INPUT. Output mode. Possible values are:

0 Normal terminal output.

1 No terminal output (for most program call uses).

2 Output via supplied routine (for applications requiring special treatment of output).

entry_var

INPUT. Entry required when op_mode is 2. The entry variable must refer to a routine with a calling interface with a fixed character string and a bin(15) byte count. The string must never exceed 80 characters. If op_mode is 0 or 1, this argument is ignored. Refer to the Discussion section for more information about this argument.

rqst_no

OUTPUT. Returned request number from operations that add a new request to the spool queue.

ret_info

OUTPUT. Returned information, depending on action requested. When a file is copied to the queue, this argument returns the total size of the request (file size multiplied by the number of copies). When a file is opened with the -OPEN option, this argument returns the file unit number. For all other actions, the argument is ignored.

code

OUTPUT. Standard error code.

Example

The following PL/I statement shows how to call SP$REQ. Because op_mode is 0, the terminal displays any error message text. The reference to TNOU is optional in this case.

```
CALL SP$REQ ('FILE1 -ftn -cop 2', 0, TNOU, reqno, file_size, code);
```

Discussion

SP$REQ handles all user requests issued with the SPOOL command, as well as any requests by user programs to place a file in the spool queue. If op_mode is 0 or 1, entry_var can be 0.

SP$REQ interprets control codes embedded in files when queued entries are printed. Possible control codes are shown below.

Character	Interpretation
000 000 header	Set a new page header and reset the page number count. The header string is supplied after the two-byte code. This forces pagination mode and causes a page eject. Margins are reset to their default values, as defined in the environment file.
000 header	Same as above.
001 000	Enter no-format mode.
001 001	Enter FORTRAN-format mode.
001 002	Enter COBOL-format mode.
001 003	Enter pagination mode.
001 004	Enter no-header mode. This is the same as pagination mode, except that no page header is printed.
001 005 n	Enter raster plot mode. The following characters hold a decimal count of words to print.
001 036 text	Same as 000 001, except that the page counter is not reset.
002 001 n	Set left margin to column number given as a decimal string after code.
002 002 n	Set right margin to column number given as a decimal string after code.
002 003 n	Set top margin to line number given as a decimal string after code.
002 004 n	Set bottom margin to line number given as a decimal string after code.
002 005	Wrap around text if line exceeds paper width.
002 006	Truncate lines that exceed paper width.
003 n	Skip to EVFU channel defined by second byte. (n) must be in the range of 1 to 12 inclusive.

002 000 text	Set the printer characteristics as specified by text, which may be in mixed cases. The possible values of text are:	
	SET_PORTRAIT	Print paper in portrait. A printer is printing in portrait when it prints text across the shorter width of the paper.
	SET_LANDSCAPE	Print paper in landscape. A printer is printing in landscape when it prints text across the longer width of the paper.
	SET_PAPER_BIN n	For laser printers or other printers that use more than one paper bin. Selects the paper bin specified by n, where n can range from 1 - 9. Manual feed of paper is supported by MANUAL.
	SET_FONT fontname	Select the font specified by fontname, where fontname is a string of up to 32 characters. The string cannot contain spaces.

Loading and Linking Information

V-mode and I-mode with unshared libraries: Load VSPOO$.BIN. Code in SP$LIB.RUN.

R-mode: Not supported.

PRINTER/PLOTTERS

The printer/plotter subroutines are used to drive and control the Versatec printer/plotter. The subroutines are O$AL14 and T$VG.

O$AL14

Purpose

O$AL14 provides the IOCS interface to the Versatec printer. Because O$AL14 is in R-mode only, the calling program must be either FTN (as in the Usage description below) or PMA.

Usage

```
INTEGER*2 unit
INTEGER*2 buffer(1)
INTEGER*2 hwcnt
INTEGER*2 altrtn

CALL O$AL14 (unit, buffer(1), hwcnt, altrtn)
```

Parameters

unit

INPUT. Indicates the line printer unit number, with the following possible values:

0	PR0, first controller
1	PR1, first controller
2	PR2, second controller
3	PR3, second controller

buffer(1)

INPUT. The name of the array from which data is to be moved. Handling of the buffer turns on the first character in the buffer. See Discussion below.

hwcnt

INPUT. Number of halfwords to be transferred.

altrtn

INPUT. A parameter not used by this routine, but maintained for coding purposes.

Discussion

The action taken by O$AL14 depends upon the data in the buffer and the current vertical control mode (first character of buffer).

O$AL14 has three vertical control modes:

1. Forms control
2. Header line and pagination control
3. No-control

The default mode is forms control. O$AL14 checks the first character in the data buffer for a .SOM. (ASCII '001). This character signifies a change in the control mode.

If the first character is a .SOM., O$AL14 makes a change in control mode, determined by the character following the .SOM.:

000	Enter no-control mode.
001	Enter control mode.
036	New header line but do not reset page count.
037	Enter new page size specified by the 16-bit number contained in the next computer halfword.
All others	Enter header control mode.

When entering header control mode, the characters following the .SOM. are stored internally in O$AL14 for use as the header line.

All change of mode commands cause a page eject before any further action.

If the first character is not a .SOM., the line is printed according to the current vertical control mode. These three mode descriptions follow.

Forms Control: In this mode, the first character in a buffer is never printed but is used for forms control. The character interpretations are:

0	Skip one line.
1	Eject to top of next page.
+	Print over last line (if printer model allows).
Other	No action.

Header Line and Pagination: In this mode O$AL14 permits a header line followed by three blank lines, followed by 56 text lines. The header line is 42 characters followed by a page count which is kept automatically by O$AL14 when in this mode.

No-control: In this mode no automatic actions are taken except that any line containing a form-feed character will cause a page eject with no further action.

Load information: This subroutine calls T$VG.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

T$VG

Purpose

T$VG moves raw data from a buffer and prints the data on the Versatec printer via a controller designed for use with the Versatec printer/plotter.

Usage

```
DCL T$VG ENTRY(FIXED BIN(15), PTR, FIXED BIN(15),
               FIXED BIN(15), FIXED BIN(31));

CALL T$VG (physical_unit, addr(buffer), nhwds,
           instruction, status);
```

Parameters

physical-unit

INPUT. Currently always 0, since the controller supports only one device.

addr(buffer)

INPUT. Pointer to the address of user's buffer.

nhwds

INPUT. The number of halfwords in the buffer. The maximum is 512.

instruction

INPUT. A number from 0 to 10 that specifies an action that the device is to take. These instructions are described in detail in the Discussion that follows.

status

OUTPUT. A two-halfword array. Device status is returned in status(2). status is returned only on a status request instruction.

The interpretation of the bits that are set in status(2) is as follows:

Bit	Meaning
1	Always 0.
2	If set (=1), then paper is low.
3	If reset (=0), then printer/plotter is ready. Otherwise, printer/plotter is not ready.
4	If reset (=0), printer/plotter is online. Otherwise, printer/plotter is offline.
5-16	Always 0.

Discussion

Printer/Plotter Instructions: The instruction parameter supplies data affecting forms control and mode control (Print mode, Plot mode, Simultaneous Print/Plot mode), as follows:

0	Return printer/plotter status in status(2). The contents of the status vector, status, are described in the calling sequence description. T$VG waits until the output buffer is empty or until there is a timeout before returning status.
1	End-of-transmission. This instruction initiates a print cycle and a paper advance. If the paper on the printer/plotter is installed in roll form, this roll is advanced eight inches; if the paper is fanfolded, it is spaced to the top of the next form.
2	Reset. The reset instruction clears the buffer and initializes all logic in the printer/plotter.
3	Form feed. The form feed initiates a print cycle and a paper advance. If the paper on the printer/plotter is installed in roll form, the paper is advanced 2-1/2 inches; If the paper is fanfolded, it is advanced to the top of the next form.
4	Clear buffer.
5	Reserved.

6	Print the contents of buffer. (Print mode only -- see below.)
7	Make a plot, using the contents of buffer. (Plot mode only -- see below.)
8	Simultaneous print/plot PRINT. (SPP mode only -- see below.)
9	Simultaneous print/plot PLOT. (SPP mode only -- see below.)
10	Return status of output queue in status(2.) If there is no room for the number of halfwords specified by the parameter nhwds, set status(2) to 0. If there is room for the number of halfwords specified by nhwds, set status(2) to a nonzero value.

Print Mode: The Versatec printer/plotter may be operated as if it were a line printer. The printer/plotter accepts 6- or 8-bit ASCII code. Control commands are transmitted by using the instructions described for the calling sequence or by transmitting the following ASCII control codes:

ASCII Code (Octal)	Meaning
'004	End of transmission.
'014	Form feed.
'012	Line feed. The transmission of the (LF) code causes a print cycle and a paper advance of one line, except when the 012 code follows either the printing of a full buffer or a carriage return (015).
'015	Carriage return. A (CR) code causes a print cycle and a paper advance of one line, provided the buffer has at least one character entered and provided the buffer is not full.

When the 8-bit (128-character) ASCII character set is used, there are no ASCII control codes.

Plot Mode: The printer/plotter performs plot operations that are standard to all printer/plotter devices connected via the controller to the Prime computer. Plot data consists of 8-bit, binary, unweighted bytes. Each dot that is plotted at the printer/plotter corresponds to a single bit in the buffer. If bit is 1, a black dot is plotted at the

point corresponding to the bit position in the buffer. Bit 1 of a memory halfword (2 bytes) is the most significant (leftmost) bit, and bit 16 of memory halfword is the least significant (rightmost) bit.

Simultaneous Print/Plot (SPP) Mode: SPP mode operation permits direct overlay of character data which is generated by an internal matrix character generator, with plotting data, which is generated on a bit-to-dot correspondence. The SPP mode is an optional feature on some printer/plotters. The SPP process makes use of both a print buffer and a plot buffer, both specified in calls to T$VG. For example, using the Versatec Printer/Plotter Model 1100A in SPP mode, the SPP operation consists of first, placing up to 132 ASCII characters in the PRINT buffer (Instruction = 8); and then placing 128 bytes of plot data in the buffer (Instruction = 9) ten times. When the plot data is transmitted to the printer/plotter, the plot buffer is scanned, and a single row of dots, corresponding to the binary content of the plot buffer, is printed. During the scanning process, the print buffer is also scanned. The corresponding dots of each print character are OR'd with the plot buffer output; thus an overlay is formed consisting of the printed and plotted data. Since the vertical height of an ASCII character for the Model 1100A Printer/Plotter is ten raster scans, the user must make ten calls to plot data before the print buffer is completely printed and ready for new data. Table 7-2 shows the number of raster scans per print line for the various models of Versatec printer/plotter optionally available with Prime computer configurations.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

Caution

For SPP mode, do not attempt to transfer more than the maximum number of characters to the print buffer.

SPP mode requires a series of calls to the T$VG driver. For instance, in the example given, each print instruction was followed by ten plot instructions. Do not interrupt such a sequence with other instructions, because printer/plotter output will be incorrect.

Table 7-2
Maximum Buffer Length for Versatec Printer/Plotters

PLOT				PRINT No. Scans/Print Lines	
Model	Bits	Bytes	Chars.	64 Chars.	96 or 128 Chars.
220a	560	70	80 (70 in spp)	8	10
1100a	1024	128	132	10	12
1600a	1600	200	100	20	20
2000a	1856	232	232	10	12
2160a	2880	360	180	20	20

CARD PROCESSING SUBROUTINES

Card-reader subroutines drive and control serial and parallel interface card readers.

Card Reading Operation: The user must insert the card deck in the card reader and give the command:

```
ASSIGN  CRn
```

n =0 or 1 for the device sub-unit number

The user then fills the input buffer from the card reader by calling one of the following subroutines:

- I$AC03 or I$AC15 for parallel interface cardreaders
- I$AC09 for serial interface cardreaders
- T$CMPC or T$PMPC (from the operating system library)
  Normally a user does not call these directly, but instead calls an I$Axx subroutine, which itself calls a T$xxxx subroutine.

The user may issue a status request call to check if the input buffer is empty. If the buffer is empty, the online status bit (bit 9 in the status word) is reset.

These card-reading subroutines, as well as the card-writing subroutines and card-code-translator subroutines, are described on the following pages.

Note

Under PRIMOS II, the card reader is never offline.

I$AC03

Purpose

Reads ASCII input from the parallel interface card reader. Because this subroutine is in R-mode only, the calling program must be either FTN (as in the Usage description below) or PMA.

Usage

```
INTEGER   unit
INTEGER   buffer(1)
INTEGER   hwcnt
INTEGER   altrtn

CALL I$AC03 (unit, buffer(1), hwcnt, altrtn)
```

Parameters

unit

INPUT. The physical unit, or device, from which data is to be moved:

0	CR0, first controller
1	CR1, second controller

buffer(1)

OUTPUT. Name of data area to receive input from card reader.

hwcnt

INPUT. Number of halfwords to be transferred.

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error.

Discussion

Card Format: Cards are expected to be in 029 format. '026' cards may be read by preceding the deck by a card containing '$6' in columns 1 and 2. The conversion done for '026' cards is shown below.

Card Code (026 Symbol)	Converted to (Character)
#	=
%	(
<	)
@	'
`	+

The driver can be switched back to '029' format by '$9' in columns 1 and 2.

Load Information: This subroutine calls T$CMPC.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

I$AC09

Purpose

The subroutine I$AC09 reads ASCII input from a serial interface card reader.

Usage

```
      INTEGER*2 unit
      INTEGER*2 buffer(1)
      INTEGER*2 hwcnt
      INTEGER*2 altrtn

CALL I$AC09 (unit, buffer, hwcnt, altrtn)
```

Parameters

unit

INPUT. The physical unit, or device, from which data is to be moved:

0	CR0, first controller
1	CR1, second controller

buffer(1)

OUTPUT. Name of data area to receive input from card reader.

halfword-count

INPUT. Number of halfwords to be transferred.

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error.

Discussion

I$AC09 translates card codes to characters in memory as follows:

Card Code (026 Symbol)	Converted to (Character)
#	=
%	(
<	)
+	`
`	+
@	'

Card codes read are either 026 or 029. The last card in the deck is .Q..

Errors: The ERRVEC(3) may have the following octal values. (See Appendix B for a discussion of ERRVEC.) Combinations are possible.

'200	Online
'40	Illegal ASCII
'20	DMX overrun
'4	Hopper empty
'2	Motion check
'1	Read check

Load Information: I$AC09 calls F$AT to fetch the arguments.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

I$AC15

Purpose

Reads and interprets (prints) a card from a parallel interface card reader. Because this subroutine is R-mode only, the calling program must be either FTN (as in the Usage description below) or PMA.

Usage

```
INTEGER*2 unit
INTEGER*2 buffer(1)
INTEGER*2 hwcnt
INTEGER*2 altrtn

CALL I$AC15 (unit, buffer(1), hwcnt, altrtn)
```

Parameters

unit

INPUT. The physical unit, or device, from which data is to be moved:

0	CR0, first controller
1	CR1, second controller

buffer(1)

OUTPUT. Name of data area to receive data from card reader.

hwcnt

INPUT. Number of halfwords to be transferred.

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error.

Discussion

Load Information: This subroutine calls T$PMPC.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

T$CMPC

Purpose

The T$CMPC routine is the raw data mover that transfers information on a card from the MPC card reader to the user's space. T$CMPC is called by the IOCS card-reader driver I$AC03. The user normally reads cards under program control using either FORTRAN READ statements or a call to I$AC03. However, it is possible to call T$CMPC directly.

Usage

```
DCL T$CMPC ENTRY(FIXED BIN(15), PTR, FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(31);

CALL T$CMPC(physical_unit, addr(buffer), count, instr,
            status);
```

Parameters

physical_unit

INPUT. Device from which data is to be moved:

0	CR0, first controller
1	CR1, second controller

addr(buffer)

INPUT. A pointer to a buffer that is to hold the information from a card being read in the card reader.

count

INPUT. The number of halfwords to be read from the current card.

instr

INPUT. An octal-code instruction needed by the card reader. Valid instructions are:

Instruction	Meaning
100000 (octal)	Return status.
40000 (octal)	Read card in ASCII format.
60000 (octal)	Read card in binary format.
100001 (octal)	Return status of hardware.

status

OUTPUT. A three-halfword vector:

status(1) Not used.

status(2) Card-reader status: If status is explicitly requested by instr ('100000), this halfword returns a value indicating the state of buffer (not of the hardware). Otherwise the status bits returned are defined as follows:

Octal Value	Condition
200	Online
40	Illegal ASCII
20	DMX overrun
4	Hopper empty
2	Motion check
1	Read check

status(3) Number of halfwords moved.

Example: The following FORTRAN example reads an 80-character card of ASCII data and places the contents in CARDS.

```
   40 DO 70 I = 1, 23
   50 CALL T$CMPC (0, LOC(CARDS), 40, :40000, STATUS)
C    Now, save "CARDS" contents,
C    either by printing as line records (O$ALxx, etc.)
C    or by backing up the card deck (O$ACxx, etc.):
   60 CALL O$xxxx./* ...But why not save on disk or tape?
   70 CONTINUE
```

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

O$AC03

Purpose

O$AC03 punches output to the parallel interface card punch. Because this subroutine is in R-mode only, the calling program must be either FTN (as in the Usage description below) or PMA.

Usage

```
    INTEGER*2 unit
    INTEGER*2 buffer(1)
    INTEGER*2 hwcnt

CALL O$AC03 (unit, buffer(1), hwcnt)
```

Parameters

unit

INPUT. Card punch sub-unit number:

0	CR0, first controller
1	CR1, second controller

buffer(1)

INPUT. Name of data area supplying output to be punched.

hwcnt

INPUT. Number of halfwords to be punched.

Discussion

Load Information: This subroutine calls T$PMPC.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

O$AC15

Purpose

Punches output to the parallel interface card punch and prints on card. Because this subroutine is in R-mode only, the calling program must be either FTN (as in the Usage description below) or in PMA.

Usage

```
INTEGER*2 unit
INTEGER*2 buffer(1)
INTEGER*2 hwcnt
INTEGER*2 altrtn

CALL O$AC15 (unit, buffer(1), hwcnt, altrtn)
```

Parameters

unit

INPUT. Card punch sub-unit number:

0	CR0, first controller
1	CR1, second controller

buffer(1)

INPUT. Name of data area supplying output to be punched.

hwcnt

INPUT. Number of halfwords to be punched.

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error.

Discussion

Load Information: This subroutine calls T$PMPC.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

T$PMPC

Purpose

T$PMPC is the raw data mover for the card punch. It is called by O$AC03, O$AC15, and I$AC15, the card punch drivers. These routines may also be called by the user.

Usage

```
DCL T$PMPC ENTRY(FIXED BIN(15), PTR, FIXED BIN(15),
                 FIXED BIN(15), FIXED BIN(31));

CALL T$PMPC (physical_unit, addr(buffer), count,
             inst, status);
```

Parameters

physical_unit

INPUT. Device from which data is to be moved:

0	CR0, first controller
1	CR1, second controller

addr(buffer)

INPUT. A pointer to a buffer supplying the data output to the card reader. Data is packed two characters per halfword. In binary mode, card punches are mapped into a 16-bit halfword as follows:

Bit	Punch Row
1-4	Not used
5	12
6	11
7-16	0-9

count

INPUT. The number of halfwords to punch on a card from buffer.

instr

INPUT. An instruction needed by the card punch. Instructions are:

Bit Set	Instruction	Meaning
1	'100000	Read status.
3	'20000	Process in binary mode.
4	'10000	Feed a card.
5	'4000	Read a card.
6	'2000	Punch a card.
7	'1000	Print a card.
8	'400	Stack a card.

To punch a card, instr would be an octal 12400 meaning:

1. Feed a card.
2. Punch a card.
3. Stack a card.

status

OUTPUT. Three halfword status vector:

status(1) Not used.

status(2) Device status returned for a read request (instr = '4000):

Value	Condition
'200	Online
'4	Illegal code
'10	Hardware error
'4	Operator intervention required

status(3) Number of halfwords read.

MAGNETIC TAPES

The magnetic tape subroutines drive and control 7-track and 9-track magnetic tape devices. Their functions are summarized in Table 7-3.

Table 7-3
Functions of Magnetic Tape Subroutines

9-Track	
C$M05	Control for 9-track ASCII and binary.
C$M13	Control for 9-track EBCDIC.
O$AM05	Write ASCII.
I$AM05	Read ASCII.
O$BM05	Write binary.
I$BM05	Read binary.
O$AM13	Write EBCDIC.
I$AM13	Read EBCDIC.
7-Track	
C$M10	Control for 7-track ASCII and binary.
C$M11	Control for 7-track BCD.
O$AM10	Write ASCII.
I$AM10	Read ASCII.
O$BM10	Write binary.
I$BM10	Read binary.
O$AM11	Write BCD.
I$AM11	Read BCD.

Note

For descriptions of the subroutines listed in Table 7-3, see Appendix E "Other Obsolete Subroutines". The subroutine T$MT has replaced all of these subroutines except O$AM13 and I$AM13.

T$MT

Purpose

The T$MT routine is the raw data mover that moves information from magnetic tape to user address space, or from the user space to tape. T$MT also performs other tape operations, such as backspacing, forward spacing, and density setting. If T$MT is called without the code argument, and an error condition is encountered, T$MT exits to the user command level, rather than to the calling program. If T$MT is called with the code argument, T$MT returns the appropriate error code to the calling program.

T$MT is used by several tape controllers. Table 7-4 gives version numbers and controller IDs for the different drive types. Instructions associated with particular versions are indicated in the Usage section below.

Table 7-4
Controller Id

Version	Device ID	Controller #	Drive Type
0	'014	2081	Pertec
1	'114	2081	Kennedy, separate formatter
2	'214	2269/2270	Kennedy, two-board controller
3	'314	2023	Telex(1600/6250 bpi)
4	'013	2301	Cipher Streamer (1600/3200 bpi)
5	'113	2047 or 6105	DEI Cartridge Drive for 2250
6	'213	2382	60-Megabyte (QIC-02) cartridge tape controller
-	'313	9610	Model 4587 quad-density tape drive

Note

The Version 6 cartridge tape controller performs a limited sub-set of the normal tape drive functions for T$MT described below, as it is designed to provide inexpensive system backup and data storage. See further comments on Version 6 under instr and statv, especially for how T$MT handles invalid commands to Version 6.

Usage

```
DCL T$MT ENTRY(FIXED BIN(15), PTR, FIXED BIN(15),
               FIXED BIN(15), 6 FIXED BIN(15)[, FIXED BIN(15)]);

CALL T$MT (physical_unit, buffno, nhw, instr, statv [, code]);
```

Parameters

physical_unit

INPUT. A sub-unit for this physical device, the magnetic tape drive -- valid numbers are (logical drive numbers) 0 through 7.

buffno

INPUT. Location of the buffer from which to read or write a record of information. It must be an octal number. If neither a read nor a write operation, buffno is 0.

nhw

INPUT. Number of halfwords to transfer. This number must be between 0 and 6K halfwords.

instr

INPUT. The instruction request to the magnetic tape drivers. The following instructions are valid for all tape drivers, except where noted.

Note

If one of the following instructions not valid for a QIC-02 drive is still submitted to a T$MT call for that cartridge tape controller, then T$MT sets bit 6 in statv(2) -- for Uncorrectable Error -- and bit 13 in statv(4) -- for Illegal Command. T$MT then returns immediately without setting the returned error code. (This is also true for a "Read record backwards" instruction, the last instruction in the next list.)

Octal	Hexadecimal	Meaning of instr
000040	0020	Rewind to BOT, 7- or 9-track, or QIC-02.
022100	2440	Backspace one file mark, 9-track.

Octal	Hexadecimal	Meaning of instr
020100	2040	Backspace one file mark, 7-track. (Not valid for Model 4587.)
062100	6440	Backspace one record, 9-track.
060100	6040	Backspace one record, 7-track. (Not valid for Model 4587.)
022220	2490	Write file mark, 9-track, and QIC-02.
020220	2090	Write file mark, 7-track. (Not valid for Model 4587.)
062200	6480	Forward one record, 9-track, and QIC-02.
060200	6080	Forward one record, 7-track. (Not valid for Model 4587.)
022200	2480	Forward one file mark, 9-track, and QIC-02.
020200	2080	Forward one file mark, 7-track. (Not valid for Model 4587.)
100000	8000	Select transport, 7- or 9-track, or QIC-02, and get status.
042620	4590	Write record, two characters per halfword, 9-track, and QIC-02 (on QIC-02 the initial WRITE requires up to two minutes).
042220	4490	Write record, one character per halfword, 9-track.
042200	4480	Read record, one character per halfword, 9-track.
042600	4580	Read record, two characters per halfword, 9-track, and QIC-02.
052200	5480	Read and correct record, one character per halfword, 9-track. (Not valid for Model 4587.)
052600	5580	Read and correct record, two characters per halfword, 9-track. (Not valid for Model 4587.)

Octal	Hexadecimal	Meaning of instr
040220	4090	Write binary record, one character per halfword, 7-track. (Not valid for Model 4587.)
040620	4190	Write binary record, two characters per halfword, 7-track. (Not valid for Model 4587.)
044220	4890	Write BCD record, one character per halfword, 7-track. (Not valid for Model 4587.)
044620	4990	Write BCD record, two characters per halfword, 7-track. (Not valid for Model 4587.)
040200	4080	Read binary record, one character per halfword, 7-track. (Not valid for Model 4587.)
040600	4180	Read binary record, two characters per halfword, 7-track. (Not valid for Model 4587.)
044200	4880	Read BCD record, one character per halfword, 7-track. (Not valid for Model 4587.)
044600	4980	Read BCD record, two characters per halfword, 7-track. (Not valid for Model 4587.)
140000	C000	Return controller id.

Note

The following instructions are valid only with certain versions of tape controllers, as noted. If an instruction is submitted to a controller that does not support it, T$MT returns an error code of E$IVCM (invalid command) in code.

Octal	Hexadecimal	Meaning of instr
004020	0810	Erase from current position to EOT on tape. (Model 4587 controller)
100020	8010	Erase a 3.5 inch gap on the tape (version 2, 3, 4, or Model 4587 controller).

Octal	Hexadecimal	Meaning of instr
100040	8020	Unload. Rewind tape and place drive offline (version 2, 3, 4 or Model 4587 controller).
100060	8030	Set density to 800 bpi (version 2, 3, 4, or Model 4587 controller).
100100	8040	Set density to 1600 bpi (version 2, 3, 4, or Model 4587 controller).
100120	8050	Set density to 6250 bpi (version 2, 3, 4, or Model 4587 controller).
100140	8060	Set density to 3200 bpi (version 2, 3, 4, or Model 4587 controller).
100160	8070	Set low speed (version 4 or Model 4587 controller).
100200	8080	Set high speed (version 4 or Model 4587 controller).
100220	8090	Re-tension tape (version 5 or 6 controller)
100240	80A0	Erase entire tape (version 6 controller)
100260	80B0	Go to end of data (version 6 controller)
043500	4740	Read record backwards (version 3 controller). As already noted, Version 6 treats this as an invalid instruction, handled like those from the previous list.
100300	80C0	Set drive default density (version Model 4587 controller)
100360	80F0	Select buffered mode. (Model 4587 controller)
100320	80D0	Erase to EOT. (Model 4587 controller)
100340	80E0	Select unbuffered mode (Model 4587 controller).

Note

In buffered mode, records can be lost when a write operation is terminated abnormally. When the write operation is terminated abnormally, an unknown number of data records may remain in the Model 4587 tape drive buffer, and not be written to tape. This is because the Model 4587 tape drive acknowledges receipt of a record when it has been read into the buffer, even though it has not yet written to tape. This problem does not occur in nonbuffered mode, because in nonbuffered mode the tape drive acknowledges each record only after it is written to tape.

Any of the following conditions can cause a write operation to terminate abnormally in buffered mode:

- A warm start (bit 6 in statv(2) is set). After a warm start during buffered mode, you should backspace to the last file mark, or rewind.

- A tape drive fault (bit 14 in statv(4) is set). After a tape drive fault, subsequent write commands are not rejected, but the same fault will be returned unless the fault condition is corrected.

- An "unrecoverable write error" (bit 16 in statv(4) is set). After an unrecoverable write error, all subsequent write commands are rejected. When a write fails on an unrecoverable error, the error is reported in the status information for the next operation to be completed after the failed write operation. The error will be reported either in a subsequent series of write operations, or in the first non-write operation after the controller detects the write failure.

  After each write operation in buffer mode, the application should check bit 16 of statv(4) to see if an unrecoverable write error occurred. If this bit is not set, there has been no error on previous write operations; the current write operation, however, may not yet have been performed. If bit bit 16 of statv(4) is set, an unrecoverable write error has occurred during an earlier write operation. In this case, the tape may be physically damaged, and the user should restart the application on another tape.

User applications are responsible for ensuring that no record loss occurs because of mode selection. To avoid loss of records, do not run any application without knowing which mode is currently selected. Note that the Model 4587 tape drive defaults to non-buffered mode after system boot. It is good practice for an application always to select the mode that it requires before it runs, and to reselect non-buffered mode when it completes execution. See Using Your Quad Density Tape Drive for more information about how to program a Model 4587 tape drive.

statv

OUTPUT. 6-halfword status vector. If this is the last argument, then only the first three halfwords are set. If the code argument follows, then additional halfwords may be set, depending on the controller being used. Each of the statv halfwords are described below:

statv(1) Status flag:

Value	Meaning
1	Operation in progress
0	Operation finished

statv(2) Hardware status word from controller. Possible values are:

Bits	Meaning When Set
01	Vertical parity (read) error
02	Runaway tape error.
03	CRC error (Always 0 for Version 6 or Model 4587)
04	LRC error (Always 0 for Version 6 or Model 4587)
05	False gap or More Record Data than was requested during a read operation
06	Uncorrectable error. For Model 4587, may indicate that selected drive is not connected. May also indicate condition signalled by bits 1, 2, 7, 14, or 15 in statv(2), or by bits 5, 12, 13, 14, or 16 in statv(4).
07	Read-after-write error detected during 7 retries (for Version 6: Read-after-write error).
08	File mark detected.
09	Transport is ready.
10	Transport is online.
11	End of tape was detected.
12	Selected transport is re-winding.
13	Selected transport is at load point (beginning of tape).

Bits	Meaning When Set
14	Transport is write-protected (file-protected); write commands are rejected.
15	DMX overrun or no formatter. For Model 4587, may indicate a FIFO parity error.
16	Indicates that rewind is complete. For Versions 2 and 6 controllers, this bit has no meaning and is always 0. For Model 4587, set while drive is erasing to EOT.

statv(3) Number of halfwords transferred (read and write operations only).

statv(4) Hardware status for version 1, 2, and 3 controllers. Bits 0 and 1 specify density of tape:

00	800 bpi
10	1600 bpi
11	6250 bpi

Other bits in statv(4) used by all controllers:

Bits	Meaning When Set
05	Illegal operation to tape drive attempted (write to a file protected drive).
13	Illegal PIO command issued (OTA not defined as meant for tape controller.)

statv(4) For Version 6 controllers only:

Bits	Meaning When Set
01	(Always set) to indicate GCR density.
02	Always zero.
03	Not used. Always zero.
04	Not used. Always zero.
05	Illegal operation to tape drive attempted (write to a file protected drive).
06	No cartridge present.
07	This is set when the controller has performed an error correction operation on a read or write operation.
08	Read error, bad block transfer.

09 Read error, filler block transfer.

10 Read or write aborted due to unrecoverable error.

11 Always zero.

12 Always zero.

13 Illegal PIO command issued (OTA not defined as meant for tape controller.)

14 Device fault or device reset.

15 Verify failure upon master clear or power up of controller.

16 Record not completely written or incomplete record read. Always zero. (EOT).

statv(4) For Model 4587 only:

Bits	Meaning When Set
1-3	Indicate density setting. 100 : PE 101 : GCR 110 : NRZI 111 : DPE 011 : Default to drive panel control.
4	100 IPS selected. (When not set, 50 IPS selected.)
5	Tape operation not legal for Model 4587.
6	Currently in buffered mode.
7	Error Correction/Retries necessary on last Read or Write.
8-11	Unused (always 0).
12	Fatal memory error, or SCSI hung. OCP '17 needed to clear error.
13	OTA not defined for Model 4587 was received and rejected.
14	Tape drive fault or tape drive reset.
15	DRAM degraded mode.

16 Write operation aborted by uncorrectable write error. No write commands accepted until command requiring a reverse tape movement is received and executed.

statv(5-6) Reserved.

code

OUTPUT. Specifies that the appropriate error code is to be returned to the calling program. code requires statv to be a six-halfword array.

The possible error codes returned are:

E$NASS Device specified in physical-unit, not assigned.

E$IVCM Invalid command (e.g. attempt to set density on version 0 controller).

E$DNCT Device specified in physical-unit not connected, or no controller.

E$BNWD Invalid number of halfwords (nhw <=0 or >6144).

Discussion

Magnetic tape I/O is not buffered under PRIMOS. A call to T$MT returns immediately before the operation is complete. When the magnetic tape operation is completed, the status flag in the user space is set to 0. Therefore, a user program may do another computation while waiting. If a user initiates another call to T$MT before the first call has completed its magnetic tape operation, the second call does not return to the user until the first magnetic tape operation has been completed.

Density Selection

It is assumed that tapes are written with one density. For versions 0 through 2 controllers and drives, the user should first set this density with the drive control panel switches. Version 3-4 controllers automatically adjust to the correct tape density. Version 5 controllers have a set density. If density is not set automatically, the user must manually set the drive to the right density before the first record is read. The rest of the tape will be read (or written) using that density. The Model 4587 returns an error if the drive is not at Load Point (BOT) when the user attempts to change the density.

For example, if the user set the density to 6250 bpi with the ASSIGN command and read the first record of a 1600 bpi tape, then the rest of the tape would be read using 1600 bpi. If after reading that record, a record was written onto the tape (without rewinding to the load point), then that record would also be written at 1600 bpi. If the tape was rewound and then a record was written, the density would be switched to 6250 bpi. Although the density setting of 6250 bpi is remembered, it will not go into effect until a record is written at the load point.

If the user assigns a tape without specifying a density, the unit will be left at the density from the previous use. The default density (at system initialization time) is 1600 bpi.

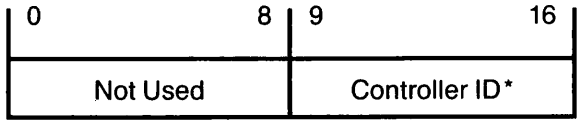
Read Record Backwards

This request causes the tape to read a record while moving the tape backwards. It is sometimes possible to read a record backwards when a bad tape prevents reading the record in the forward direction. After the record is read, it will be necessary to reorganize the data. The halfwords of the record will be in reverse order. Each halfword will have the bytes reversed. The bits within each byte will be in correct order.

Instruction to Get Controller Id

The controller id may be used by software that intends to support all tape drives, but takes advantage of special features that are available only with a particular controller. For example, the ERASE command is only available with version 2 and 3 controllers.

Figure 7-1 shows how buf(1) must be set up for this instruction ('140000)



*ID from Table 7-4

BUFF(2) When instr is '140000
Figure 7-1

Use of the T$MT Wait Semaphore

While waiting for an operation to complete (that is, for status-word 1 to go to 0), a process can do one of several things. It can loop while checking the status-done word, do another operation (such as get status), or use a wait semaphore.

Looping on a wait for statv(1) to go from 1 to 0 uses up CPU time while the process waits for the tape operation to complete. This is not a good practice for two reasons. First, it ties up the CPU needlessly and slows down system performance in general. Second, it causes the process to waste some of its time slice without doing useful work. This will result in the process being scheduled extra time and the real time of program execution will be longer than necessary.

This problem can be solved by using a semaphore. If the process waits on a semaphore, the wait time is not counted against its time slice. Therefore, as soon as the tape operation completes, the process will be scheduled to run again to finish up its time slice.

The program T$MT contains a wait semaphore that can be used for this purpose. This semaphore is used to queue tape requests. If the process makes a tape request when the controller is busy with another operation, the process is put on the wait semaphore.

When the program wants to wait for a tape operation to complete, it can call T$MT with a request for status. Since the tape controller is already busy with the previous operation, the process will be put on the T$MT wait semaphore.

Since the status request is fast and doesn't affect the tape, it is a convenient tape operation to use to provide the semaphore wait. A scratch status vector should be used so that the status from the original call is not destroyed.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

Example: A FORTRAN Example of wait code is given here, preceded by the proper data definitions for variables:

```
          . . .

          INTEGER CODE, CODE2        /* RETURN CODES
          INTEGER STATV(6)           /* STATUS VECTOR SET BY T$MT
          INTEGER UNIT               /* MAG TAPE DRIVE NUMBER (0-7)
          INTEGER BUF (1024)         /* OUTPUT BUFFER
          INTEGER XSTATV (6)         /* SCRATCH VECTOR FOR WAIT

          . . .
          CALL T$MT (UNIT, LOC(BUF), ,:042620,STATV, CODE)
                                     /*WRITE 1024

          . . .                      /* OVERLAP EXECUTION WITH IO

    C     NOW WAIT FOR TAPE WRITE TO COMPLETE:

    100   IF (STATV(1).EQ.0) GOTO 120 /* SEE IF IO IS ALREADY DONE
             CALL T$MT (UNIT,LOC(0),0,:100000,XSTATV, CODE2) /* WAIT
             GOTO 100
    120   . . .
```

Error Recovery on Writing

The two error recovery schemes described here are based on different record formats. The first algorithm can be used when records contain only data. The other scheme requires that the records contain extra information for error recovery.

Notes

A user cannot generate an error recovery procedure for the QIC-02 tape drive while that drive is in use. Either the drive itself performs such or an uncorrectable error has occurred.

The Model 4587 drive performs its own error recovery on both reads and writes.

The following schemes are provided as alternatives to using the IOCS routines that FORTRAN uses. The error recovery provided in the IOCS routines correspond to that described for Simple Write Error Recovery.

Simple Write Error Recovery: The aim of the simple error recovery program is to get by a possible bad spot on the tape by erasing part of the tape where the error occurred and rewriting the record after that gap.

The program does not try to rewrite the record on the same spot on the tape even though repeated tries on the same spot may improve the tape enough to permit the write to succeed. The tape is considered marginal at that spot and may not be readable at a later date.

Only two controllers will erase an entire tape. The version 3 controller (MPC-3), which supports the 6250 bpi tape drives, has an ERASE command. The QIC-02 cartridge tape drive also accepts an "Erase entire tape" instruction ('100240 or (HEX)80A0).

The version 2, 3, and 4 controllers have an "Erase 3.5 inch gap" ('100020 or (HEX)8010) to erase over a badspot on a tape, using the following technique.

Program steps for write error recovery:

1. Check if error recovery is possible. Don't attempt error recovery if the tape drive is offline or not ready, or the tape is file-protected.

2. Erase a 3.5 inch gap on the tape:

   - If a version 2, 3, or 4 controller, perform "erase 3.5 inch gap" operation

- Otherwise perform a "write file mark" operation, followed by perform a "backspace record" operation, followed by a check that the filemark detect bit is set in the status word.

3. Attempt to rewrite the record.

4. If the record was not written successfully, repeat steps 2 and 3 up to twenty times. Note that step 2 is done once on the first repetition, twice on the second repetition, thrice on the third, and so on, up to twenty retrys (a maximum of five feet of erased tape).

Write Error Recovery with Sequence Numbers: There is a drawback to the first scheme. Since the tape is bad at the spot where the error recovery is being done, it is possible for errors to occur while backspacing. For example, if the bad record has a gap in the middle of it, the program might detect two short records when backspacing. If the program has some way of identifying records, the program can be sure that it has not lost position during error recovery.

One way to do this is to include a sequence number with every record. Then when error recovery is attempted, the program backspaces two records and then reads a record. This record should contain the sequence number of the last good record before the error record.

