Programmer's Library Reference Manual

SR–0113 C

Cray Research, Inc.

The UNICOS operating system is derived from the AT&T UNIX System V operating system. UNICOS is also based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California.

Requests for copies of Cray Research, Inc. publications should be sent to the following address:

Cray Research, Inc.
Distribution Center
2360 Pilot Knob Road
Mendota Heights, MN 55120

NEW FEATURES

COS:

CALLCSP	Loads and executes an absolute program from a dataset
TRIMLEN	Returns the number of characters in a string
STARTSP, CLOSEV, SETSP, and ENDSP	Offer a new and better way of handling beginning-of volume (BOV) and end-of-volume (EOV) conditions on tape I/O jobs. Cray Research recommends the use of these routines over the older CONTPIO, PROCBOV, PROCEOV, and SVOLPROC routines.
AQIO routines	Permit programs to delay program execution during I/O processing and to stop processing requests already queued. New routines also allow concurrent read and write operations to execute without forcing a wait by COS.
TSMT, MTTS	Include new parameters to handle real-time clock values on different machine types
JCCYCL	Returns the machine cycle time in picoseconds

UNICOS:

ACPTBAD, SKIPBAD	Make an area of bad data on a tape available to you by transferring it to a buffer or permits you to skip over it
AQIO routines	Permit the transfer of data and the execution of other statements in a program to proceed concurrently
GETTP, SETTP	Permit positioning information to be set and received for tape files
FSUP	Writes a specified value as a blank in a formatted I/O operation
BUFTUNE	Supports the use of barriers when multitasking
TSECND	Gives timing information for a multitasked program
ACTTABLE	Returns additional accounting information, such as the Task Accounting Table, the Generic Resource Table, and Fast Secondary Storage (FSS) utilization information
GETARG	Returns a Fortran command-line argument
IARGC	Gives the number of command-line arguments for a command
ISHELL	Executes a UNICOS shell command from a program
SYMDUMP	Performs a snapshot dump of a running job
EXIT	Ends the execution of a Fortran program

This release also contains miscellaneous technical changes to numerous routines.

RECORD OF REVISION PUBLICATION NUMBER SR-0113

Requests for copies of Cray Research, Inc. publications should be directed to the Distribution Center. Comments about these publications should be directed to the following address:

CRAY RESEARCH, INC.
Technical Publications
1345 Northland Drive
Mendota Heights, Minnesota 55120

Revision	Description
March 1986	Original printing. This manual and the System Library Reference Manual, CRI publication SM-0114, obsolete the Library Reference Manual, CRI publication SR-0014. This manual supports the Cray operating system COS release 1.15 and the UNICOS release 1.0 running on CRAY X-MP and CRAY-1 computer systems.
October 1986	This manual supports COS release 1.16 and UNICOS release 2.0 running on the CRAY X-MP and CRAY-1 computer systems. Several routines are now available under UNICOS as well as COS. These include the table management routines, Fortran I/O routines, word-addressable I/O routines, multitasking routines, flowtrace routines, and the machine characteristics routines. The manual style has changed to reflect UNICOS on-line style. Miscellaneous technical and editorial changes are also included. All trademarks are now documented in the record of revision.
June 1987	This reprint with revision includes documentation to support the UNICOS release 3.0 and COS release 1.16 running on the CRAY X-MP and CRAY-1 computer systems. The following routines are now available under UNICOS: VAX conversion routines, IBM conversion routines, miscellaneous conversion routines, logical record I/O routines, and additional miscellaneous routines. The multitasking barrier routines have been added for UNICOS. A miscellaneous UNICOS libraries and routines section has been added. TCP/IP routines have been removed and are now in the TCP/IP Network Library Reference Manual, publication SR-2057. Specific changes made to the routines are documented in the *New Features* section following the table of contents. Miscellaneous technical and editorial changes are also included.
July 1988	This reprint with revision includes documentation to support the UNICOS 4.0 release and the COS 1.17 release running on the CRAY Y-MP, CRAY X-MP, and CRAY-1 computer systems. The Boolean arithmetic routines are now documented with their own pages, as are three Fortran interfaces to C routines: **GETENV**, **GETOPT**, and **UNAME**. A new set of routines (**STARTSP**, **SETSP**, **CLOSEV** and **ENDSP**) to handle tape volume switching under COS replace the obsolete set (**CONTPIO**, **CHECKTP**, **PROCBOV**, **PROCEOV**, **SWITCHV**, and **SVOLPRC**). The base set of Asynchronous Queued I/O (AQIO) routines has been ported to UNICOS, and new routines have been added to the base set on COS. Eleven new level 2 Basic Linear Algebra Subprograms (BLAS2) have been added to the scientific library routines. The **SYMDUMP** and **TSECND** routines have been added to UNICOS, and the **TRIMLEN** and **CALLCSP** routines to COS. Miscellaneous technical changes to existing routines and editorial changes to this manual are also included.