Program steps for error recovery:

1. Check if error recovery is possible. Don't attempt error recovery if the tape drive is offline or not ready, or the tape is file-protected.

2. Position the tape after the last good record.

   - Backspace two records. This will place the tape before the last good record.

   - Read a record and verify that its sequence number matches the one expected for the last good record.

   - If the 'good' record can't be read, then it is possible that the tape is not positioned correctly. Backspace several records and read those records to find the sequence number of the last good record written.

3. Erase a 3.5 inch gap on the tape.

   - If a version 2, 3, or 4 controller, perform "erase 3.5 inch gap" operation

- Otherwise perform a "write file mark" operation, followed by perform a "backspace record" operation, followed by a check that the filemark detect bit is set in the status word.

4. Attempt to write the record again.

5. If the record was not written successfully, repeat steps 1-4 up to twenty times, lengthening the gap each time.

Error Recovery on Reading

Error recovery when reading a tape involves repeatedly rereading the record. The problem of losing position can occur when doing error recovery. Therefore, the procedure can be improved by verifying the sequence number each time a record is read. Note that the Model 4587 drive performs its own error recovery on both reads and writes.

Program steps for read error recovery:

1. Check that error recovery is possible. Don't attempt error recovery if the tape drive is offline or not ready.

2. Backspace and reread the record eight times.

3. If unsuccessful, backspace eight records (or to the load point if less than eight records away), space forward seven records and then read the problem record. This sequence draws the tape over the tape cleaner and could dislodge a possible dirt particle.

4. Repeat steps 1-3 eight times.

PART III

SMLC/AMLC SUBROUTINES

8
Synchronous and Asynchronous Controllers

Part III of this Volume describes the subroutines and the control block configurations used with Synchronous and Asynchronous Controllers.

Chapter 8, containing all this information, first describes T$SLC0, the R-mode subroutine used for Synchronous Multi-Line Controllers. Thirteen Tables follow the description of T$SLC0, each giving control block bit configurations dependent on the key option used within T$SLC0. The last two subroutines described in this chapter, ASNLN$ and T$AMLC, are used for Asynchronous Multi-Line Controllers.

Routine	Function
T$SLC0	Communicate with SMLC driver.
ASNLN$	Assign AMLC line.
T$AMLC	Communicate with AMLC driver.
AS$LST	List specified asynchronous line characteristics.
AS$LIN	Return asynchronous line number.
AS$SET	Set asynchronous line characteristics.
NT$LTS	Returns information about a PRIMOS line used for LAN terminal service.

SYNCHRONOUS CONTROLLERS

This section defines the raw data mover for the assigned SMLC line. See the System Administrator's Guide for a discussion of SMLC lines.

There can be a maximum of two synchronous controllers configured: two MDLCs, two ICS1s, or one of each. An MDLC supports up to four synchronous lines; an ICS1 supports one synchronous line. The ICS1 also provides asynchronous support on the same board. The number of synchronous lines available depends on the controller configuration in use. Present possible configurations are given below.

Controller 1	Controller 2	Max Synchronous Lines
MDLC(4)	MDLC(4)	8
MDLC(4)	MDLC(2)	6
MDLC(4)	ICS1	5
MDLC(4)	--	4
MDLC(2)	ICS1	3
MDLC(2)	--	2
ICS1	ICS1	2
ICS1	--	1

Note: MDLC(4) is a four-line MDLC; MDLC(2) is a two-line MDLC.

Caution

R-mode subroutines can be called from FTN and PMA in R-mode only. If you call an R-mode routine from a program in a different mode, the results are unpredictable. Refer to the FORTRAN and PMA chapters in Volume I for information on declaring parameters in FTN and PMA, respectively.

T$SLC0

Purpose

The SMLC driver is loaded in PRIMOS. A user program communicates with the driver via FORTRAN-format calls to T$SLC0. The driver communicates with the user address space via buffers in the user address space specified by the user program. The data structure used by the driver is a control block created by the user in the user address space. It contains pointers to the user status buffer and to buffers containing a message to be transmitted or set to receive a message. A separate control block is required for each line.

Usage

```
INTEGER*2 key, line, block(1), nhwds

CALL T$SLC0 (key, line, LOC(block), nhwds)
```

Parameters

key

INPUT. Indicates the desired operation and may hold the following values:

1 Stop line. Only key + line required for the subroutine to execute on this key.

2 Define control block. The block is structured as in Table 8-1. It defines an area to store status information and, optionally, a message chain for reception or transmission.

3 Array block contains five halfwords which are to be output to the controller. See Tables 8-2 through 8-11 for details.

4 Array block contains a halfword which is to be used as the next data set control word. See Table 8-12 for details.

5 Array block contains two halfwords which are to be used as the next receive/transmit enable words. See Table 8-13 for details.

6 The calling user process will go to sleep. It will waken at the next SMLC interrupt or after approximately one second. It will run with a full time slice interval. The value line is ignored, as are addr(block) and nhwds. If, however, the user process does not own any SMLC lines, the call will return immediately.

7 Return model number. When using this key, nhwds must equal 1. Model number will be returned in block.

For MDLC model numbers, the value will be given in octal. The possible model numbers and their associated protocols are the following.

Model Number (Octal)	Protocols
0	HSSMLC
5646	BISYNC and HDLC
5647	BISYNC and PACKET
5650	BISYNC and 1004/UT200/7020
5651	HDLC and 1004/UT200/7020
5652	PACKET and 1004/UT200/7020
5653	HDLC and PACKET
5654	BISYNC and GRTS

ICS1 model numbers are specified in decimal. Listed below are their model numbers, the number of synchronous or asynchronous lines they support, and the Protocols they use.

Model Number	Lines Supported sync	async	Protocols
5181	1	8	BISYNC
5141	1	4	BISYNC

line

INPUT. Octal line number 0-7.

LOC(block)

INPUT. Pointer to address of user's block. User's block must reside entirely within one page.

nhwds

INPUT. Number of halfwords in block.

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

Discussion

Before calling T$SLC0 to configure a line (key = 3), a call with a key of 7 should be made to see if the controller contains the proper protocol and to determine what the line configuration should be. If an error occurs during initialization, the following error messages are printed:

No SMLCxx -(controller address)
No CONTROLLER CONFIGURED for SMLCyy (logical number)
UNDEFINED CONTROLLER ID for SMLCxx (controller address)

It is the responsibility of the caller to see that the line configuration is correct for the model of MDLC being used.

Timing: The user space program runs asynchronously with message transfers. A call to T$SLC0 returns immediately after executing whatever control function was required. The progress of the communication must be monitored by the user program by examination of the user space status buffer contents.

Assigning Communication Lines: The communications lines must be assigned to a user space before they can be used. The proper command is:

```
        SMLC00
        SMLC01
        SMLC02
        SMLC03
ASSIGN  SMLC04
        SMLC05
        SMLC06
        SMLC07
```

given at the user terminal. One or more lines may be assigned to a given user.

Table 8-1
Key = 2 SMLC Control Block

Halfword 0	Last receiver/transmitter enable word sent to the HSSMLC by the driver. (This halfword is written into but not read by the driver.)	
	Bit 15 = 1	Transmitter on
	Bit 16 = 1	Receiver on
Halfword 1	Bit 1	Valid line enable order in bits 2-16
	Bits 2-16	Line enable order. See Table 8-4, Halfword 0.
Halfword 2	Bits 1-4	Data set status mask (DSSM)
	Bits 5-8	Required data set status (RDSS)
	Bit 9	Set: No data set order - ignore Word 2
	Bits 13-16	Data set control order (DSCO)

Note

Issue DSCO, wait for
(DS status .AND. DSSM) = RDSS,
then issue line enable order.

Halfword 3	Spare
Halfword 4	Pointer to top of status buffer
Halfword 5	Pointer to bottom + 1 of status buffer
Halfword 6	Pointer to next halfword in status buffer to receive the status information. (This halfword is written into but not read by the driver.)

Note

The status buffer must be completely contained in the same page as the control block.

Table 8-1 (continued)
Key = 2 SMLC Control Block

Halfword 7	Bits 1-2	'01' there exists a continuation control block
	Bits 3-6	Halfword count of next block - 8
	Bit 7	0
	Bits 8-16	Offset in current 512 halfword page of next block

Note

The continuation block must reside in the same page as the control block from which it was continued.

Halfword 8	Bit 16:	
	1	Transmit
	0	Receive

Note

If Halfword 8 is given (nhwds > 8) then at least one DMC address pair must be given.

Halfwords 9-10	DMC start and end address pointers.
11-12	Up to four pairs may be specified to
13-14	allow for channel chaining.
15-16	

Note

Transmit/receive buffers may reside in any page, but their starting and ending address pointers must reside in the same page.

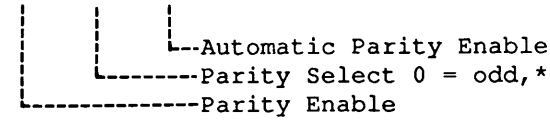
Table 8-2
Key = 3 Line Configuration Control Block (Bits 10-16)

Halfword 0 Bits 10 through 16 are constant for all controllers and protocols. Descriptions for Bits 1 through 9 follow descriptions of these constants.

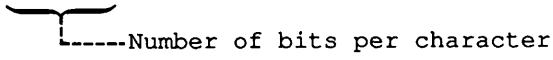
Bit 10 Enable formatter option (BISYNC, UT200, ICL 7020, 1004, PACKET SWITCH depending on HSSMLC options)

Bit 11 Enable reporting of data set changes by interrupt and status halfword.

Bits 12-14



Bits 15-16



*If automatic parity is enabled with 8-bit data enabled, no parity will be generated or checked (that is, no 9-bit data formats).

Automatic parity-enable appends a parity bit to the data while parity-enable steals the most significant bit of each data byte.

Table 8-3
Key = 3 Line Configuration Control Block (HSSMLC, bits 1-9).

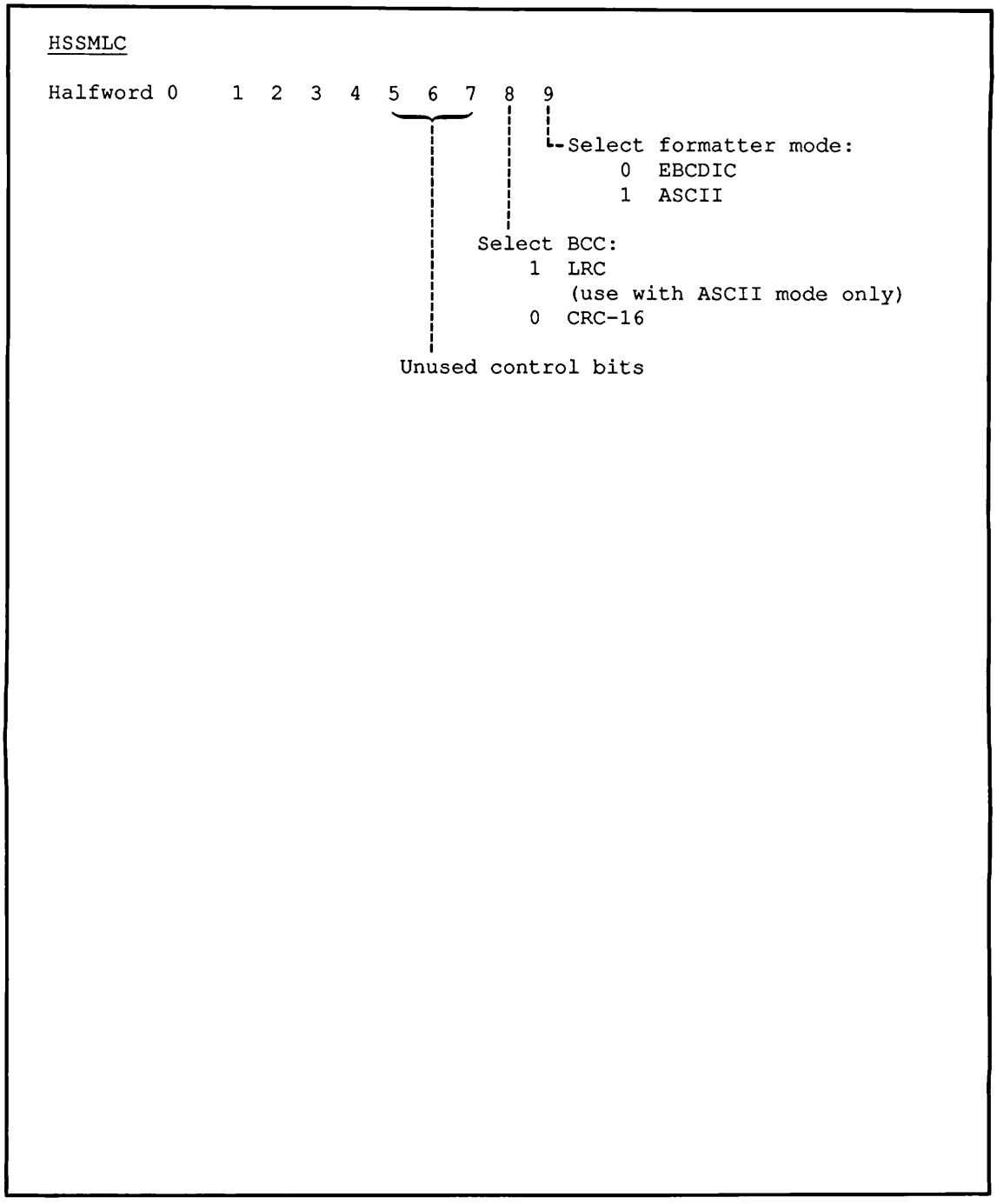
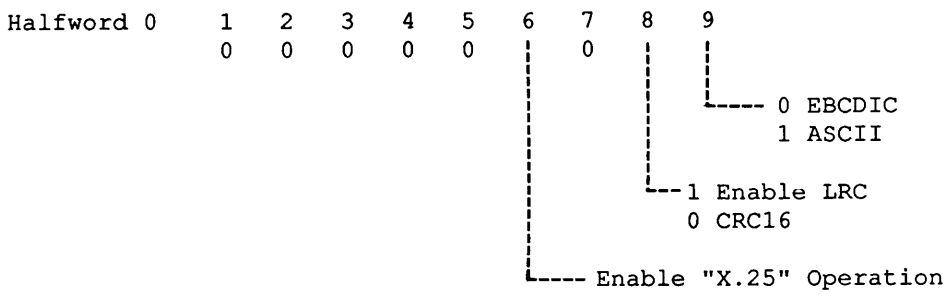
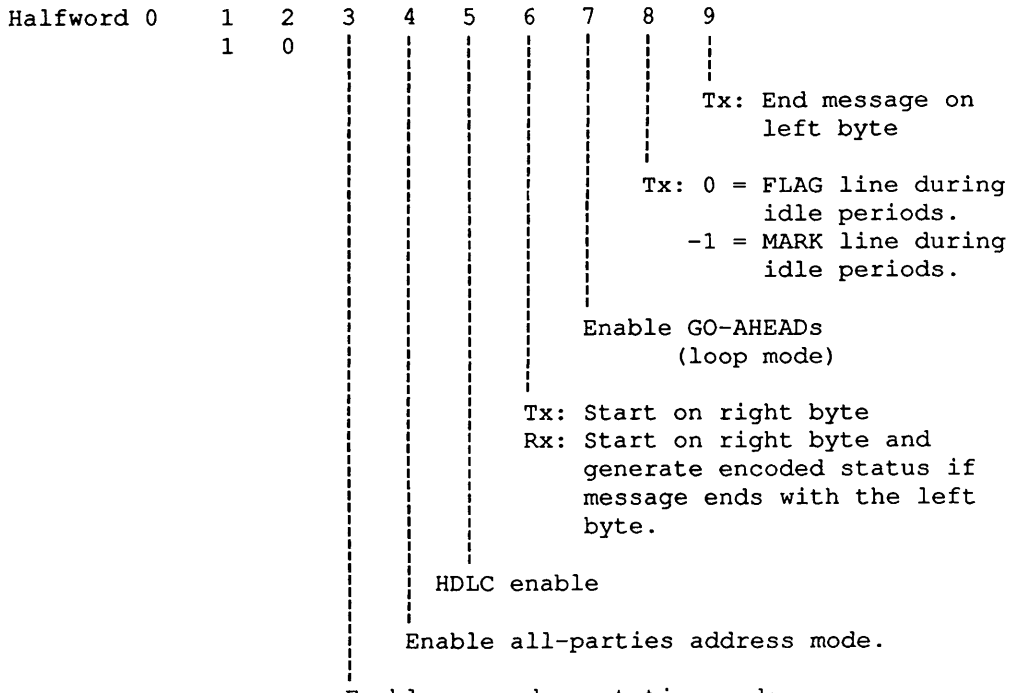


Table 8-4
Key = 3 Line Configuration Control Block (5646, Bits 1-9)

5646
BISYNC



HDLC



Secondary station mode, HDLC mode, Loop mode, and all-parties address mode are enabled on a line-pair basis only.

Table 8-5
Key = 3 Line Configuration Control Block (5647, Bits 1-9)

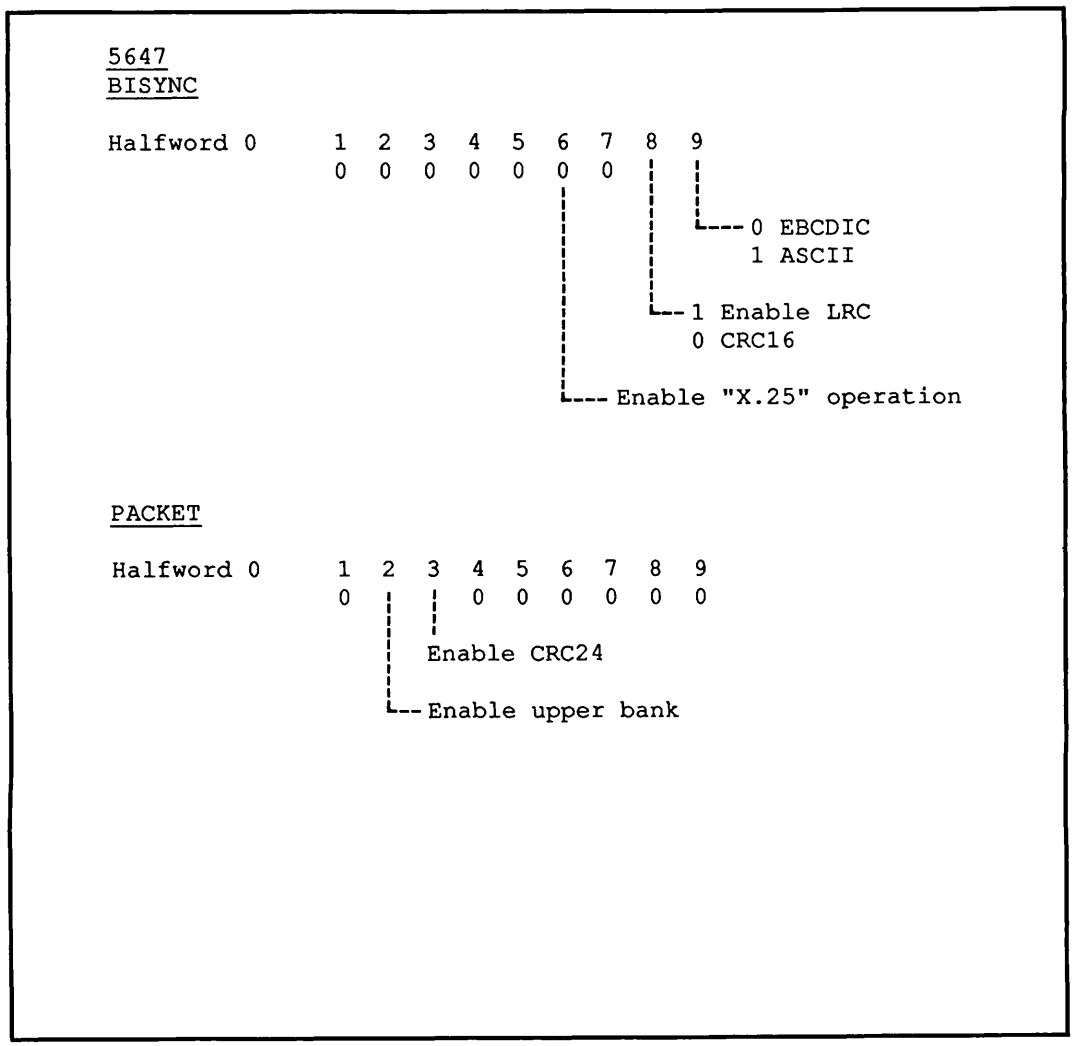
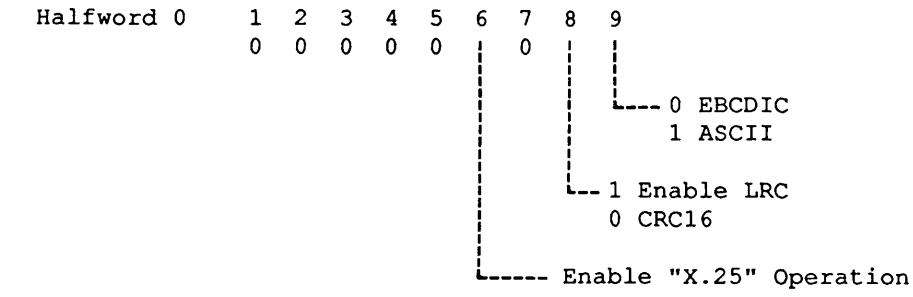
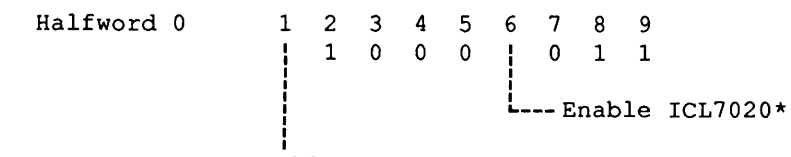


Table 8-6
Key = 3 Line Configuration Control Block (5650, Bits 1-9)

5650
BISYNC



ICL7020/UT200/1004



Recommended Configurations

1004	'140722	
UT200	'40723	(Add '40 to enable DSS interrupts)
ICL7020	'42723	

* Default protocol is UT200

Table 8-7
Key = 3 Line Configuration Control Block (5651, Bits 1-9)

```
5651
ICL7020/UT200/1004

Halfword 0     1  2  3  4  5  6  7  8  9
               |  0  0  0  0  |  0  1  1
               |              |
               |         Enable ICL7020*
               |
             Enable 1004*

             Recommended Configurations

             UNIVAC      '100722
             UT200          '723       (Add '40 to enable DSS interrupts)
             ICL7020       '2723

HDLC

Halfword 0     1   2   3   4   5   6   7   8   9
               |   1   0   |   |   |   |   |   |
               |           |   |   |   |   |   Tx: End message on
               |           |   |   |   |   |       left byte
               |           |   |   |   |   |
               |           |   |   |   |   Tx: 0 = FLAG line during
               |           |   |   |   |           idle periods.
               |           |   |   |   |      -1 = MARK line during
               |           |   |   |   |           idle periods.
               |           |   |   |   |
               |           |   |   |   L--Enable GO-AHEADs
               |           |   |   |           (loop mode)
               |           |   |   |
               |           |   |   L--- Tx: Start on right byte
               |           |   |        Rx: Start on right byte and
               |           |   |            generate encoded status
               |           |   |            if message ends with the
               |           |   |            left byte.
               |           |   |
               |           |   L--- HDLC enable
               |           |
               |           L----- Enable all-parties address mode
               |
               L---------------Enable secondary station mode
```

Secondary Station Mode, HDLC mode, Loop mode, and all-parties address mode are enabled on a line-pair basis <u>only</u>.

*Default protocol is UT200

Table 8-8
Key = 3 Line Configuration Control Block (5652, Bits 1-9)

5652
ICL7020/UT200/1004

```
Halfword 0     1   2   3   4   5   6   7   8   9
               |   0   0   0   0   |   0   1   1
               |                   |
               |                 Enable ICL7020
               |
               |--- Enable 1004 (UT200=Default)
```

Recommended Configurations

1004	'100722	
UT200	'723	(Add '40 to enable
ICL7020	'2723	DSS interrupts)

PACKET

```
Halfword 0     1   2   3   4   5   6   7   8   9
               0   |   |   0   0   0   0   0   0
                   |   |
                   |   |- Enable CRC24
                   |
                   |------ Enable upper bank
```

Table 8-9
Key = 3 Line Configuration Control Block (5653, Bits 1-9)

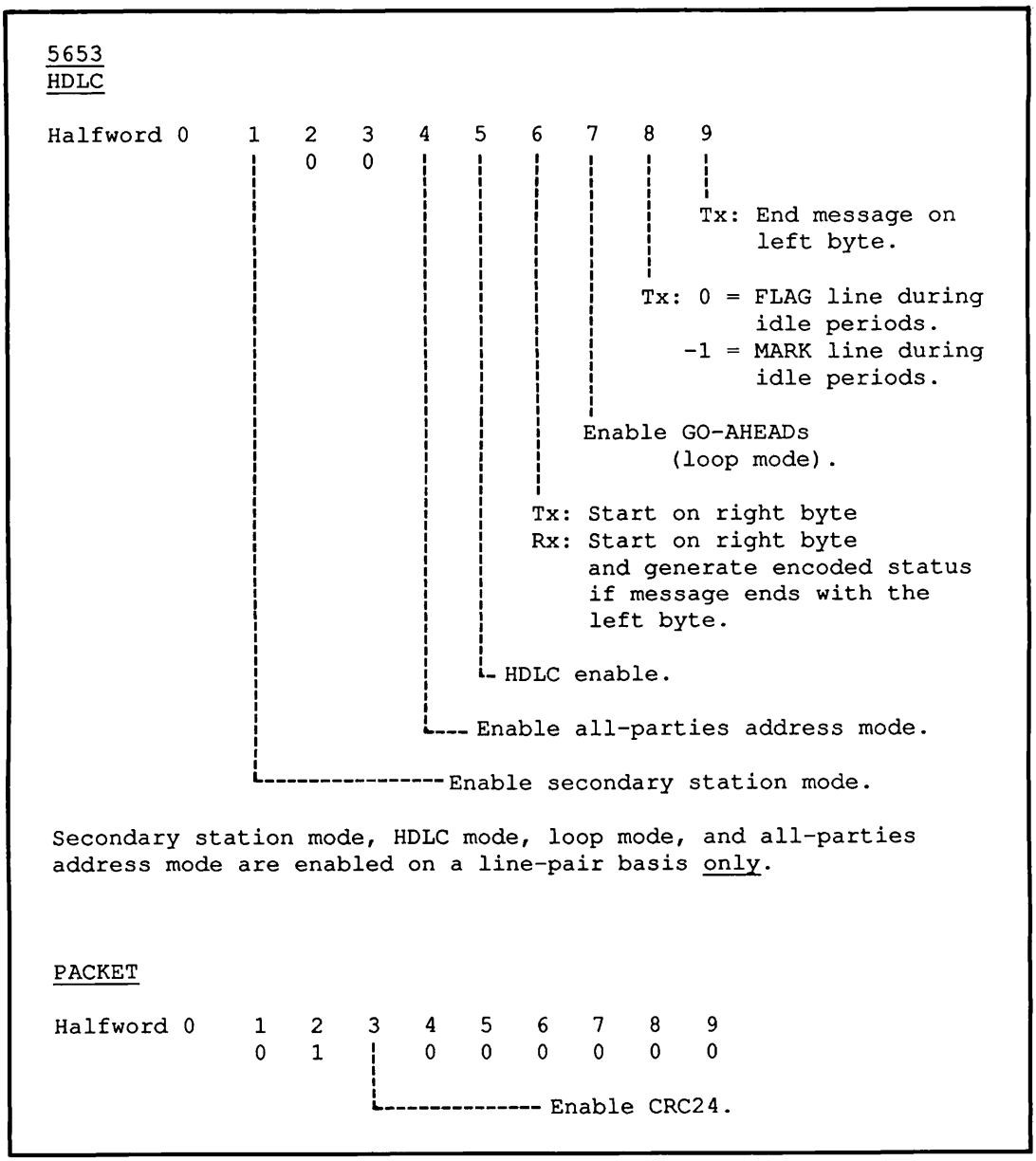


Table 8-10
Key = 3 Line Configuration Control Block (5654, Bits 1-9)

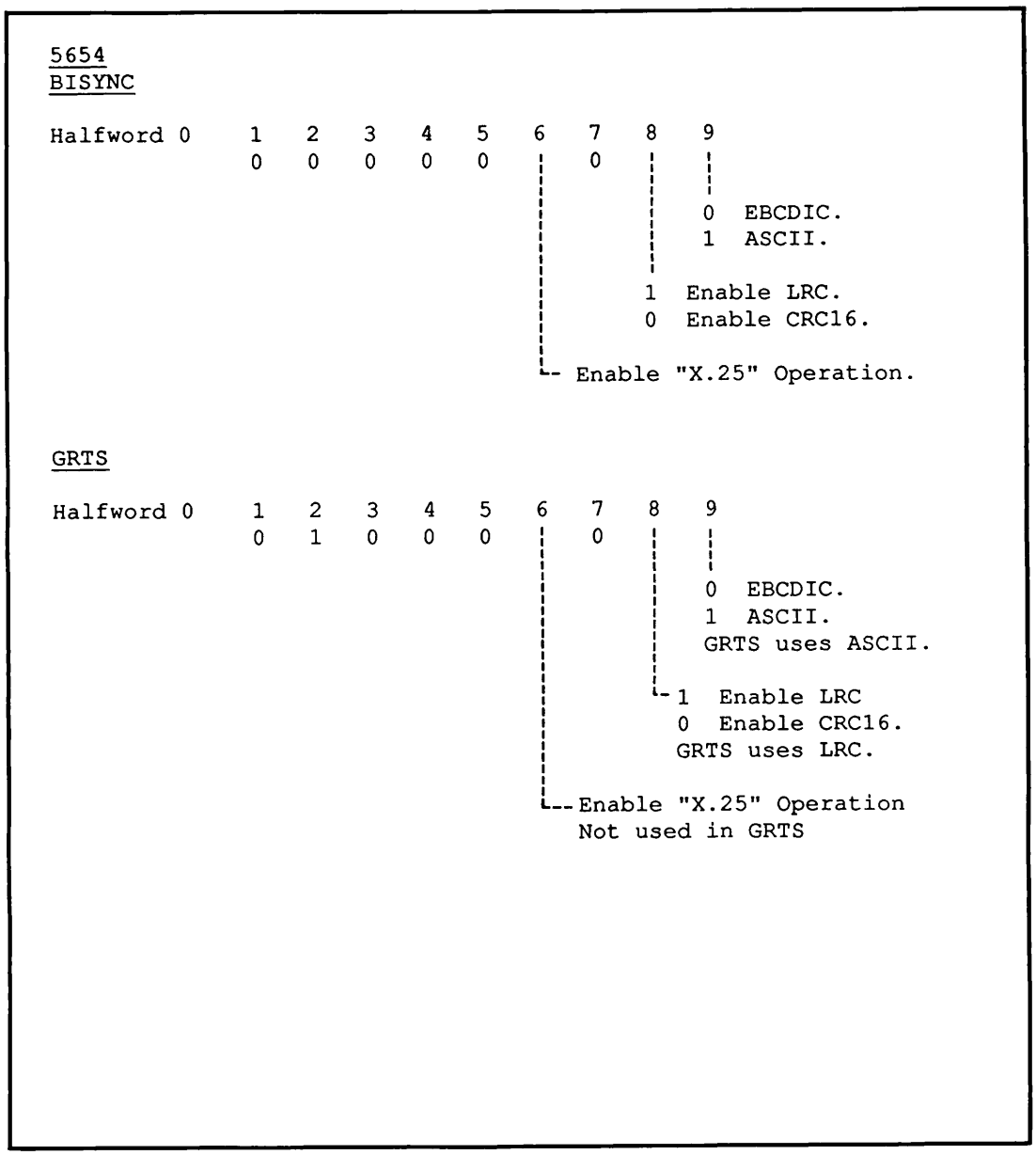


Table 8-11
Key = 3 Line Configuration Control Block (Halfwords 1-4)

Halfword 1	Word configuration - Transmitter bit settings as for Halfword 0.		
Halfword 2	Special character (OTA '00 : Function '10)		
	Bits 7-8	00	Character 1
		01	Character 2
		10	Character 3
		11	Character 4
	Bits 9-16	Character	
Halfword 3	Special character bit settings as for Halfword 2		
Halfword 4	Clock selection:		
	0	Reset internal clock to default 9.6 Kbps.	
	1	Switch internal clock to 62.5 Kbps.	

Table 8-12
Key = 4 Data Set Control Bits (OTA '00:Function '00)

Bit 13	Not used
Bit 14	Speed Select
Bit 15	Request to send (RTS)
Bit 16	Data Terminal Ready (DTR)

Table 8-13
Key=5 Receive/Transmit Enable (OTA '00:Function '15)

Halfword 0	Bit 11	Select internal as receive clock
	Bit 12	Select internal as transmit clock
	Bit 13-14:	
	00	Normal (transmit out, receive in)
	01	Loop full duplex (transmit out, receive in)
	10	Echo full duplex (receive in, transmit out)
	11	Loop half duplex (pair combinations must be: 1-2, 2-1, 3-4, 4-3)
	Bit 15:	
	1	Enable transmitter
	0	Disable transmitter
	Bit 16:	
	1	Enable receiver
	0	Disable receiver
Halfword 1	Bit 16:	
	1	Enable transmitter
	0	Enable receiver

Note

Transmitter and receiver must be enabled/disabled separately.

ASYNCHRONOUS CONTROLLERS

Applications that require the use of asynchronous controllers are serviced by two subroutines: ASNLN$ for line assignment requests and T$AMLC for raw data movement. The description of these subroutines follow.

ASNLN$

Purpose

ASNLN$ (Assign AMLC line) allows user programs to request the assignment of a line directly.

Usage:

```
DCL ASNLN$ ENTRY(FIXED BIN(15), FIXED BIN(15), CHAR(6),
                 FIXED BIN(15), FIXED BIN(15), FIXED BIN(15));

CALL ASNLN$ (key, line, protocol, config, lword, code);
```

Parameters

key

INPUT. Indicates the desired assignment option and may be one of the following:

0	Unassign AMLC line.
1	Assign AMLC line.
2	Unassign all AMLC lines owned by caller.

line

INPUT. Indicates the desired line number to be addressed for the keyed operation.

protocol

INPUT/OUTPUT. Indicates the desired protocol (input or output). Blanks cause a default to TRAN (transparent), signifying both input and output. Refer to the System Administrator's Guide for a complete discussion of AMLC protocols.

config

INPUT. Indicates the desired config setting. 0 indicates no change desired.

lword

INPUT. Indicates the desired line characteristics. The buffer number used for the line cannot be changed by a user program using this interface.

code

OUTPUT. Status returned to caller, either a 0 for success or one of the error codes, as listed in Appendix A.

Description

This routine, a direct entrance call, performs the assignment and unassignment of AMLC lines for a caller. A user may own more than one assigned line. The caller may also set line characteristics, protocol, etc. This routine will only allow a caller to assign a line that has a corresponding LBT entry of 0, which means that the line is assignable. The buffer used for the assigned line is dynamically chosen within ASNLN$.

Refer to the System Administrator's Guide for protocol, config, and lword values.

Loading and Linking Information

NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode

T$AMLC

Purpose

T$AMLC is a direct entrance call. It performs raw data movement, provides status information about assigned AMLC lines, and transfers characters between the caller's buffer and a desired assigned line's buffer. The caller must own the desired line, that is, the line has been assigned with the ASNLN$ routine or a Primos command.

Usage

```
DCL T$AMLC ENTRY(FIXED BIN(15), PTR OPTIONS (SHORT), FIXED BIN(15),
                 FIXED BIN(15), 2 FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15));

CALL T$AMLC (line, ADDR(buffer), ch_cnt,
             key, statv, ch_pos,
             errcode);
```

Parameters

line

INPUT. Indicates the AMLC line number desired for the data movement operation.

ADDR(buffer)

INPUT. Pointer to the address of the caller's buffer.

ch_cnt

INPUT. Indicates the desired number of characters to move. No maximum limit is enforced, other than the limits of the fixed bin(15) parameter: 32,767 characters.

key

INPUT. Indicates the desired operation. Valid keys are:

1 Input ch_cnt characters.

2 Input ch_cnt characters or until .NL. is encountered. statv(1) will hold the actual number of characters read.

3 Output ch_cnt characters. Maximum is ch_cnt. This key assures the caller that ch_cnt characters will be output. For example, an error is not returned if the line's input or output buffer is smaller than ch_cnt. T$AMLC will output blocks of data from the caller's buffer into the available space in the line's output buffer until ch_cnt is exhausted. A one-second wait is issued between output chunks to allow time for the line's output buffer to clear. In most cases, the entire ch_cnt should be output at once.

4 Load statv such that statv(1) = number of characters in input buffer. statv(2) = state of carrier. 0 = carrier, not 0 = no carrier.

5 Return status of output buffer such that statv(1) = 1 if room for ch_cnt in output buffer. statv(1) = 0 if not enough room for ch_cnt. statv(2) = state of carrier.

6 Input all available characters in the input buffer. Maximum = ch_cnt. This key will place all the available characters from the line's input vbuffer into the caller's buffer. statv(1) = number of characters actually input.

7 Return additional output buffer status. (Refer to key 5.) statv(1) = amount of character space remaining in the output buffer.

8 Flush input buffer.

9 Flush output buffer.

10 Flush both output and input buffers.

11 Output characters to available space in output buffer. This key will output as many characters as possible into the line's output buffer. A wait will not be done to exhaust ch_cnt.

After execution on this key, statv(1) = ch_cnt minus the number of characters actually output, i.e. statv(1) = number of chars that were not successfully output. If statv(1) = 0, then all characters were output.

statv

OUTPUT. Indicates the two-halfword status vector, subdivided into statv(1) and statv(2); these will output values dependent on the key that is input.

ch_pos

INPUT. The caller may wish to indicate a starting position within the buffer addressed by loc(buf_ad). ch_pos applies for both input and output keys. This is an optional argument. If omitted, the default is to start with the first character.

Note

If ch_pos is used, the first character position should be indicated by 1. (There is no character at position 0.) Also, ch_pos is not updated within T$AMLC.

code

OUTPUT. Optional argument to return error status. If code is present, error messages will not be printed at the caller's terminal.

Loading and Linking Information

```
FTNLIB   --   R-mode
NPFTNLB  --   V-mode (unshared)
PFTNLB   --   V-mode
SVCLIB   --   R-mode (maintained for PRIMOS-II)
```

AS$LST

Purpose

AS$LST retrieves asynchronous line characteristics.