PREFACE

The Programmer's Library Reference Manual describes Fortran subprograms and functions available to users of the Cray operating systems COS and UNICOS executing on CRAY Y-MP, CRAY X-MP, and CRAY-1 computer systems. It supplements the information contained in the other manuals in the UNICOS documentation set.

The System Library Reference Manual, publication SM-0114, describes internal system subprograms, Cray Assembly Language (CAL) subprograms, and Cray Pascal subprograms used by the Pascal compiler. The Cray Y-MP, CRAY X-MP, and CRAY-1 C Library Reference Manual, publication SR-0136, describes the C libraries available under both COS and UNICOS on CRAY X-MP and CRAY-1 computer systems.

The following Cray Research, Inc. (CRI) manuals provide additional information about UNICOS and related subjects. Unless otherwise noted, all publications referenced in this manual are CRI publications.

Introductory manuals:

- UNICOS Overview for Users, publication SG-2052
- UNICOS Primer, publication SG-2010
- TCP/IP Network User Guide, publication SG-2009
- UNICOS Text Editors Primer, publication SG-2050
- UNICOS Tape Subsystem User's Guide, publication SG-2051
- UNICOS Source Code Control System (SCCS) User's Guide, publication SG-2017
- UNICOS Index for CRAY Y-MP, CRAY X-MP, and CRAY-1 Computer Systems, publication SR-2049

UNICOS reference manuals:

- UNICOS User Commands Reference Manual, publication SR-2011
- UNICOS User Commands Ready Reference, publication SQ-2056
- UNICOS System Calls Reference Manual, publication SR-2012
- UNICOS File Formats and Special Files Reference Manual, publication SR-2014
- Fortran (CFT) Reference Manual, publication SR-0009
- CFT77 Reference Manual, publication SR-0018
- CAL Assembler Version 2 Reference Manual, publication SR-2003
- Cray C Reference Manual, publication SR-2024
- UNICOS **vi** Reference Card, publication SQ-2054
- UNICOS **ed** Reference Card, publication SQ-2055
- Network Library Reference Manual, publication SR-2057

CONVENTIONS

The following conventions are used throughout UNICOS documentation:

command(1)	Refers to an entry in the UNICOS User Commands Reference Manual, publication SR-2011.
command(1BSD)	Refers to an entry in the UNICOS User Commands Reference Manual, publication SR-2011.
command(1M)	Refers to an entry in the UNICOS Administrator Commands Reference Manual, publication SR-2022.
system call(2)	Refers to an entry in the UNICOS System Calls Reference Manual, publication SR-2012.
routine(3*X*)	Refers to an entry in the appropriate CRI library reference manual The letter or letters following the number 3 indicate that the routine is either COS-only or that the routine belongs to a specific UNICOS library, as follows:

(3M)	UNICOS math library
(3SCI)	UNICOS scientific library
(3F)	UNICOS Fortran library
(3IO)	UNICOS I/O library
(3U)	UNICOS utility library
(3DB)	UNICOS debugging library

entry(4*X*)	Refers to an entry in the UNICOS File Formats and Special Files Reference Manual, publication SR-2014. The letter following the number 4 indicates the section reference.
entry(info)	Refers to an entry in the info section, which contains topical information that is not available in the UNICOS on-line manuals. The info man pages are not published in hard-copy form.

All sections begin with an entry called **intro**, and the entries that follow the **intro** page are alphabetized. Some entries may describe several routines. In such cases, the entry is usually alphabetized under its major name.

In this manual, **bold** indicates all literal strings, including command names, directory names, file names, path names, library routine names, man page entry names, options, shell or system variable code names, system call names, C structures, and C reserved words.