Usage

```
DCL AS$LST ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
                  PTR, PTR, FIXED BIN(15), FIXED BIN(15),
                  FIXED BIN(15));

CALL AS$LST (line_number, key, version, list_ptr, errlist_ptr,
             list_length, errcount, code);
```

Parameters

line_number

INPUT. The asynchronous line about which you wish to retrieve information. Specify -1 to retrieve the characteristics of your login line.

key

INPUT. A key that indicates the source of the information that AS$LST is to return. The valid keys are:

key	action
K$PLST	Return the parameters specified by the parameter list pointed to by list_ptr. (See below.)
K$GTAL	Return all parameters. An asynchronous line has 29 retrievable characteristics. A full list of characteristics and values is provided below.

version

INPUT. The version number of the AS$LST internal structure. For PRIMOS Revision 22, set this parameter to 1.

list_ptr

INPUT -> OUTPUT. A pointer to an array in your program that consists of pairs of 16 bit halfwords. In the first entry of each pair, you can specify a line characteristic that you want AS$LST to return; AS$LST returns the corresponding value of that

characteristic to the second entry in the pair. If you want AS$LST to return only the characteristics specified in this array, you must set key to K$PLST. If you want AS$LST to return values for all 29 asynchronous line characteristics, set key to K$GTAL; in this case, you need not specify anything in the array.

errlist_ptr

INPUT -> OUTPUT. A pointer to an array of 16 bit halfwords in your program. AS$LST returns error codes for invalid characteristics to this array if key is set to K$PLST. This array should be the same size as the array pointed to by list_ptr. Errors are returned as pairs of numbers, the first number in each pair being an index to the list you specified in list_ptr, and the second number being the error code. For example, if you specify two characteristics in the array pointed to by list_ptr and the second characteristic you specified contains an error, the errlist_ptr array will contain a pair consisting of the number 2 followed by an error code. The error codes returned to this list are as follows:

E$DPAR Duplicate parameter. Returned each time a valid characteristic number is duplicated in the list.

E$PNR Parameter not retrievable. Returned if you specify a characteristic number that is invalid or a valid characteristic number that may not be retrieved.

list_length

INPUT -> OUTPUT. If you set key to K$PLST, you must set list_length to the number of asynchronous line characteristics specified in the array pointed to by list_ptr. If you set key to K$GTAL, you must set list_length to the declared size of the array of pairs of halfwords; AS$LST will change the list_length value to the number of pairs that it returns. The arrays pointed to by list_ptr and errlist_ptr should be the same size.

error_count

The number of errors returned to the error list pointed to by errlist_ptr. Errors are returned to this list only if you set key to K$PLST.

code

OUTPUT. Standard error code. The possible codes are:

E$OK The operation completed successfully.

E$BLEN The array was not large enough for the number of characteristics to be returned.

E$BVER The version number is not correct.

E$BPAR A bad parameter was specified in the list pointed to by list_ptr. If this error is returned, there are one or more error codes in the error list pointed to by errlist_ptr.

Discussion

AS$LST returns the characteristics of the asynchronous line that you specify. You can use AS$LST to retrieve all of the line's characteristics or any subset of them. AS$LST can return the characteristics of local asynchronous lines and Network Terminal Service (NTS) lines. These can be terminal lines or assignable lines. AS$LST cannot list characteristics of remote users; you must use DUPLX$ to list characteristics of remote users.

The information returned by AS$LST consists of pairs of numbers: the first number in each pair is the characteristic, and the second number is the value of the characteristic. Each pair consists of two 16 bit halfwords.

To make AS$LST return only certain characteristics, you must set key to K$PLST; then you must specify the number of each characteristic that you want to retrieve as the first number in one of the pairs in the list_ptr array.

AS$LST can return the following asynchronous line characteristics:

```
2   Echo              0 -- No echo
                      1 -- Echo

5   Reverse flow control   0 -- No Xon/Xoff
                           1 -- Xon/Xoff

11   Line Speed      -1 -- Other
                      0 -- 110 bps
                      1 -- 134.5 bps
                      2 -- 300 bps
                      3 -- 1200 bps
                      4 -- 600 bps
                      5 -- 75 bps
                      6 -- 150 bps
```

		7 -- 1800 bps 8 -- 200 bps 9 -- 100 bps 10 - 50 bps 11 - 75/100 bps 12 - 2400 bps 13 - 4800 bps 14 - 9600 bps 15 - 19200 bps 16 - 48000 bps 17 - 56000 bps 18 - 64000 bps 30 - 3600 bps 31 - 7200 bps
12	Flow control	0 -- No Xon/Xoff 1 -- Xon/Xoff
13	Line feed	0 -- No line feed 1 -- Line feed after carriage return
21	Parity check	0 -- No parity checking 3 -- Parity checking and generation
50	Char length	5 -- 5 bits 6 -- 6 bits 7 -- 7 bits 8 -- 8 bits
51	Stop bits	1 -- 1 stop bit 2 -- 2 stop bits
52	Parity type	0 -- Parity odd 1 -- Parity even
53	Line protocol	0 -- TTY 1 -- TRAN 2 -- TT8BIT 3 -- TTYUPC 4 -- TTY8 5 -- TTYNOP 6 -- TTYHS 7 TRANHS 8 -- TTY8HS 9 -- TTYHUP
75	Data sense enable	0 -- Don't use reverse channel protocol 1 -- Use reverse channel protocol
76	Data set sense	0 -- If Data set sense is off(1); do XON If Data set sense is on(0); do XOFF 1 -- If Data set sense is off(1); do XOFF If Data set sense is on(0); do XON

77 Input error detection 0 -- Disable error detection
1 -- Enable error detection

78 Data set control 0 -- Off
1 -- On

79 Loop line 0 -- Do not loop
1 -- Loop

80 User number Terminal line: process number associated with line.
Assignable line: Owner process number of line.

81 Auto speed detect 0 -- Do not enable auto speed detect
1 -- Enable auto speed detect

82 Line type 0 -- Terminal line
1 -- Assignable line

83 Logout on disconnect 0 -- Disable logout on disconnect
1 -- Enable logout on disconnect

Note

Do not attempt to set characteristics numbered 84 or greater.

84 Jumper 1 speed Same possible values as characteristic 11.

85 Jumper 2 speed Same possible values as characteristic 11.

86 Jumper 3 speed Same possible values as characteristic 11.

87 Clock speed Same possible values as characteristic 11.

88 Buffer number Terminal line: Buffer number associated with process.
Assignable line: Buffer number associated with line.

89 Clock line 0 -- Line is not the clock line
1 -- Line is the clock line

90 Flow control type 0 -- None
1 -- Input buffer
2 -- On controller board
3 -- Both input buffer and on controller board

91 Controller type 0 -- Unknown
1 -- ICS1
2 -- ICS2
3 -- ICS3
4 -- AMLC/DMT
5 -- AMLC/DMQ

		6 -- NTS
92	AMLQ buffer	Any legal buffer size.
105	Received XOFF	0 -- Did not receive XOFF 1 -- Received XOFF

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

AS$LIN

Purpose

AS$LIN returns asynchronous line number.

Usage

```
DCL AS$LIN ENTRY (FIXED BIN(15), FIXED BIN(15));

CALL AS$LIN (line_number, code);
```

Parameters

line_number

OUTPUT. The line number of your asynchronous line.

code

OUTPUT. Error status code. The possible codes are:

E$OK	The operation completed successfully.
E$BLIN	Bad line number.
E$LNP	Line not present on system.
E$RMLN	Illegal operation on remote line.

Discussion

AS$LIN returns the line number of the asynchronous line attached to the caller's terminal. This subroutine returns local line numbers; it does not return remote line numbers. Do not use this subroutine for remote login terminals that use a modem. This subroutine can be used for terminals connected to the system by a direct connect modem.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

AS$SET

Purpose

AS$SET sets asynchronous line characteristics.

Usage

```
DCL AS$SET ENTRY (FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
                  PTR, PTR, FIXED BIN(15), FIXED BIN(15),
                  FIXED BIN(15));

CALL AS$SET (line_number, key, version, list_ptr, errlist_ptr,
             list_length, errcount, code);
```

Parameters

line_number

INPUT. The asynchronous line to be set. Specify -1 if you wish to set your login stream line.

key

INPUT. A key that indicates the source of the information for setting the line characteristics. The possible options are:

key	action
K$PLST	Indicates that the source of the information is the list pointed to by list_ptr.
K$SLS	Indicates that the source of the information is the system login settings defined in the System Login Characteristics Table. This Table is described in the System Administrator's Guide, Volume II.

version

INPUT. The version number of the AS$SET internal structure. For PRIMOS Revision 22, set this parameter to 1.

list_ptr

INPUT. A pointer to an array of 16-bit halfwords in your program. Use this array to specify the line characteristics and their values that you wish to set. Specify this information as pairs of

numbers, the first number in each pair being the characteristic and the second number being the value. The numbers for characteristics and their values are listed in the description of AS$LST. Each pair of numbers requires two 16-bit halfwords.

errlist_ptr

INPUT -> OUTPUT. A pointer to an array of 16 bit halfwords in your program. AS$SET returns error codes for invalid characteristics to this array. This array should be the same size as the array pointed to by list_ptr. Errors are returned as pairs of numbers, the first number in each pair being the index to the list you specified in list_ptr and the second number being the error code. For example, if you specify two characteristics in the array pointed to by list_ptr and the second characteristic you specified contained an error, the errlist_ptr array will contain a pair consisting of the number 2 followed by an error code. The error codes returned in this list are as follows:

E$DPAR Duplicate parameter. Returned each time a valid characteristic number is duplicated in the list.

E$IPS Invalid parameter setting. Returned for a valid characteristic number with an invalid value.

E$ITLB Invalid terminal line buffer. Returned if you try to set the User Number characteristic for an assignable line.

E$PNS Parameter not settable. Returned if you specify a characteristic number that is invalid or a valid characteristic number that may not be set.

list_length

INPUT. The number of characteristics specified in the array pointed to by list_ptr. The arrays pointed to by list_ptr and errlist_ptr should be the same size. If you specify K$SLS for the key parameter, you must set the list_length to zero.

errcount

OUTPUT. The number of errors returned to the array pointed to by errlist_ptr.

code

OUTPUT. Standard error code. The possible codes are:

E$OK The operation completed successfully.

E$BLEN The error list is not large enough for the number of errors to be returned. No characteristics have been set.

E$BPAR One or more errors were returned to the array pointed to by errlist_ptr. No characteristics have been set.

E$BVER Incorrect version number. No characteristics have been set.

Discussion

AS$SET sets characteristics of asynchronous lines. It can set the characteristics of local asynchronous lines and Network Terminal Service (NTS) lines. These can be terminal lines or assignable lines. Characteristics set using AS$SET remain set for the duration of the current session. AS$SET cannot set characteristics of remote users; you must use DUPLX$ to set characteristics of remote users.

AS$SET sets characteristics based on the key parameter. If key is K$PLST, AS$SET uses the information you supply in the area pointed to by list_ptr to set characteristics. If key is KSLS, ASSET resets all characteristics to system defaults.

You use AS$SET to change the values of characteristics; characteristics not specified in AS$SET remain set to their existing values. You specify values for characteristics as pairs of code numbers, the first number in each pair specifying the characteristic, and the second number specifying the value for that characteristic. You can specify these pairs in any sequence.

The names and valid values for characteristics are shown in the discussion of AS$LST. Do not attempt to set characteristics numbered 84 or greater. Do not attempt to set a value for User Number for an assignable line or a value for Buffer Number for a terminal line.

Before setting any characteristics, AS$SET validates all of the listed pairs. If it detects invalid values or duplicate entries, it does not set any line characteristics, but instead writes a paired entry (characteristic and error code) for each error into the area pointed to by errlist_ptr and sets code to E$BPAR.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

NT$LTS

Purpose

NT$LTS returns information about a PRIMOS line used for LAN terminal service (LTS).

Usage

```
DCL NT$LTS ENTRY (FIXED BIN(15), FIXED BIN(15), CHAR(16) VAR,
                  FIXED BIN(15), CHAR(6), FIXED BIN(15));

CALL NT$LTS (primos_line_number, media_type, LTS_name,
             LTS_line, MAC_address, code);
```

Parameters

primos_line_number

INPUT. A PRIMOS line number. The line number can range from 1024 to 1535. The line can be either a login or an assignable line across network terminal service.

media_type

OUTPUT. The media type. Possible values are:

0 IEEE 802.3 Ethernet

LTS_name

OUTPUT. The LTS name corresponding to the media type and Media Access Control (MAC) address for the LTS. This field is blank and NT$LTS returns a zero return code if the unconfigured LTS option is utilized and this is not a configured LTS.

LTS_line

OUTPUT. Physical line number on LTS for connected line. Must range from 0 to 7.

MAC_address

OUTPUT. The Media Access Control (MAC) address for the connection corresponding to the PRIMOS line number requested.

code

OUTPUT. The status code. Possible values are:

E$OK The call to NT$LTS was completed without error.

E$NTNS NTS is not currently started.

E$BDEV Specified line is not a valid network terminal service line.

E$LNOC The PRIMOS line is not currently connected.

Discussion

NT$LTS is a direct-entrance call that returns the media type, LAN Terminal Service (LTS) logical name, LTS line number, and Media Access Control (MAC) address for network terminal service lines in PRIMOS.

If the LTS logical name is not configured, as in the case of unconfigured LAN Terminal Services allowed on the Local Area Network (LAN), the LTS logical name returned is null. NT$LTS returns valid information only if the NTS line is connected.

Effective for PRIMOS Rev. 22.0 and subsequent revisions.

Example

The following code from an external logout program receives an LTS_name from NT$LTS, determines whether it is the name of a configured LTS, and if it is, calculates the proper surcharge for the amount of time used.

```
DCL chargeable_lts_names(3) char(16) var init('lts1','lts2','
    lab-connect');

DCL LTS_rates(0:3) fixed bin(15) init(0,1,3,2); /* In cents/min. */

CALL NT$LTS(my_line, media_type, LTS_name, LTS_line, MAC_address,code);
 If code = 0 then
    If LTS_name ^= '' then do ; /* Configured LTS - see if we should
                                   apply surcharge for use */
       charge_index = Index (chargable_lts_names,' ' ||LTS_name||' ');
       charge_index = 0;
       Do i = 1 to 3 while ( charge_index = 0 );
            If LTS_name = chargable_lts_names(i) then
                 charge_index = i;
            end;
       LTS_charge = LTS_rates(charge_index) * connect_time;
```

PART IV

APPLICATION LIBRARY

9
Introduction to Application Library

GENERAL DESCRIPTION

Part IV of this Volume contains descriptions of the Application Library, called APPLIB for its subroutines written in R-Mode and VAPPLB for those in V-Mode. The Application Library is a user-oriented library that provides a set of service routines, designed for ease of use. In many cases, the APPLIB or VAPPLB routines call a lower-level routine, filling in arguments that the caller isn't concerned about. The routines may also reformat the data that the lower-level routine returns. The use of APPLIB or VAPPLB routines avoids a duplication of effort and provides a consistent interface for the terminal user.

All of these routines are written as FORTRAN functions. When used as such, they return one -- and only one -- of the following:

- A status indication (a FORTRAN logical .TRUE. or .FALSE.)
- An appropriate value
- An alternate value or format of a returned argument
- A code that must be decoded

All error detection, reporting, and, if possible, recovery are performed by the routine, which returns only an indication of success or failure. This simplified error-reporting scheme assures the user that the error is reported and all possible recovery procedures have been tried.

These routines may be used either as subroutines or as functions that return a value. If you use them as functions, be aware that the logical value returned is a .TRUE. or .FALSE. according to FORTRAN conventions. If you use them as subroutines, be aware that there is no code parameter provided for error detection. You are therefore urged to use the function form.

Since FORTRAN logical values are returned by these functions (as opposed to PL/I logical values, for example, returning a single bit), the Usage descriptions are given in FTN. Programmers in other languages should consult the chapter treating that language in Volume I to see how to handle these values.

HOW TO USE PART IV

Refer to the following chapters for descriptions of the indicated categories of functions provided by the Applications library:

Chapter 10:	String Manipulation Routines
Chapter 11:	User Query Routines
Chapter 12:	System Information Routines
Chapter 13:	Randomizing Routines
Chapter 14:	Conversion Routines
Chapter 15:	File System Routines
Chapter 16:	Parsing Routine

FORMAT SUMMARY

Below is a brief summary of the calling sequences for all the VAPPLB and APPLIB routines. The type codes are defined as:

Type Code	Description
L	LOGICAL
I	INTEGER*2 or INTEGER*4
I*2	INTEGER*2
R	REAL
DP	DOUBLE PRECISION or REAL*8

Group	Name	Type	Arguments
String	CSTR$A	L	(A,ALEN,B,BLEN)
	CSUB$A	L	(A,ALEN,AFC,ALC,B,BLEN,BFC,BLC)
	FILL$A	I	(NAME,NAMLEN,CHAR)
	FSUB$A	L	(STRING,LENGTH,FCHAR,LCHAR,FILCHAR)
	GCHR$A	I	(FARRAY,FCHAR)
	JSTR$A	L	(KEY,STRING,LENGTH)
	LSTR$A	L	(A,ALEN,B,BLEN,FCP,LCP)
	LSUB$A	L	(A,ALEN,AFC,ALC,B,BLEN,BFC,BLC,FCP,LCP)
	MCHR$A	I	(TARRAY,TCHAR,FARRAY,FCHAR)
	MSTR$A	I*2	(A,ALEN,B,BLEN)
	MSUB$A	I*2	(A,ALEN,AFC,ALC,B,BLEN,BFC,BLC)
	NLEN$A	I*2	(NAME,NAMLEN)
	RSTR$A	L	(STRING,LENGTH,COUNT)
	RSUB$A	L	(STRING,LENGTH,FCHAR,LCHAR,COUNT)
	SSTR$A	L	(STRING,LENGTH,COUNT,FILCHAR)
	SSUB$A	L	(STRING,LENGTH,FCHAR,LCHAR,COUNT,FILCHAR)
	TREE$A	I	(NAME,NAMLEN,FSTART,FLEN)
	TYPE$A	L	(KEY,STRING,LENGTH)
User Query	RNAM$A	L	(MSG,MSGLEN,NAMKEY,NAME,NAMLEN)
	RNUM$A	L	(MSG,MSGLEN,NUMKEY,VALUE)
	RNAM$A	L	(MSG,MSGLEN,NAMKEY,NAME,NAMLEN)
	RNUM$A	L	(MSG,MSGLEN,NUMKEY,VALUE)
	YSNO$A	L	(MSG,MSGLEN,DEFKEY)
Information	CTIM$A	DP	(CPUTIM)
	DATE$A	DP	(DATE)
	DOFY$A	DP	(DOFY)
	DTIM$A	DP	(DSKTIM)
	EDAT$A	DP	(EDATE)
	TIME$A	DP	(TIME)
Randomizing	RAND$A	DP	(SEED)
	RNDI$A	DP	(SEED)
Conversion	CASE$A	L	(KEY,STRING,LENGTH)
	CNVA$A	L	(NUMKEY,NAME,NAMLEN,VALUE)
	CNVB$A	I	(NUMKEY,VALUE,NAME,NAMLEN)
	ENCD$A	L	(ARRAY,WIDTH,DEC,VALUE)
	FDAT$A	DP	(DATMOD,DATE)
	FEDT$A	DP	(DATMOD,DATE)
	FTIM$A	DP	(TIMMOD,TIME)

Group	Name	Type	Arguments
File System	CLOS$A	L	(FUNIT)
	DELE$A	L	(NAME,NAMLEN)
	EXST$A	L	(NAME,NAMLEN)
	GEND$A	L	(FUNIT)
	OPEN$A	L	(OPNKEY+TYPKEY+UNTKEY,NAME,NAMLEN,FUNIT)
	OPNP$A	L	(MSG,MSGLEN,OPNKEY+TYPKEY+UNTKEY,NAME, NAMLEN,FUNIT)
	OPNV$A	L	(OPNKEY+TYPKEY+UNTKEY,NAME,NAMLEN,FUNIT, VERKEY,WTIME,RETRY)
	OPVP$A	L	(MSG,MSGLEN,OPNKEY+TYPKEY+UNTKEY,NAME, NAMLEN,FUNIT,VERKEY,WTIME,RETRY)
	POSN$A	L	(POSKEY,FUNIT,POS)
	RPOS$A	L	(FUNIT,POS)
	RWND$A	L	(FUNIT)
	TEMP$A	L	(TYPKEY,NAME,NAMLEN,FUNIT)
	TRNC$A	L	(FUNIT)
	TSCN$A	L	(KEY,FUNITS,ENTRY,MAXSIZ, ENTSIZ,MAXLEV,LEV,CODE)
	UNIT$A	L	(FUNIT)
Parsing	CMDL$A	L	(KEY,KWLIST,KWINDX,OPTBUF,BUFLEN OPTION,VALUE,KWINFO)

NAMING CONVENTIONS

All APPLIB and VAPPLB routines follow a consistent naming convention designed to avoid the possibility of a conflict with user-written routines and system routines. They all have a four-letter mnemonic name and the suffix $A. For example, the routine to open a temporary file is named TEMP$A.

Note

While all subroutines in APPLIB/VAPPLB use a $A suffix, not every subroutine with this suffix belongs to this library.

Subroutines used internally by APPLIB routines have a suffix of $$A. Do not use these subroutines under ordinary circumstances.

Keys

Many routines have options which are specified by named parameter keys which all begin with the prefix A$. All parameter keys are defined in a $INSERT file named SYSCOM>A$KEYS.INS.language. The key names following the A$ prefix are three- or four-letter mnemonics specifying the allowable options for the various routines. They are INTEGER*2 data types. In addition, the FORTRAN version of this file supplies all the appropriate FUNCTION type declarations for the application routines. Volume I provides a listing of SYSCOM>A$KEYS with the

decimal value for each key. Please read the chapter on your language interface to see how to use this file.

LIBRARY IMPLEMENTATION POLICIES

VAPPLB and its R-mode version, APPLIB, exist as independent libraries in the UFD LIB.

> **Caution**
>
> R-mode subroutines can be called from FTN and PMA in R-mode only. If you call an R-mode routine from a program in a different mode, the results are unpredictable. Refer to the FORTRAN and PMA chapters in Volume I for information on declaring parameters in FTN and PMA, respectively.

The routines have been coded to make them easily callable from most other languages, including PL/I and 1977 ANSI FORTRAN, both of which can automatically generate string length arguments following string arguments. As a result, in the argument pair name, namlen, the name is often updated by an application routine, but the namlen argument is never modified. If the namlen argument is not 0 or positive, an error message is displayed on the user terminal. Where applicable, the function value returned is .FALSE.. The function NLEN$A can be used to determine the operational length of a returned name.

All application routines that either accept keys as arguments, or call other routines which do, use the SYSCOM>A$KEYS file to define those keys. Also, these routines do not take advantage of any particular numerical values these keys may have, in case it should become necessary either to change these values or to add new keys with numerical values which do not fit the previous pattern. For example, there are no computed GOTOs on keys and no range checks for validity of a key. In this way, if a new SYSCOM>A$KEYS file is created, both the user programs and the routines they call will always agree on the meaning of a given key. The same is true of the declared types of the application functions.

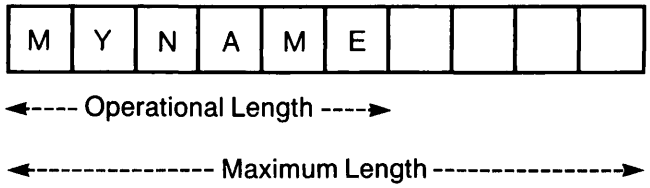
Library Building

All routines are compiled into a single binary file which is then converted into the appropriate library file with the EDB utility. At present, the only difference between the R-mode and V-mode build procedures is the FTN compile option used. For APPLIB, all routines are compiled for 64R-mode loading (LOAD). For VAPPLB, all routines are compiled for 64V-mode loading (SEG and BIND). An unshared version of VAPPLB exists in NVAPPLB. In addition, all routines included in VAPPLB

are pure procedure and may be loaded into the shared portion of a shared procedure. When you use BIND and then specify VAPPLB, the system selects APPLICATION_LIBRARY.RUN, another version of the shared routines used for the EPF environment.

STRING MANIPULATION ROUTINES

The string manipulation routines operate on packed strings, unless stated otherwise. Most of the routines in this section require that the maximum length of a string (in characters) be passed as an argument. The maximum length is the actual storage allocated for that string in bytes or characters (including any trailing blanks). The operational length of a string does not include trailing blanks, so it may be shorter than the maximum length. (See Figure 9-1.) Since the length of a string is specified as an INTEGER*2 variable, the maximum possible length is 32767 characters.



Maximum Length and Operational Length
Figure 9-1

The majority of routines that operate on entire strings first truncate them to their operational length. The routines that operate on substrings treat any trailing blanks as part of the substring.

All string-length specifications and substring-delimiting character positions are checked for validity and must conform to the following rules:

- Maximum string-length specifications must be greater than or equal to 0. A value of 0 indicates a null or empty string.
- Substring-delimiting character positions must be greater than or equal to 0. The length of the substring must be less than or equal to the physical string length. The beginning character position must be less than or equal to the ending character position. A value of 0 for either the starting or ending character position indicates a null substring.

If these rules are violated, an error message will be displayed and the logical functions will be .FALSE..

USER QUERY ROUTINES

These routines provide a convenient means to input data from the user's terminal. Each routine can prompt the terminal user with a customized message, and then process the user's response.

FILE SYSTEM ROUTINES

The file system routines in the Applications library give the user a simple and consistent way to specify the most common file system operations. Accordingly, the Applications library does not provide the user with the full capabilities of the file system routines since for detailed operations it is best to use the file system routines themselves (Volume II). This library supports both Sequential Access Method (SAM) and Direct Access Method (DAM) files. There is no support for segment directory files, as the MIDAS subsystem provides the higher level functions with these files.

All routines except Open, Delete, and Check for File Existence use only the file unit and not the filename. File units are explained in Volume II. Also, each routine carries the name of its function, as above, with arguments consisting of only the relevant information, usually only the file unit number. Note that all filenames, except scratch files, may be pathnames.

The only complicated routines are the five OPEN routines, because of the many ways programs can obtain the name of the file they wish to open and the various options for verification or error recovery. Five different routines exist to perform the varying levels of complexity. In this way, the simple operations are represented by simple calling sequences. Only complex operations need complex argument lists.

All OPEN routines allow selection of the file type (SAM or DAM) and all but TEMP$A allow specification of the open mode (READ, WRITE, or READ/WRITE). TEMP$A (scratch) files are always opened for READ/WRITE. Table 9-1 shows the routines available for opening.

Table 9-1
Ways to Open a File

Open name.	OPEN$A
Open funit.	OPNP$A
Open name, verify, and delay.	OPNV$A
Open funit, verify, and delay.	OPVP$A
Open scratch file.	TEMP$A

All OPEN routines can choose the file unit number upon which a file will be opened. The A$GETU key is used for this purpose and the PRIMOS file unit selected by the routine will be returned to the user (in the argument funit). If A$GETU is not used, the user must provide the routine with a usable file unit number.

Several of these subroutines have arguments called verkey, which allows verification of the validity of the file operation requested. Verification provides the following options:

1. Verify that the file is new; otherwise, verify that it is all right to modify a file which already exists.

2. Verify that the file may be modified and determine whether an existing file is to be overwritten or appended.

3. Verify that the file exists; that is, do not allow creation of a new file. Note that if the open mode is READ, this is the only possible verification option.

In case of failure of an operation, the argument wtime allows the subroutine to delay the time specified, then try again the number of times allowed by retry. wtime provides the following options:

1. If, and only if, the file is "IN USE", wait a supplied number of seconds (elapsed time) and try again.

2. Repeat step 1 a specified number of times.

SYSCOM>A$KEYS

The keys needed for FORTRAN programs are given in:

SYSCOM>A$KEYS>A$KEYS.INS.FTN

The Pascal and PL/I programmers should use the A$KEYS.INS.file in SYSCOM>A$KEYS that is applicable to their language.

The listings from the SYSCOM UFD use octal values. Refer to the Appendix section of Volume I for a listing of keys with decimal values.

10
String Routines

SUMMARY OF STRING MANIPULATION ROUTINES

This chapter contains descriptions of the following string manipulation routines from the APPLIB/VAPPLB subroutines library.

CSTR$A	Compare two strings for equality.
CSUB$A	Compare two substrings for equality.
FILL$A	Fill a string with a character.
FSUB$A	Fill a substring with a given character.
GCHR$A	Get a character from a packed string.
JSTR$A	Left-justify, right-justify, or center a string within a field.
LSTR$A	Locate one string within another.
LSUB$A	Locate one substring within another.
MCHR$A	Move a character between packed strings.
MSTR$A	Move one string to another.
MSUB$A	Move one substring to another.
NLEN$A	Determine the operational length of a string.
RSTR$A	Rotate string left or right.
RSUB$A	Rotate substring left or right.
SSTR$A	Shift string left or right.
SSUB$A	Shift substring left or right.
TREE$A	Test for pathname.
TYPE$A	Determine string type.

CSTR$A

Purpose

CSTR$A is a logical function used to compare two strings for equality. The function returns a logical value of .TRUE. if each character in string a matches the corresponding character in string b, or if both strings are null (length equal to 0). Otherwise, the function returns .FALSE.. Only the operational lengths are used in the comparison. (Trailing blanks are ignored.) If the two strings are not of equal length, the result is .FALSE..

Usage

```
INTEGER*2  a(1), alen, b(1), blen
LOGICAL    log

log = CSTR$A(a, alen, b, blen)
```

Parameters

a

INPUT. String to be compared, packed two characters per halfword. Internal data type of the array does not matter.

alen

INPUT. Length of a, in characters. Length must be 0 or greater.

b

INPUT. String to be compared against, packed two characters per halfword. Internal data type of the array does not matter.

blen

INPUT. Length of b in characters. Length must be 0 or greater.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by CSTR$A

CSUB$A and NLEN$A

CSUB$A

Purpose

CSUB$A is a logical function used to compare substrings for equality.

Usage

```
      INTEGER*2  a(1), alen, afc, alc
      INTEGER*2  b(1), blen, bfc, blc
      LOGICAL    log

      log = CSUB$A(a, alen, afc, alc,
     x             b, blen, bfc, blc)
```

Parameters

a

INPUT. Array containing substring to be compared, packed two characters per halfword. Internal data type of the array does not matter.

alen

INPUT. Length of a, in characters. Length must be 0 or greater.

afc

INPUT. First character position of substring in a

alc

INPUT. Last character position of substring in a.

b

INPUT. Array containing substring to be compared against, packed two characters per halfword. Internal data type of the array does not matter.

blen

INPUT. Length of b in characters. Length must be 0 or greater.

bfc

INPUT. First character position of substring in b.

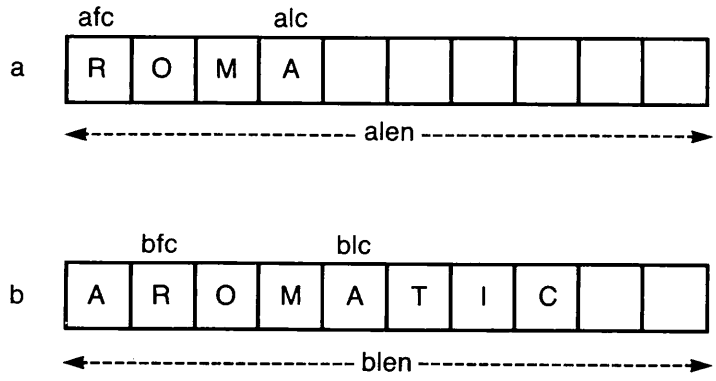
blc

INPUT. Last character position of substring in b.

Discussion

If each character in the a substring matches the corresponding character in the b substring, or both substrings are null (length equal to 0), the function will be .TRUE.. If two corresponding characters do not match, or if the lengths of the substrings are not equal, the function will be .FALSE..

Figure 10-1 is a representation of the arguments to CSUB$A.



Arguments to CSUB$A
Figure 10-1

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

FILL$A

Purpose

FILL$A is an INTEGER function that fills the name buffer with the fill character char supplied. The function is INTEGER*2 or INTEGER*4, but its value is always 0.

Usage

```
      INTEGER*2  name(1), namlen, char(1)
      INTEGER*2  rt_val
C      rt_val may also be declared INTEGER*4

      rt_val = FILL$A(name, namlen, char)
             (or)
      CALL  FILL$A(name, namlen, char)
```

Parameters

name

INPUT. Name of buffer to fill, packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of name in characters (INTEGER*2).

char

INPUT. Fill character,in FORTRAN A1 format, used to fill the buffer. The single char is loaded into the left byte of char; its data type does not matter.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

Other Routines Called by FILL$A

CSUB$A and NLEN$A

FSUB$A

Purpose

FSUB$A is a logical function used to fill a character substring with a specified character. The substring delimited by fchar and lchar is filled with the character specified in filchar. The string parameters are checked for validity. If an error is found, the function is .FALSE. and a message is printed. If all parameters are valid, the function will be .TRUE..

Usage

```
INTEGER*2  string(1), length, fchar, lchar, filchar
LOGICAL    log

LOG = FSUB$A(string, length, fchar, lchar, filchar)
        (or)
CALL  FSUB$A(string, length, fchar, lchar, filchar)
```

Parameters

string

INPUT/OUTPUT. String containing substring to be filled, packed two characters per halfword. Data type does not matter.

length

INPUT. Length of string in characters.

fchar

INPUT. First character position of substring.

lchar

INPUT. Last character position of substring.

filchar

INPUT. Fill character in FORTRAN A1 format. Left byte of filchar holds the character; its data type does not matter.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

GCHR$A

Purpose

GCHR$A is an INTEGER*2 or INTEGER*4 function which extracts a single character from a packed string. It is intended for use only by FORTRAN programmers. The function value will be the accessed character in FORTRAN A1 format (with blank padding on the right). The character returned will be left-justified and padded with blanks.

Usage

```
      INTEGER*2  farray(1), fchar
      INTEGER*2  rt_val
C   rt_val may be declared INTEGER*4

      rt_val = GCHR$A(farray, fchar)
             (or)
      CALL  GCHR$A(farray, fchar)
```

Parameters

farray

INPUT. Source that is the packed array. Its internal data type does not matter.

fchar

INPUT. Character position in farray to be returned.

Discussion

This routine replaces the FORTRAN statement:

```
CHAR = FARRAY(FCHAR)
```

where FARRAY is declared LOGICAL*1 (IBM FORTRAN) or of a one-character data type.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

JSTR$A

Purpose

This logical function is used to left-justify, right-justify, or center a string within itself. The function is .TRUE. if justification is successful; it is .FALSE. if the length is less than 0 or if a bad key is used.

Usage

```
INTEGER*2  key string(length), length
LOGICAL    log

LOG = JSTR$A(key, string, length)
        (or)
CALL  JSTR$A(key, string, length)
```

Parameters

key

INPUT. Determines direction of justification. Possible values are:

A$RGHT Right-justify

A$LEFT Left-justify

A$CNTR Center

string

INPUT/OUTPUT. String to be justified, packed two characters per halfword. Data type does not matter.

length

INPUT. Length of string in characters. It must be greater than 0.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by JSTR$A

NLEN$A, FILL$A, MSUB$A, GCHR$A.

LSTR$A

Purpose

LSTR$A is a logical function used to locate one string within another.

Usage

```
INTEGER*2  a(1), alen, b(1), blen
INTEGER*2  fcp, lcp
LOGICAL    log

log = LSTR$A(a, alen, b, blen, fcp, lcp)
        (or)
CALL  LSTR$A(a, alen, b, blen, fcp, lcp)
```

Parameters

a

INPUT. String to be located, packed two characters per halfword. Internal data type of the array does not matter.

alen

INPUT. Number of characters in a.

b

INPUT. String to be searched, packed two characters per halfword. Data type does not matter.

blen

INPUT. Length of b, in characters.

fcp

OUTPUT. First character position in b of substring that matches string a.

lcp

OUTPUT. Last character position in b of substring that matches string a.

Discussion

LSTR$A searches string b for the first occurrence of string a. If string a is found, the function returns .TRUE., and fcp and lcp will be equal to the character positions of the substring in b that matches string a. If string a is not found, or if either string is null (length equal to 0), the function returns .FALSE., and fcp and lcp will be equal to 0. Each string is logically truncated to its operational length before the search is performed (trailing blanks are ignored).

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by LSTR$A

LSUB$A and NLEN$A

LSUB$A

Purpose

This logical function is used to locate one substring within another.

Usage

```
INTEGER*2  a(1), alen, afc, alc
INTEGER*2  b(1), blen, bfc, blc
INTEGER*2  fcp, lcp
LOGICAL    log

log = LSUB$A(a, alen, afc, alc, b, blen, bfc, blc, fcp, lcp)
        (or)
CALL  LSUB$A(a, alen, afc, alc, b, blen, bfc, blc, fcp, lcp)
```

Parameters

a

INPUT. Array containing substring to be located, packed two characters per halfword. Data type does not matter.

alen

INPUT. Length of a, in characters (INTEGER*2).

afc

INPUT. First character position of substring in a.

alc

INPUT. Last character position of substring in a (INTEGER*2).

b

INPUT. Array containing substring to be searched, packed two characters per halfword. Data type does not matter.

blen

INPUT. Length of b, in characters (INTEGER*2).

bfc

INPUT. First character position of substring in b.

blc

INPUT. Last character position of substring in b.

fcp

OUTPUT. First character position in b of substring that matches substring in a.

lcp

OUTPUT. Last character position in b of substring that matches substring in a.

Discussion

LSUB$A searches the substring contained in b for the first occurrence of the substring contained in a. If a match is found, fcp and lcp will be equal to the character positions in b of the matching substring and the function is .TRUE..

If a matching substring cannot be found or if either substring is null (length equal to 0), the function will be .FALSE. and fcp and lcp will be equal to 0. (.TRUE. and .FALSE. are the FORTRAN logical values.)

Figure 10-1, included in the description of CSUB$A, illustrates the passage of arguments to both LSUB$A and CSUB$A.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

MCHR$A

Purpose

MCHR$A is an INTEGER function that moves a character from one packed string to another.

Usage

```
      INTEGER*2  tarray(1), tchar, farray(1), fchar
      INTEGER*2  rt_val
C      rt_val may be declared INTEGER*4

      rt_val = MCHR$A(tarray, tchar, farray, fchar)
             (or)
      CALL MCHR$A(tarray, tchar, farray, fchar)
```

Parameters

tarray

INPUT. Returned array of characters, packed two per halfword, firs
t
character on the left.

tchar

INPUT. Position in tarray of the character to be received.

farray

INPUT. Source string. Data type does not matter.

fchar

INPUT. Character position in farray of character to be moved.

Discussion

This routine replaces the FORTRAN statement:

```
TARRAY(TCHAR) = FARRAY(FCHAR)
```

when TARRAY and FARRAY are declared LOGICAL*1 (IBM FORTRAN) or of a one-character data type. Only one character in TARRAY is replaced.

The function value will be the character that was moved in FORTRAN A1 format, that is, the character in the left-most byte, right padded with blanks.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

MSTR$A

Purpose

MSTR$A is an INTEGER*2 or INTEGER*4 function used to move the source string to the destination string.

Usage

```
      INTEGER*2  a(1), alen, b(1), blen
      INTEGER*2  rt_val
C       rt_val may be declared INTEGER*4

      rt_val = MSTR$A(a, alen, b, blen)
             (or)
      CALL MSTR$A(a, alen, b, blen)
```

Parameters

a

INPUT. Source string, packed two characters per halfword. Data type does not matter.

alen

INPUT. Length of a, in characters.

b

OUTPUT. Destination string, packed two characters per halfword. Data type does not matter.

blen

INPUT. Length of b, in characters.

Discussion

If the source string is longer than the destination string, it will be truncated. If it is shorter, it will be padded with blanks. The source and destination strings may overlap. The function value will be equal to the number of characters moved (excluding blank padding). If either string is null (length equal to 0), no characters are moved and the function value will be equal to 0.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by MSTR$A

MSUB$A

MSUB$A

Purpose

MSUB$A is an integer function used to move the source substring contained in a to the destination substring contained in b.