Italic indicates variable information usually supplied by you and words or concepts being defined.

All entries are based on the following common format; however, most entries contain only some of these parts:

NAME shows the name of the entry and briefly states its function.

SYNOPSIS presents the syntax of the routine. The following conventions are used in this section:

Brackets [ ] around an argument indicate that the argument is optional.

DESCRIPTION discusses the entry in detail.

IMPLEMENTATION provides details for using the command or routine with specific machines or operating systems; normally this will tell you under which operating system the routine is implemented.

NOTES points out items of particular importance.

CAUTIONS describes actions that can destroy data or produce undesired results.

WARNINGS describes actions that can harm people, damage equipment, or damage system software.

EXAMPLES shows examples of usage.

FILES lists files that are either part of the entry or related to it.

RETURN VALUE describes possible error returns.

MESSAGES describes the informational, diagnostic, and error messages that may appear.

BUGS indicates known bugs and deficiencies.

SEE ALSO lists entries that contain related information and specifies the manual title for each entry.

All entries in this manual that are applicable to your Cray computer system are available on-line through the **man**(1) command. To retrieve an entry, type the following, substituting the desired entry name for *entry*:

man *entry*

If there is more than one entry with the same name, all entries with that name will be printed. To retrieve the entry for a particular section, type the following, substituting the desired section name for *section* and the desired entry name for *entry*:

man *section entry*

For further information on the **man** command, see **man**(1).

READER COMMENTS

If you have any comments about the technical accuracy, content, or organization of this manual, please tell us. You can contact us in any of the following ways:

- Call our Technical Publications department at (612) 681-5729 during normal business hours (Central Time).

- Send us electronic mail from a UNICOS or UNIX system, using one of the following electronic mail addresses:

  ihnp4!cray!publications
  or
  sun!tundra!hall!publications

- Use the postage-paid Reader's Comment form at the back of this manual.

- Write to us at the following address:

  Cray Research, Inc.
  Technical Publications Department
  1345 Northland Drive
  Mendota Heights, MN 55120

We value your comments and will respond to them promptly.

CONTENTS

3. COS DATASET MANAGEMENT SUBPROGRAMS

4. LINEAR ALGEBRA SUBPROGRAMS

5. FAST FOURIER TRANSFORM ROUTINES

6. SEARCH ROUTINES

7. SORTING ROUTINES

8. CONVERSION SUBPROGRAMS

9. PACKING ROUTINES

10. BYTE AND BIT MANIPULATION ROUTINES

11. HEAP MANAGEMENT AND TABLE MANAGEMENT

12. I/O ROUTINES

13. DATASET UTILITY ROUTINES

14. MULTITASKING ROUTINES

15. TIMING ROUTINES

16. PROGRAMMING AID ROUTINES

17. SYSTEM INTERFACE ROUTINES

18. INTERFACE TO C LIBRARY ROUTINES

19. MISCELLANEOUS UNICOS ROUTINES

1. INTRODUCTION

This manual describes Fortran programming subprograms provided in the standard COS libraries $ARLIB, $FTLIB, $IOLIB, $SCILIB, $SYSLIB, and $UTLIB, and those subprograms supported by UNICOS on the CRAY Y-MP, CRAY X-MP, and CRAY-1 computer systems. The Cray Assembly Language (CAL) subprograms and subprograms called by code generated by the Cray Fortran compiler or the Cray Pascal compiler are described in the System Library Reference Manual, publication SM-0114. Routines generated in the form of in-line code are generally not included in this manual, but they are described in the Fortran (CFT) Reference Manual, publication SR-0009, and the CFT77 Reference Manual, publication SR-0018.

The routines are divided into functional sections. A brief description of each section follows:

Section	Description
1	Introduction
2	Common Mathematical Subprograms - General arithmetic, exponentiation, logarithmic, trigonometric, character, type conversion, and Boolean functions
3	COS Dataset Management Subprograms - COS Job Control Language (JCL) routines
4	Linear Algebra Subprograms - Basic linear algebra, linear recurrence, matrix inverse and multiplication, filter, gather/scatter, and LINPACK/EISPACK routines
5	Fast Fourier Transform Routines - Computing Fourier analysis and Fourier synthesis routines
6	Search Routines - Maximum and minimum search and vector search routines
7	Sorting Routines - **ORDERS** optimized sort routine
8	Conversion Subprograms - Foreign dataset conversion (IBM, CDC, and VAX), numeric conversion, and miscellaneous conversion routines
9	Packing Routines - Packing and unpacking data routines
10	Byte and Bit Manipulation Routines - Routines for comparing, moving, and searching at the element level
11	Heap Management and Table Management Routines - Routines for manipulating and managing memory within heaps and tables
12	I/O Routines - Dataset positioning, auxiliary **NAMELIST**, logical record, random access dataset, and output suppression routines
13	Dataset Utility Routines - Routines for positioning, copying, and skipping datasets
14	Multitasking Routines - Task, lock, event, and history trace buffer routines
15	Timing routines - Time-stamp and time/date routines
16	Programming Aids Routines - Flowtrace, traceback, dump, Exchange Package processing, and hardware performance routines
17	System Interface Routines - JCL symbol, control statement processing, job control, floating-point interrupt, bidirectional memory transfer, and special purpose interface routines

Section	Description
18	Interfaces to C Library Routines - C library interface routines available under UNICOS and documented in the CRAY Y-MP, CRAY X-MP, and CRAY-1 C Library Reference Manual, publication SR-0136 and the UNICOS System Calls Reference Manual, publication SR-2012.
19	Miscellaneous UNICOS Routines - X Window System routines and libraries.

SUBPROGRAM CLASSIFICATION

Unless otherwise noted, all routines in this manual are described as Fortran subroutines or functions. In some cases (e.g., **SECOND**), the routine may be called as either a subroutine or a function. The Fortran compilers will, however, enforce consistency in any one compilation unit.

Programs written in C can call library functions intended for use by Fortran programs. The C programmer is responsible for passing arguments by address and not by value, as is the normal case in C.

C programs can also be written to accommodate Fortran users. Such programs must be written to accept arguments passed by address rather than passed by value, as in the normal case in C.

Pascal programs can call library functions intended for use by Fortran programs. Similarly, Fortran codes can invoke subroutines and functions written in Pascal. Unlike C, the Pascal compiler passes all arguments by address, and supports several predefined conversion functions to facilitate communication with Fortran routines. See the Pascal Reference Manual, publication SR-0060, for information regarding parameter passing, data formats, and restrictions.

LINKAGE METHODS

The externally-callable library routines are accessed by one of two methods: call-by-address or call-by-value. Subroutines are always called by address. Fortran accesses intrinsic library functions or user functions named in a **VFUNCTION** directive in either call-by-address or call-by-value mode, depending on context.

In call-by-address mode, addresses of arguments are stored sequentially in memory. Functions return their results in registers. Subroutines return results through their argument lists (for information on the calling sequence, see the Macros and Opdefs Reference Manual, CRI publication SR-0012).

In call-by-value mode, arguments are loaded into either scalar (S) or vector (V) registers, and the function returns its result in S1 or V1. S2 or V2 is used for complex or double-precision functions. Vector functions must also have the vector length present in the vector length (VL) register.

Linkage macros generate code to handle subprogram linkage between compiled routines and CAL-assembled routines. These linkage macros and their uses follow.

Macro	Description
CALL	Provides linkage to call-by-address routines
CALLV	Provides linkage to call-by-value routines
ENTER	Reserves space for parameter addresses, saves B and T registers, and sets up traceback linkage
EXIT	Initiates a return from a routine to its caller and restores any B or T registers not considered scratch

Linkage macros should be used whenever possible to maintain compatibility with future CRI software. See the Macros and Opdefs Reference Manual for detailed descriptions of linkage macros and linkage conventions.

All Cray library subroutines can use any of the A, S, V, VL, VM, B70 through B77, and T70 through T77 registers as scratch registers; therefore, the calling routine should not depend on any of these registers being preserved. These routines, however, preserve the contents of registers B01 through B65 and T00 through T67 (all registers are numbered in octal).

NOTE

CRI reserves the right to make future use of any of the A, S, V, VL, VM, B66-B77, and T70-T77 registers in any library subroutine. You cannot depend on the contents of these registers being preserved in any library routine.

CRI also reserves subroutine names beginning with the characters I00 for internal use only.