Usage

```
INTEGER*2  a(1), alen, afc, alc
INTEGER*2  b(1), blen, bfc, blc
INTEGER*2  rt_val

rt_val = MSUB$A(a, alen, afc, alc, b, blen, bfc, blc)
       (or)
CALL  MSUB$A(a, alen, afc, alc, b, blen, bfc, blc)
```

Parameters

a

INPUT. Array containing source substring, packed two characters per halfword. Data type does not matter.

alen

INPUT. Length of a, in characters.

afc

INPUT. First character position of substring in a, packed two characters per halfword. Data type does not matter.

alc

INPUT. Last character position of substring in a.

b

INPUT/OUTPUT. Array containing destination substring, packed two characters per halfword. Data type does not matter.

blen

INPUT. Length of b, in characters (INTEGER*2).

bfc

INPUT. First character position of substring in b.

blc

INPUT. Last character position of substring in b.

Discussion

If the source substring is longer than the destination substring, it will be truncated. If it is shorter, it will be padded with blanks. The source and destination substrings may overlap.

If either substring is null (length equal to 0), no characters are moved and the function will be equal to 0. Otherwise it is equal to the number of characters moved (excluding blanks used for padding).

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by MSUB$A

MCHR$A

NLEN$A

Purpose

NLEN$A is an INTEGER*2 function that returns, as its function value, the actual length (not including trailing blanks) of the ASCII string in name.

Usage

```
INTEGER*2  name(1), namlen
INTEGER*2  rt_val

rt_val = NLEN$A(name, namlen)
         (or)
CALL  NLEN$A(name, namlen)
```

Parameters

name

INPUT. Name buffer to be tested, packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of the variable name, possibly containing blanks.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

RSTR$A

Purpose

RSTR$A is a logical function used to rotate a character string left or right. The string is truncated to its operational length before the rotate is performed; therefore, trailing blanks are not included in count. If length is less than 0, the function returns .FALSE., otherwise the function returns .TRUE..

Usage

```
INTEGER*2  string(1), length, count
LOGICAL    log

log = RSTR$A(string, length, count)
        (or)
CALL  RSTR$A(string, length, count)
```

Parameters

string

INPUT/OUTPUT. String to be rotated, packed two characters per halfword. Data type does not matter.

length

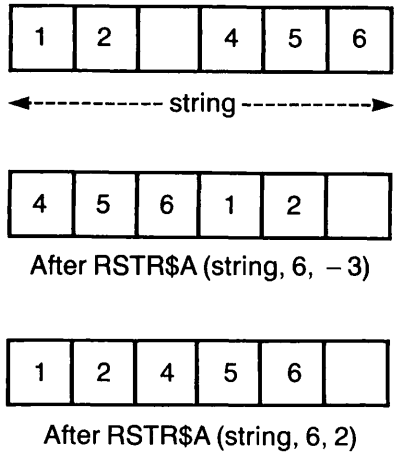
INPUT. Length of string in characters.

count

INPUT. Number of positions to rotate string. Negative count causes left rotate, positive count right rotate.

Discussion

This routine uses an algorithm that minimizes temporary storage and execution time. One word of temporary storage is used and the number of iterations necessary to rotate a string is equal to the length in characters of the string. A character is moved directly from its original position to its final destination position. Figure 10-2 shows the results of two calls to RSTR$A.



Use of RSTR$A
Figure 10-2

Example

Perhaps you have COBOL programs that are to be converted to CBL programs. During conversion CBL often automatically corrects some of the incompatibilities from your old COBOL programs. Refer to the COBOL to CBL Conversion Guide (MAN10002-1LA) to handle the more unusual conversion situations.

Furthermore, perhaps your COBOL programs previously had been submitted to the SEG link/loader. After you update these programs to CBL, you will also want to BIND them. Thereafter the system will dynamically load your programs. You no longer risk overwriting one executable runfile with another.

The following example of a program performing the character rotations above also shows what happens to ROTATE.COBOL when it is renamed and then recompiled as ROTATE.CBL. As indicated, you can BIND and RESUME the program as is. (Nevertheless, it would be prudent to first make the recommended changes.)

```
OK SLIST  ROTATE.COBOL
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ROTATE.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 STRING1  PIC X(32) VALUE '12 456
       01 LENGTH   COMP.
       01 CNT      COMP.
       PROCEDURE DIVISION.
       001-BEGIN.
           MOVE 6 TO LENGTH.
           MOVE -3 TO CNT.
           CALL 'RSTR$A' USING STRING1, LENGTH, CNT.
           EXHIBIT STRING1.
           MOVE 2 TO CNT.
           CALL 'RSTR$A' USING STRING1, LENGTH, CNT.
           EXHIBIT STRING1.
           STOP RUN.
OK CNAME ROTATE.COBOL ROTATE.CBL
OK CBL ROTATE

[CBL Rev. 20.2 Copyright (c) Prime Computer, Inc. 1985]

ERROR 175  SEVERITY 1  LINE 7  COLUMN 8  [OBSERVATION, SEMANTICS]
COMPUTATIONAL items with no picture clause are assumed to be s9(4).

ERROR 175  SEVERITY 1  LINE 8  COLUMN 8  [OBSERVATION, SEMANTICS]
COMPUTATIONAL items with no picture clause are assumed to be s9(4).

[2 OBSERVATIONS IN PROGRAM: ROTATE.CBL]
OK BIND -LO ROTATE -LI VCOBLB -LI VAPPLB -LI

[BIND Rev. 20.2 Copyright (c) 1985, Prime Computer, Inc.]
BIND COMPLETE

OK R ROTATE
STRING1 = 45612
STRING1 = 12456
OK
```

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

RSUB$A

Purpose

RSUB$A is a logical function used to rotate a character substring left or right. Only the characters of the substring contained in string are affected. The parameters are checked for validity. If there is an error, a message is printed and the function will be .FALSE.. If no error occurs, the function will be .TRUE..

Usage

```
INTEGER*2  string(1), length, fchar, lchar, count
LOGICAL    log

LOG = RSUB$A(string, length, fchar, lchar, count)
        (or)
CALL  RSUB$A(string, length, fchar, lchar, count)
```

Parameters

string

INPUT/OUTPUT. String containing substring to be rotated, packed two characters per halfword. Data type does not matter.

length

INPUT. Length of string in characters.

fchar

INPUT. First delimiting character position of substring.

lchar

INPUT. Last delimiting character position of substring.

count

INPUT. Number of positions to rotate substring. A negative count causes left rotate, a positive count causes right rotate.

Discussion

This routine uses an algorithm that minimizes temporary storage and execution time. One word of temporary storage is used and the number of iterations necessary to rotate a string is equal to the length in characters of the string. A character is moved directly from its original position to its final destination position.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by RSUB$A

MCHR$A

SSTR$A

Purpose

SSTR$A is a logical function used to shift a character string left or right. The string is shifted the specified number of characters, and the vacated positions are padded with the specified fill character. Trailing blanks are not included in the shift. If length is less than 0, an error message is printed, the function is .FALSE., and no characters are shifted. If no error occurs, the function is .TRUE..

Usage

```
INTEGER*2  string(1), length, count, fil_ch(1)
LOGICAL    log

log = SSTR$A(string, length, count, fil_ch)
        (or)
CALL  SSTR$A(string, length, count, fil_ch)
```

Parameters

string

INPUT/OUTPUT. Character string to be shifted, packed two characters per halfword. Data type does not matter.

length

INPUT. Length of string in characters. Must be greater than or equal to 0.

count

INPUT. Number of positions to shift string. A negative count causes left shift, positive count causes right shift.

fil_ch

INPUT. Fill character which will pad the vacated positions. fil_chis specified in FORTRAN A1 format (two characters per halfword and blank-padded on the right). Data type does not matter.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by SSTR$A

FSUB$A, MCHR$A, and NLEN$A.

SSUB$A

Purpose

SSUB$A is a logical function used to shift a character substring left or right. The substring is shifted the specified number of characters and the vacated positions are padded with the specified fill character. Any trailing blanks are included in the shift. The parameters are checked for validity. An error causes a message to be printed and the function will be .FALSE.. If no error occurs, the function will be .TRUE.. (.TRUE. and .FALSE. are the FORTRAN logical values.) If the substring is null, or length is equal to 0, there will be no shift.

Usages

```
INTEGER*2  string(1), length, fchar, lchar, count, fil_ch
LOGICAL    log

log = SSUB$A(string, length, fchar, lchar, count, fil_ch)
        (or)
CALL  SSUB$A(string, length, fchar, lchar, count, fil_ch)
```

Parameters

string

INPUT/OUTPUT. String containing substring to be shifted, packed two characters per halfword. Data type does not matter.

length

INPUT. Length of string in characters.

fchar

INPUT. First delimiting character position of substring.

lchar

INPUT. Last delimiting character position of substring.

count

INPUT. Number of positions to shift substring. A negative count causes left shift, positive count causes right shift.

fil_ch

INPUT. Fill character with which to pad vacated positions. filchar is specified in A1 format (two characters per halfword and right-padded with blanks). Data type does not matter.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by SSUB$A

FSUB$A and MCHR$A.

TREE$A

Purpose

TREE$A checks a pathname for syntactical correctness.

Usage

```
INTEGER*2  name(1), namlen, fst, flen
LOGICAL    log

log = TREE$A(name, namlen, fst, flen)
        (or)
CALL TREE$A(name, namlen, fst, flen)
```

Parameters

name

INPUT. Array containing filename, packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of name in characters.

fst

OUTPUT. Number returned indicating the character position for the first character in the final name within name.

flen

OUTPUT. Length in characters of the final name within the filename name.

Discussion

TREE$A is a logical function that scans a filename and checks it for syntactical correctness. If the pathname is syntactically correct, the function is .TRUE. and if not, it is .FALSE.. In addition, the location of the final name (or entire name if not part of a pathname) can be determined from the values returned in fst and flen. Note that if the name is empty, fst and flen are both 0.

The following example illustrates the use of TREE$A from a CBL program. The program has already been submitted to BIND. Its runfile is located

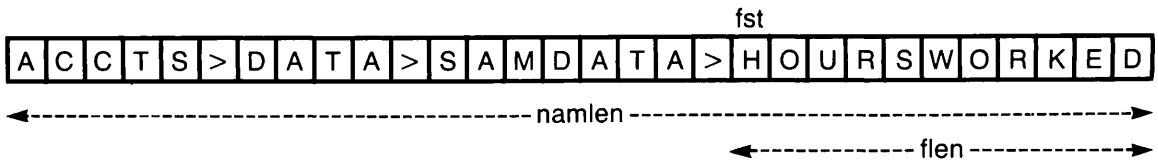
within the current subdirectory, along with the source file (slist TREE.CBL). It is now Resumed, and interactively executes as shown. Following the example, Figure 10-3 shows the data layout of the arguments to TREE$A.

Example

```
OK, SLIST TREE.CBL

        IDENTIFICATION DIVISION.
        PROGRAM-ID. TREE.
        ENVIRONMENT DIVISION.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01 NAME                        PIC X(32) VALUE SPACES.
        01 NAMLEN                      PIC S9(4) COMP.
        01 FSTART                      PIC S9(4) COMP.
        01 FLEN                        PIC S9(4) COMP.
        01 ASCIILEN                    PIC S99.
        PROCEDURE DIVISION.
        001-BEGIN.
            DISPLAY 'ENTER FILENAME'.
            ACCEPT NAME.
            DISPLAY 'ENTER LENGTH OF NAME'.
            ACCEPT ASCIILEN.
            MOVE ASCIILEN TO NAMLEN.
            CALL 'TREE$A' USING NAME, NAMLEN, FSTART, FLEN.
            EXHIBIT NAME.
            EXHIBIT NAMLEN.
            EXHIBIT FSTART.
            EXHIBIT FLEN.
            STOP RUN.

OK, R TREE.RUN
ENTER FILENAME
ACCTS>DATA>SAMDATA>HOURSWORKED
ENTER LENGTH OF NAME
30
NAME = ACCTS>DATA>SAMDATA>HOURSWORKED
NAMLEN =        30
FSTART =        20
FLEN =         11
OK,
```



Arguments to TREE$A
Figure 10-3

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by TREE$A

GCHR$A and NLEN$A.

TYPE$A

Purpose

TYPE$A is a logical function that tests a character string to determine if it can be interpreted as the type specified by key.

Usage

```
INTEGER*2  key, string(1), length
LOGICAL    log

log = TYPE$A(key, string, length)
        (or)
CALL TYPE$A(key, string, length)
```

Parameters

key

INPUT. Indicates the type of test that String will undergo. Possible keys are:

A$NAME	Can string be interpreted as a name?
A$BIN	Can string be interpreted as a binary number?
A$DEC	Can string be interpreted as a decimal number?
A$OCT	Can string be interpreted as an octal number?
A$HEX	Can string be interpreted as a hexadecimal number?

string

INPUT. The string to be tested, packed two characters per halfword. Data type does not matter.

length

INPUT. Length of string, in characters.

Discussion

A string is interpreted as a name if it contains at least one alphabetic or special character other than a leading plus or minus; a binary number if it contains only the digits 0 through 1; a decimal

number if it contains only the digits 0 through 9. It is an octal number if it contains only the digits 0 through 7, and is hexadecimal if it contains only the digits 0 through 9 and the characters A through F (uppercase only). A number may have a leading sign and any number of blanks between the sign and the first digit. However, embedded blanks within the number itself are not allowed. A number must also have at least one digit.

Leading and trailing blanks are ignored. The function is .TRUE. if string satisfies the conditions required by the key used; otherwise it is .FALSE.. A null string (length equal to 0) will return a function value of .TRUE. only if key is A$NAME.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by TYPE$A

GCHR$A and NLEN$A.

11
User Query Routines

SUMMARY OF USER QUERY ROUTINES

This chapter describes the following User Query Routines, found in the APPLICATION subroutines library.

RNAM$A	Prompt and read a name.
RNUM$A	Prompt and read a number (binary, decimal, octal, or hexadecimal). INTEGER*4
YSNO$A	Ask question and obtain a YES or NO answer.

RNAM$A

Purpose

RNAM$A is a logical function that prints the supplied message prompt and appends a colon (:) to it. It then reads a user response from the command stream. If the response is not a legal name, or if the name provided is too long for the supplied buffer, an error message will be typed and the message prompt will be repeated. If no name is provided, or if name contains illegal values (such as a digit or a plus or minus sign), the value of the function will be .FALSE.. If a legal name is provided, the function value will be .TRUE.. The caller should be aware that COMANL and RDTK$$ (Volume II) are called to read the user response, and therefore the previous command line entered is unavailable.

Usage

```
INTEGER*2  msg(1), msglen, namkey
INTEGER*2  name(1), namlen
LOGICAL    log

log = RNAM$A(msg, msglen, namkey, name, namlen)
        (or)
CALL  RNAM$A(msg, msglen, namkey, name, namlen)
```

Parameters

msg

INPUT. Message text, packed two characters per halfword. Data type does not matter.

msglen

INPUT. Message length in characters.

namkey

INPUT. Indicates options for character handling. Keys cannot be combined. Valid keys are:

A$FUPP Force uppercase.

A$UPLW Do not force uppercase.

A$RAWI Read line as raw uninterpreted text.

name

OUTPUT. Returned name, packed two characters per halfword. Data type is ASCII. It must begin with a non-character that is also not a plus or a minus sign.

namlen

INPUT. Length of name buffer in characters (maximum 80).

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

RNUM$A

Purpose

RNUM$A is a logical function used to accept numeric data from the user terminal.

Usage

```
INTEGER*2  msg(1), msglen, numkey
INTEGER*4  value
LOGICAL    log

log = RNUM$A(msg, msglen, numkey, value)
        (or)
CALL  RNUM$A(msg, msglen, numkey, value)
```

Parameters

msg

INPUT. Message text, packed two characters per halfword. Data type does not matter.

msglen

INPUT. Message length in characters.

numkey

INPUT. Indicates the data type to be verified. Valid keys are:

A$DEC	Decimal
A$BIN	Binary
A$OCT	Octal
A$HEX	Hexadecimal

value

OUTPUT. Returned value.

Discussion

The routine prints the user-supplied message and appends the colon (:) to it. It then reads a user response and if the response is not a legal number or if the number provided has too many digits for an INTEGER*4 value, the error will be reported and the message will be repeated. If no number is provided, the value of the function will be .FALSE. and value will be 0. If a legal number is provided, the function will be .TRUE. and the value will be returned in value.

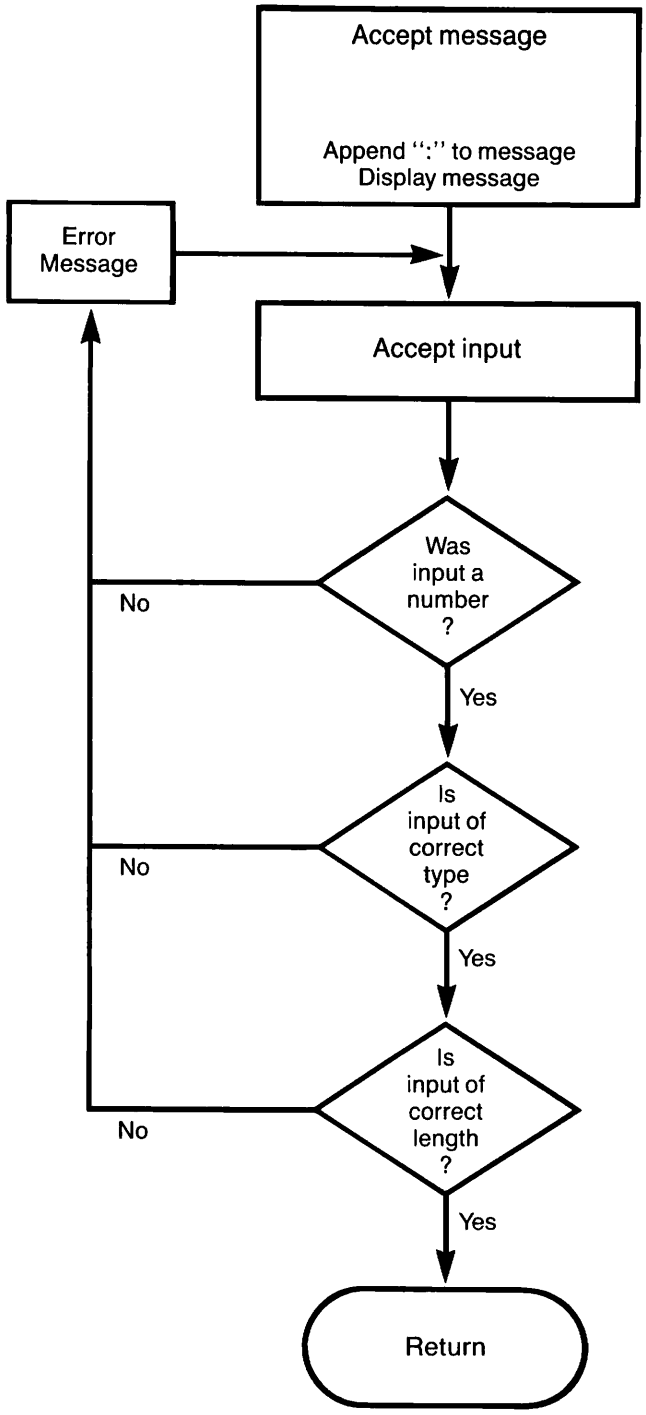
Numbers may be immediately preceded by "+" or "-". Binary numbers may have a maximum of 31 digits, octal a maximum of 11 digits, decimal a maximum of 10 digits, and hexadecimal a maximum of 8 digits. Negative binary, octal, or hexadecimal should not be entered in two's complement, but the same as a negative decimal number.

The caller should be aware that COMANL and RDTK$$ (see Volume II) are called to read the user response, and therefore the previous command line is unavailable.

The operation of this subroutine is shown in Figure 11-1.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode



How RNUM$A Works
Figure 11-1

YSNO$A

Purpose

YSNO$A is a logical function that prints the supplied message and appends the character "?" to it. It then reads a user response. If the answer is "YES" or "OK", the function returns .TRUE.. If the answer is "NO", the function value returns .FALSE.. If an illegal answer is provided or if no default is accepted, the message is repeated. User responses may be abbreviated to the first one or two characters.

Usage

```
INTEGER*2  msg(1), msglen, defkey
LOGICAL    log

log = YSNO$A(msg, msglen, defkey)

CALL  YSNO$A(msg, msglen, defkey)
```

Parameters

msg

INPUT. Message text, packed two characters per halfword. Data type does not matter.

msglen

INPUT. Message length in characters.

defkey

INPUT. A key specifying the default. Valid keys are:

A$NDEF	No default accepted.
A$DNO	Default is "NO".
A$DYES	Default is "YES".

Example

```
OK, SLIST YESNO1.PASCAL
program main;
{                                                                    }
{ FORTRAN logicals are incompatible with Pascal boolean data types.}
{ Therefore, interfacing to the applications library from Pascal    }
{ can be a problem. The following program shows the easiest way to }
{ determine True and False when calling FORTRAN subroutines with    }
{ logicals.                                                         }
{                                                                    }
{ Note: This program assumes that the type of logical returned is   }
{       a LOGICAL*2, and only occupies two bytes of memory.          }
{                                                                    }
const
%INCLUDE 'SYSCOM>A$KEYS.INS.PASCAL';

type
  msgtype = packed array[1..8] of char;

var
  msg    : msgtype;
  msglen : integer;

function ysno$a(var s : msgtype; {Pass by ref, msg                  }
                    l : integer; {Pass by value, length of msg      }
                    k : integer) {Pass by value, default keys       }
              :integer; extern; {Returns FORTRAN logical as integer}

begin
  writeln;
  msg := 'Yes | No';
  msglen := 8;
  if ysno$a(msg, msglen, a$ndef) = ord(true) then
    writeln('Ok!')
  else
    writeln('Absolutely NO!')
end.
```

This program, stored as YESNO1.PASCAL, may be compiled, loaded, and executed with the following dialogue.

```
OK, PASCAL YESNO1
[PASCAL Rev. 20.2.B2 Copyright (c) 1986, Prime Computer, Inc.]
0000 ERRORS [PASCAL Rev. 20.2]
OK, BIND
[BIND Rev. 20.2 Copyright (c) 1985, Prime Computer, Inc.]
: LO YESNO1
: LI PASLIB
: LI VAPPLB
: LI
BIND COMPLETE
: FILE
OK, RESUME YESNO1

Yes | No? YES
Ok!
OK, R YESNO1

Yes | No? NO
Absolutely NO!
OK,
```

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

12
System Information Routines

SUMMARY OF SYSTEM INFORMATION ROUTINES

This chapter describes the following System Information Routines, found in the APPLICATION subroutines library.

CTIM$A	CPU time since login.
DATE$A	Today's date, American style.
DOFY$A	Today's date as day of year ("Julian" date).
DTIM$A	Disk time since login.
EDAT$A	Today's date, European (military) style.
TIME$A	Time of day.

CTIM$A

Purpose

CTIM$A is a double precision function that returns CPU time elapsed since login, in seconds as the function value, and as centiseconds in the cputim argument.

Usage

```
INTEGER*4  cputim
REAL*8     rt_val

rt_val = CTIM$A(cputim)
        (or)
CALL  CTIM$A(cputim)
```

Parameters

cputim

OUTPUT. CPU time in centiseconds.

Discussion

The function value will be CPU time elapsed since login, in seconds. This value may be received as REAL*8.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

DATE$A

Purpose

DATE$A is a double-precision function that returns the date in the argument date in the form "DAY, MON DD YYYY" (for example, TUE, FEB 23 1982).

The value of the function is the date in the form "MM/DD/YY" (for example, 02/23/82). This value must be received as REAL*8.

Note that this routine is good for the period January 1, 1977 through December 31, 2076.

Usage

```
INTEGER*2  date(16)
REAL*8     rt_val

rt_val = DATE$A(date)
         (or)
CALL  DATE$A(date)
```

Parameters

date

OUTPUT. Date in the form DAY, MON DD YYYY. The data type does not matter as long as it is at least 16 characters long.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

DOFY$A

Purpose

DOFY$A is a double-precision function that returns the day of the year in the form "DDD" in the dofy argument. The value of the function is the date in the form YR.DDD, suitable for printing in FORMAT F6.3. This value can be received as either REAL*4 or REAL*8. This routine is good for the period January 1, 1977 through December 31, 2076.

Usage

```
      INTEGER*2  dofy(1)
      REAL*8     rt_val
C   rt_val may also be declared REAL

      rt_val  = DOFY$A(dofy)
              (or)
      CALL  DOFY$A(dofy)
```

Parameters

dofy

OUTPUT. Day of year in the form "DDD" ("Julian" date). The data type does not matter as long as it is at least four characters long.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

DTIM$A

Purpose

DTIM$A is a double-precision function that returns disk time since login as centiseconds is the dsktim argument. The function value will be disk time since login in seconds. This value may be received as either REAL*4 or REAL*8.

Usage

```
      INTEGER*4  dsktim
      REAL*8     rt_val
C  rt_val may also be declared REAL*4

      rt_val = DTIM$A(dsktim)
             (or)
      CALL  DTIM$A(dsktim)
```

Parameters

dsktim

OUTPUT. Disk time in centiseconds.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

EDAT$A

Purpose

EDAT$A is a double-precision function. It returns the date in the European (military) form 'DAY, DD MON YEAR' in the argument edate (for example, TUE, 23 FEB 1982).

The value of the function is the date in the form DD.MM.YY (for example, 23.03.82). This value must be received in a REAL*8 variable. The routine is good for the period January 1, 1977 through December 31, 2076.

Usage

```
INTEGER*2  edate(16)
REAL*8     rt_val

rt_val = EDAT$A(edate)
         (or)
CALL  EDAT$A(edate)
```

Parameters

edate

OUTPUT. Date in the form "DAY, DD MON YEAR".

Discussion

The data type of the edate array does not matter as long as it is at least 16 characters long.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

Other Routines Called by EDAT$A

DATE$A

TIME$A

Purpose

TIME$A is a double-precision function that returns the time of day in the form HR:MN:SC. The value of the function is the time of day in decimal hours. This value may be received as either REAL*4 or REAL*8.

Usage

```
INTEGER*2  time(8)
REAL*8     rt_val

rt_val = TIME$A(time)
        (or)
CALL  TIME$A(time)
```

Parameters

time

OUTPUT. Time of day in the form HH:MM:SS, packed two characters per halfword. Data type does not matter as long as it is at least eight characters long.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

13
Randomizing Routines

SUMMARY OF RANDOMIZING ROUTINES

This chapter describes the following two Randomizing Routines, found in the APPLICATION subroutines library.

RAND$A	Generate random number and update "seed," based upon a 32-bit word size and using the Linear Congruential Method.
RNDI$A	Initialize random number generator "seed."

RAND$A

Purpose

RAND$A is a random-number generator.

Usage

```
      INTEGER*4  seed
      REAL*8     rt_val
C   rt_val may also be declared REAL*4

      rt_val =  RAND$A(seed)
              (or)
      CALL  RAND$A(seed)
```

Parameters

seed

INPUT/OUTPUT. Input is previous seed, output is new seed.

Discussion

RAND$A is a double-precision function that updates a seed to a new seed based upon the following linear congruential method:

```
U(I)=FLOAT(K(I))/M

    K(I)   B*K(I-1) modulo M

    B      16807.0

    M      2**31-1 = 2147483647.0
```

B and M are from Lewis, Goodman, and Miller, "A Pseudo-random Number Generator for the System/360," IBM Systems Journal, vol. 8, no. 2, 1969, pp. 136-145.

K(I-1) is the input value of seed and K(I) is the returned value.

The value of the function is U(I) which represents a probability and is between 0.0 and 1.0. This value may be received as either REAL*4 or REAL*8.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

RNDI$A

Purpose

Initialize random number generator seed.

Usage

```
      INTEGER*4  seed
      REAL*8     rt_val
C  rt_val may be declared REAL

      rt_val  =  RNDI$A(seed)
              (or)
      CALL  RNDI$A(seed)
```

Parameters

seed

OUTPUT. The time of day. The granularity of the returned time of day value varies from system to system.

Discussion

RNDI$A is a double-precision function that is used to initialize a random number generator. The function value is the time of day in seconds. This value may be received as either REAL*4 or REAL*8. If the function value is exactly 0, 1234567 and 12345.67 will be returned instead.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

14
Conversion Routines

SUMMARY OF CONVERSION ROUTINES

This chapter describes the following Conversion Routines, found in the APPLICATION subroutines library.

CASE$A	Convert a string from lowercase to uppercase or uppercase to lowercase.
CNVA$A	Convert ASCII number to binary.
CNVB$A	Convert binary number to ASCII.
ENCD$A	Make a number printable if possible.
FDAT$A	Convert the DATMOD field (as returned by RDEN$$) in format DAY, MON DD YYYY
FEDT$A	Convert the DATMOD field (as returned by RDEN$$) in format DAY, DD MON YYYY.
FTIM$A	Convert the TIMMOD field (as returned by RDEN$$).

CASE$A

Purpose

CASE$A is a logical function that converts a string from uppercase to lower, or from lowercase to upper. The function will be .FALSE. if length is less than 0, otherwise .TRUE..

Usage

```
INTEGER*2  key, string(1), length
LOGICAL    log

log  =  CASE$A(key, string, length)
      (or)
CALL  CASE$A(key, string, length)
```

Parameters

key

INPUT. Indicates the desired conversion option. Valid keys are:

A$FUPP Convert all alphabetic characters in string from lowercase to uppercase.

A$FLOW Convert all alphabetic characters in string from uppercase to lowercase.

string

INPUT/OUTPUT. Array containing character string to be converted, packed two characters per halfword, any data type.

length

INPUT. Length of string in characters.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by CASE$A

GCHR$A and MCHR$A

CNVA$A

Purpose

CNVA$A is a logical function that converts an ASCII digit string into its binary value for decimal, octal, and hexadecimal numbers. The numbers may be explicitly signed. Leading and trailing blanks are ignored, as well as blanks between the sign and the number. However, blanks within the number are not allowed. If the number converts successfully, the function is .TRUE. and value is the converted binary value. If conversion, is not successful, the function is .FALSE. and value is 0. Note that for decimal conversions overflow will be considered as unsuccessful, whereas for octal and hexadecimal conversions overflow is ignored.

(.TRUE. and .FALSE. are FORTRAN logical values.)

Usage

```
INTEGER*2  numkey, name(1), namlen
INTEGER*4  value
LOGICAL    log

log = CNVA$A(numkey, name, namlen, value)
          (or)
CALL  CNVA$A(numkey, name, namlen, value)
```

Parameters

numkey

INPUT. Specifies data type of number to be converted. Possible values are:

A$DEC	Decimal
A$BIN	Binary
A$OCT	Octal
A$HEX	Hexadecimal

name

INPUT. Array containing ASCII digit string, packed two characters per halfword. Maximum lengths for the input string's original data type are: binary, 31; octal, 11; decimal, 10; hexadecimal, 8. Maximum does not include leading signs or blanks.

namlen

INPUT. Length of name in characters.

value

OUTPUT. Returned converted binary value.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by CNVA$A

GCHR$A and NLEN$A

CNVB$A

Purpose

CNVB$A is an INTEGER*2 function used to convert a binary number to an ASCII digit string.

Usage

```
INTEGER*2  numkey, name(1), namlen
INTEGER*4  value
INTEGER*2  rt_val

rt_val = CNVB$A(numkey, value, name, namlen)
        (or)
CALL  CNVB$A(numkey, value, name, namlen)
```

Parameters

numkey

INPUT. Number base to which value is converted. Valid keys are:

A$BIN	Binary number with leading blanks
A$BINZ	Binary number with leading 0s
A$DEC	Signed decimal number with leading blanks
A$DECU	Unsigned decimal number with leading blanks
A$DECZ	Signed decimal number with leading 0s
A$OCT	Octal number, leading blanks
A$OCTZ	Octal number, leading 0s
A$HEX	Hexadecimal, leading blanks
A$HEXZ	Hexadecimal, leading 0s

value

INPUT. Binary number to be converted.

name

OUTPUT. Array containing returned ASCII digit string packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of name in characters. Maximum length for binary is 31, octal is 11, decimal is 10, and hexadecimal is 8. Maximum does not include leading signs or 0s.

Discussion

CNVB$A converts a binary number into an ASCII digit string for decimal, octal, and hexadecimal numbers. The returned digit string is right-justified in name and preceded by leading blanks or 0s depending upon numkey specification.

If value is negative and the number is to be treated as signed decimal, the digit will begin with an initial minus sign. If value is negative, binary, octal, and hexadecimal numbers will be in two's-complement form. If the number converts successfully, the function value is the number of digits and if not, it is 0.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by CNVB$A

FILL$A and MCHR$A

ENCD$A

Purpose

ENCD$A is a logical function that converts a numeric value to a FORTRAN format.

Usage

```
INTEGER*2  array(1), width, dec
REAL*8     value
LOGICAL    log

log = ENCD$A(array, width, dec, value)
        (or)
CALL  ENCD$A(array, width, dec, value)
```

Parameters

array

OUTPUT. Array to receive value, packed two characters per halfword. Data type does not matter.

width

INPUT. Field width as in format Fw.d (should be even).

dec

INPUT. Places to right of decimal point as shown in format Fw.d.

value

INPUT. Double-precision value to be encoded (REAL*8).

Discussion

ENCD$A attempts to encode value in the supplied Fw.d format if it will fit. If not, the dec argument is decremented (moving the decimal point to the right) until it will fit. If dec reaches 0, or is originally supplied as 0, value will be encoded in Iw format if the number will fit into a 32-bit integer. If not, and if the field is wide enough (width > 7), the value will be encoded in E format. If the field is not wide enough, it will be filled with asterisks.

The formats are:

F	A number that includes a decimal fraction. The d is the number of digits after the decimal point, and w is the total number of positions (including the decimal point) in the field. The maximum is 32767.
I	An integer, with w digits. Maximum is 32767.
E	A floating point number in scientific format (xxE+yy), where xx represents the characteristic and yy is the mantissa or exponent.

Examples are:

Fw.d: 123.4

I: 12345

E: 1.23456E+99

Note that the largest value of width is 16. If it is larger than 16, only the first 16 characters of array are used.

The function returns .TRUE. if the encoding was successful, and .FALSE. if the field was filled with asterisks. Note that array is the only argument that is actually modified in the calling program.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

FDAT$A

Purpose

FDAT$A is a REAL*8 function that converts the datmod field, returned as halfword 20 of buffer by RDEN$$, to the format DAY, MON DD YYYY (for example, TUE, FEB 23 1982).

The function value is the datmod field converted to MM/DD/YY (for example, 02/23/82). It must be received in a REAL*8 variable. The routine is good for the period January 1, 1972 to December 31, 2071.

RDEN$$ must be called before this subroutine. Since RDEN$$ is considered obsolete, this subroutine has limited use.

Usage

```
INTEGER*2  datmod, date(16)
REAL*8     rt_val

rt_val = FDAT$A(datmod, date)
        (or)
CALL  FDAT$A(datmod, date)
```

Parameters

datmod

INPUT. Date returned by RDEN$$. This is the date the file was last modified and is in the format YYYYYYYMMMMDDDDD. YYYYYYY is the year modulo 100, MMMM is the month, and DDDDD is the day.

date

OUTPUT. Array containing the date as a character string, packed two characters per halfword. Date is in the DAY, MON DD YEAR format. Data type does not matter as long as the array is at least 16 characters long.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by FDAT$A

CNVB$A

FEDT$A

Purpose

FEDT$A converts the datmod field, returned as halfword 20 of buffer by RDEN$$, to the DAY, DD MON YEAR format in date (for example, TUE, 23 FEB 1982). The function value is datmod converted to a DD.MM.YY format (for example, 23.02.82). It must be received in a REAL*8 variable. The routine includes the period January 1, 1972 through December 31, 2071.

RDEN$$ must be called before this subroutine. Since RDEN$$ is considered obsolete, this subroutine has limited use.

Usage

```
INTEGER*2  datmod, date(16)
REAL*8     rt_val

rt_val = FEDT$A(datmod, date)
         (or)
CALL  FEDT$A(datmod, date)
```

Parameters

datmod

INPUT. Date returned by RDEN$$. This is the date that the file was last modified and is in the format YYYYYYYMMMMDDDDD. YYYYYYY is the year modulo 100, MMMM is the month, and DDDDD is the day.

date

OUTPUT. Array containing the date as a character string, packed two characters per halfword. Date is in the 'DAY, DD MON YEAR format. Data type does not matter as long as the array is at least 16 characters long.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by FEDT$A

FDAT$A

FTIM$A

Purpose

FTIM$A is a REAL*4 or REAL*8 function that converts the timmod field, returned as halfword 21 of buffer by RDEN$$, to the HH:MM:SS format. The function value is the timmod field converted to decimal hours and may be received as either REAL*4 or REAL*8.

Usage

```
      INTEGER*2  timmod, time(8)
      REAL*8     rt_val
C  rt_val may also be declared REAL

      rt_val = FTIM$A(timmod, time)
             (or)
      CALL  FTIM$A(timmod, time)
```

Parameters

timmod

INPUT. Time at which a file was last modified, formatted as 'seconds since midnight' divided by four.

time

OUTPUT. Array containing the time a file was last modified as a character string in the format 'HH:MM:SS'. Data type does not matter as long as array is at least eight characters long.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

Other Routines Called by FTIM$A

CNVB$A

15
File System Routines

SUMMARY OF FILE SYSTEM ROUTINES

This chapter describes the following File System Routines, found in the APPLICATION subroutines library.

CLOS$A	Close a file.
DELE$A	Delete a file.
EXST$A	Check for file existence.
GEND$A	Position to end-of-file.
OPEN$A	Open supplied name.
OPNP$A	Read name and open.
OPNV$A	Open supplied name with verification and delay.
OPVP$A	Read name and open with verification and delay.
POSN$A	Position file.
RPOS$A	Return position of file.
RWND$A	Rewind file.
TEMP$A	Open a scratch file with unique name.
TRNC$A	Truncate file.
TSCN$A	Scan the file system structure.
UNIT$A	Check for file open.

CLOS$A

Purpose

CLOS$A is a logical function that closes the file open on funit. If the operation is successful, the function is .TRUE.; otherwise, the function is .FALSE..

Usage

```
      INTEGER*2  funit
      LOGICAL*2  log

      log = CLOS$A(funit)
              (or)
      CALL  CLOS$A(funit)
```

Parameters

funit

INPUT. File unit to be closed.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

DELE$A

Purpose

DELE$A is a logical function that deletes the file named in name. If the operation is successful, the function is .TRUE.; otherwise the function is .FALSE..

Usage

```
INTEGER*2  name(1), namlen
LOGICAL*2  log

log = DELE$A(name, namlen)
        (or)
CALL  DELE$A(name, namlen)
```

Parameters

name

INPUT. Filename (may be a pathname) packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of name in characters.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

Other Routines called by DELE$A

TREE$A, UNIT$A, NLEN$A

EXST$A

Purpose

EXST$A is a logical function that returns .TRUE. if the file exists and .FALSE. if the file does not exist or if an error was encountered.

Usage

```
      INTEGER*2  name(1), namlen
      LOGICAL*2  log

      log = EXST$A(name, namlen)

      CALL  EXST$A(name, namlen)
```

Parameters

name

INPUT. Filename (may be a pathname) packed two characters pe halfword. Data type does not matter.

namlen

INPUT. Length of name in characters.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

Other Routines Called by EXST$A

TREE$A, UNIT$A, NLEN$A.

GEND$A

Purpose

GEND$A is a logical function that positions the file open on funit to end-of-file. If the operation is successful, the function is .TRUE., otherwise, the function is .FALSE..

Usage

```
INTEGER*2  funit
LOGICAL*2  log

LOG = GEND$A(funit)
        (or)
CALL  GEND$A(funit)
```

Parameters

funit

INPUT. PRIMOS file unit whose file is acted upon.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

OPEN$A

Purpose

OPEN$A is a logical function that opens a file of the given name on funit. If the operation is successful, the function value is .TRUE., and if the operation is unsuccessful, the function value is .FALSE..