2. COMMON MATHEMATICAL SUBPROGRAMS

This section is divided into the following categories of mathematical subprograms:

- General arithmetic functions
- Exponential and logarithmic functions
- Trigonometric functions
- Character functions
- Type conversion functions
- Boolean functions

NOTE

In general, real functions have no prefix, integer functions are prefixed with I, double-precision functions are prefixed with D, and complex functions are prefixed with C (for example **ABS, IABS, DABS,** and **CABS**). Arguments are given in their type: *real, integer, complex, logical, Boolean*, and *double* (double-precision); results are given as *r, i, z, l, b*, and *d* for real, integer, complex, logical, Boolean, and double-precision, respectively. Functions with a type different from their arguments are noted. Real functions are usually the same as the entry name.

IMPLEMENTATION

All routines in this section are available to users of both the COS and UNICOS operating systems.

GENERAL ARITHMETIC FUNCTIONS

The general arithmetic functions are based upon ANSI standards, with the exception of the pseudo-random number routines (**RANF, RANGET**, and **RANSET**), which are CRI extensions.

The following table contains the purpose, name, and entry of each general arithmetic function.

In the routine descriptions, complex arguments are represented such that

$$x = x_r + i * x_i$$

where x_r is the real portion and $i * x_i$ is the imaginary portion of the complex number. Arguments and results are of the same type unless otherwise indicated.

Base values raised to a power and 64-bit integer division are implicitly called from Fortran. Details on calls from CAL are documented in the System Library Reference Manual, publication SM-0114.

General Arithmetic Routines		
Purpose	Name	Entry
Compute absolute value for real, integer, double-precision, and complex numbers	**ABS** **IABS** **DABS** **CABS**	**ABS**
Compute the imaginary portion of a complex number	**AIMAG**	**AIMAG**
Compute real and double-precision truncation	**AINT** **DINT**	**AINT**
Compute the conjugate of a complex number	**CONJG**	**CONJG**
Find the positive difference of real, integer, or double-precision numbers	**DIM** **IDIM** **DDIM**	**DIM**
Compute the double-precision product of two real numbers	**DPROD**	**DPROD**
Remainder of x_1/x_2 for integer, real, and double-precision numbers	**MOD** **AMOD** **DMOD**	**MOD**
Find the nearest whole number for real and double-precision numbers	**ANINT** **DNINT**	**ANINT**
Find the nearest integer for real and double-precision numbers	**NINT** **IDNINT**	**NINT**
Obtain and establish a pseudo-random number seed	**RANGET** **RANSET**	**RAN**
Obtain the first or next number in a series of pseudo-random numbers	**RANF**	
Transfer the sign of a real, integer, or double-precision number	**SIGN** **ISIGN** **DSIGN**	**SIGN**

EXPONENTIAL AND LOGARITHMIC FUNCTIONS

The CRI exponential and logarithmic functions are similar to the ANSI standard functions. Each function has variations for real, double-precision, and complex values except the common logarithm function, which only addresses real and double-precision values. Complex arguments are represented such that

$$x = x_r + i * x_i$$

where x_r is the real portion and $i * x_i$ is the imaginary portion of the complex number.

The following table contains the purpose, name, and entry of each exponential and logarithmic function.

Exponential and Logarithmic Functions		
Purpose	Name	Entry
Compute the natural logarithm for real, double-precision, and complex numbers	**ALOG** **DLOG** **CLOG**	**ALOG**
Compute the common logarithm for real and double-precision numbers	**ALOG10** **DLOG10**	**ALOG10**
Compute exponents for real, double-precision, and complex numbers	**EXP** **DEXP** **CEXP**	**EXP**
Compute the square root for real, double-precision, and complex numbers	**SQRT** **DSQRT** **CSQRT**	**SQRT**

TRIGONOMETRIC FUNCTIONS

The trigonometric functions are based on the ANSI standard, except for the cotangent function, which is a CRI extension.

The following table contains the purpose, name, and entry of each trigonometric function.