Usage

```
INTEGER*2  opnkey, typkey, untkey
INTEGER*2  name(1), namlen, funit
LOGICAL*2  log

log = OPEN$A(opnkey+typkey+untkey, name, namlen, funit)
        (or)
CALL  OPEN$A(opnkey+typkey+untkey, name, namlen, funit)
```

Parameters

opnkey

INPUT. Indicates the desired operation. Valid Keys are:

A$READ	Open for reading.
A$WRIT	Open for writing.
A$RDWR	Open for reading and writing.

typkey

INPUT. Indicates the desired file type. Valid Keys are:

A$SAMF	SAM file
A$DAMF	DAM file
A$CAMF	CAM file

untkey

INPUT. Indicates how funit is to be handled. Key is:

A$GETU	Choose a PRIMOS file unit number to be returned in funit. Omission of this key requires user input of a legal file unit number in funit.

name

INPUT. File name (or pathname) packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of name in characters.

funit

INPUT/OUTPUT. PRIMOS file unit returned. While always an output, funit must also input a legal file unit number if A$GETU is not specified in untkey.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by OPEN$A

TREE$A, UNIT$A, and NLEN$A.

OPNP$A

Purpose

OPNP$A is a logical function that gets a name from the user and opens it on funit. If the operation is successful, the function value is .TRUE. and if the operation is unsuccessful or no name is supplied, the function value is .FALSE..

Usage

```
      INTEGER*2  msg(1), msglen, opnkey, typkey, untkey
      INTEGER*2  name(1), namlen, funit
      LOGICAL*2  log

      log = OPNP$A(msg, msglen, opnkey+typkey+untkey, name, namlen,
     x             funit)
               (or)
      CALL  OPNP$A(msg, msglen, opnkey+typkey+untkey, name, namlen,
     x             funit)
```

Parameters

msg

OUTPUT. Array containing prompt for name message, packed two characters per halfword. Data type does not matter.

msglen

INPUT. Length of msg in characters.

opnkey

INPUT. Indicates the desired operation. Key values may be:

A$READ Open for reading.

A$WRIT Open for writing.

A$RDWR Open for reading and writing.

typkey

INPUT. Indicates the type of file. Key values may be:

A$SAMF SAM file

A$DAMF DAM file

untkey

INPUT. Indicates how funit is to be used. Key is:

A$GETU Choose a PRIMOS file unit number to be returned in funit. Omission of this key requires that the caller input a unit number in funit.

name

OUTPUT/INPUT. Filename (or pathname) packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of name in characters.

funit

INPUT/OUTPUT. PRIMOS file unit returned. While always an output, funit must also input a legal file unit number if A$GETU is not specified in untkey.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by OPNP$A

RNAM$A, NLEN$A, TREE$A, and UNIT$A.

OPNV$A

Purpose

OPNV$A is a logical function that opens a file of the given name on funit. Note that the functions of verification and delay as described here are different from those in the File System Subroutines in Volume II.

Usage

```
      INTEGER*2  opnkey, typkey, untkey
      INTEGER*2  name(1), namlen, funit
      INTEGER*2  verkey, wtime, retry
      LOGICAL*2  log

      log = OPNV$A(opnkey+typkey+untkey, name, namlen, funit,
     x             verkey, wtime, retry)
               (or)
      CALL  OPNV$A(opnkey+typkey+untkey, name, namlen, funit,
     x             verkey, wtime, retry)
```

Parameters

opnkey

INPUT. Indicates the desired operation. Valid keys are:

A$READ Open for reading.

A$WRIT Open for writing.

A$RDWR Open for reading and writing.

typkey

INPUT. Indicates the type of file. Valid keys are:

A$SAMF SAM file

A$DAMF DAM file

untkey

INPUT. Indicates how to handle funit. Key is:

A$GETU Choose a PRIMOS file unit number to be returned in funit. Omission of this key requires that the caller input a valid unit number in funit.

name

INPUT. Filename (may be a pathname) packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of name in characters. If namlen is 0 or less, the function value is .FALSE..

funit

INPUT/OUTPUT. PRIMOS file unit returned. While always an output, funit must also input a legal file unit number if A$GETU is not specified in untkey.

verkey

INPUT. Indicates type of verification procedure to follow during execution of this routine. Valid keys are:

A$NVER	No verification.
A$VNEW	Verify new or ask if OK to modify old file.
A$OVAP	Same as A$VNEW except user is prompted to "OVERWRITE" or "APPEND" if file already exists.
A$VOLD	Verify old; return .FALSE. if not old file.

wtime

INPUT. Number of seconds to wait if FILE IN USE.

retry

INPUT. Number of times to retry if FILE IN USE.

Discussion

If wtime and retry are specified as nonzero, and the file to be opened is IN USE, the open is retried the specified number of times, with wtime seconds (elapsed time) between each attempt. If the number of retries expires, or if either wtime or retry is initially 0 and the file is IN USE, the function returns .FALSE..

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by OPNV$A

RNAM$A, TIME$A, NLEN$A, EXST$A, UNIT$A, TREE$A, and GEND$A.

Verification

If verification is not requested (verkey = A$NVER), OPNV$A is identical in function to OPEN$A. If verification is requested (verkey other than A$NVER), the following actions will be taken according to the value of verkey:

A$VNEW	If the file already exists and opnkey is either A$WRIT or A$RDWR, the user is asked if it is OK to modify the old file. If the answer is "NO", the function returns .FALSE.. If the answer is "YES", the file is opened.
A$OVAP	This is the same as A$VNEW except that if an old file is to be modified, the user is also asked if the file should be overwritten or appended. If the answer is "APPEND", the file is positioned to end of file.
A$VOLD	This is the default case if opnkey = A$READ. If any other key is specified, and if the named file does not already exist, a new file is not created and the function returns .FALSE..

Errors

If any errors not covered above occur while opening the file or positioning it (A$OVAP), the function returns .FALSE.. If the open is ultimately successful, the function returns .TRUE..

OPVP$A

Purpose

OPVP$A is a logical function that gets a filename from the user and opens it on funit. Note that the functions of verification and delay as described below perform differently from the File System Subroutines in Volume II.

Usage

```
      INTEGER*2  msg(1), msglen, opnkey, typkey, untkey
      INTEGER*2  name(1), namlen, funit
      INTEGER*2  verkey, wtime, retry
      LOGICAL*2  log

      log = OPVP$A(msg, msglen, opnkey+typkey+untkey, name(1),
     x             namlen, funit, verkey, wtime, retry)
               (or)
      CALL  OPVP$A(msg, msglen, opnkey+typkey+untkey, name(1),
     x             namlen, funit, verkey, wtime, retry)
```

Parameters

msg

INPUT. Array containing prompt message, packed two characters per halfword. Data type does not matter.

msglen

INPUT. Length of msg in characters.

opnkey

INPUT. Indicates desired operation. Valid keys are:

A$READ	Open for reading.
A$WRIT	Open for writing.
A$RDWR	Open for reading and writing.

typkey

Indicates type of file being accessed. Valid keys are:

A$SAMF	SAM file
A$DAMF	DAM file

untkey

INPUT. Indicates how to handle funit. Key is:

A$GETU Choose a file unit number to be returned in funit. Omission of this key requires the routine to input a valid file unit number in funit.

name

OUTPUT. Array containing filename (may be pathname), packed two characters per halfword. Data type does not matter.

namlen

INPUT. Length of name in characters. If namlen is 0 or less, the function value is .FALSE..

funit

INPUT/OUTPUT. Primos file unit returned. While always an output, funit must also input a legal file unit number if A$GETU is not specified in untkey.

verkey

INPUT. Indicates the verification option desired. Valid keys are:

A$NVER No verification.

A$VNEW Verify new file or ask if OK to modify old file.

A$OVAP Same as A$VNEW except user is prompted to "OVERWRITE" or "APPEND" if file already exists.

A$VOLD Verify old. Function value is .FALSE. if not old.

wtime

INPUT. Number of seconds to wait if FILE IN USE.

retry

INPUT. Number of times to retry if FILE IN USE.

Discussion

If wtime and retry are specified as nonzero, and the file to be opened is IN USE, the open will be retried the specified number of times, with wtime seconds (elapsed time) between attempts. If the number or retries expires, or if either wtime or retry is initially 0 and the file is in use, the function returns .FALSE..

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by OPVP$A

RNAM$A, TIME$A, NLEN$A, EXST$A, UNIT$A, TREE$A, and GEND$A.

Verification

If verification is requested, the following are the possible actions, according to the value of verkey:

A$VNEW	If the file already exists and opnkey is A$WRIT or A$RDR, the user will be asked if it is OK to modify the old file. If the answer is "NO", the function returns .FALSE.. If "YES", the file is opened.
A$OVAP	If an old file is to be modified (as answered "YES" for A$VNEW), the user is also asked if the file should be overwritten or appended. If the answer is "APPEND", the file will be positioned to end of file.
A$VOLD	Default case if opnkey = A$READ. If any other key is specified, and if the named file does not already exist, a new file will not be created and the prompt message will be repeated.

Errors

If any errors not covered above occur while opening the file or positioning it (A$OVAP), or a name is not supplied when requested, the function returns .FALSE.. If the open is ultimately successful, the function returns .TRUE..

POSN$A

Purpose

POSN$A is a logical function that positions the file open on funit to the specified position. If the operation is successful, the function is .TRUE., and if unsuccessful, the function is .FALSE..

Usage

```
INTEGER*2  poskey, funit
INTEGER*4  pos
LOGICAL*2  log

log = POSN$A(poskey, funit, pos)
       (or)
CALL  POSN$A(poskey, funit, pos)
```

Parameters

poskey

INPUT. Indicates the desired position. Valid keys are:

A$ABS Absolute position

A$REL Relative position

funit

INPUT. PRIMOS file unit to which the file is assigned.

pos

INPUT. The position (relative or absolute).

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

RPOS$A

Purpose

RPOS$A is a logical function that returns the current absolute position of the file open on funit. If the operation is successful, the function is .TRUE.; otherwise the function is .FALSE..

Usage

```
INTEGER*2  funit
INTEGER*4  pos
LOGICAL*2  log

log = RPOS$A(funit, pos)
        (or)
CALL  RPOS$A(funit, pos)
```

Parameters

funit

INPUT. PRIMOS file unit opened on the file being queried.

pos

OUTPUT. Returned absolute position.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

RWND$A

Purpose

RWND$A is a logical function that rewinds the file open on funit. If the operation is successful, the function is .TRUE. Otherwise the function is .FALSE..

Usage

```
      INTEGER*2  funit
      LOGICAL*2  log

      log = RWND$A(funit)
              (or)
      CALL  RWND$A(funit)
```

Parameters

funit

INPUT. PRIMOS file unit holding file to be rewound.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

TEMP$A

Purpose

This routine opens a unique temporary file in the current directory for reading and writing. This file will be named T$xxxx where xxxx is a four-digit decimal number between 0000 and 9999 inclusive. The actual name opened will be returned in the name buffer. If the operation is successful, the function value is .TRUE. and if the operation is unsuccessful, the function value is .FALSE..

Usage

```
      INTEGER*2  typkey, untkey, name(6), namlen, funit
      LOGICAL*2  log

      log = TEMP$A(typkey+untkey, name, namlen, funit)
              (or)
      CALL  TEMP$A(typkey+untkey, name, namlen, funit)
```

Parameters

typkey

INPUT. Indicates the file type to be loaded into the unique temporary file. Valid keys are:

A$SAMF SAM file

A$DAMF DAM file

untkey

INPUT. Indictes how to handle funit. Key is:

A$GETU Choose a file unit number to be returned in funit. If A$GETU is omitted, the caller must input a valid file unit number in funit.

name

OUTPUT. Returned name (six characters, packed two characters per halfword). Data type does not matter.

namlen

INPUT. Length of name buffer in characters (must be at least six).

funit

INPUT/OUTPUT. Indicates the file unit. While always given for output, it is only required for input if (A$GETU) has been omitted from untkey.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by TEMP$A

FILL$A

TRNC$A

Purpose

TRNC$A is a logical function that truncates the file open on funit. If the operation is successful, the function is .TRUE.; otherwise the function is .FALSE.

Usage

```
      INTEGER*2  funit
      LOGICAL*2  log

      log = TRNC$A(funit)
              (or)
      CALL  TRNC$A(funit)
```

Parameters

funit

INPUT. PRIMOS file unit holding the file to be truncated.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

TSCN$A

Purpose

TSCN$A is a logical function that scans the file system tree structure (starting with the home directory). It uses the file subroutines RDEN$$ and SGDR$$ to read directory and segment directory entries into the entry array.

Usage

```
      INTEGER*2  key, funits(1), entry(1,1)
      INTEGER*2  maxsiz, entsiz, maxlev, lev, code
      LOGICAL*2  log

      log = TSCN$A((key, funits, entry, maxsiz, entsiz, maxlev,
     x             lev, code)
                    (or)
      CALL TSCN$A(key, funits, entry, maxsiz, entsiz, maxlev,
     x             lev, code)
```

Parameters

key

INPUT. Indicates the desired scan. Valid keys are:

A$TREE	Scan full tree.
A$NUFD	Do not scan subdirectories.
A$NSEG	Do not scan segment directories.
A$CUFD	Scan current directory only.
A$DLAY	Pause when popping up to directory.

funits

INPUT. Array of unit numbers maxlev long.

entry

OUTPUT. Array maxsiz * maxlev long.

> **Caution**
>
> The above two-dimensional array may be passed from a FORTRAN program only.

maxsiz

INPUT. Size of each entry in entry array.

entsiz

INPUT. Set to size of current entry.

maxlev

INPUT. Maximum number of levels to scan.

lev

OUTPUT. Current level.

code

OUTPUT. Standard return code of 0 for success or one of the standard error codes.

Discussion

Each call to TSCN$A returns the next file on the current level or the first file on the next lower level of the structure. The variable lev is used to keep track of the current level. For example, after the first call to TSCN$A (with lev=0), lev will be returned as 1, and entry(1,1) will contain the directory entry describing the first file in the home directory. If this file is a subdirectory, following the next call to TSCN$A lev will be 2, and entry(1,2) will contain the entry for the first file in the subdirectory. Thus, for the directory represented in Figure 15-1, TSCN$A in a loop would return the names in the order shown in Figure 15-2.

The values of key have the following meanings:

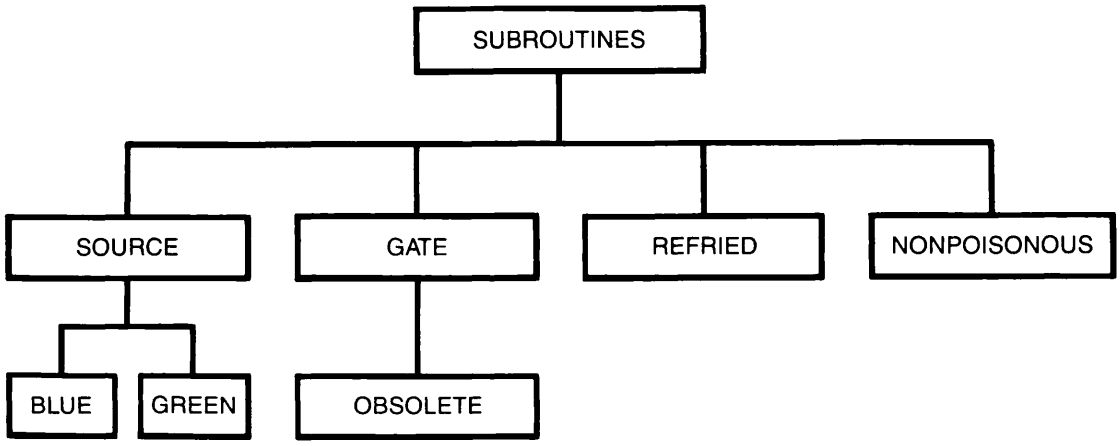
A$TREE	All entries in the directory structure are returned up to maxlev levels deep. (Levels below level maxlev are ignored.)

A$NUFD — When a subdirectory is encountered (in the home directory), its entry is returned, but no files under that subdirectory are returned. In the absence of segment directories, this effectively limits the scan to the home directory.

A$NSEG — Files inside segment directories are not returned.

A$CUFD — This is a logical combination of A$NUFD and A$NSEG -- only files in the home directory are returned.

A$DLAY — This key is identical to A$TREE except that directory entries are returned twice, once on the way down (as for A$TREE), and again on the way up. (This is necessary, for example, to implement a tree-delete function since a directory cannot be deleted until it has been emptied.)



A Directory to be Searched by TSCN$A
Figure 15-1

```
SOURCE
SOURCE > BLUE
SOURCE > GREEN
GATE
GATE > OBSOLETE
NONPOISONOUS
REFRIED
OK,
```

Result of TSCN$A Sample Program on Figure 15-1
Figure 15-2

The following items should be considered when using TSCN$A:

1. For the first call of TSCN$A, lev should be equal to 0. Thereafter it should not be modified until EOF is reached on the top level directory at which point lev will be reset to 0.

2. The entries in the entry array are in RDEN$$ format. For entries inside a segment directory, all information from the directory entry is first copied down a level. Entry(2,lev) is set to 0 and entry(3,lev) is then set to a 16-bit entry number. For nested segment directories, the type field of the entry is set appropriately by opening the file with SRCH$$. (The file is then immediately closed again.)

3. The parameter entsiz is set to the number of halfwords returned by RDEN$$. Inside segment directories, it should be ignored.

4. The type fields in the entry array -- entry(20,1) -- should not be modified. (TSCN$A uses them to walk up and down the tree.)

5. When TSCN$A requires a file unit, it uses units(lev). By using the RDEN$$ and SGDR$$ read-position and set-position functions carefully, it is possible to reuse file units dynamically.

6. TSCN$A returns .TRUE. until a non-file system code is returned or until E$EOF is returned with lev=0 (top level). E$EOF on lower levels of the structure is suppressed, and code is returned as 0.

7. TSCN$A requires owner rights in the home directory.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Example

The following FORTRAN program illustrates how TSCN$A can be used to perform a directory LISTF. The previous figures, 15-1 and 15-2, show the results of the program run in a sample directory.

```
$INSERT SYSCOM>ERRD.INS.FTN
$INSERT SYSCOM>KEYS.INS.FTN
$INSERT SYSCOM>A$KEYS.INS.FTN
C
      INTEGER MAXLEV,MAXSIZ
      PARAMETER (MAXLEV=16)  /* MAXIMUM LEVELS TO SCAN
      PARAMETER (MAXSIZ=24)  /* MAXIMUM SIZE OF EACH SLICE IN ENTRY
      INTEGER I,L,ENTRY(MAXSIZ,MAXLEV),UNITS(MAXLEV),CODE,NLEV$A
      LOGICAL TSCN$A
      DATA UNITS/1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16/
C
10    L=0                    /* INITIALIZE LEVEL COUNTER
100   IF(TSCN$A(A$TREE,UNITS,ENTRY,MAXSIZ,I,MAXLEV,L,CODE))GOTO 105
      IF (CODE.NE.E$EOF) CALL ERRPR$(E$NRTN, CODE, 0, 0, 0, 0)
      CALL EXIT              /* ALL DONE IF E$EOF
      GOTO 10                /* RESTART IF 'S' TYPED
C
105   DO 200 I=1,L           /* CONSTRUCT PATHNAME
        IF (ENTRY(2,I).EQ.0) GOTO 150/* BRANCH IF SEGDIR
        CALL TNOUA(ENTRY(2,I), NLEN$A(ENTRY(2,I), 32))
        GOTO 170
C
150     CALL TNOUA('(', 1)   /* FORMAT SEGDIR ENTRY NUMBER
        CALL TODEC(ENTRY(3,I))
        CALL TNOUA(')', 1)
C
170     IF (I.NE.L) CALL TNOUA(' > ', 3)/* PATHNAME SEPARATOR
200     CONTINUE
      CALL TONL
      GOTO 100
      END
```

UNIT$A

Purpose

UNIT$A is a logical function that returns .TRUE. if a file unit is open and .FALSE. if it is not open.

Usage

```
INTEGER*2  funit
LOGICAL*2  log

log = UNIT$A(funit)
        (or)
CALL  UNIT$A(funit)
```

Parameters

funit

INPUT. PRIMOS file unit whose open status is being queried.

Loading and Linking Information

```
APPLIB   --   R-Mode
NVAPPLB  --   V-Mode (unshared)
VAPPLB   --   V-Mode
```

16
Parsing Routine

PARSING ROUTINE

This chapter describes the command line Parsing Routine CMDL$A, found in the APPLICATION subroutines library. The subroutine description includes a thorough discussion of strategies for using this subroutine. Examples are included.

CMDL$A

Note

For Pascal and PL/I programmers, CMDL$A is obsolete and has been replaced with CL$PIX.

Purpose

CMDL$A is a logical function for parsing a PRIMOS command line. CMDL$A is designed to facilitate the design and implementation of user interfaces in a program. It provides a means to break a character string into tokens (words or expressions) and return information regarding each token.

Usage

```
      INTEGER*2  key, kwlist(1), kwindx, optbuf(1)
      INTEGER*2  buflen, option, kwinfo(10)
      INTEGER*4  value
      LOGICAL    log

      log = CMDL$A(key, kwlist, kwindx, optbuf, buflen, option,
     x             value, kwinfo)
               (or)
      CALL  CMDL$A(key, kwlist, kwindx, optbuf, buflen, option,
     x             value, kwinfo)
```

Parameters

key

INPUT. Indicates the desired subroutine function. Valid keys are:

A$READ Return the next keyword entry in the command line.

A$NEXT Call COMANL to get the next command line, turn on default processing, and return the first keyword entry in the new command line.

A$RSET Reset the command line pointer to the beginning of the command line and turn on default processing. Use of this key does not return a keyword entry.

A$RAWI Return the remainder of the command line as raw text and turn on the end-of-line indicator. Text starts at the token following the option (if present) of the last keyword entry read.

A$NKWL Turn on default processing and return the next keyword entry in the command line. This key allows the calling program to switch keyword lists in the middle of a command line.

A$RCMD Permits the use of a keyword without a preceding minus sign as the first token on a line (may only be used for lines subsequent to the initial command line).

kwlist

INPUT. A one-dimensional integer array containing control information, a table of keyword entry descriptions, and a list of default keywords. See Kwlist Format later in this chapter for a complete description.

kwindx

OUTPUT. A keyword index returned as an INTEGER*2 variable identifying the keyword in an entry. Possible values are:

< 0	Unrecognized keyword or CMDL$A was called with a key of A$RSET or A$RAWI.
0	End of line.
> 0	Valid keyword.

optbuf

INPUT. Packed array that normally contains the text of a keyword option. However, if an unrecognized keyword is encountered, optbuf contains the text of that keyword. The data type does not matter.

buflen

INPUT. Specified length of optbuf in characters. It must be 0 or greater.

option

OUTPUT. Returned INTEGER*2 variable that describes the option following a keyword. Possible values are:

A$NONE No option, or option was null, optbuf will be blank.

A$NAME option was a name.

A$NUMB option was a number, result of numeric conversion returned in value.

A$NOVF option was a number and conversion resulted in overflow (decimal numbers only)

value

OUTPUT. Returned INTEGER*4 variable equal to the binary value of an option if it was a number. Otherwise, it is 0.

kwinfo

OUTPUT. A ten-halfword integer array that returns miscellaneous information and must be dimensioned in the calling program. kwinfo(1) is equal to the number of characters in optbuf and kwinfo(2) through kwinfo(10) are reserved for future use.

Discussion

CMDL$A was designed to simplify the processing of a PRIMOS command line while, at the same time, providing the user with a great deal of flexibility in defining the command environment.

This routine will parse a command line, one keyword entry at a time, and return information about each entry it encounters. A keyword entry is defined as a -keyword followed by an option. A default keyword entry is defined as an option that is not preceded by a -keyword but, by virtue of its position in the command line, implies a specified -keyword (e.g., FTN SNARF, where SNARF implies the default keyword -INPUT). Defaults may only occur at the beginning of a command line.

CMDL$A returns the following information for each keyword entry in the command line:

- Integer that identifies the -keyword (kwindx)
- Text of the keyword option, if present (optbuf)
- Option type (option)

- Results of numeric conversion, if option was a number (value)
- Number of characters in the text of an option (kwinfo(1))

Note

CMDL$A does not perform any action other than returning information about the command line.

Loading and Linking Information

APPLIB -- R-Mode
NVAPPLB -- V-Mode (unshared)
VAPPLB -- V-Mode

Other Routines Called by CMDL$A

CNVA$A, CNVB$A, CSUB$A, FILL$A, JSTR$A, MSUB$A, MSTR$A, NLEN$A and SSUB$A.

Defining a Command Environment

The following is a list of considerations that should be taken into account when defining a command environment:

1. A keyword may have, at most, one option following it.
2. A keyword must begin with a dash (-).
3. A keyword may not be a decimal number (e.g., -99).
4. Register-setting parameters (described with the R-mode EXECUTE command in the LOAD and SEG Reference Guide) are not recognized.
5. Default keywords are only allowed at the beginning of a command line. The first -keyword encountered turns off default processing and all remaining options on the command line must be preceded by a -keyword. (This restriction can be circumvented by using a key of A$NKWL; however the user must be aware of the fact that when default processing is in effect each option is treated as if it were preceded by a -keyword.)
6. A key of A$RAWI (or an option type of A$RAWI) will turn on the end-of-line indicator and any further attempts to read from the current command line will return an end-of-line condition. To turn off the end-of-line indicator, CMDL$A must be called with a key of A$RSET or A$NEXT.

7. A buffer length that is too small to contain the text of an option will cause that option to be truncated and an error message to be displayed.

8. Default keyword entries that have a numeric option should be avoided as PRIMOS may intercept them as register settings.

9. A negative hexadecimal option that consists only of alphabetic characters (such as -FF) is always interpreted as a -keyword.

10. Keyword entries in the keyword table with the same keyword index are considered synonyms. A keyword may have any number of synonyms, each with different option specifications. However, if a keyword with synonyms is also a default and default processing is in effect, the option specifications for the synonyms is ignored. (In other words, a default keyword option always implies the first keyword in a synonym chain.)

11. Null entries in the command line are only permitted for keywords that have an option status of A$OPTL. All other null entries will be treated as either a missing option or an unrecognized keyword.

12. Calls to CMDL$A and RDTK$$ on the same command line should be avoided, as CMDL$A uses RDTK$$ to perform a look ahead when a -keyword is encountered.

13. All text is forced to uppercase unless enclosed in quotes or read as raw text (A$RAWI).

Kwlist Format

The kwlist array consists of three sections. The first section contains control information, the second contains the keyword entry table, and the third contains the default list.

Control Information

Word 1 — Number (n) of keyword entries in table, must be greater than 0.

Word 2 — Maximum length of keyword text in characters, must be greater than or equal to 2 and not more than 80. All keywords must have the same length and therefore it may be necessary to pad them with blanks.

Keyword Entry Table

Words 1 to n — Text of keyword. The actual number of characters must be equal to the maximum keyword length.

Word n+1 — Keyword index, must be greater than 0.

Word n+2 — Minimum number of characters in the keyword to match, including leading minus sign. The number must be no less than 2 and no greater than the maximum keyword length. A 0 or negative value causes the keyword to be ignored when the table is searched. This allows keyword text to exist as documentation.

Word n+3 — Option status; possible values are:

- A$NONE — No option may follow keyword.
- A$OPTL — option may or may not follow keyword.
- A$REQD — option must follow keyword.

Word n+4 — Option type; possible values are:

- A$NONE — If status is A$NONE.
- A$BIN — option must be a binary number.
- A$DEC — option must be a decimal number.
- A$OCT — option must be an octal number.
- A$HEX — option must be a hexadecimal number.
- A$NAME, — option must be a name.
- A$NBIN — option may be a name or a binary number.

A$NDEC option may be a name or a decimal number.

A$NOCT option may be a name or an octal number.

A$NHEX option may be a name or a hexadecimal number. If the option consists of all alphabetic characters, which also constitute a valid hexadecimal number, it will be interpreted as such -- for example, FACE.

A$RAWI option is the remainder of the command line after the current -keyword is read as raw text. Use of this option will turn on the end-of-line indicator in the same manner as a key of A$RAWI.

Default List

Word 1 Number (n) of default keywords, must be greater than or equal to 0.

Words 2 to n+1 List of keyword indices, previously defined in the keyword entry table, which will be used when default processing is in effect. A default keyword entry may not have an option status of A$NONE.

Error Messages

The function value will be false if any of the following errors occur:

```
BAD KEY
BUFFER LENGTH LESS THAN ZERO
NAME TOO LONG.  (name text)
UNRECOGNIZED KEYWORD.  (keyword text)
BAD KEYWORD OPTION.  (option text)
MISSING KEYWORD OPTION.
NO. OF KEYWORD ENTRIES MUST BE .GT. ZERO.
MAX KEYWORD LENGTH MUST BE .GE. 2 AND .LE. 80.
1ST CHARACTER OF KEYWORD MUST BE '-'.  (keyword text)
KEYWORD MAY NOT BE A NUMBER.  (keyword text)
KEYWORD INDEX MUST BE .GT. ZERO.  (keyword text)
MIN CHARACTERS TO MATCH MUST BE .LE. MAX KEYWORD LENGTH.
   (keyword text)
INVALID OPTION STATUS.  (keyword text)
INVALID OPTION TYPE.  (keyword text)
NO. OF DEFAULTS MUST BE .GE. ZERO.
DEFAULT NOT DEFINED IN KEYWORD LIST.  (default index)
INVALID DEFAULT OPTION STATUS.  (keyword text)
MIN CHARACTERS TO MATCH MUST BE .GE. 2. (keyword text)
UNDETERMINED ERROR> (text of last keyword or option read)
```

Example(s)

```
C         TEST PROGRAM FOR CMDL$A
C
        IMPLICIT INTEGER*2 (A-Z)
        INTEGER*4 VALUE
        DIMENSION BUFFER(10),KWLIST(128),INFO(10)
$INSERT SYSCOM>A$KEYS
C
        DATA KWLIST /11,14,
       * '*max chars: 14',1,0,A$REQD,A$DEC,
       * '-NDECIMAL',2,2,A$OPTL,A$NDEC,
       * '-OCTAL',4,2,A$REQD,A$OCT,
       * '-NOCTAL',4,3,A$OPTL,A$NOCT,
       * '-HEXADECIMAL',5,2,A$REQD,A$HEX,
       * '-NHEXADECIMAL',6,3,A$OPTL,A$NHEX,
       * '-NAME',7,5,A$REQD,A$NAME,
       * '-MAYBE',8,6,A$OPTL,A$NAME,
       * '-NONE',9,5,A$NONE,A$NONE,
       * '-QUIT',10,2,A$NONE,A$NONE,
       * '-TITLE',99,2,A$OPTL,A$RAWI,
       * 4,1,2,8,7/
C
C
        BUFLEN = 20
        KEY = A$READ
  10    IF (CMDL$A(KEY,KWLIST,KWINDX,BUFFER,BUFLEN,TYPE,VALUE,INFO))
       *GO TO 15
        PRINT 99
  99    FORMAT(/'TRY AGAIN,TURKEY !')
        CALL EXIT
  15    IF (KWINDX.EQ.10) CALL EXIT
        IF (KWINDX.NE.A$NONE) GO TO 20
        KEY = A$NEXT
        GO TO 10
  20    KEY = A$READ
        PRINT 100 BUFFER,KWINDX,TYPE,VALUE,INFO(1)
  100   FORMAT(/10A2/'KWINDX  TYPE  VALUE  CHARS'/2X,4(I3,6X))
        GO TO 10
        END
```

PART V

SORT LIBRARIES AND FORTRAN MATRIX LIBRARY

17
Sort Libraries

GENERAL OVERVIEW

Part V of this Volume presents descriptions of the Sort Libraries and the MATHLB (Matrix) Library. Chapter 17 describes several sort subroutines available to the user in either R-Mode or V-Mode libraries. Chapter 18 describes the R-Mode subroutines available in MATHLB.

SORT SUBROUTINE LIBRARIES

PRIMOS contains many subroutines for performing disk or internal sorts. These subroutines are contained in four libraries:

- VSRTLI
- SRTLIB
- VMSORT
- MSORTS

After a brief survey of these libraries, there is a summary of the subroutines in each library, followed by important information on records, collating sequence, keys, tag/nontag sorts, and the use of open file units. Finally, each sort routine receives a detailed description.

VSRTLI: is the V-mode sort library. It contains the routine SUBSRT, which sorts a single input file on ASCII keys, and the routine ASCS$$, that sorts and merges up to 10 input files and handles a variety of key types. These two routines accept larger records and more keys than their corresponding R-mode versions, which are located in SRTLIB. Both SUBSRT and ASCS$$ call SRTF$S, another VSRTLI routine. SRTF$S sorts up to 20 input files and accepts a variety of key types.

VSRTLI also contains a set of cooperating sort routines and cooperating merge routines. These allow you to use your own input and output procedures. Strategies for using these cooperating routines are discussed in the sections called Cooperating Sort Subroutines and Cooperating Merge Subroutines, below. A sample program that uses the cooperating sort subroutines is included.

SRTLIB: is the R-mode sort library. It contains the R-mode versions of SUBSRT and ASCS$$.

VMSORT: is the V-mode library containing routines that perform different types of in-memory sorts (heap, bubble, partition exchange, radix exchange, straight insertion, binary search, and diminishing increment). VMSORT also has a binary-search and table-building subroutine.

MSORTS: is the R-mode version of VMSORT.

Table 17-1 shows the subroutines by function. Table 17-2 shows which subroutines are located in each sort library.

> Caution
>
> R-mode subroutines can be called from FTN and PMA in R mode only. If you call an R-mode routine from a program in a different mode, the results are unpredictable. Refer to the FORTRAN and PMA chapters in Volume I for information on declaring parameters in FTN and PMA, respectively.

Table 17-1
Sort Routines by Function

Sort one file on ASCII key(s).	SUBSRT
Sort (multiple key types) or merge sorted files.	ASCS$$
Merge sorted files.	MRG1$S
Return next merged record to sort.	MRG2$S
Close merged input files.	MRG3$S
Sort one or several input files.	SRTF$S
Prepare sort table and buffers.	SETU$S
Get input records.	RLSE$S
Sort tables prepared by SETU$S.	CMBN$S
Get sorted records.	RTRN$S
Close all sort units.	CLNU$S
Heap sort.	HEAP
Partition exchange sort.	QUICK
Diminishing increment sort.	SHELL
Radix exchange sort.	RADXEX
Insertion sort.	INSERT
Bubble sort.	BUBBLE
Binary search or build binary table.	BNSRCH

Table 17-2
Sort Subroutines by Library

SRTLIB	VSRTLI	MSORTS	VMSORT
SUBSRT	SUBSRT	HEAP	HEAP
ASCS$$	ASCS$$	QUICK	QUICK
	SRTF$S	SHELL	SHELL
	SETU$S	RADXEX	RADXEX
	RLSE$S	INSERT	INSERT
	CMBN$S	BUBBLE	BUBBLE
	RTRN$S	BNSRCH	BNSRCH
	CLNU$S		
	MRG1$S		
	MRG2$S		
	MRG3$S		

Record Types

The following record types are handled by the VSRTLI library routines.

Compressed Source: Record with compressed blanks, delimited by a new line character ('212). Compressed source lines cannot contain data which may be interpreted as a blank compression indicator ('221) or new line character.

Uncompressed Source: Record with no blank compression, delimited by a newline character ('212). Uncompressed source lines cannot contain data which may be interpreted as a new line character.

Variable Length: Record stored with length (in halfwords) in the first halfword. This length does not include the first halfword which contains the count. Files containing records of this type are also called binary files (not the same as object files produced by a compiler).

Note

To sort variable length records, you must supply character varying strings for subroutine parameters.

Fixed Length: Record containing data but no length information. The length must be defined as the maximum line size. (If a new line character is appended to each record to make the file acceptable input to EDITOR (ED), the character must be included in the length.)

Default Record Type: The default depends upon the key types specified. (See Key Definitions, below.) The input type defaults to variable length if the key specifies a single-precision (16-bit) integer, double-precision (32-bit) integer, or single- or double-precision real number. Otherwise, the default is compressed source. If the output type is not specified, it is assumed to be the same as the input type. SRTLIB routines use only compressed-source and variable records.

Note

If multiple input files are used, they must all contain records of the same type.

Record Length

The maximum record length allowed is 508 characters in R-mode and 32760 characters in V-mode. "WARNING-LINE TRUNCATED" is printed whenever the data (not including record delimiters) exceeds the maximum record length and the excess data is ignored. Output record length defaults to the input record length.

Collating Sequence

You may sort ASCII keys using the EBCDIC rather than the ASCII collating sequence. This option is specified in the spcls(2) parameter of SRTF$S and SETU$S.

Key Definitions

A sort key is a portion of the record that determines the position of the record in the sorted output. Most routines allow you to sort on multiple keys. For many routines you must specify the starting and ending positions of the keys in the data record. Specify the appropriate columns (for character data) or bytes (for binary data). Each key must start and end on a byte boundary. An improperly defined key (for example, a key whose ending byte is greater than the record length) produces unpredictable results. With compressed source records, the key is padded with spaces.

In R-mode you may specify 20 keys each with a maximum length of 312 characters.

In V-mode you may specify up to 64 key fields, with a total length less than or equal to the maximum record length of 32760 characters. For fixed-length records, the routines verify that key fields are contained

within the record length. Other restrictions on key length are mentioned below. Each key type is specified as a parameter. The available key types are the following:

ASCII Keys: Character strings, stored one character per byte. ASCII keys are limited only by the length of the record in V mode.

Signed Numeric ASCII Keys: Require one byte per digit and include the following types:

Numeric ASCII, leading separate sign

Numeric ASCII, trailing separate sign

Numeric ASCII, leading embedded sign

Numeric ASCII, trailing embedded sign

A space is treated as a positive sign. Signed numeric ASCII keys may be as long as 63 digits plus sign.

When the sign is separate, a positive number has a plus sign(+) and a negative number has a minus sign(-). If the sign is embedded, a single character is used to represent the digit and sign. Embedded sign characters are:

Digit	Positive	Negative
0	0,-,+,{	};-
1	1 A	J
2	2 B	K
3	3 C	L
4	4 D	M
5	5 E	N
6	6 F	O
7	7 G	P
8	8 H	Q
9	9 I	R

Unsigned Numeric ASCII Keys: Stored one digit per byte and are limited only by the length of the record.

Integer and Real Keys: Include the following types:

Key	Byte Length	Range
Single-precision integer	2	-32767 to +32767
Double-precision integer	4	-2**31 to +2**31-1
Single-precision real	4	±(10**-38 to 10**38)
Double-precision real	8	±(10**-9902 to 10**9825)
Unsigned integer	2	0 to 65535

Packed Decimal Keys: Stored two digits per byte. The last byte contains the final digit plus sign. A negative field has a hex D in the sign nibble. All other four-bit combinations in the sign nibble represent a positive sign. A packed field must have an odd number of digits and may have up to 63 digits plus sign.

Arguments

Numeric parameters are INTEGER*2 (fixed bin(15)), unless otherwise noted; refer to the sample declaration statements for specific information. Character strings such as pathnames are received as integer arrays by the subroutines.

Tag Sorts

Some sort subroutines offer two types of sorts: tag sorts and nontag sorts. You can choose the type of sort by setting a parameter. This choice has the following meaning. When a routine cannot perform a sort completely in the memory allocated, it creates temporary work files in which it stores sorted pieces of the data. These sorted pieces are then merged to create the output file. A tag sort stores the input records separately from the key data. After all the keys have been sorted and merged, the corresponding records are located and returned. This last phase may be time-consuming for a very large file. A nontag sort stores each input record with its sort key. This method eliminates the search for each record after the merge, but requires more disk space. Furthermore, a nontag sort is not always faster than a tag sort because merging records and keys requires more I/O than merging keys only.

The following are some criteria (in suggested order of importance) for selecting a tag sort versus a nontag sort:

- If disk space is a problem, use a tag sort.
- If the input file is small, use either type of sort.
- If the input file is large, use a nontag sort.
- If the input file is partially ordered, use a nontag sort.
- If the input file is not ordered, use a tag sort.

Using Open File Units

The SRTF$S, SETU$S, and MRG1$S subroutines allow you to use open file units for input and output files. Using open file units can save you time. If an input or output file is already open, you need not close it only to have the sort routine open it again.

When you use an open file unit, specify 0 for the pathname length parameter of the sort routine. Before you call the sort routine, be sure that the file pointer is positioned at the beginning of the open file.

VSRTLI (V-MODE) SUBROUTINES

VSRTLI routines follow a consistent naming convention to avoid possible conflict between user-written routines and system routines. All entry points end with the suffix $S (except SUBSRT and ASCS$$, which remain the same for compatibility with earlier versions of the library). Subroutines that are used internally by the VSRTLI routines have a suffix of $$S and should not be called from user routines.

For the VSRTLI routines, you may specify up to 64 keys. The maximum record length is 32760 bytes.

SUBSRT

Purpose

SUBSRT sorts a single input file containing compressed source records. The file is sorted on up to 64 ASCII keys in ascending order. Maximum record length is 32760 bytes (characters).

Usage

```
DCL SUBSRT ENTRY(CHARACTER(80), FIXED BIN(15), CHARACTER(80),
                 FIXED BIN(15), FIXED BIN(15), 64 FIXED BIN(15),
                 64 FIXED BIN(15), FIXED BIN(15), FIXED BIN(31));

CALL SUBSRT (path_1, len_1, path_2, len_2, numkey, nstart, nend,
             npass, nitem);
```

Parameters

path_1

INPUT. Input pathname, up to 80 characters.

len_1

INPUT. Length of input pathname in characters.

path_2

INPUT. Output pathname, up to 80 characters.

len_2

INPUT. Length of output pathname in characters.

numkey

INPUT. Number of keys (pairs of starting and ending columns, or starting and ending bytes if binary). (Maximum is 64, default is 1.)

nstart

INPUT. Array containing starting columns/bytes of keys. (Each must be $\geq$ 1).

nend

INPUT. Array containing ending columns/bytes of keys. (Each must be $\leq$ the maximum record length.)

npass

OUTPUT. Number of passes made during the sort.

nitem

OUTPUT. Number of items returned in the output file.

Loading and Linking Information

VSRTLI -- V-mode

(For the R-mode version of SUBSRT, see SRTLIB (R-MODE) SUBROUTINES, later in this chapter.)

ASCS$$

Alternate Name

A nonstandard alternate name for this subroutine is ASCSRT. Avoid this calling form.

Purpose

ASCS$$ sorts and merges compressed-source or variable-length records. Maximum record length is 32760 bytes. A variety of key types may be used, with ascending and descending keys within the same sort or merge. (The R-mode version handles fewer key types.) When equal keys are sorted, the input order is maintained.

Usage

```
DCL   ASCS$$ ENTRY(CHARACTER(80), FIXED BIN(15), CHARACTER(80),
                   FIXED BIN(15), FIXED BIN(15), 64 FIXED BIN(15),
                   64 FIXED BIN(15), FIXED BIN(15), FIXED BIN(31),
                   64 FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
                   CHARACTER(80*MGCNT), MGCNT(FIXED BIN 15),
                   PTR, FIXED BIN(15), 64 FIXED BIN(15),
                   FIXED BIN(15), FIXED BIN(15), MGCNTFIXED BIN(15));

CALL  ASCS$$(path_1, len_1, path_2, len_2, numkey, nstart, nend,
             npass, nitem, nrev, ispce, mgcnt, mgbuff, len,
             ADDR(buffer), msize, ntype, linsiz, nunits, units);
```

Parameters

path_1

INPUT. Input pathname, up to 80 characters.

len_1

INPUT. Length of input pathname in characters.

path_2

INPUT. Output pathname, up to 80 characters.

len_2

INPUT. Length of output pathname in characters.

numkey

INPUT. Number of keys (pairs of starting and ending columns, or starting and ending bytes if binary). (Maximum is 64, with a default of 1.)

nstart

INPUT. Array containing starting columns/bytes of keys. (Each must $\geq$ 1.)

nend

INPUT. Array containing ending columns/bytes of keys. (Each must be $\leq$ linsiz.)

npass

OUTPUT. Number of passes made during the sort.

nitem

OUTPUT. Number of items in output file.

nrev

INPUT. Array containing sort order for each key:

0 Ascending

1 Descending

Default is 0 (ascending).

ispce

INPUT. Option to specify treatment of blanks:

0 Include blank lines in sort (default).

1 Delete blank lines.

mgcnt

INPUT. Number of merge files (up to 10). (These files are merged with the input file.)

mgbuff

INPUT. Array containing merge filenames, up to 80 characters each (Pathnames may be used.)

len

INPUT. Array containing length of merge filenames in characters.

ADDR(buffer)

INPUT. Obsolete -- specify as 0.

msize

INPUT. Size (≤65536) of common block for sort in halfwords. Should be record size times maximum number of records expected. If nonzero, msize must be at least 1024 (one page) and no more than 64 pages. If larger, the message "WARNING-PRESORT BUFFER SHOULD NOT BE LARGER THAN ONE SEGMENT" is issued, and the default is used. Default is one segment (65536 halfwords).

ntype

INPUT. Optional. Array containing type of each key:

1	ASCII
2	16-bit integer
3	Single-precision real
4	Double-precision real
5	32-bit integer
6	Numeric ASCII, leading separate sign
7	Numeric ASCII, trailing separate sign
8	Packed decimal
9	Numeric ASCII, leading embedded sign
10	Numeric ASCII, trailing embedded sign
11	Numeric ASCII, unsigned
12	ASCII, lowercase sorts equal to uppercase
13	Unsigned integer

Default is all ASCII keys.

linsiz

INPUT. Optional. Maximum size of record in characters (bytes). Default is 32760.

nunits

SCRATCH. Obsolete. May be omitted.

units

SCRATCH. Obsolete. May be omitted.

Loading and Linking Information

VSRTLI -- V-mode

(For R-mode version of ASCS$$, see SRTLIB (R-MODE) SUBROUTINES later in this chapter.)

SRTF$S

Purpose

SRTF$S sorts a maximum of 20 input files into a single output file. It is called by the previous two sorts.

Usage

```
DCL SRTF$S ENTRY(CHAR(80,INCNT), INCNT FIXED BIN(15),
                 INCNT FIXED BIN(15), FIXED BIN(15),
                 CHARACTER(80), FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15), 64 FIXED BIN(15),
                 64 FIXED BIN(15), 64 FIXED BIN(15),
                 64 FIXED BIN(15), FIXED BIN(15), 5 FIXED BIN(15),
                 5 FIXED BIN(15), 5 FIXED BIN(15), FIXED BIN(15));

CALL SRTF$S(inbuff, inlen, inunts, incnt, path2, len2, outunt,
            numkey, nstart, nend, nrev, ntype, code, inrec,
            outrec, spcls, msize);
```

Parameters

inbuff

INPUT. Array containing input filenames. Ignored if open units are used.

inlen

INPUT. Array containing lengths of input pathnames in characters (up to 80 characters each). Specify 0 for pathname lengths if open units are used.

inunts

INPUT. Array containing input file units (if open units are used).

incnt

INPUT. Number of input files (up to 20).

path2

INPUT. Output file pathname, up to 80 characters. Ignored if an open unit is used.

len2

INPUT. Length of output pathname in characters. Specify 0 if an open unit is used.

outunt

INPUT. Output file unit (if an open unit is used).

numkey

INPUT. Number of keys (pairs of starting and ending columns, or starting and ending bytes if binary). (Maximum is 64, with a default of 1.)

nstart

INPUT. Array containing starting columns/bytes of keys (Each must be $\geq$ 1.)

nend

INPUT. Array containing ending columns/bytes of keys (Each must be $\leq$ inrec(2).)

nrev

INPUT. Array containing sort order for each key:

0	Ascending (default)
1	Descending

ntype

INPUT. Array containing type of each key:

1	ASCII
2	16-bit integer
3	Single-precision real
4	Double-precision real
5	32-bit integer
6	Numeric ASCII, leading separate sign
7	Numeric ASCII, trailing separate sign
8	Packed decimal
9	Numeric ASCII, leading embedded sign

10 Numeric ASCII, trailing embedded sign

11 Numeric ASCII, unsigned

12 ASCII, lowercase sorts equal to uppercase

13 Unsigned integer

Default is all ASCII keys.

code

OUTPUT. Return code (Refer to Appendix A for more information.)

inrec

INPUT. Array containing input record information:

inrec(1) Input record type:

1 Compressed source (blanks compressed)

2 Variable length

3 Fixed length (inrec(2) must be specified)

4 Uncompressed source (no blank compression)

Default depends on the key types specified in argument ntype.

inrec(2) Maximum input record size in characters (bytes). Default is 32760. Required for sorting fixed-length records.

inrec(3-5) Must be 0; reserved for future use.

outrec

INPUT. Array containing output record information:

outrec(1) Output record type. (See inrec.)

outrec(2) Maximum output record size in characters (bytes).

outrec(3-5) Must be 0; reserved for future use.

spcls

INPUT. Array containing special options:

spcls(1) Space option:

0 Include blank lines in sort (default).

1 Delete blank lines.

spcls(2) Collating sequence:

0 Default (Prime ECS)

1 Prime ECS

2 EBCDIC

3 ASCII-8

4 ISO-7

spcls(3) Tag/nontag option:

0 Default (tag sort)

1 Tag sort

2 Nontag sort

spcls(4-5) Must be 0; reserved for future use.

msize

INPUT. Size of presort buffer in pages (units of 1024 halfwords), not greater than 64. (Note that the units used here are pages which differ from the halfwords used by ASCS$$. Default is one segment; 64 pages.)

Loading and Linking Information

VSRTLI -- V-mode

COOPERATING SORT SUBROUTINES

This section describes the following five subroutines:

- SETU$S
- RLSE$S
- CMBN$S
- RTRN$S
- CLNU$S

These routines allow you to use your own input and output procedures. If you use these routines, you must use all of them and call them in the order listed above to ensure that the sort is done correctly. These subroutines are available in V-mode only. All parameters are INTEGER*2 in FTN; refer to the sample dcl statements for parameter declarations in PL/I.

The cooperating sort routines are used as follows. SETU$S creates a table in which the sort is to be done, setting record size, record type, and other attributes. It also determines whether the records are to be read directly from the input files into the sort area or whether they are to be accepted from an input procedure. It determines whether, after sorting, the records are to be sent directly to the output file or are to be postprocessed by an output procedure.

After calling SETU$S and giving it the necessary information, your program should call RLSE$S. If you specified to SETU$S that records were to be read from a preprocessing input procedure, you must supply the input procedure. The procedure should call RLSE$S once for each record to be sorted, supplying the record in the rlbuff parameter. If you specified to SETU$S that records were to be read directly from input file(s), your program should call RLSE$S only once and should not use the RLSE$S parameters. In this case, RLSE$S simply reads the records from the input file(s) into the sort area.

Next, your program should call the sort procedure, CMBN$S, to do the actual sorting. Since SETU$S should already have stored all information about record size, type, and collating sequence, CMBN$S accepts no parameters.

After calling CMBN$S, your program must call RTRN$S to obtain the sorted records. If you specified to SETU$S that records were to be postprocessed by an output procedure, RTRN$S uses its rtbuff parameter to return records for postprocessing. If you specified to SETU$S that records were to be returned directly to an output file, RTRN$S writes the records to the output file.

Finally, your program must call CLNU$S to close files opened by RLSE$S and RTRN$S and to delete temporary sort files.

These cooperating sort subroutines allow great flexibility in a sort operation because the program that calls them can process the records extensively before and after sorting. However, there is a tradeoff in speed. Because input and output procedures involve a procedure call for each record, and because preprocessing and postprocessing take time, sorting with these routines is generally slower than sorting with other routines.

An example of combined use of these subroutines is provided later in this section.

SETU$S

Purpose

SETU$S checks the parameters that you supply and sets up the tables for the requested sort.

Usage

```
DCL SETU$S ENTRY(CHARACTER(80,INCNT), INCNT FIXED BIN(15),
                 INCNT FIXED BIN(15), FIXED BIN(15),
                 CHARACTER(80), FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15), 64 FIXED BIN(15), 64 FIXED BIN(15),
                 64 FIXED BIN(15), 64 FIXED BIN(15), FIXED BIN(15),
                 5 FIXED BIN(15), 5 FIXED BIN(15), 5 FIXED BIN(15),
                 FIXED BIN(15), FIXED BIN(15), FIXED BIN(15));

CALL SETU$S(inbuff, inlen, inunts, incnt, path2, len2, outunt,
            numkey, nstart, nend, nrev, ntype, code, inrec,
            outrec, spcls, msize, iproc, oproc);
```

Parameters

inbuff

INPUT. Array containing input filenames (Ignored if open units are used.)

inlen

INPUT. Array containing lengths of input pathnames in characters (up to 80 characters each). (Specify 0 for pathname lengths if open units are used.)

inunts

INPUT. Array containing input file units (if open units are used).

incnt

INPUT. Number of input files (up to 20).

path2

INPUT. Output file pathname, up to 80 characters. (Ignored if an open unit is used.)

len2

INPUT. Length of output pathname in characters. (Specify 0 if an open unit is used.)

outunt

INPUT. Output file unit (if an open unit is used).

numkey

INPUT. Number of keys (pairs of starting and ending columns, or starting and ending bytes if binary). (Maximum is 64, default is 1.)

nstart

INPUT. Array containing starting columns/bytes of keys. (Each must be $\geq$ 1.)

nend

INPUT. Array containing ending columns/bytes of keys. (Each must be $\leq$ inrec(2).)

nrev

INPUT. Array containing sort order for each key:

0	Ascending (default)
1	Descending

ntype

INPUT. Array containing type of each key:

1	ASCII
2	Single-precision integer
3	Single-precision real
4	Double-precision real
5	Double-precision integer
6	Numeric ASCII, leading separate sign
7	Numeric ASCII, trailing separate sign
8	Packed decimal
9	Numeric ASCII, leading embedded sign

10 Numeric ASCII, trailing embedded sign

11 Numeric ASCII, unsigned

12 ASCII, lowercase sorts equal to uppercase

13 Unsigned integer

Default is all ASCII keys.

code

OUTPUT. Return code. (Refer to Appendix A for more information.)

inrec

INPUT. Array containing input record information:

inrec(1) Input record type:

1 Compressed source (blanks compressed)

2 Variable length

3 Fixed length (inrec(2) must be specified)

4 Uncompressed source (no blank compression)

Default depends on the key types specified in ntype.

inrec(2) Maximum input line size in characters (bytes). Default is 32760. Required for sorting fixed-length records.

inrec(3-5) Must be 0; reserved for future use.

outrec

INPUT. Array containing output record information:

outrec(1) Output record type. (See inrec.)

outrec(2) Maximum output line size in characters (bytes).

outrec(3-5) Must be 0; reserved for future use.

spcls

INPUT. Array containing:

spcls(1) Space option:

0 Include blank lines in sort (default).

1 Delete blank lines.

spcls(2) Collating sequence:

0 Default (ASCII)

1 ASCII

2 EBCDIC

spcls(3) Tag/nontag option:

0 Default (tag sort)

1 Tag sort

2 Nontag sort

spcls(4-5) Must be 0; reserved for future use.

msize

INPUT. Size of common presort buffer in pages (units of 1024 halfwords), no greater than 64. The size should be at least the product of the size of one record and the maximum number of records expected. Default is one segment (64 pages).

iproc

INPUT. Input data source (used by RLSE$S):

0 Input file

1 Input procedure

oproc

INPUT. Output data destination (used by RTRN$S):

0 Output file

1 Output procedure

Loading and Linking Information

VSRTLI -- V-mode

RLSE$S

Purpose

RLSE$S reads records into the sort area. Depending upon the value of iproc in the preceding SETU$S call, RLSE$S reads records either from input file(s) or from the buffer specified in the RLSE$S call.

Usage

```
DCL RLSE$S ENTRY(CHAR(*), FIXED BIN(15));

CALL RLSE$S(rlbuff, length);
```

Parameters

rlbuff

INPUT. Buffer containing next record for sort.

length

INPUT. Length of record in characters or bytes. This is not necessarily the full length of rlbuff.

Discussion

If you use an input procedure, you should call RLSE$S once for each record to be read.

If you use an input file instead of an input procedure, you should call RLSE$S only once per input file. If input is from a file, multiple calls to RLSE$S result in multiple occurrences of each record when sorted.

Source records passed from an input procedure (when inrec(1) = 1 in the SETU$S call) must end with a new line character ('212). Otherwise RLSE$S issues the message, "WARNING-LINE TRUNCATED," and the last character is overwritten by a NEWLINE character.

You may prefer to sort a text file as fixed-length records by reading the records into the program with RDLIN$ rather than by sorting them as source records.

Loading and Linking Information

VSRTLI -- V-mode

CMBN$S

Purpose

CMBN$S performs the internal sort. It uses the records provided by RLSE$S together with the tables, collating sequence, and other information provided by SETU$S. If the sort cannot be done within allocated memory, CMBN$S merges the strings previously sorted. (Refer to Tag Sorts, earlier in this chapter, for details.)

Usage

```
DCL CMBN$S entry;

CALL CMBN$S;
```

Loading and Linking Information

VSRTLI -- V-mode

RTRN$S

Purpose

RTRN$S returns the records sorted by CMBN$S. Records are returned either to an output procedure or to an output file, depending on the value of the oproc argument in the last call to SETU$S.

Usage

```
DCL RTRN$S ENTRY(CHAR(*), FIXED BIN(15));

CALL RTRN$S(rtbuff, length);
```

Parameters

rtbuff

OUTPUT. Buffer containing next sorted record. This parameter should be large enough to hold the longest record sorted.

length

OUTPUT. Length of record in characters or bytes. When all records have been returned, a call to RTRN$S returns a record length of 0.

Discussion

If you use an output procedure, each call to RTRN$S calls the next sorted record. The record is placed in rtbuff. If you want to save the record, you must write it to an output file.

If you use an output file instead of an output procedure, you should call RTRN$S only once. In this case, the RTRN$S parameters are not used but they are writeable, so you should include dummy variables. Refer to the list of integers and the CALL RTRN$S statement in the sample included with the CLNU$S description.

Loading and Linking Information

VSRTLI -- V-mode

CLNU$S

Purpose

CLNU$S closes all units opened by the sort routines and deletes any temporary files.

Usage

```
DCL CLNU$S entry;

CALL CLNU$S;
```

Loading and Linking Information

VSRTLI -- V-mode

SAMPLE USER INPUT PROCEDURE

The following sample program demonstrates the use of an input procedure with the sort subroutines. This input procedure selects for sorting only those records in INPUTFILE that begin with AA.

```
OK SLIST  SAMPLE.FTN
C---SAMPLE PROGRAM WHICH CALLS SORT
C------TO DEMONSTRATE THE USE OF AN
C------INPUT PROCEDURE BEFORE SORTING
$INSERT SYSCOM>KEYS.INS.FTN
$INSERT SYSCOM>ERRD.INS.FTN
C
      INTEGER
     & BUFFER(10),        /* Buffer for reading file
     & ERCODE,            /* Error code
     & INREC(5),          /* Input record information
     & OUTREC(5),         /* Output record information
     & SPCLS(5),          /* Flags for special options
     & TYPE               /* File type returned when file opened
     & DUMMY              /* Dummy variable for RTRN$S
C
      DATA
C        Input records are fixed length (20 characters):
     & INREC  / 3, 20, 0, 0, 0 /,
C        Output records are uncompressed source (to allow editing):
     & OUTREC / 4, 20, 0, 0, 0 /,
C        No special options:
     & SPCLS  / 0, 0, 0, 0, 0 /
```

```
C
C      Open the input file
       CALL SRCH$$ (K$READ, 'INPUTFILE', 9, 1, TYPE, ERCODE)
       IF (ERCODE .NE. 0) CALL ERRPR$ (K$NRTN, ERCODE, 0,0,0,0)
C
C      Initialize sort tables
       CALL SETU$S
      &        (0,              /* no input filenames
      &         0,              /* no lengths of filenames
      &         0,              /* no input file units
      &         0,              /* no input filenames
      &         'OUTPUTFILE',   /* this is the output filename
      &         10,             /* its name is 10 chars long
      &         0,              /* no output file unit specified
      &         1,              /* sort file on one key
      &         1,              /* start sort at column one
      &         20,             /* end sort at column twenty
      &         0,              /* sort in ascending order
      &         1,              /* the key is all ASCII characters
      &         ERCODE,         /* an error code will be returned
      &         INREC,          /* input record information
      &         OUTREC,         /* output record information
      &         SPCLS,          /* use options requested (none&)
      &         0,              /* use default for presort buffer
      &         1,              /* input data is from procedure
      &         0)              /* output is to file
       IF (ERCODE .NE. 0) CALL ERRPR$(K$NRTN, ERCODE,0,0,0,0)
C
C      With everything initialized, Read records from input file:
100    READ (5, 200, END=300) BUFFER
200    FORMAT (10A2)
C
C      Select the records to be sorted, and pass them so as to
C      sort with the record length (i.e, 20 characters):
       IF (BUFFER(1) .EQ. 'AA') CALL RLSE$S(BUFFER,20)
       GO TO 100                /* read next record
C
C      At end of input file, come here to finish up the sort:
300    CALL CMBN$S              /* do the actual sort
       CALL RTRN$S(DUMMY,DUMMY) /* send records to the output file
       CALL CLNU$S              /* clean up after sorting
C
C      Now close the input file:
       CALL SRCH$$ (K$CLOS, 0, 0, 1, 0, ERCODE)
       IF (ERCODE .NE. 0) CALL ERRPR$(K$NRTN,ERCODE,0,0,0,0)
       CALL EXIT
       END
```

This program may be compiled, loaded, and run with the following dialog:

```
OK FTN  SAMPLE  -64V  -DCLVAR

0000 ERRORS [<.MAIN.>FTN-REV19.3]
OK SEG  -LOAD
[SEG Rev. 20.2.B2 Copyright (c) 1986, Prime Computer, Inc.]
$ LO SAMPLE
$ LI VSRTLI
$ LI
LOAD COMPLETE
$ EXEC
```

Note

When compiling an F77 program, use the -INTS option for FTN compatibility.

The following listings show INPUTFILE and the sorted OUTPUTFILE.

```
OK SLIST  INPUTFILE

AA   EMPLOYEE1
BB   EMPLOYEE5
BB   EMPLOYEE3
CC   EMPLOYEE4
AA   EMPLOYEE2
AA   EMPLOYEE6
CC   EMPLOYEE7
AA   EMPLOYEE0
EE   EMPLOYEE8

OK SLIST OUTPUTFILE

AA   EMPLOYEE0
AA   EMPLOYEE1
AA   EMPLOYEE2
AA   EMPLOYEE6
```

COOPERATING MERGE SUBROUTINES

This section describes the merge subroutines MRG1$S, MRG2$S, and MRG3$S.

To merge two or more sorted files with no special processing, use MRG1$S. If you want to postprocess the merged records, you may use the three merge subroutines as follows:

- Call MRG1$S and supply it with specifications about the operation to be performed and the files and records to be used.
- Call MRG2$S to get the merged records one at a time.
- Finally, call MRG3$S to close units and delete temporary files opened by the other subroutines.

The cooperating merge routines are similar to the cooperating sort subroutines described earlier. However, the merge routines differ from the sort routines in their handling of output files. If you call MRG1$S and supply an output file rather than an output procedure, MRG1$S calls MRG2$S and MRG3$S itself. Do not call MRG2$S and MRG3$S yourself if output is to a file.

MRG1$S

Purpose

MRG1$S merges two to eleven previously sorted files into a single output file.

Usage

```
DCL MRG1$S ENTRY(CHARACTER(80,INCNT), INCNT FIXED BIN(15),
                 INCNT FIXED BIN(15), FIXED BIN(15),
                 CHARACTER(80), FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15), 64 FIXED BIN(15), 64 FIXED BIN(15),
                 64 FIXED BIN(15), 64 FIXED BIN(15), FIXED BIN(15),
                 5 FIXED BIN(15), 5 FIXED BIN(15), 5 FIXED BIN(15),
                 FIXED BIN(15));

CALL MRG1$S(inbuff, inlen, inunts, incnt, tree2, len2, outunt,
            numkey, nstart, nend, nrev, ntype, code, inrec,
            outrec, spcls, oproc);
```

Parameters

inbuff

INPUT. Array containing input filenames. Ignored if open units are used.

inlen

INPU. Array containing lengths of input pathnames in characters (up to 80 characters each). Specify 0 for pathname lengths if open units are used.

inunts

INPUT. Array containing input file units (if open units are used).

incnt

INPUT. Number of input files.

tree2

INPUT. Output file pathname, up to 80 characters. Ignored if an open unit is used.

len2

INPUT. Length of output pathname in characters. Specify 0 if an open unit is used.

outunt

INPUT. Output file unit (if an open unit is used).

numkey

INPUT. Number of keys (pairs of starting and ending columns, or starting and ending bytes if binary). Maximum is 64, default is 1.

nstart

INPUT. Array containing starting columns/bytes of keys. (Each must be $\geq$ 1.)

nend

INPUT. Array containing ending columns/bytes of keys. (Each must be $\leq$ inrec(2).)

nrev

INPUT. Array containing sort order for each key:

0	Ascending (default)
1	Descending

ntype

INPUT. Array containing type of each key:

1	ASCII
2	16-bit integer
3	Single-precision real
4	Double-precision real
5	32-bit integer
6	Numeric ASCII, leading separate sign
7	Numeric ASCII, trailing separate sign
8	Packed decimal
9	Numeric ASCII, leading embedded sign

10 Numeric ASCII, trailing embedded sign

11 Numeric ASCII, unsigned

12 ASCII, lowercase sorts equal to uppercase.

13 Unsigned integer

Default is all ASCII keys.

code

OUTPUT. Return code. (Refer to Appendix A for more information.)

inrec

INPUT. Array containing input record information:

inrec(1) Input record type:

1 Compressed source (blanks compressed)

2 Variable length

3 Fixed length (inrec(2) must be specified)

4 Uncompressed source (no blank compression)

Default depends on the key type specified in ntype.

inrec(2) Maximum input record size in characters (bytes). Required for sorting fixed-length records. Default is 32760.

inrec(3-5) Must be 0; reserved for future use.

outrec

INPUT. Array containing output record information:

outrec(1) Output record type. (See inrec.)

outrec(2) Maximum output record size in characters (bytes).

outrec(3-5) Must be 0; reserved for future use.

spcls

INPUT. Array containing:

spcls(1) Space option:

0 Include blank lines in sort (default).

1 Delete blank lines.

spcls(2) Collating sequence:

0 Default (ASCII)

1 ASCII

2 EBCDIC

spcls(3-5) Must be 0; reserved for future use.

oproc

INPUT. Output data destination (for use by MRG2$S):

0 Output file

1 Output procedure

Loading and Linking Information

VSRTLI -- V-mode

MRG2$SS

Purpose

This subroutine is used only after MRG1$S has been called to set up the merge area, record and file specifications, and collating keys. MRG2$S returns the next merged record. Do not call MRG2$S when output is to a file.

Usage

```
DCL MRG2$S ENTRY(CHAR(*), FIXED BIN(15));

CALL MRG2$S(rtbuff, length);
```

Parameters

rtbuff

OUTPUT. Buffer containing next merged record. Should be large enough to hold longest record merged.

length

OUTPUT. Length (in characters) of the record returned. Once all records have been returned, MRG2$S returns a length of 0.

Loading and Linking Information

VSRTLI -- V-mode

MRG3$S

Purpose

This subroutine is called only after MRG1$S and MRG2$S have been called. MRG3$S closes all units opened by the other merge routines. Do not call MRG3$S when output is to a file.

Usage

```
DCL MRG3$S ENTRY;

CALL MRG3$S;
```

Loading and Linking Information

VSRTLI -- V-mode

SRTLIB (R-MODE) SUBROUTINES

SRTLIB, the R-mode version of VSRTLI, holds two subroutines: SUBSRT and ASCS$$. See the discussion at the beginning of this chapter for the differences in these subroutines when used in R-mode instead of V-mode.

SUBSRT

Purpose

SUBSRT sorts a single input file containing compressed source records. The file is sorted on up to 20 ASCII keys in ascending order. Maximum record length is 508 bytes (characters). Maximum key length for all keys is 312 characters.

Usage

```
      INTEGER*2  path_1(40), len_1, path_2(40), len_2, numkey,
     x           nstart(20), nend(20), npass, nitem

      CALL SUBSRT(path_1, len_1, path_2, len_2, numkey, nstart,
     x            nend, npass, nitem)
```

Parameters

path_1

INPUT. Input pathname, up to 80 characters.

len_1

INPUT. Length of input pathname in characters.

path_2

INPUT. Output pathname, up to 80 characters.

len_2

INPUT. Length of output pathname in characters.

numkey

INPUT. Number of keys (pairs of starting and ending columns, or starting and ending bytes if binary). Maximum is 20, default is 1.

nstart

INPUT. Array containing starting columns/bytes of keys. (Each must be $\geq$ 1.)

nend

INPUT. Array containing ending columns/bytes of keys. (Each must be $\leq$ the maximum record length.)

npass

OUTPUT. Number of passes made during the sort.

nitem

OUTPUT. Number of items returned in the output file.

Loading and Linking Information

SRTLIB -- R-mode

(For the V-mode version of SRTLIB, see VSRTLI (V-MODE) SUBROUTINES, earlier in this chapter.)

ASCS$$

Alternate Name

A nonstandard alternate name for this subroutine is ASCSRT. Avoid this calling form.

Purpose

ASCS$$ sorts and merges compressed-source or variable-length records. Maximum record length is 508 bytes. A variety of key types may be used, with ascending and descending keys within the same sort or merge. (The V-mode version handles more key types.) When equal keys are sorted, the input order is maintained. Maximum total length for keys is 312 characters.

Usage

```
      INTEGER*2   path_1(1), len_1, path_2(11), len_2, numkey,
     x            nstart(1), nend(1), npass, nrev(1), ispce,
     x            mgcnt, mgbuff(1), len(1), msize, ntype(1),
     x            linsiz, nunits, units
      INTEGER*4   nitem

      CALL ASCS$$(path_1, len_1, path_2, len_2, numkey, nstart,
     x            nend, npass, nitem, nrev, ispce, mgcnt, mgbuff,
     x            len, loc(buffer), msize, ntype, linsiz, nunits,
     x            units)
```

Parameters

path_1

INPUT. Input pathname, up to 80 characters.

len_1

INPUT. Length of input pathname in characters.

path_2

INPUT. Output pathname, up to 80 characters.

len_2

INPUT. Length of output pathname in characters.

numkey

INPUT. Number of keys (pairs of starting and ending columns, or starting and ending bytes if binary). Maximum is 20, default is 1.

nstart

INPUT. Array containing starting columns/bytes of keys. (Each must be $\geq$ 1.)

nend INPUT. Array containing ending columns/bytes of keys. (Each must be $\leq$ the maximum record length.)

npass

OUTPUT. Number of passes made during the sort.

nitem

OUTPUT. Number of items returned in output file.

nrev

INPUT. Array containing order for each key:

0 Ascending

1 Descending

ispce

INPUT. Whether to take blanks into account:

0 Sort blank lines.

1 Delete blank lines.

mgcnt

INPUT. Number of merge files (up to 10). These file are merged with the input file.

mgbuff

INPUT. Array containing merge filenames, up to 80 characters each. Pathnames may be used.

len

INPUT. Array containing lengths of merge filenames in characters.

loc(buffer)

INPUT. Location of presort buffer.

msize

INPUT. Size of presort buffer in halfwords. The presort buffer size should be as large as possible on P100 and P200 systems. On virtual memory systems, the best size must be determined by experimentation.

ntype

INPUT. Optional. Array containing type of each key (default is ASCII):

1	ASCII (default)
2	16_bit integer
3	Single_precision real
4	Double_precision real
5	32_bit integer

linsiz

INPUT. Optional. Maximum size of record in characters (bytes). Default is 508.

nunits

INPUT. Optional. Number of file units available. (Four are used by ASCS$$.)

units

INPUT. Optional. Array containing available file units.

Loading and Linking Information

SRTLIB -- R-mode

(For the V-mode version of ASCS$$, see VSRTLI (V-MODE) SUBROUTINES, earlier in this chapter.)

MSORTS AND VMSORT SUBROUTINES

The MSORTS and VMSORT libraries contain several in-memory sort subroutines and a binary-search and table-building routine. MSORTS and VMSORT contain the same subroutines, except that MSORTS is the R-mode version and VMSORT is the V-mode version.

The reference for most of the algorithms and timing studies is Donald Knuth, "Sorting and Searching," The Art of Computer Programming, vol. 3, Reading, MA: Addison-Wesley, 1973. The timing figures quoted are based upon Knuth's algorithms on his fictional machine (MIX). Since these routines are more general, the timing formulas quoted here should be used only as an indication of the relative merits of each algorithm and not as exact computational tools.

The following routines are included in MSORTS and VMSORT:

HEAP	Heap sort - based upon binary trees
QUICK	Quick sort - partition-exchange
SHELL	Shell sort - diminishing increment
RADXEX	Radix exchange sort
INSERT	Straight insertion sort
BUBBLE	Bubble sort - interchange
BNSRCH	Binary search

The binary search routine (BNSRCH) can be used either for table lookup in an ordered table or for building a sorted table.

All routines accept multiword entries and multiword keys located anywhere within the entry. All entries must be equal length and keywords must be contiguous (no secondary keys).

The calling sequences for these routines are similar. However, each sort has slightly different requirements. Except for RADXEX, all routines have the same first five parameters.

Parameters Common to More Than One Subroutine

ptable

INPUT. Pointer to the first halfword of the table. (Not a PL/I pointer.) For example, if the table is in an array table(a,b), the parameter ptable = LOC (table). For routines in MSORTS, ptable is a full 16-bit pointer and can be in the upper 32K of memory. For VMSORT, ptable is a two-halfword pointer.

nentry

INPUT. Number of table entries (not halfwords) in the table -- that is, items to be sorted or searched. This parameter is a full 16-bit count, since there can be more than 32K entries in the table.

nhwds

INPUT. Number of halfwords per entry. nhwds must be more than 0. If nhwds is greater than 32K, there can be only a single entry.

fhword

INPUT. First halfword within the entry of the key field.

nkhwds

INPUT. Number of halfwords in the key field. nkhwds must be greater than 0 and less than or equal to nhwds. fhword + nkhwds - 1 must be no more than nhwds. (In other words, the key field must be contained within an entry.)

npass

OUTPUT. Number of passes made during the sort (0 if error).

altbp

INPUT. Alternate return for bad parameters (used only with FORTRAN -- use 0 for other languages).

RADXEX replaces the nkhwds parameter with the following:

fbit

INPUT. First bit within fhword of the key. fbit must be greater than 0 and fhword + (nbit + fbit - 2)/16 must be no more than nhwds. (In other words, the key field must be contained within an entry.)

nbit

INPUT. Number of bits in the key. The key field must be contained within an entry.

The routines HEAP, QUICK, RADXEX, and BUBBLE also require temporary arrays of the following sizes (in halfwords):

BUBBLE	tarray(nkhwds)
HEAP,QUICK	tarray(nhwds)
RADXEX	tarray(2*nbit)

These arrays are work arrays used internally by the subroutines. Their space is provided by the user, but the subroutine initializes them.

All routines except RADXEX sort the table in increasing order where the key is treated as a single, signed, multiword integer. For example, the numbers 5, -1, 10, -3 would be sorted to -3, -1, 5, 10. Since for RADXEX the key need not begin on a word boundary, RADXEX treats the key as a single, unsigned, multiword (or partial-word) integer. Thus, RADXEX would sort the same four numbers to 5, 10, -3, -1.

Loading and Linking Information

SRTLIB -- R-mode

(For the V-mode version of ASCS$$, see VSRTLI (V-MODE) SUBROUTINES, earlier in this chapter.)

BNSRCH

Purpose

BNSRCH sets up a binary table and performs a binary search.

Usage

```
DCL BNSRCH ENTRY(ADDR(TABLE), FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15), FIXED BIN(15), CHARACTER(NKHWDS),
                 CHARACTER(NHWDS), FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15), FIXED BIN(15));

CALL BNSRCH(ptable, nentry, nhwds, fhword, nkhwds, skey, fentry,
            index, opflag, altnf, altbp);
```

Parameters

Most of the BNSRCH parameters are described in the section, Parameters Common to More Than One Subroutine. The additional parameters are described below.

skey

INPUT. Search key array.

fentry

OUTPUT. Found entry array.

index

OUTPUT. Entry number of found entry.

opflag

INPUT. Operation key:

0	Locate.
1	Locate and delete.
2	Locate or insert.
3	Locate and update.

altnf

INPUT. Alternate return.

Discussion

Simple binary searching (opflag=0) tests each entry's key field for a match with skey. If the entry is found, it is returned in fentry and the entry number is put into index. If the entry is not found, the alternate return (altnf) is taken. If altnf is not specified, the normal return is taken, and the entry is deleted from the table as well as returned in fentry. In this case, index specifies where the entry was.

Opflag=2 is the same as opflag=0 if the entry is found. However, if the entry is not found, index is set to 0 before the return. altnf is not taken.

Opflag=3 is the same as opflag=0 if the entry is not found. If the entry is found, the contents of fentry and the found entry are interchanged, thus updating the table and returning the old entry.

Loading and Linking Information

MSORTS -- R-mode
VMSORT -- V-mode

BUBBLE

Purpose

This routine performs a bubble (simple interchange) sort.

Usage

```
DCL BUBBLE ENTRY(ADDR(TABLE), FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15), FIXED BIN(15), CHARACTER(2*NKHWDS),
                 FIXED BIN(15), FIXED BIN(15), FIXED BIN(15));

CALL BUBBLE(ptable, nentry, nhwds, fhword, nkhwds, tarray, npass,
            altbp, incr);
```

Parameters

Most of the BUBBLE parameters are described in the section, Parameters Common to More Than One Subroutine. The additional parameters are described below.

tarray

SCRATCH. Temporary array.

incr

INPUT. Used to sort nonadjacent entries. (Refer to the discussion of incr in the description of INSERT, below.) Default is 1 (sort adjacent entries).

Discussion

Running Time: If N is the number of entries, the average running time for this routine is proportional to N**2. Bubble sorting is good only for very small N, but is not as good as insertion sorting.

Loading and Linking Information

MSORTS -- R-mode
VMSORT -- V-mode

HEAP

Purpose

A heap sort is based on a nonthreaded binary tree structure. The algorithm consists of two parts: convert the table into a "heap," and then sort the heap by an efficient top-down search of the tree. The formal definition of a heap is as follows:

> The keys K(1), K(2), K(3),..., K(N) constitute a "heap" if K(J/2)>K(J) for 1<(J/2)<J<N.

Usage

```
DCL HEAP ENTRY(ADDR(TABLE), FIXED BIN(15), FIXED BIN(15),
               FIXED BIN(15), FIXED BIN(15), CHARACTER(2*NHWDS),
               FIXED BIN(15), FIXED BIN(15));

CALL HEAP(ptable, nentry, nhwds, fhword, nkhwds, tarray, npass, altbp);
```

Parameters

Most of the HEAP parameters are described in the section, Parameters Common to More Than One Subroutine. The additional parameter is the following:

tarray

> SCRATCH. Temporary array.