Trigonometric Functions		
Purpose	Name	Entry
Compute the arcsine for real and double-precision numbers	**ASIN** **DASIN**	**ASIN**
Compute the arccosine for real and double-precision numbers	**ACOS** **DACOS**	**ACOS**
Compute the arctangent with one real or double-precision argument	**ATAN** **DATAN**	**ATAN**
Compute the arctangent with two real or double-precision arguments	**ATAN2** **DATAN2**	**ATAN2**
Compute the cosine for real, double-precision, and complex numbers	**COS** **DCOS** **CCOS**	**COS**
Compute the hyperbolic cosine for real or double-precision numbers	**COSH** **DCOSH**	**COSH**
Compute the sine for real, double-precision, and complex numbers	**SIN** **DSIN** **CSIN**	**SIN**
Compute the hyperbolic sine for real or double-precision numbers	**SINH** **DSINH**	**SINH**
Compute the tangent real and double-double-precision numbers	**TAN** **DTAN**	**TAN**
Compute the cotangent for real and double-precision numbers	**COT** **DCOT**	**COT**
Compute the hyperbolic tangent for real or double-precision numbers	**TANH** **DTANH**	**TANH**

CHARACTER FUNCTIONS

Character functions compare strings, determine the lengths of strings, and return the index of a substring within a string. The character functions are ANSI standard functions.

The comparison functions return a logical value of true or false when two character arguments are compared according to the ANSI collating sequence. These four functions are found under the entry LGE(3F).

The routines for determining the length of a string and the index of a substring are found under the entries LEN(3F) and INDEX(3F), respectively.

TYPE CONVERSION FUNCTIONS

Type conversion functions change the type of an argument. The following table contains the purpose, name, and entry of each type conversion routine.

In the routine description, complex arguments are represented such that $x = x_r + i * x_i$. Arguments and results are of the same type unless indicated otherwise.

<table>
<tr><th colspan="3">Type Conversion Routines</th></tr>
<tr><th>Purpose</th><th>Name</th><th>Entry</th></tr>
<tr><td>Convert type character to integer</td><td>ICHAR</td><td rowspan="2">CHAR</td></tr>
<tr><td>Convert type integer to character</td><td>CHAR</td></tr>
<tr><td>Convert to type complex</td><td>CMPLX</td><td>CMPLX</td></tr>
<tr><td>Convert to type double-precision</td><td>DBLE</td><td rowspan="2">DBLE</td></tr>
<tr><td>Convert integer to double-precision</td><td>DFLOAT</td></tr>
<tr><td>Convert to type integer</td><td>INT
IFIX
IDINT</td><td>INT</td></tr>
<tr><td>Convert a 64-bit integer to a 24-bit integer</td><td>INT24</td><td rowspan="2">INT24</td></tr>
<tr><td>Convert a 24-bit integer to a 64-bit integer</td><td>LINT</td></tr>
<tr><td>Convert to type real</td><td>REAL
FLOAT
SNGL</td><td>REAL</td></tr>
</table>

BOOLEAN FUNCTIONS

The Boolean functions perform logical operations and bit manipulations.

The scalar subprograms in the following table are external versions of Fortran in-line functions. These functions can be passed as arguments to user-defined functions. They are all called by address; results are returned in register S1. All Boolean functions are CRI extensions.

Boolean Arithmetic Routines		
Purpose	Name	Entry
Compute the logical product	**AND**	**AND**
Compute the logical complement	**COMPL**	**COMPL**
Compute the logical equivalence	**EQV**	**EQV**
Count the number of leading 0 bits	**LEADZ**	**LEADZ**
Return a bit mask	**MASK**	**MASK**
Compute the logical difference (same as XOR)	**NEQV**	**NEQV**
Compute the logical sum	**OR**	**OR**
Count the number of bits set to 1	**POPCNT**	**POPCNT**
Compute the bit population parity	**POPPAR**	**POPPAR**
Perform a left circular shift	**SHIFT**	**SHIFT**
Perform a left shift with zero fill	**SHIFTL**	**SHIFTL**
Perform a right shift with zero fill	**SHIFTR**	**SHIFTR**
Compute the logical difference (same as NEQV)	**XOR**	**XOR**

NAME

ABS, IABS, DABS, CABS – Computes absolute value (Cray Fortran instrinsic function)

SYNOPSIS

r=**ABS**(*real*)

i=**IABS**(*integer*)

d=**DABS**(*double*)

r=**CABS**(*complex*)

DESCRIPTION

These functions evaluate $y=|x|$. The argument range for **ABS, IABS,** and **DABS** is $|x| <$ inf. **CABS** has an argument range of $|x_r|, |x_i| <$ inf.