Discussion

Running Time: If N is the number of entries, the average running time is proportional to 23*N*ln(N) and the maximum is 26*N*ln(N). A heap sort tends to be inefficient if N<2000, but for N>2000 it outperforms all other sorts except QUICK.

Loading and Linking Information

MSORTS -- R-mode
VMSORT -- V-mode

INSERT

Purpose

Straight insertion sorting involves "percolating" each element into its final position.

Usage

```
DCL INSERT(ADDR(TABLE), FIXED BIN(15), FIXED BIN(15),
           FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
           FIXED BIN(15), FIXED BIN(15));

CALL INSERT(ptable, nentry, nhwds, fhword, nkhwds, npass, altbp,
            incr);
```

Parameters

Most of the INSERT parameters are described in the section, Parameters Common to More Than One Subroutine. The additional parameter is the following:

incr

INPUT. Used to sort nonadjacent entries. (Refer to the Discussion below.)

Discussion

The incr parameter is used to sort nonadjacent entries. For example, if incr = 3, then every third entry is included in the sort. The following is an example of a sort with incr = 3.

input: 10 9 8 7 6 5 4 3 2 1 0

output: 1 9 8 4 6 5 7 3 2 10 0

The default is incr = 1.

Running Time: Let N be the number of entries. Although the average running time is proportional to N**2, insertion sorting is very good for small tables (N<13) and tends to be very efficient for nearly ordered tables, even for large N.

Loading and Linking Information

MSORTS -- R-mode
VMSORT -- V-mode

QUICK

Purpose

QUICK performs a partition exchange sort. It is a variation of the basic quicksort called a median-of-three quicksort.

Usage

```
DCL QUICK ENTRY(ADDR(TABLE), FIXED BIN(15), FIXED BIN(15),
                FIXED BIN(15), FIXED BIN(15), CHARACTER(2*NHWDS),
                FIXED BIN(15), FIXED BIN(15));

CALL QUICK(ptable, nentry, nhwds, fhword, nkhwds, tarray, npass,
           altbp);
```

Parameters

Most of the QUICK parameters are described in the section, Parameters Common to More Than One Subroutine. The additional parameter is the following:

tarray

SCRATCH. Temporary array.

Discussion

Running Time: If N is the number of entries, the average running time is proportional to 12*N*ln(N), but the maximum time is on the order of N**2. On the average, QUICK is the fastest sort in MSORTS. However, in the worst case QUICK may be the slowest sort in MSORTS. The worst case is a completely ordered table. QUICK should not be used on tables that are already well-ordered.

Loading and Linking Information

MSORTS -- R-mode
VMSORT -- V-mode

RADXEX

Purpose

RADXEX is a radix-exchange sort that treats the key as a series of binary bits. It is based on both the method of radix sorting (as in the case of a card sorter) and that of partitioning. RADXEX does not sort signed numbers. It sorts the numbers 5, -1, 10, -3 to 5, 10, -3, -1. RADXEX has the advantage that the key does not have to start on a halfword boundary. In addition, the key need not be an integral number of halfwords long.

Usage

```
DCL RADXEX ENTRY(ADDR(TABLE), FIXED BIN(15), FIXED BIN(15),
                 FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
                 CHARACTER(4*NBIT), FIXED BIN(15),
                 FIXED BIN(15));

CALL RADXEX(ptable, nentry, nhwds, fhword, fbit, nbit, tarray,
            npass, altbp);
```

Parameters

Most of the RADXEX parameters are described in the section, Parameters Common to More Than One Subroutine. The additional parameter is the following:

tarray

SCRATCH. Temporary array used as partition stack.

Discussion

Running Time: If N is the number of entries, the average running time is proportional to 14*N*ln(N). Radix exchange is very fast (on the order of a QUICK sort) for large N, but is inefficient if equal keys are present.

Loading and Linking Information

MSORTS -- R-mode
VMSORT -- V-mode

SHELL

Purpose

A SHELL sort (named after Donald Shell) is a diminishing increment sort. SHELL utilizes the straight insertion sort (INSERT) on each of its passes to order the nonadjacent elements that are one INC apart. INC is then decreased on each pass. Increments are chosen by the formula:

```
INC=(3**k-1)/2
```

where the initial increment is chosen so that INC(k + 2)>N, and subsequent increments are chosen by decrementing k within the function.

Usage

```
DCL SHELL ENTRY(ADDR(TABLE), FIXED BIN(15), FIXED BIN(15),
                FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
                FIXED BIN(15));

CALL SHELL(ptable, nentry, nhwds, fhword, nkhwds, npass, altbp);
```

Parameters

The SHELL parameters are described in the section, Parameters Common to More Than One Subroutine.

Discussion

Running Time: If N is the number of entries, the average running time is proportional to N**1.25 and the maximum time is N**1.5. A complete timing analysis on the SHELL sort is not possible, but SHELL is very good for N<2000. For N>2000, the HEAP sort is better.

Loading and Linking Information

MSORTS -- R-mode
VMSORT -- V-mode

18
Fortran Matrix Library (MATHLB)

MATHLB contains a set of subroutines that perform matrix operations, solve systems of simultaneous linear equations, and generate permutations and combinations of elements. Table 18-1 lists the MATHLB subroutines.

> Caution
>
> The MATHLB subroutines, and all other R-mode subroutines, can be called from FTN and PMA in R-mode only. If you call an R-mode routine from a program in a different mode, the results are unpredictable. Refer to the FORTRAN and PMA chapters in Volume I for information on declaring parameters in FTN and PMA, respectively.
>
> In this chapter, each subroutine description includes the FTN statement(s) to declare the subroutine's parameters. If you want to call a MATHLB subroutine from PMA, refer to the PMA chapter in Volume I for information on declaring parameters in PMA.

Table 18-1
Summary of Available Matrix Operations

Operation	Single-Precision	Double-Precision	Integer	Complex
Setting matrix to identity matrix	*MIDN	DMIDN	IMIDN	CMIDN
Setting matrix to constant matrix	MCON	DMCON	IMCON	CMCON
Multiplying matrix by a scalar	MSCL	DMSCL	IMSCL	CMSCL
Matrix addition	MADD	DMADD	IMADD	CMADD
Matrix subtraction	MSUB	DMSUB	IMSUB	CMSUB
Matrix multiplication	MMLT	DMMLT	IMMLT	CMMLT
Calculating transpose matrix	*MTRN	DMTRN	IMTRN	CMTRN
Calculating adjoint matrix	MADJ	DMADJ	IMADJ	CMADJ
Calculating inverted matrix	*MINV	DMINV		CMINV
Calculating signed cofactor	*MCOF	DMCOF	IMCOF	CMCOF
Calculating determinant	*MDET	DMDET	IMDET	CMDET
Solving a system of linear equations	LINEQ	DLINEQ		CLINEQ
Generating permutations			PERM	
Generating combinations			COMB	

* For square matrices only

SUBROUTINE CONVENTIONS

The following conventions are used in the subroutine descriptions in this chapter.

Names

Most of the MATHLB subroutines have single-precision, double-precision, integer, and complex forms. A subroutine with more than one form appears in this chapter under its single-precision name. Alternate forms of the routine are shown in sample declaration and CALL statements. If a routine's single-precision name is XXXX, then its double-precision, integer, and complex names are DXXXX, IXXXX, and CXXXX, respectively.

Arguments and Data Modes

In most cases, matrices, arrays, and constants must be of the same data mode (REAL, DOUBLE PRECISION, INTEGER*2, or COMPLEX) as the subroutine. One exception is that work arrays are sometimes integer arrays even if the routine is real, double precision, or complex. The sample declaration statements indicate the data types of all parameters. Generally, you choose which form of the subroutine to use based on the data mode of the parameters you want to use. Matrix dimensions and error flags must be declared as INTEGER*2. Unless otherwise noted, all parameters are required.

Matrices

For the purposes of the MATHLB subroutines, a matrix is defined to be an array with two subscripts. An n x m matrix is an array with n rows and m columns; the first subscript has dimension n and the second subscript has dimension m. The dimensions passed as subroutine arguments must agree with the array sizes declared in the calling program; otherwise the elements will not be accessed properly.

Note

Except where otherwise noted, you may use the same matrix for more than one parameter of a subroutine (provided the dimensions are correct). For example, in matrix addition you may specify: A = A + B. In this case, the value of A will change; the new value of A will be the original value of A added to B.

Work Arrays and Matrices

Parameters described as work arrays or work matrices must always be distinct from one another in the calling program. Space for work arrays and matrices is provided by the user, but values for these parameters are supplied by the subroutine. In the parameter descriptions, these parameters are labeled "SCRATCH."

COMB

Purpose

COMB computes the next combination of nr out of n elements with a single interchange of elements at each call. The first call to COMB returns the combination 1, 2, 3, ..., nr. This subroutine is self-initializing and proceeds through all n!/(nr! * (n - nr)!) combinations. At the last combination it returns a value of last = 1 and resets itself. You may reinitialize the COMB subroutine by passing a restrt value of 1 along with new values for n and nr. The restrt parameter is optional and can be omitted. If you do not want to reinitialize the routine, either omit restrt from the calling sequence or set it to a value of 0.

Usage

```
      INTEGER icomb(nr), n, nr, iw1(n), iw2(n),
     x        iw3(n), last, restrt

      CALL COMB (icomb, n, nr, iw1, iw2, iw3, last, restrt)
```

Parameters

icomb

OUTPUT. Array with one subscript of dimension nr. Contains the combination returned by the subroutine.

n

INPUT. The number of elements from which combinations are taken.

nr

INPUT. The number of elements in each combination.

iw1

SCRATCH. Work array with one subscript of dimension n.

iw2

SCRATCH. Work array with one subscript of dimension n.

iw3

SCRATCH. Work array with one subscript of dimension n.

last

OUTPUT. When returned with value of 1, indicates that the currently returned combination is the last.

restrt

INPUT. Optional. When restrt = 1, the subroutine is reinitialized with new values for n and nr. When restrt = 0 or is omitted, the subroutine is not reinitialized.

Note

The calling program should not attempt to modify icomb, iw1, iw2, or iw3. For further details, see Gideon Ehrlich, "Loopless Algorithms for Generating Permutations, Combinations, and Other Combinatorial Configurations," Journal of the ACM, vol. 20, no. 3, July 1973, pp. 500-513.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

LINEQ

Purpose

LINEQ solves the system of n linear equations in n unknowns represented by (cmat) (xvect) = (yvect) where cmat is the square matrix of coefficients, xvect is the vector of unknowns upon which cmat operates, and yvect is the resulting vector. LINEQ solves the system of equations for xvect.

Note

For double-precision or complex numbers, use DLINEQ or CLINEQ, respectively.

Usage

LINEQ:

```
REAL*4      xvect(n), yvect(n), cmat(n,n), work(npl,npl)
INTEGER*2   n, npl, ierr
CALL LINEQ  (xvect, yvect, cmat, work, n, npl, ierr)
```

DLINEQ:

```
REAL*8      xvect(n), yvect(n), cmat(n,n), work(npl,npl)
INTEGER*2   n, npl, ierr
CALL DLINEQ (xvect, yvect, cmat, work, n, npl, ierr)
```

CLINEQ:

```
COMPLEX     xvect(n), yvect(n), cmat(n,n), work(npl,npl)
INTEGER*2   n, npl, ierr
CALL CLINEQ (xvect, yvect, cmat, work, n, npl, ierr)
```

Parameters

xvect

OUTPUT. Array with one subscript of dimension n. The n x 1 column vector on which cmat operates to result in yvect.

yvect

INPUT. Array with one subscript of dimension n. The n x 1 column vector that results when cmat operates on xvect.

cmat

INPUT. The n x n matrix of coefficients.

work

SCRATCH. An npl x npl work matrix.

n

INPUT. Integer indicating the dimension of the set of linear equations.

npl

INPUT. Integer indicating the dimension of work. Must be equal to n + 1.

ierr

OUTPUT. Integer error flag. Returns one of three possible values:

ierr	Meaning
0	Solution found satisfactorily
1	Coefficient matrix singular
2	npl < > n + 1

If ierr < > 0, no modifications are made to xvect.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MADD

Purpose

MADD adds the n x m matrix mat2 to the n x m matrix mat1 and returns the sum in the n x m matrix mats. In component form: mats (i,j) = mat1 (i,j) + mat2 (i,j) as i goes from 1 to n and j goes from 1 to m.

Note

For double-precision, integer, or complex numbers, use DMADD, IMADD, or CMADD, respectively.

Usage

MADD:

```
REAL*4     mats(n,m), mat1(n,m)
INTEGER*2  n, m
CALL MADD  (mats, mat1, mat2, n, m)
```

DMADD:

```
REAL*8     mats(n,m), mat1(n,m)
INTEGER*2  n, m
CALL DMADD (mats, mat1, mat2, n, m)
```

IMADD:

```
INTEGER*2  mats(n,m), mat1(n,m), n, m
CALL IMADD (mats, mat1, mat2, n, m)
```

CMADD:

```
COMPLEX    mats(n,m), mat1(n,m)
INTEGER*2  n, m
CALL CMADD (mats, mat1, mat2, n, m)
```

Parameters

mats

OUTPUT. Sum of mat1 and mat2. An n x m matrix.

mat1

INPUT. An n x m matrix.

mat2

INPUT. An n x m matrix.

n

INPUT. Integer indicating the number of rows in each matrix.

m

INPUT. Integer indicating the number of columns in each matrix.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MADJ

Purpose

This subroutine calculates the adjoint of the n x n matrix mati and stores it in the n x n matrix mato. Each element of the output matrix is the signed cofactor of the corresponding element of the input matrix.

Note

For double-precision, integer, or complex numbers, use DMADJ, IMADJ, or CMADJ, respectively.

Usage

MADJ:

```
      REAL*4     mato(n,n), mati(n,n)
      INTEGER*2  n, iw1(n), iw2(n), iw3(n), iw4(n), ierr
      CALL MADJ  (mato, mati, n, iw1, iw2, iw3, iw4, ierr)
```

DMADJ:

```
      REAL*8     mato(n,n), mati(n,n)
      INTEGER*2  n, iw1(n), iw2(n), iw3(n), iw4(n), ierr
      CALL MADJ  (mato, mati, n, iw1, iw2, iw3, iw4, ierr)
```

IMADJ:

```
      INTEGER*2  mato(n,n), mati(n,n), n, iw1(n),
     x           iw2(n), iw3(n), iw4(n), ierr
      CALL MADJ  (mato, mati, n, iw1, iw2, iw3, iw4, ierr)
```

CMADJ:

```
      COMPLEX    mato(n,n), mati(n,n)
      INTEGER*2  n, iw1(n), iw2(n), iw3(n), iw4(n), ierr
      CALL MADJ  (mato, mati, n, iw1, iw2, iw3, iw4, ierr)
```

Parameters

mato

OUTPUT. Adjoint of mati. An n x n matrix.

mati

INPUT. An n x n matrix.

n

INPUT. Integer indicating the dimension of mati.

iw1

SCRATCH. Work array with one subscript of dimension n.

iw2

SCRATCH. Work array with one subscript of dimension n.

iw3

SCRATCH. Work array with one subscript of dimension n.

iw4

SCRATCH. Work array with one subscript of dimension n.

ierr

OUTPUT. Integer error flag. Returns one of two possible values:

ierr	Meaning
0	Adjoint successfully constructed
1	n < 2; no adjoint may be constructed

Note

You must supply different names for mato and mati.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MCOF

Purpose

This subroutine calculates the signed cofactor of the element mat (i,j) of the n x n matrix mat and stores this value in cof. If i = 0 and j = 0 the determinant of mat is calculated.

Note

For double-precision, integer, or complex numbers, use DMCOF, IMCOF, or CMCOF, respectively.

Usage

MCOF:

```
      REAL*4     cof, mat(n,n)
      INTEGER*2  n, iw1(n), iw2(n), iw3(n),
     x           iw4(n), i, j, ierr
      CALL MCOF  (cof, mat, n, iw1, iw2, iw3, iw4, i, j, ierr)
```

DMCOF:

```
      REAL*8     cof, mat(n,n)
      INTEGER*2  n, iw1(n), iw2(n), iw3(n), iw4(n), i, j, ierr
      CALL DMCOF (cof, mat, n, iw1, iw2, iw3, iw4, i, j, ierr)
```

IMCOF:

```
      INTEGER*2  cof, mat(n,n), n, iw1(n), iw2(n), iw3(n),
     x           iw4(n), i, j, ierr
      CALL IMCOF (cof, mat, n, iw1, iw2, iw3, iw4, i, j, ierr)
```

CMCOF:

```
      COMPLEX    cof, mat(n,n)
      INTEGER*2  n, iw1(n), iw2(n), iw3(n), iw4(n), i, j, ierr
      CALL CMCOF (cof, mat, n, iw1, iw2, iw3, iw4, i, j, ierr)
```

Parameters

cof

OUTPUT. Signed cofactor of mat (i,j). Integer.

mat

INPUT. An n x n matrix.

n

INPUT. Integer indicating the dimension of mat.

iw1

SCRATCH. Work array with one subscript of dimension n.

iw2

SCRATCH. Work array with one subscript of dimension n.

iw3

SCRATCH. Work array with one subscript of dimension n.

iw4

SCRATCH. Work array with one subscript of dimension n.

i

INPUT. Integer indicating the row position of the element of mat whose signed cofactor is to be calculated.

j

INPUT. Integer indicating the column position of the element of mat whose signed cofactor is to be calculated.

ierr

OUTPUT. Integer error flag. Returns one of two possible values:

ierr	Meaning
0	Cofactor calculated successfully
1	No cofactor calculated, for one or more of the following reasons:

1. n < 2; no cofactor possible
2. i=j=n=0; no determinant
3. i=0 and j<>0 or
   j=0 and i<>0; subscript error
4. i>n and/or j>n;
   subscript error

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MCON

Purpose

This subroutine sets every element of the n x m matrix mat equal to the constant con.

Note

For double-precision, integer, or complex numbers, use DMCON, IMCON, or CMCON, respectively.

Usage

MCON:

```
REAL*4      mat(n,m), con
INTEGER*2   n, m
CALL MCON   (mat, n, m, con)
```

DMCON:

```
REAL*8      mat(n,m), con
INTEGER*2   n, m
CALL DMCON (mat, n, m, con)
```

IMCON:

```
INTEGER*2   mat(n, m), n, m, con)
CALL IMCON (mat, n, m, con)
```

CMCON:

```
COMPLEX     mat(n,m), con
INTEGER*2   n, m
CALL CMCON (mat, n, m, con)
```

Parameters

mat

OUTPUT. An n x m matrix.

n

INPUT. Integer indicating the number of rows in mat.

m

INPUT. Integer indicating the number of columns in mat.

con

INPUT. Constant to which all elements of mat are to be set equal.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MDET

Purpose

This subroutine calculates the determinant of the n x n matrix mat and stores it in det.

Note

For double-precision, integer, or complex numbers, use DMDET, IMDET, or CMDET, respectively.

Usage

MDET:

```
REAL*4     det, mat(n,n)
INTEGER*2  n, iw1(n), iw2(n), iw3(n), iw4(n), ierr
CALL MDET  (det, mat, n, iw1, iw2, iw3, iw4, ierr)
```

DMDET:

```
REAL*8     det, mat(n,n)
INTEGER*2  n, iw1(n), iw2(n), iw3(n), iw4(n), ierr
CALL DMDET (det, mat, n, iw1, iw2, iw3, iw4, ierr)
```

IMDET:

```
INTEGER*2  det, mat(n,n), n, iw1(n), iw2(n), iw3(n),
x          iw4(n), ierr
CALL IMDET (det, mat, n, iw1, iw2, iw3, iw4, ierr)
```

CMDET:

```
COMPLEX    det, mat(n,n)
INTEGER*2  n, iw1(n), iw2(n), iw3(n), iw4(n), ierr
CALL CMDET (det, mat, n, iw1, iw2, iw3, iw4, ierr)
```

Parameters

det

OUTPUT. Determinant of mat.

mat

INPUT. An n x n matrix.

n

INPUT. Integer indicating the dimension of mat.

iw1

SCRATCH. Work array with one subscript of dimension n.

iw2

SCRATCH. Work array with one subscript of dimension n.

iw3

SCRATCH. Work array with one subscript of dimension n.

iw4

SCRATCH. Work array with one subscript of dimension n.

ierr

OUTPUT. Integer error flag. Returns one of two possible values:

ierr	Meaning
0	Determinant calculated successfully
1	n = 0; no determinant possible

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MIDN

Purpose

This subroutine sets the n x n matrix mat equal to the n x n identity matrix. That is:

```
mat (i,j) = 0 if i <> j
          = 1 if i =  j
```

Note

For double-precision, integer, or complex numbers, use DMIDN, IMIDN, or CMIDN, respectively.

Usage

MIDN:

```
REAL*4     mat(n,n)
INTEGER*2  n
CALL MIDN  (mat, n)
```

DMIDN:

```
REAL*8     mat(n,n)
INTEGER*2  n
CALL DMIDN (mat, n)
```

IMIDN:

```
INTEGER*2  mat(n,n), n
CALL IMIDN (mat, n)
```

CMIDN:

```
COMPLEX    mat(n,n)
INTEGER*2  n
CALL CMIDN (mat, n)
```

Parameters

mat

INPUT/OUTPUT. An n x n matrix.

n

INPUT. Integer indicating the dimension of mat.

Discussion

The form of the subroutine (that is, the data mode of mat) determines the representation of 1 in the matrix, as follows:

Data Mode	Subroutine	Representation of 1
Single-precision	MIDN	1.(SP)
Double-precision	DMIDN	1.(DP)
Integer	IMIDN	1
Complex	CMIDN	(1.,0) (each SP)

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MINV

Purpose

This subroutine calculates the inverse of the n x n matrix mati and stores it in mato, if successful. The inverse of mati is mato if and only if:

```
mati * mato = mato * mati = I
```

where * denotes matrix multiplication and I is the n x n identity matrix. You must supply an npl x npn work matrix, where npl = n + 1 and npn = n + n.

Note

For double-precision or complex numbers, use DMINV or CMINV, respectively. There is no integer form of this subroutine as there is no guarantee that the inverse of an integer matrix is an integer matrix.

Usage

MINV:

```
REAL*4     mato(n,n), mati(n,n), work(npl,npn)
INTEGER*2  n, npl, npn, ierr
CALL MINV  (mato, mati, n, work, npl, npn, ierr)
```

DMINV:

```
REAL*8     mato(n,n), mati(n,n), work(npl,npn)
INTEGER*2  n, npl, npn, ierr
CALL DMINV (mato, mati, n, work, npl, npn, ierr)
```

CMINV:

```
COMPLEX    mato(n,n), mati(n,n), work(npl,npn)
INTEGER*2  n, npl, npn, ierr
CALL CMINV (mato, mati, n, work, npl, npn, ierr)
```

Parameters

mato

OUTPUT. Inverse of mati. An n x n matrix.

mati

INPUT. An n x n matrix.

n

INPUT. Integer indicating the dimension of mati.

work

SCRATCH. An npl x npn work matrix.

npl

INPUT. Integer indicating the number of rows in work. Must be equal to n + 1.

npn

INPUT. Integer indicating the number of columns in work. Must be equal to n + n.

ierr

OUTPUT. Integer error flag. Returns one of three possible values:

ierr	Meaning
0	Inverted matrix stored in mato.
1	Matrix is singular; no inversion possible; mato is filled with zeros.
2	npl < > n + 1 and/or npn < > n + n. No calculations performed.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MMLT

Purpose

This subroutine multiplies the n1 x n2 matrix matl (on the left) by the n2 x n3 matrix matr (on the right) and stores the resulting n1 x n3 product matrix in matp.

Note

For double-precision, integer, or complex numbers, use DMMLT, IMMLT, or CMMLT, respectively.

Usage

MMLT:

```
      REAL*4     matp(n1,n3), matl(n1,n2), matr(n2,n3)
      INTEGER*2  n1, n2, n3
      CALL MMLT  (matp, matl, matr, n1, n2, n3)
```

DMMLT:

```
      REAL*8     matp(n1,n3), matl(n1,n2), matr(n2,n3)
      INTEGER*2  n1, n2, n3
      CALL DMMLT (matp, matl, matr, n1, n2, n3)
```

IMMLT:

```
      INTEGER*2  matp(n1,n3), matl(n1,n2), matr(n2,n3),
     x           n1, n2, n3
      CALL IMMLT (matp, matl, matr, n1, n2, n3)
```

CMMLT:

```
      COMPLEX    matp(n1,n3), matl(n1,n2), matr(n2,n3)
      INTEGER*2  n1, n2, n3
      CALL CMMLT (matp, matl, matr, n1, n2, n3)
```

Parameters

matp

OUTPUT. Product of matl (on the left) and matr (on the right). An n1 x n3 matrix.

matl

INPUT. An n1 x n2 matrix.

matr

INPUT. An n2 x n3 matrix.

n1

INPUT. Integer indicating the number of rows in matl.

n2

INPUT. Integer indicating the number of columns in matl (same as number of rows of matr).

n3

INPUT. Integer indicating the number of columns in matr.

Note

The name you supply for matp must be different from those you supply for matl and matr. However, matl and matr may have the same name. For example:

```
CALL MMLT (A, B, C, N1, N2, N3)     LEGAL
CALL MMLT (A, B, B, N, N, N)        LEGAL
CALL MMLT (A, A, A, N, N, N)        ILLEGAL
CALL MMLT (A, A, B, N, N, N)        ILLEGAL
CALL MMLT (A, B, A, N, N, N)        ILLEGAL
```

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MSCL

Purpose

This subroutine multiplies the n x m matrix mati by the scalar constant scon and stores the resulting n x m matrix in mato. By components, scalar multiplication is defined as follows: mato (i,j) = scon * mati (i,j) for i from 1 to n, j from 1 to m.

Note

For double-precision, integer, or complex numbers, use DMSCL, IMSCL, or CMSCL, respectively.

Usage

MSCL:

```
REAL*4     mato(n,m), mati(n,m), scon
INTEGER*2  n, m
CALL MSCL  (mato, mati, n, m, scon)
```

DMSCL:

```
REAL*8     mato(n,m), mati(n,m), scon
INTEGER*2  n, m
CALL DMSCL (mato, mati, n, m, scon)
```

IMSCL:

```
INTEGER*2  mato(n,m), mati(n,m), n, m, scon
CALL IMSCL (mato, mati, n, m, scon)
```

CMSCL:

```
COMPLEX    mato(n,m), mati(n,m), scon
INTEGER*2  n, m
CALL CMSCL (mato, mati, n, m, scon)
```

Parameters

mato

OUTPUT. The product of mati and scon. An n x m matrix.

mati

INPUT. An n x m matrix.

n

INPUT. Integer indicating the number of rows in mati.

m

INPUT. Integer indicating the number of columns in mati.

scon

INPUT. Scalar constant.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MSUB

Purpose

This subroutine subtracts the n x m matrix mat2 from the n x m matrix mat1 and stores the difference in the n x m matrix matd.

Note

For double-precision, integer, or complex numbers, use DMSUB, IMSUB, or CMSUB, respectively.

Usage

MSUB:

```
REAL*4     matd(n,m), mat1(n,m), mat2(n,m)
INTEGER*2  n, m
CALL MSUB  (matd, mat1, mat2, n, m)
```

DMSUB:

```
REAL*8     matd(n,m), mat1(n,m), mat2(n,m)
INTEGER*2  n, m
CALL DMSUB (matd, mat1, mat2, n, m)
```

IMSUB:

```
INTEGER*2  matd(n,m), mat1(n,m), mat2(n,m), n, m
CALL IMSUB (matd, mat1, mat2, n, m)
```

CMSUB:

```
COMPLEX    matd(n,m), mat1(n,m), mat2(n,m)
INTEGER*2  n, m
CALL CMSUB (matd, mat1, mat2, n, m)
```

Parameters

matd

OUTPUT. The difference mat1 - mat2. An n x m matrix.

mat1

INPUT. An n x m matrix.

mat2

INPUT. An n x m matrix.

n .

INPUT. Integer indicating the row dimension of the input matrices.

m

INPUT. Integer indicating the column dimension of the input matrices.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

MTRN

Purpose

This subroutine calculates the transpose of the n x n matrix mati and stores it in the n x n matrix mato. The relationship between mati and mato is as follows: mato (i,j) = mati (j,i) for i, j = 1 to n.

Note

For double-precision, integer, or complex numbers, use DMTRN, IMTRN, or CMTRN, respectively.

Usage

MTRN:

```
REAL*4     mato(n,n), mati(n,n)
INTEGER*2  n
CALL MTRN  (mato, mati, n)
```

DMTRN:

```
REAL*8     mato(n,n), mati(n,n)
INTEGER*2  n
CALL DMTRN (mato, mati, n)
```

IMTRN:

```
INTEGER*2  mato(n,n), mati(n,n), n
CALL IMTRN (mato, mati, n)
```

CMTRN:

```
COMPLEX    mato(n,n), mati(n,n)
INTEGER*2  n
CALL CMTRN (mato, mati, n)
```

Parameters

mato

OUTPUT. The transpose of mati. An n x n matrix.

mati

INPUT. An n x n matrix.

n

INPUT. The dimension of mati.

Note

You must supply different names for mato and mati.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

PERM

Purpose

PERM computes the next permutation of n elements with a single interchange of adjacent elements at each call. The first call to PERM returns the permutation 1, 2, 3,..., n. This subroutine is self-initializing and proceeds through all n! permutations. At the last permutation it returns a value of last = 1 and resets itself. You may reinitialize the PERM subroutine by passing a new value of n or by passing the restrt parameter with a value of 1. The restrt parameter is optional and can be omitted. If you do not want to reinitialize the routine, either omit restrt from the calling sequence or set it to a value of 0.

Usage

```
      INTEGER*2  iperm(n), n, iw1(n), iw2(n), iw3(n),
     x           last, restrt

      CALL PERM  (iperm, n, iw1, iw2, iw3, last, restrt)
```

Parameters

iperm

OUTPUT. Array with one subscript of dimension n. Contains the permutation returned by the subroutine.

n

INPUT. The number of elements from which permutations are to be generated. A new value of n reinitializes the subroutine. n must be greater than 2.

iw1

SCRATCH. Work array with one subscript of dimension n.

iw2

SCRATCH. Work array with one subscript of dimension n.

iw3

SCRATCH. Work array with one subscript of dimension n.

last

OUTPUT. When returned with value of 1, indicates that the currently returned permutation is the last.

restrt

INPUT. Optional. When restrt = 1, the subroutine is reinitialized. When restrt = 0 or is omitted, the subroutine is not reinitialized.

Note

The calling program should not attempt to modify iperm, iw1, iw2, or iw3. For further details, see Gideon Ehrlich, "Loopless Algorithms for Generating Permutations, Combinations, and Other Combinatorial Configurations," Journal of the ACM, vol. 20, no. 3, July 1973, pp. 500-513.

Loading and Linking Information

V-mode with shared libraries: Load VMXLIB.

V-mode with unshared libraries: Load NVMXLIB.

R-mode: Load MATHLB.

APPENDIXES

A
Error Handling

INTRODUCTION

This appendix discusses PRIMOS error messages and codes, and error-handling conventions for Rev. 17 and later.

ERROR CODES

In most languages, error codes may be treated as data names rather than as numbers. For a full explanation of each error code with comments on possible causes and solutions of the error indicated, see Volume 0 of the Advanced Programmer's Guide.

All the file management system routines described in Subroutines Reference II: File System, and most other new subroutines, employ error-handling procedures that are standard to PRIMOS subsystems. These procedures replace the older systems using ERRVEC (Appendix B) and the altrtn argument used for the IOCS subroutines in this volume.

The Return Code Parameter

All error codes, formerly placed in ERRVEC, are now returned to the user in a 16-bit halfword user-supplied integer variable called code in the Subroutines Reference series. For example, in the call:

```
CALL PRWF$$ (KEY,UNIT,LOC(BFR),NW,POS,RNW,CODE)
```

CODE is an integer that PRWF$$ sets to the appropriate return code. CODE should always be checked for zero or nonzero to ensure that errors do not go unnoticed. An example is:

```
CALL CREA$$ (NAME,NAMLEN,OPASS,NPASS,CODE)
IF (CODE.NE.0) GOTO 99
```

Standard System Error Code Definitions

Standard system error codes are variables with standardized names. In all cases, E$OK means no error. Any other value identifies a particular error or exceptional (not necessarily error) condition. All reference to specific code values should be by the standardized names in languages where they are available. For convenience, all names are defined in files in the SYSCOM directory on Volume 1 of the master disk. They are:

FORTRAN 77	ERRD.INS.FTN
FORTRAN IV	ERRD.INS.FTN
PASCAL	ERRD.INS.PASCAL
PL1G and PL/I	ERRD.INS.PL1
C	ERRD.INS.CC
PMA	ERRD.INS.PMA
BASIC/VM	Not available
COBOL	Not available

These should be included in the program with $INSERT for FORTRAN and PMA, or %INCLUDE for Pascal and PL/I.

THE ERROR-HANDLING SUBROUTINES ER$PRINT AND ER$TEXT

ER$PRINT and ER$TEXT take advantage of this error-handling facility provided by the code parameter, but they also allow error-handling in user-defined subroutines without the need for an altrtn argument.

Refer to Subroutines Reference III: Operating System for a full description of these subroutines.

B
Error Handling for I/O Subroutines

INTRODUCTION

The following describes obsolete error-handling procedures for the I/O subroutines. These procedures have been replaced by return codes and the subroutine ER$PRINT. (See Appendix A).

Generally, error-message and status information from PRIMOS I/O subroutines and some older PRIMOS routines are placed in a system-wide error vector, ERRVEC, described later in this appendix. If an error occurs, the user program returns to PRIMOS command level and the error and/or status information is placed in ERRVEC. Upon completion of a call to an I/O subroutine, status information is also placed in ERRVEC, which the user may access through a call to GINFO or PRERR. The contents of this vector are listed later in this appendix.

If the FORTRAN user so desires, it is possible to take an alternate return if an error occurs. This is specified by use of the altrtn parameter in the call to the I/O subroutine invoked by the user program. If the user specifies an altrtn, then the location of the return and the action taken are entirely up to the user.

SUBROUTINES FOR ERROR HANDLING

There are three subroutines that can be used for setting or retrieving information in ERRVEC. They are: ERRSET, GETERR, PRERR. Each is described on the following pages.

ERRSET

Purpose

ERRSET sets ERRVEC, a system vector, then performs an alternate return or displays the message stored in ERRVEC and returns control to the system.

Usage

```
DCL ERRSET ENTRY(CHAR(4), FIXED BIN(15));

CALL ERRSET (altval, altrtn);
```

(or)

```
DCL ERRSET ENTRY(CHAR(4), FIXED BIN(15), CHAR(*), FIXED BIN(15));

CALL ERRSET (altval, altrtn, messag, num);
```

(or)

```
DCL ERRSET ENTRY(CHAR(4), FIXED BIN(15), CHAR(6),
                 CHAR(*), FIXED BIN(15));

CALL ERRSET (altval, altrtn, name, messag, num);
```

Note

In Form 1, altval must have the value 100000 octal, and altrtn specifies where control is to pass. If altrtn is 0, the message stored in ERRVEC is printed and control returns to the system. Forms 2 and 3 are similar; therefore, the arguments are described collectively as shown below.

Parameters

altval

INPUT. A two-halfword array that contains an error code that replaces ERRVEC(1) and ERRVEC(2). altval(1) must not be equal to 100000 octal.

altrtn

INPUT. A FORTRAN or PL1 label in the calling program, here preceded by a dollar sign. If altrtn is nonzero, control goes to altrtn. If altrtn is 0, the message stored in ERRVEC is displayed and control returns to PRIMOS.

name

INPUT. The name of a three-halfword array containing a six-letter word. This name replaces ERRVEC(3), ERRVEC(4), and ERRVEC(5). If name is not an argument in the call, ERRVEC(3) is set to 0.

messag

OUTPUT. An array of characters stored two per halfword. A pointer to this messag is placed in ERRVEC(7).

num

INPUT. The number of characters in messag. The value of num replaces ERRVEC(8).

Discussion

If a message is to be displayed, first, six characters starting at ERRVEC(3) are printed at the terminal. Then the operating system checks to determine the number of characters to be printed. This information is contained in ERRVEC(8). The message to be printed is pointed to by ERRVEC(7). The operating system only prints the number of characters from the message (pointed to by ERRVEC(7)) that are indicated in ERRVEC(8). If ERRVEC(3) is 0, only the message pointed to by ERRVEC(7) is printed. The message stored in ERRVEC may also be printed by the PRERR command or the PRERR subroutine. The contents of ERRVEC may be obtained by calling subroutine GETERR.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

GETERR

Purpose

A user calls GETERR to obtain the contents of ERRVEC.

Usage

```
DCL GETERR ENTRY(CHAR(*), FIXED BIN(15);

CALL GETERR (xervec, n);
```

Parameters

xervec

INPUT. Array assigned to hold the contents of ERRVEC.

n

INPUT. The number of halfwords to move from ERRVEC into xervec.

Discussion

GETERR moves n halfwords from ERRVEC into xervec. To access information from ERRVEC use the proper offsets into xervec to achieve the following:

On an Alternate Return:	On a Normal Return:
ERRVEC(1) Error code	PRWFIL: ERRVEC(3) Record number ERRVEC(4) Word number
ERRVEC(2) Alternate value	SEARCH: ERRVEC(2) File type

Loading and Linking Information

```
FTNLIB   --   R-mode
NPFTNLB  --   V-mode (unshared)
PFTNLB   --   V-mode
SVCLIB   --   R-mode (maintained for PRIMOS-II)
```

PRERR

Purpose

PRERR displays an error message on the user's terminal.

Usage

CALL PRERR

Loading and Linking Information

FTNLIB -- R-mode
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

Example

A user wants to retain control on a request to open a unit for reading if the name was not found by SEARCH. To accomplish this, the program calls SEARCH and gets an alternate return. It then calls to GETERR and determines if an error occurred other than NAME NOT FOUND. To print the error message while maintaining program control, the user calls PRERR.

DESCRIPTION OF ERRVEC

ERRVEC consists of eight halfwords; their contents are as follows:

Word	Content	Remarks
ERRVEC(1)	Code	Indicates origin of error and nature of error.
(2)	Value	On alternate return, this is the value of the A-register. On normal return, this may have special meaning. (Refer to PRWFIL and SEARCH error codes below.)

Word	Content	Remarks
(3) (4) (5) (6)	X X X X X X X X	ERRVEC(3), ERRVEC(4), and ERRVEC(5) contain a six-character filename of the routine that caused the error. (ERRVEC(6) is available for expansion of names.)
(7)	Pointer to message	For PRIMOS supervisor use.
(8)	Message length	For PRIMOS supervisor use.

PRWFIL Error Codes

Code		Meaning
PD	UNIT NOT OPEN	
PE	PRWFIL EOF (End of File)	Number of halfwords left (Information is in R ERRVEC(2)).
PG	PRWFIL EOF (Beginning of File)	Number of halfwords left (Information is in ERRVEC(2)).

PRWFIL Normal Return

ERRVEC(3)	Record number
ERRVEC(4)	Word number

PRWFIL Read-Convenient

ERRVEC(2)	Number of halfwords read.

SEARCH Error Codes

ERRVEC(1) Code, with one of the following values:

Code	Meaning
SA	SEARCH, BAD PARAMETER
SD	UNIT NOT OPEN (truncate)
SD	UNIT OPEN ON DELETE
SH	<Filename> NOT FOUND
SI	UNIT IN USE
SK	UFD FULL
SL	NO UFD ATTACHED
SQ	SEG-DIR-ER
DJ	DISK FULL

SEARCH Normal Return

ERRVEC (2) Type, with one of the following values:

Type	Meaning
0	File is SAM.
1	File is DAM.
2	Segment directory is SAM.
3	Segment directory is DAM.
4	UFD is SAM.

C
SVC Information

SUPERVISOR CALL INSTRUCTIONS CALLED BY PRIMOS SUBROUTINES

This Appendix defines Supervisor Call (SVC) Instructions that are called by PRIMOS subroutines. The SVC instructions are all described in this documentation set unless otherwise noted. SVC numbers used by PRIMOS are listed in Table C-1.

SVC INTERFACE FOR I/O CALLS

The I/O subroutines described in Chapter 7 interface with the operating system by means of SVC instructions. This appendix describes these interfaces.

SVC INTERFACE CONSIDERATIONS

Disk

The disk interfaces with virtual memory through a SVC instruction to perform a READ or WRITE operation on a single physical record of a physical disk. The disk must be assigned to the terminal by the ASSIGN command. Refer to RRECL and WRECL in Chapter 5. For information about the SVC instruction, refer to the Assembly Language Programmer's Guide.

Magnetic Tape

MPC Line Printer

Output to the parallel interface line printer is accomplished through SVC calls. Refer to T$LMPC in Chapter 7.

MPC Card Reader

Input from the parallel interface card reader is controlled through SVC calls. Refer to T$CMPC in Chapter 7.

OPERATING SYSTEM RESPONSE TO AN SVC INSTRUCTION

The operating system response to an SVC instruction includes a "return-to-sender" capability. The format is an SVC instruction followed by a halfword encoded as follows:

Bits	Meaning
1	Use interlude routine
2	Return to sender
3-4	Zero
5-10	SVC class
11-16	SVC subclass

When bit 1 is set, the operating system assumes the location preceding the SVC instruction is a subroutine entry point and looks for the arguments back through that entry point.

When bit 2 is set, the operating system either performs the requested function or, if the class and subclass are not recognized, returns to the caller at the location following the SVC code halfword.

The four legal syntaxes are:

1.

```
      .
      .
      .
     SVC
     OCT    00xxyy
     DAC
     DAC
      .
      .
      .
     OCT    0
```

2.

```
     Ent  DAC  **
          SVC
          OCT  10xxyy
```

3.

```
      .
      .
      .
     SVC
     OCT  04xxyy
     (return-to-sender location)
     DAC
     DAC
      .
      .
      .
     OCT  0
```

4.

```
     Ent  DAC  **
          SVC
          OCT  14xxyy
          (return-to-sender location)
            .
            .
            .
```

In all cases above:

xx = 6-bit class

yy = 6-bit subclass

The following classes are currently assigned:

0 RTOS

1 File system miscellaneous

2 Sequential file I/O

3 Direct file I/O

4 -

5 DOSVM only; never reflected

6 Command input/output

7 Typers

10 Mag tape

11 Line printer

12 Card reader/punch

13 SMLC

77 Reserved for customer use

Table C-1
SVC Numbers Used by PRIMOS

Number	Associated Call
	AC$CAT (object-path, category-name, code)
	AC$CHG (name, acl-ptr, code)
	AC$DFT (name, code)
	AC$LST (name, acl-ptr, max-entries, acl-name, acl-type, code)
	AC$SET (key,name, acl-ptr, code)
	APSFX (in-pathname, out-pathname, suffix, status)
	ASNLN$ (key,line,protocol,config,lword,status)
*1500	ATCH$$ (ufdnam,namlen,ldisk,passwd,(key code))
!1400	ATTAC$ (ufdnam,namlen,ldisk,passwd,(key,loc (code)))
0100	ATTACH (ufdnam,ldisk,paswd,(key,altrtn))
*0507	BREAK$ (offon)
*0601	C1IN (char)
	CALAC$ (name, id-ptr, acess-needed, access-gotten, code)
	CAT$DL (name, code)
0602	CMREAD (char)
*1515	CNAM$$ (oldnam,oldlen,newnam,newlen,code)
!0113	CNAME (oldnam,newnam,altrtn)
1415	CNAME$ (oldnam,oldlen,newnam,newlen,loc(code))
*0604	CNIN$ (buff,charcnt,statv(3))
*0600	COMANL
*1516	COMI$$ (filnam,namlen,unit,code)
!1416	COMIN$ (filnam,namlen,unit,loc(code))
0603	COMINP (filnam,unit,(altrtn))
*1523	COMO$$ (key,filnam,namlen,xxxxxx,code)
!0401	CONECT (tgtnam,tgtusr,lun,data,statv,lintyp)
*1501	CREA$$ (ufdnam,namlen,opass,npass,code)
!1401	CREAT$ (ufdnam,namlen,opass,npass,loc(code))
0506	D$INIT (pdev)
	DIR$RD (key, unit, return-ptr, max-return-len, code)
!0410	DISCON (lun,data,statv)
*0705	DUPLX$ (key)
	ENT$RD (unit, name, return-ptr, max-return-len, code)
*1524	ERKL$$ (key,erasec,killc,code)
*1402	ERRPR$ (key,code),text,txtlen,name,namlen)
!0106	ERRTN (altrtn,name,msg,msglen)
0114	ERRSET (altval,altrtn,name,msglen)
*0105	EXIT
!0400	FAMSVC (a1,a2,a3,a4,a5,a6,altrtn)
*0115	FORCEW (key,unit)
!0402	GETCON (target,user,data,statv)
0110	GETERR (buff,nw)
	GETID$ (if-ptr, max-groups, code)
0112	GINFO (buff,nw)
*1504	GPAS$$ (ufdnam,namlen,opass,npass,code)
!1404	GPASS$ (ufdnam,namlen,opass,npass,code)
	GPATH$ (key, funit, buffer, bufflen, pathlen, code)
	ISACL$ (unit, code)
	NAMEQ$ (filnam1, namlen1, filnam2, namlen2)

Table C-1 (continued)
SVC Numbers Used by PRIMOS

Number	Associated Call
!0412	NETLNK (statv)
!0406	NETWAT
!0407	NTSTAT (key,p1,p2,array)
	PA$DEL (partition-name, code)
	PA$LST (name, acl-ptr, max-entries, code)
	PA$SET (partition-name, acl-ptr, code)
0111	PRERR
*1506	PRWF$$ (key,Funit,loc(bf),bflen,pos32,rnw,code)
0300	PRWFIL (key,unit,loc(buff),n,pos,altrtn)
!1406	PRWFL$ (key,unit,loc(buff),nw,pos,rnw,loc(code))
	Q$READ (buf, buflen, type, code)
	Q$SET (key, ufdnam, namlen, amount, code)
*1507	RDEN$$ (key,funit,bf,bfln,rnw,nam32,namln,code)
!1407	RDENT$ (key,unit,buff,buflen,Rnw,name32,namlen,loc(code))
!0202	RDLIN (unit,line,nw,altrtn)
*1525	RDLIN$ (unit,line,nw,code)
*1517	RDTK$$ (key,info(8),buff,buflen,code)
!1417	RDTKN$ (key,info(8),buff,buflen,loc(code))
!0404	RECEIV (lun,loc(buff),nw,statv)
*0505	RECYCL
*1520	REST$$ (rvec,name,namlen,code)
!1420	RESTO$ (rvec,name,namlen,loc(code))
0103	RESTOR (rvec,name,altrtn)
*1521	RESU$$ (name,namlen)
!1421	RESUM$ (name,namlen)
0104	RESUME (name)
!0403	RJCON (target,user,statv,numtyp)
!0500	RREC (loc(buff),buflen,n,ra,pdev,(altrtn))
0516	RRECL (loc(buff),buflen,n,ra32,pdev,(altrtn))
*1510	SATR$$ (key,name,namlen,array,code)
!1410	SATTR$ (key,name,namlen,array,loc(code))
0102	SAVE (rvec,name)
!1422	SAVE$ (rvec,name,namlen,loc(code))
*1522	SAVE$$ (rvec,name,namlen,code)
!1411	SEARC$ (key,name,namlen,unit,type,loc(code))
0101	SEARCH (key,name,unit,(altrtn))
!1414	SEGDR$ (key,unit,entrya,entryb,loc(code))
*1512	SGDR$$ (key,funit,entrya,entryb,code)
* --	SEM$DR (semnum,code)
* --	SEM$NF (semnum,code)
* --	SEM$TN (semnum,int32,int32,code)
* --	SEM$TS (senmun,code) (int fcn)
* --	SEM$WT (semnum,code)
* --	SLEEP$ (int32)
!*1513	SPAS$$ (opass,npass,loc(code))
1413	SPASS$ (key,name,namlen,unit,type,code)
*1511	SRCH$$ (key,name,namlen,unit,type,code)

Table C-1 (continued)
SVC Numbers Used by PRIMOS

Number	Associated Call
	SRSFX$ (key, pathname, unit, type, n-suffixed, suffix-list, basename, suffix-used, status)
*0513	T$AMLC (line,loc(buff),nw,inst.statv)
*0512	T$CMPC (unit,loc(buff),nw,inst,statv)
*0511	T$LMPC (unit,loc(buff),nw,inst,statv)
*0515	T$PMPC (unit,loc(buff),nw,inst,statv)
*0510	T$MT (unit,loc(buff),nw,inst,statv)
*0514	T$VG (unit,loc(buff),nw,inst,statv)
!1001	T$SLC0 (key,line,loc(buff),nw)
*0502	TIMDAT (buff,buflen)
*0702	TNOU (msg,charcnt)
*0703	TNOUA (msg,charcnt)
!0405	TRNMIT (lun,loc(buff),cnt,statv)
	TSRC$$ (ation+newfil, pathname, funit, chrpos, type, code)
	UPDATE
!0411	UNLINK
!0501	WREC (loc(buff),buflen,n,ra,pev,(altrtn))
0517	WRECL (loc(buff),buflen,n,ra32,pdev,(altrtn))
!0203	WTLIN (unit,line,nw,(altrtn))
*1526	WTLIN$ (unit,line,nw,code)

* = Also direct entrance call
! = Not described in volumes of the Subroutines Reference Guide

D
Obsolete Indication and Control Subroutines

OVERVIEW

The subroutines described in this appendix return a message or an error indicator value in AC5, or set a value depending on some machine condition. The subroutines are:

DISPLY

OVERFL

SLITE

SLITET

SSWTCH

Note

The subroutines described in this appendix are not currently available in V-mode under PRIMOS.

DISPLY

Purpose

DISPLY updates the sense light settings according to a1. The bit values of a1 (1=on, 0=off) correspond to switch/light settings which are displayed on the computer control panel.

Usage

```
INTEGER*2   a1
CALL DISPLY (a1)
```

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode

OVERFL

Purpose

If entry into F$ER was made, a1 (in location AC5) is given a value 1; otherwise it is set to 2. F$ER is left in the no error condition. OVERFL is called to check if an overflow condition has occurred.

Usage

```
INTEGER*2   a1
CALL OVERFL (a1)
```

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode

SLITE

Purpose

SLITE sets the sense light specified in a1 on or sets all sense lights off. If a1=0, all sense lights are reset off.

Usage

```
INTEGER*2  a1
CALL SLITE (a1)
CALL SLITE (0)
```

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode

SLITET

Purpose

SLITET tests the setting of a sense light specified by a1. The result of this test (1 for on, 2 for off) is in the location specified by r.

Usage

```
INTEGER*2   a1, r
CALL SLITET (a1,r)
```

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode

SSWTCH

Purpose

SSWTCH tests the setting of a sense switch specified by a1. The result of this test (1=set, 2=reset) is stored in the location specified in r.

Usage

```
INTEGER*2   a1, r
CALL SSWTCH (a1,r)
```

Loading and Linking Information

FTNLIB -- R-mode
SVCLIB -- R-mode

E
Other Obsolete Subroutines

This appendix contains descriptions for obsolete subroutines. Table E-1 lists the obsolete subroutines described in this appendix.

Table E-1
Obsolete Subroutines

O$AD07
C$Mxx
O$AMxx
I$AMxx
O$BMxx
I$BMxx

O$AD07

Purpose

O$AD07 writes ASCII from buffer onto a disk file open on file_unit. O$AD07 is obsolete and has been replaced with WTLIN$.

Usage

```
DCL O$AD07 ENTRY(FIXED BIN(15), CHAR(*) VARYING,
                 FIXED BIN(15), FIXED BIN(15));

CALL O$AD07 (file_unit, buffer, count, altrtn);
```

Parameters

file_unit

INPUT. The PRIMOS file unit (funit) number from 0 through 32761. (Users may assign 2 through 32761.) Since a file unit has a position and access method, a user program need not keep track of a file's position and access. Examples of file unit strategy are given with SRCH$$ in Volume II.

buffer

INPUT. The name of a data area in memory from which data is moved to the disk file.

count

INPUT. The number of halfwords to be transferred, or the length in halfwords of a buffer or filename.

altrtn

INPUT. The value of a numeric label in the user's FORTRAN program, to be used as an alternate return from the subroutine in case of error. The label number should be preceded by a $. FORTRAN calls may omit the argument or give it the value of 0 if no alternate return is wanted. PL/I programs may also use altrtn, but its label must be in the same stack frame used for the code of the calling module. Other calling languages should omit the argument, but not use 0.

Discussion

Information is written on the disk in compressed ASCII format. Multiple blank characters are replaced with the character DC1 (221 octal) followed by a halfword count. Trailing blanks are removed and the end of record indicated by the NEWLINE character, or NEWLINE followed by null.

Loading and Linking Information

FTNLIB -- R-mode
NPFTNLB -- V-mode (unshared)
PFTNLB -- V-mode
SVCLIB -- R-mode (maintained for PRIMOS-II)

Table E-2
Obsolete Magnetic Tape Subroutines

9-Track

C$M05	Control for 9-track ASCII and binary.
C$M13	Control for 9-track EBCDIC.
O$AM05	Write ASCII.
I$AM05	Read ASCII.
O$BM05	Write binary.
I$BM05	Read binary.
O$AM13	Write EBCDIC.
I$AM13	Read EBCDIC.

7-Track

C$M10	Control for 7-track ASCII and binary.
C$M11	Control for 7-track BCD.
O$AM10	Write ASCII.
I$AM10	Read ASCII.
O$BM10	Write binary.
I$BM10	Read binary.
O$AM11	Write BCD.
I$AM11	Read BCD.

Note

T$MT has replaced all the subroutines in Table E-2 except O$AM13 and I$AM13.

C$Mxx

Purpose

These subroutines provide particular control functions for either 7-track or 9-track magnetic tape machines, as shown in Table E-2. Since they share identical subroutine calling formats, their Usage descriptions are given the generic name C$Mxx, where xx stands for the numbers of the particular subroutine needed.

Usage

```
DCL C$Mxx ENTRY(FIXED BIN(15), FIXED BIN(15), FIXED BIN(15),
                FIXED BIN(15));

CALL C$Mxx (key, reserved, physical_unit,
            altrtn);
```

Parameters

key

INPUT. Indicates a user option and may be one of the following:

-4	Rewind to BOT (Beginning of Tape).
-3	Backspace one file mark.
-2	Backspace one record.
-1	Write file mark.
1	Open to read.
2	Open to write.
3	Open to read/write.
4	Close. (Write file mark and rewind).
5	Move forward one record.
6	Move forward one file mark.
7	Rewind to BOF (Beginning of file)
8	Select device and read status.

reserved

Not used.

physical_unit

INPUT. 0-7 (0-3 for PRIMOS II), depending on which device is ASSIGNed).

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error.

Discussion

These routines call T$MT and ERRSET.

Error Messages: If an error occurs during subroutine execution, the user program returns to PRIMOS command level and the error and/or status information is placed in a system-wide error vector called ERRVEC. When an I/O subroutine returns, status information is also placed in ERRVEC. The user may access it through a call to GINFO or PRERR. These subroutines return values to ERRVEC(1) and ERRVEC(2) that may be interpreted as indicated below. For a thorough discussion of ERRVEC, refer to Appendix B.

Message	Meaning	ERRVEC(1)	ERRVEC(2)
C$Mxx EOF	End of file	IE	1
C$Mxx EOT	End of tape	ID	2
C$Mxx MTNO	Magtape not operational	ID	3
C$Mxx PERR	Parity error	ID	4
C$Mxx HERR	Hardware error	ID	5
C$Mxx BADC	Bad call	ID	6

Loading and Linking Information

```
FTNLIB   --   R-mode
NPFTNLB  --   V-mode (unshared)
PFTNLB   --   V-mode
SVCLIB   --   R-mode (maintained for PRIMOS-II)
```

O$AMxx, I$AMxx, O$BMxx, I$BMxx

Purpose

These subroutines provide read and write functions for magnetic tape machines, as shown in Table E-2.

Since all these subroutines share the same calling sequence, the Usage description gives sub$ for the subroutine name; the name of the particular subroutine needed should be used at coding time.

Usage

```
DCL sub$ ENTRY(FIXED BIN(15), CHAR(*), FIXED BIN(15),
               FIXED BIN(15));

CALL sub$ (physical_unit, buffer, nhw, altrtn);
```

Parameters

physical_unit

INPUT. The magnetic tape controller drives multiple sub-units. Valid sub-unit numbers for this physical-device = 0, 1, 2, or 3.

buffer

INPUT/OUTPUT. Data name of the area from or to which information is tranferred.

nhw

INPUT/OUTPUT. Number of halfwords to be read or written. If nhw = 0, then the subroutine is to write a file mark.

altrtn

INPUT. Alternate return for FORTRAN programs calling this subroutine in case of end of file or other error. (See Chapter 10.)

Discussion

Error Messages: If an error occurs during subroutine execution and an altrtn has not been provided, the user program returns to PRIMOS command level and the error and/or status information is placed in a system-wide error vector called ERRVEC. When an I/O subroutine returns, status information is also placed in ERRVEC. The user may access it through a call to GINFO or PRERR. These particular subroutines return values to ERRVEC(1) and ERRVEC(2) that may be interpreted as indicated below. See Appendix B for a thorough discussion of ERRVEC.

Message	Meaning	ERRVEC(1)	ERRVEC(2)
Subroutine EOF	End of file	IE	1
Subroutine EOT	End of tape	ID	2
Subroutine MTNO	Magtape not operational	ID	3
Subroutine PERR	Parity error	ID	4
Subroutine HERR	Hardware error	ID	5
Subroutine BADC	Bad call	ID	6

Note

Parity error, PERR, occurs only after 25 parity or raw errors.

Loading Information: These subroutines all call T$MT and ERRSET.

Loading and Linking Information

```
FTNLIB   --   R-mode
NPFTNLB  --   V-mode (unshared)
PFTNLB   --   V-mode
SVCLIB   --   R-mode (maintained for PRIMOS-II)
```

F
Data Type Equivalents

To call a subroutine from a program written in any Prime language, you must declare the subroutine and its parameters in the calling program. Therefore, you must translate the PL/I data types expected by the subroutine into the equivalent data types in the language of the calling program.

The table that follows shows the equivalent data types for the Prime languages BASIC/VM, C, COBOL 74, FORTRAN IV, FORTRAN 77, Pascal, and PL/I. The leftmost column lists the generic storage unit, which is measured in bits, bytes, or halfwords for each data type. Each storage unit matches the data types listed to the right on the same row. The table does not include an equivalent data type for each generic unit in all languages. However, with knowledge of the corresponding machine representation, you can often determine a suitable workaround. For instance, to see if you can use a left-aligned bit in COBOL 74, you could write a program to test the sign of the 16-bit field declared as COMP. In addition, if a subroutine parameter consists of a structure with elements declared as BIT(n), it can be declared as an integer in the calling program. Read the appropriate language chapter in the Subroutines Reference I: Using Subroutines before using any of the equivalents shown in the table.

Note

The term PL/I refers both to full PL/I and to PL/I Subset G (PL/I-G).

Table F-1
Data Type Equivalents

Generic Unit	BASIC/VM SUB FORTRAN	C	COBOL 74	FORTRAN IV	FORTRAN 77	Pascal	PL/I
16-bit integer	INT	short enum	COMP PIC S9(1)- PIC S9(4)	INTEGER INTEGER*2 LOGICAL	INTEGER*2 LOGICAL*2	INTEGER Enumerated	FIXED BIN FIXED BIN(15)
32-bit integer	INT*4	int long	COMP PIC S9(5)- PIC S9(9)	INTEGER*4	INTEGER INTEGER*4 LOGICAL LOGICAL*4	LONGINTEGER	FIXED BIN(31)
64-bit integer			COMP PIC S9(10)- PIC S9(18)				
32-bit float single precision	REAL	float	COMP-1	REAL REAL*4	REAL REAL*4	REAL	FLOAT BIN FLOAT BIN(23)
64-bit float double precision	REAL*8	double	COMP-2	REAL*8	REAL*8	LONGREAL	FLOAT BIN(47)
128-bit float quad precision					REAL*16		
1 bit		short					BIT BIT(1)
1 left-aligned bit		short				BOOLEAN	BIT(1) ALIGNED

Table F-1 (continued)
Data Type Equivalents

Generic Unit	BASIC/VM SUB FORTRAN	C	COBOL 74	FORTRAN IV	FORTRAN 77	Pascal	PL/I
Bit string		unsigned int				SET	BIT(n)
Fixed-length character string	INT	char NAME[n] char NAME	DISPLAY PIC A(n) PIC X(n) FILLER		CHARACTER *n	CHAR PACKED ARRAY[1..n] OF CHAR	CHAR(n)
Fixed-length digit string			DISPLAY PIC 9(n)				PICTURE
Fixed-length digit string, 2 digits per byte			COMP-3				FIXED DECIMAL
Varying-length character string		struct {short LENGTH; char DATA[n]; } CVAR				STRING[n]	CHAR(n) VARYING
32-bit pointer		Pointer (32IX-mode)		LOC( )	LOC( )		POINTER OPTIONS (SHORT)
48-bit pointer		Pointer (64V-mode)				Pointer	POINTER

Notes

For a discussion of possible workarounds for some of the empty boxes in this table as well as a description of generic units for PMA, refer to the appropriate language chapter in the *Subroutines Reference Guide, Volume I.*

INDEXES

Index of Subroutines By Name

F$xxyy series	FORTRAN compiler floating-point functions.	I	B-8
FDAT$A	Convert the DATMOD field returned by RDEN$$ to DAY MON DD YYYY.	IV	14-10
FEDT$A	Convert the DATMOD field returned by RDEN$$ to DAY DD MON YYYY.	IV	14-12
FIL$DL	Delete a file identified by a pathname.	II	4-43
FILL$A	Fill a string with a character.	IV	10-6
FINFO$	Return information about a specified file unit.	II	4-45
FNCHK$	Verify a supplied string as a valid filename.	II	4-49
FORCEW	Force PRIMOS to write modified records to disk.	II	4-51
FRE$RA	De-allocate space for EPF function return information.	III	4-23
FSUB$A	Fill a substring with a given character.	IV	10-7
FTIM$A	Convert the TIMMOD field returned by REDN$$.	IV	14-14
G$METR	Return system metering information.	III	2-63
GCHAR	Get a character from an array.	III	6-25
GCHR$A	Get a character from a packed string.	IV	10-9
GEND$A	Position to end of file.	IV	15-5
GEN$PW	Generate a login validation password.	III	2-26a
GETERR	Return ERRVEC contents.	III	10-6
GETID$	Obtain the user-id and the groups to which it belongs.	II	2-21
GINFO	Return PRIMOS II information.	III	2-10
GPAS$$	Obtain the passwords of a subdirectory of the current directory.	II	2-23
GPATH$	Return the pathname of a specified unit, attach point, or segment.	II	4-53
GSNAM$	Return current PRIMOS system name.	III	2-12
GT$PAR	Parse character string into tokens.	III	6-27
GTROB$	Find out whether current attach point is on a robust partition.	II	3-18
GV$GET	Retrieve the value of a global variable.	II	6-12
GV$SET	Set the value of a global variable.	II	6-14
H$xy series	FORTRAN compiler complex number storage.	I	B-5
HEAP	Heap sort.	IV	17-51

Index of Subroutines by Function

This index lists subroutines grouped by the general functions that they perform. See the Index of Subroutines by Name to find a particular subroutine's volume, chapter, and page number.

ACCESS CATEGORY

Add an object's name to an access category.	AC$CAT
Modify an existing ACL on an object.	AC$CHG
Set an object's ACL to that of its parent directory.	AC$DFT
Make an object's ACL identical to that of another object.	AC$LIK
Obtain the contents of an object's ACL.	AC$LST
Convert an object from ACL protection to password protection.	AC$RVT
Set a specific ACL on an object.	AC$SET
Determine whether an object is accessible for a given action.	CALAC$
Delete an access category.	CAT$DL
Obtain the user-id and the groups to which it belongs.	GETID$
Obtain the passwords of a subdirectory of the current directory.	GPAS$$
Determine whether an object is ACL-protected.	ISACL$
Remove an object's priority access.	PA$DEL
Obtain the contents of an object's priority ACL.	PA$LST
Set priority access on an object.	PA$SET
Set the owner and nonowner passwords on an object.	SPAS$$

ARRAYS

Get a character from an array.	GCHAR
Store a character into an array location.	SCHAR

Return caller's maximum command environment depth.	CE$DPT
Parse command arguments according to a character string "picture" of the command line.	CL$PIX
Invoke a command from a running program.	CP$
Retrieve the value of a global variable.	GV$GET
Set the value of a global variable.	GV$SET
Return a list of commands valid at mini-command level.	LIST$CMD
Retrieve the value of a CPL local variable.	LV$GET
Set the value of a CPL local variable.	LV$SET
Return breadth of caller's current command environment.	RD$CE_DP

COMMAND_LEVEL

Call a new command level after an error.	CMLV$E
Call a new command level.	COMLV$
Return to PRIMOS.	EXIT
Initialize the command environment.	ICE$
Return serialization data.	KLM$IF
Record command error status.	SETRC$
Signal an error in a subsystem.	SS$ERR

CONDITION MECHANISM

Continue scan for on-units.	CNSIG$
Convert FORTRAN statement label to PL/I format.	MKLB$F
Create an on-unit (for FTN users).	MKON$F
Create an on-unit (for any language except FTN).	MKON$P
Create an on-unit (for PMA and PL/I users).	MKONU$

CONTROLLERS, Asynchronous, Multi-Line

DATA CONVERSION

DATE FORMATS

Convert binary date to "visual" format.	CV$FDV
Convert quadsecond date to binary format.	CV$QSD

DEVICES, Assigning or Attaching

Attach specified devices.	ATTDEV
Provide or set aside available logical file unit.	IOCS$G
Free a logical file unit number.	IOCS$F

DISK I/O

Read ASCII from disk.	I$AD07
Write binary to disk.	O$BD07
Read binary from disk.	I$BD07
Write ASCII to disk (fixed-length records).	O$AD08
Register disk format with driver.	DKGEO$

DRIVERS, Device-independent

Write ASCII data.	WRASC
Read ASCII data.	RDASC
Write binary data.	WRBIN
Read binary data.	RDBIN
Open PRIMOS file and perform other non-data transfer functions. (Primarily for IOCS applications.)	CONTRL

ENCRYPTION, of Login Password

Encrypt login validation passwords.	ENCRYPT$

EPFs

Allocating and De-Allocating Space For EPFs

Management of EPFs

Information from In-Memory User Profile

ERROR HANDLING, I/O

Set ERRVEC and perform a return or display ERRVEC message before returning control to system.	ERRSET
Obtain contents of ERRVEC.	GETERR
Display I/O error message on user terminal.	PRERR

EVENT SYNCHRONIZERS AND EVENT GROUPS

Creating, Using, and Destroying Event Synchronizers

Create an event synchronizer.	SYN$CREA
Post a notice on an event synchronizer.	SYN$POST
Wait on an event synchronizer.	SYN$WAIT
Perform a timed wait on an event synchronizer.	SYN$TMWT
Retrieve a notice from an event synchronizer.	SYN$RTRV
Destroy an event synchronizer.	SYN$DEST

Creating, Using, and Destroying Event Groups

Create an event group.	SYN$GCRE
Move an event synchronizer into an event group.	SYN$MVTO
Remove an event synchronizer from an event group.	SYN$REMV
Cause a process to wait on an event group.	SYN$GWT
Cause a process to perform a timed wait on an event group.	SYN$GTWT
Retrieve a notice from an event group.	SYN$GRTR
Destroy an event group.	SYN$GDST

Information

EXECUTABLE IMAGES

EXIT$ CONDITION

FILE SYSTEM OBJECTS

Close a file by name and return a bit string indicating closed units.	CL$FNR
Close a file system object by pathname.	CLO$FN
Close a file system object by file unit number.	CLO$FU
Close a file.	CLOS$A
Change the name of an object in the current directory.	CNAM$$
Create a new subdirectory in the current directory.	CREA$$
Create a new password directory.	CREPW$
Delete a file.	DELE$A
Create a new directory.	DIR$CR
Search for specified types of entries in a directory open on a file unit.	DIR$LS
Read sequentially the entries of a directory open on a file unit.	DIR$RD
Return entries meeting caller-specified selection criteria in a directory open on a file unit.	DIR$SE
Return the contents of a named entry in a directory open on a file unit.	ENT$RD
Generate a filename based on another name.	EQUAL$
Check for file existence.	EXST$A
Return a file system object's entryname and parent directory pathname.	EXTR$A
Delete a file identified by a pathname.	FIL$DL
Return information about a specified file unit.	FINFO$
Verify a supplied string as a valid filename.	FNCHK$
Force PRIMOS to write modified records to disk.	FORCEW
Position to end-of-file.	GEND$A
Return the pathname of a specified unit, attach point, or segment.	GPATH$

Return the size of a file system entry.	SIZE$
Open, close, delete, change access, or verify the existence of an object.	SRCH$$
Search for a file with a list of possible suffixes.	SRSFX$
Open a scratch file with unique name.	TEMP$7
Verify a supplied string as a valid pathname.	TNCHK$
Truncate file.	TRNC$A
Scan the file system structure.	TSCN$A
Open a file anywhere in the PRIMOS file structure.	TSRC$$
Check for file open.	UNIT$A
Return the minimum and maximum file unit numbers currently in use by this user.	UNITS$
Return a logical value indicating whether a wildcard name was matched.	WILD$
Write a line of characters to a file in compressed ASCII format.	WTLIN$

ISC

Access Server Names

Catalog a server's Low Level Name.	ISN$C
Look up a server's Low Level Name.	ISN$L
Recatalog a server's Low Level Name.	ISN$RC
Uncatalog a server's Low Level Name.	ISN$UC
Get the server name of a process.	SRS$GN
Get the process numbers of all processes associated with the server name.	SRS$GP
List the server names on your system.	SRS$LN

MATRIX OPERATIONS

Generate permutations.	PERM
Generate combinations.	COMB

The following groups contain subroutines for single-precision, double-precision, integer, and complex operations, respectively. (* indicates that a subroutine is not available.)

Set matrix to identity matrix.	MIDN, DMIDN, IMIDN, CMIDN
Set matrix to constant matrix.	MCON, DMCON, IMCON, CMCON
Multiply matrix by a scalar.	MSCL, DMSCL, IMSCL, CMSCL
Perform matrix addition.	MADD, DMADD, IMADD, CMADD
Perform matrix subtraction.	MSUB, DMSUB, IMSUB, CMSUB
Perform matrix multiplication.	MMLT, DMMLT, IMMLT, CMMLT
Calculate transpose matrix.	MTRN, DMTRN, IMTRN, CMTRN
Calculate adjoint matrix.	MADJ, DMADJ, IMADJ, CMADJ
Calculate inverted matrix.	MINV, DMINV, *, CMINV
Calculate signed cofactor.	MCOF, DMCOF, IMCOF, CMCOF
Calculate determinant.	MDET, DMDET, IMDET, CMDET
Solve a system of linear equations.	LINEQ, DLINEQ, *, CLINEQ

MEMORY

Allocate memory on the current stack.	ALOC$S
Move a block of memory.	MOVEW$

Output binary data to the high-speed paper-tape punch.	O$BP02
Input one character from the high-speed paper-tape reader to Register A.	P1IB
Output one character to the high-speed paper-tape punch from Register A.	P1OB
Input one character from paper tape, set high-order bit, ignore line feeds, send a line feed when carriage return is read.	P1IN
Output one character to the high-speed paper-tape punch.	P1OU

PARSING

Parse a PRIMOS command line.	CMDL$A
Parse character string into tokens.	GT$PAR

PERIPHERAL DEVICES

Line Printers

Centronics LP.	O$AL04
Parallel interface to line printer (MPC).	O$AL06
Versatec printer.	O$AL14
Move data to LPC line printer.	T$LMPC
Access a spooler queue.	SPOOL$
Place file in spool queue and perform SPOOLER command functions.	SP$REQ

Printer/Plotter

Versatec.	O$AL14
Versatec.	T$VG

Card Reader/Punch

Input from parallel card reader.	I$AC03

RANDOMIZING

Generate random number and update "seed," based upon a 32-bit word size and using the Linear Congruential Method.	RAND$A
Initialize random number generator "seed."	RNDI$A

SEARCH RULES

Locate a file using a search list and open the file. Create a file if the file sought does not exist.	OPSR$
Locate a file using a search list and a list of suffixes. Open the located file, or create a file if the file sought does not exist.	OPSRS$
Disable an optional search rule. Used to disable rules that have been enabled using SR$ENABL.	SR$ABSDS
Add a rule to the beginning of a search list or before a specified rule.	SR$ADDB
Add a rule to the end of a search list or after a specified rule.	SR$ADDE
Create a search list.	SR$CREAT
Delete a search list.	SR$DEL
Disable an optional search rule. Used to disable rules that have been enabled using SR$ENABL.	SR$DSABL
Enable an optional search rule. Enabled rules can be disabled using SR$DSABL or SR$ABSDS.	SR$ENABL
Determine if a search rule exists.	SR$EXSTR
Free list structure space allocated by SR$LIST or SR$READ.	SR$FR_LS
Initialize all search lists to system defaults.	SR$INIT
Return the names of all defined search lists.	SR$LIST
Read the next rule from a search list.	SR$NEXTR
Read all of the rules in a search list.	SR$READ

SEMAPHORES

SORTING

Get sorted records.	RTRN$S
Close all sort units.	CLNU$S
Heap sort.	HEAP
Partition exchange sort.	QUICK
Diminishing increment sort.	SHELL
Radix exchange sort.	RADXEX
Insertion sort.	INSERT
Bubble sort.	BUBBLE
Binary search or build binary table.	BNSRCH

STRINGS

Compare two strings for equality.	CSTR$A
Compare two substrings for equality.	CSUB$A
Fill a string with a character.	FILL$A
Fill a substring with a given character.	FSUB$A
Get a character from a packed string.	GCHR$A
Left-justify, right-justify, or center a string within a field.	JSTR$A
Locate one string within another.	LSTR$A
Locate one substring within another.	LSUB$A
Move a character between packed strings.	MCHR$A
Move one string to another.	MSTR$A
Move one substring to another.	MSUB$A
Compare two character strings.	NAMEQ$
Determine the operational length of a string.	NLEN$A
Rotate string left or right.	RSTR$A
Rotate substring left or right.	RSUB$A

SYSTEM INFORMATION

General System Information

System Time Information

Return CPU time since login.	CTIM$A
Return today's date, American style.	DATE$A
Return today's date as day of year (the Julian date).	DOFY$A
Return disk time since login.	DTIM$A
Return today 's date, European (military) style.	EDAT$A
Return time of day.	TIME$A

System Status and Metering Information

Return data about a disk partition.	DS$AVL
Return data about a process's environment.	DS$ENV
Return data about file units.	DS$UNI
Return a variety of metering information.	G$METR

TIMERS

Set and read various timers.	LIMIT$
Create a timer.	TMR$CREA
Destroy a timer.	TMR$DEST
Set an absolute timer.	TMR$SABS
Set an interval timer.	TMR$SINT
Set a repetitive timer.	TMR$SREP
Cancel a timer.	TMR$CANL
Return the timer type and information.	TMR$GTMR
List the identifiers of the timers within a server.	TMR$LIST

USER INFORMATION

Validate a name against composite identification.	VALID$

USER TERMINAL

Functions

Control functions for user terminal.	C$A01
Output ASCII to the user terminal or ASR punch.	O$AA01
Inhibit or enable CONTROL-P.	BREAK$
Get next character from terminal or command file.	C1IN
Get next character from command line until carriage return.	C1IN$
Move characters from terminal or command file to memory.	CNIN$
Read a line of text from the terminal or from a command file.	COMANL
Supervise the editing of input from a terminal or a command file (callable from C).	ECL$CC
Supervise the editing of input from a terminal or a command file.	ECL$CL
Read or set erase and kill characters.	ERKL$$
Output count characters to the user terminal followed by a line feed and carriage return.	TNOU
Output count characters to the user terminal.	TNOUA
Output the 16-bit integer num to the terminal.	TOVFD$
Read one character from the user terminal into Register A.	T1IB
Read one character from the user terminal.	T1IN
Write one character from Register A to the user terminal.	T1OB

Output char to the user terminal. The data type must be a 16-bit integer in F77.	T1OU
Input decimal number.	TIDEC
Input an octal number.	TIOCT
Input a hexadecimal number.	TIHEX
Output a six-character signed decimal number.	TODEC
Output a six-character unsigned octal number.	TOOCT
Output a four-character unsigned hexadecimal number.	TOHEX
Output Carriage return and Line feed.	TONL

Input from User Terminal

Read a character.	C1IN
Read a character.	C1IN$
Read a character, suppressing echo.	C1NE$
Read a line.	CL$GET
Read a specified number of characters.	CNIN$
Read a line into a PRIMOS buffer.	COMANL
Parse a command line.	RDTK$$
Read a character (function).	T1IB
Read a character (procedure).	T1IN
Read a decimal number.	TIDEC
Read a hexadecimal number.	TIHEX
Read an octal number.	TIOCT

Output to User Terminal

Print a standard error message from PRIMOS or a PRIMOS subsystem.	ER$PRINT
Print a standard error message.	ERRPR$
Provide free-format output.	IOA$
Provide free-format output, for error messages.	IOA$ER
Write characters to terminal, followed by NEWLINE.	TNOU
Write characters to terminal.	TNOUA
Write a signed decimal number.	TODEC
Write a hexadecimal number.	TOHEX
Write a NEWLINE.	TONL
Write an octal number.	TOOCT
Write a decimal number, without spaces.	TOVFD$
Write one character from Register A.	T1OB
Write one character.	T1OU

Control Output to User Terminal

Inhibit or enable BREAK function.	BREAK$
Return information about command output settings.	CO$GET
Switch input between the terminal and a file.	COMI$$
Switch output between the terminal and a file.	COMO$$
Control the way PRIMOS treats the user terminal.	DUPLX$
Read or set the erase and kill characters.	ERKL$$
Determine if there are pending quits.	QUIT$
Check for unread terminal input characters.	TTY$IN
Clear the terminal input and output buffers.	TTY$RS

Index

SURVEY

READER RESPONSE FORM

DOC10083-1LA Subroutines Reference Guide Volume IV

Your feedback will help us continue to improve the quality, accuracy, and organization of our user publications.

1. How do you rate the document for overall usefulness?

   ___excellent ___very good ___good ___fair ___poor

2. Please rate the document in the following areas:

   Readability: ___hard to understand ___average ___very clear

   Technical level: ___too simple ___about right ___too technical

   Technical accuracy: ___poor ___average ___very good

   Examples: ___too many ___about right ___too few

   Illustrations: ___too many ___about right ___too few

3. What features did you find most useful? _____

   _____

   _____

   _____

4. What faults or errors gave you problems? _____

   _____

   _____

   _____

Name: _____ Position: _____

Company: _____

Address: _____

_____ Zip: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

First Class Permit #531 Natick, Massachusetts 01760

BUSINESS REPLY MAIL

Postage will be paid by:

Prime™

Attention: Technical Publications
Bldg 10
Prime Park, Natick, Ma. 01760