ABS is the generic function name. **ABS, IABS,** and **DABS** are inline Cray Fortran code.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ACOS, DACOS – Computes the arccosine (Cray Fortran intrinsic function)

SYNOPSIS

r=**ACOS**(*real*)

d=**DACOS**(*double*)

DESCRIPTION

ACOS (generic name) and **DACOS** solve the equation $y=\arccos(x)$. The range for the real and double-precision arguments is $|x| \leq 1$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

AIMAG – Computes the imaginary portion of a complex number

SYNOPSIS

r=AIMAG(*complex*)

DESCRIPTION

This real function evaluates $y=x_i$. The argument ranges are $|x_r|, |x_i| < \mathrm{inf}$. AIMAG is in-line Cray Fortran code.

EXAMPLE

```
PROGRAM AIMTEST
REAL RESULT
RESULT=AIMAG((1.0,2.0))
PRINT *, RESULT
STOP
END
```

The preceding program gives the imaginary portion of the complex number (1.0,2.0). After running the program, RESULT=2.0.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

AINT, DINT – Computes real and double-precision truncation (Cray Fortran intrinsic function)

SYNOPSIS

r=AINT(*real*)

d=DINT(*double*)

DESCRIPTION

AINT (generic name) is in-line Fortran code. These ANSI functions evaluate $y=[x]$ with no rounding. The argument range for **AINT** is $|x| < 2^{46}$, and the range for **DINT** is $|x| < 2^{95}$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ALOG, DLOG, CLOG – Computes the natural logarithm (Cray Fortran intrinsic function)

SYNOPSIS

r=**ALOG**(*real*)

d=**DLOG**(*double*)

z=**CLOG**(*complex*)

DESCRIPTION

LOG (generic name) evaluates the following equation for real, double-precision, and complex arguments:

$y = \ln(x)$

The argument range is $0 < x < \text{inf}$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ALOG10, DLOG10 – Computes a common logarithm (Cray Fortran intrinsic function)

SYNOPSIS

r=**ALOG10**(*real*)

d=**DLOG10**(*double*)

DESCRIPTION

LOG10 (generic name) evaluates the following equation:

$y=\log(x)$

The argument range is $0<x<\text{inf}$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

AND – Computes the logical product

SYNOPSIS

l=**AND**(*logical*,*logical*)
b=**AND**(*arg*,*arg*)

DESCRIPTION

arg Argument of type integer, real, or Boolean

When given two arguments of type logical, **AND** computes a logical product and returns a logical result. When given two arguments of type integer, real, or Boolean, **AND** computes a bit-wise logical product and returns a Boolean result. The truth tables below show both the logical product and bit-wise logical product.

Logical 1	Logical 2	(Logical 1) **AND** (Logical 2)
T	T	T
T	F	F
F	T	F
F	F	F

Bit 1	Bit 2	(Bit 1) **AND** (Bit 2)
1	1	1
1	0	0
0	1	0
0	0	0

EXAMPLES

The following section of Fortran code shows the **AND** function used with two arguments of type logical.

```
LOGICAL L1, L2, L3
...
L3 = AND(L1,L2)
```

The following section of Fortran code shows the **AND** function used with two arguments of type integer. The bit patterns of the arguments and result are also shown below. For clarity, an 8-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2, I3
...
I3 = AND(I1,I2)
```

0	0	0	0	1	1	0	0

I1

0	0	0	0	1	0	1	0

I2

0	0	0	0	1	0	0	0

I3

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ANINT, DNINT – Finds the nearest whole number (Cray Fortran intrinsic function)

SYNOPSIS

r=**ANINT**(*real*)

d=**DNINT**(*double*)

DESCRIPTION

ANINT (generic name) finds the nearest whole number for real and double-precision numbers using the following equations.

$$y = [\ x + .5\ ] \text{ if } x \geq 0$$

$$y = [\ x - .5\ ] \text{ if } x < 0$$

The argument range for **ANINT** is $|\ x\ | < 2^{46}$. The range for **DNINT** is $|\ x\ | < 2^{95}$.

ANINT and **DNINT** are type real and type double-precision functions, respectively.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ASIN, DASIN – Computes the arcsine (Cray Fortran intrinsic function)

SYNOPSIS

r=ASIN(*real*)

d=DASIN(*double*)

DESCRIPTION

ASIN (generic name) and **DASIN** solve the equation $y=\arcsin(x)$. The range for both real and double-precision arguments is $|x| \leq 1$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ATAN, DATAN – Computes the arctangent for single argument (Cray Fortran intrinsic function)

SYNOPSIS

r=**ATAN**(*real*)

d=**DATAN**(*double*)

DESCRIPTION

ATAN (generic name) and DATAN solve for the equation with one real argument or one double-precision argument as follows:

$y=\arctan(x)$

The argument must be in the range $|x| < \mathrm{inf}$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ATAN2, DATAN2 – Computes the arctangent for two arguments (Cray Fortran intrinsic function)

SYNOPSIS

r=**ATAN2**(*real*,*real*)

d=**DATAN2**(*double*,*double*)

DESCRIPTION

ATAN2 (generic name) and **DATAN2** solve for two real or double-precision arguments as follows:

$$y = \arctan(x_1/x_2)$$

For real arguments, the range is $|x_1|, |x_2| < \inf$, and x_1 and x_2 are not both zero.

For double-precision arguments, the range is $|x_1|, |x_2| < \inf$, and x_1 and x_2 are not both zero.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

CHAR, ICHAR – Converts integer to character and vice versa (Cray Fortran intrinsic function)

SYNOPSIS

ch=**CHAR**(*integer*)
ch=**CHAR**(*boolean*)

i=**ICHAR**(*char*)

DESCRIPTION

CHAR (inline Fortran code) and **ICHAR** are inverse functions. **CHAR** (type character) converts an integer or Boolean argument to a character specified by the ASCII collating sequence. Type conversion routines assign the appropriate type to Boolean arguments without shifting or manipulating the bit patterns they represent. For example, **CHAR**(*i*) returns the *i*th character in the collating sequence. *integer* must be in the range 0 to 255.

ICHAR (type integer) converts a character to an integer based on the character position in the collating sequence.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

CMPLX – Converts to type complex (Cray Fortran intrinsic function)

SYNOPSIS

c=**CMPLX**(*arg* $_1$[,*arg* $_2$])

DESCRIPTION

CMPLX (type complex) converts one or two arguments into type complex. Complex and 24-bit integer arguments use a single argument. Integer, Boolean, real, and double-precision arguments can use either one or two arguments. Type conversion routines assign the appropriate type to Boolean arguments without shifting or manipulating the bit patterns they represent.

If two arguments are used, they must be of the same type. The following cases represent the evaluation of **CMPLX** when using two arguments:

> **CMPLX(I,J)** gives the value **FLOAT(I)+i*FLOAT(J)**
> **CMPLX(x,y)** gives the complex value **x+i*y**

The following cases represent the evaluation of **CMPLX** when using one argument:

> **CMPLX(X)** gives the value **X+i*0**
> **CMPLX(I)** gives the value **FLOAT(I)+i*0**
> **CMPLX(C)** where **C** is a complex number, gives the complex value **x+i*y**; that is, **CMPLX(C)=C.**

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

COMPL – Computes the logical complement

SYNOPSIS

l=COMPL(*logical*)
b=COMPL(*arg*)

DESCRIPTION

arg Argument of type integer, real, or Boolean

When given an argument of type logical, **COMPL** computes a logical complement and returns a logical result. When given an argument of type integer, real, or Boolean, **COMPL** computes a bit-wise logical complement and returns a Boolean result. The truth tables below show both the logical complement and bit-wise logical complement.

Logical	COMPL (Logical)
T	F
F	T

Bit	COMPL (Bit)
1	0
0	1

EXAMPLES

The following section of Fortran code shows the **COMPL** function used with an argument of type logical.

```
LOGICAL L1, L2
...
L2 = COMPL(L1)
```

The following section of Fortran code shows the **COMPL** function used with an argument of type integer. The bit patterns of the argument and result are also shown below. For clarity, an 8-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2
...
I2 = COMPL(I1)
```

1	1	1	1	0	0	0	0

I1

0	0	0	0	1	1	1	1

I2

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

CONJG – Computes the conjugate of a complex number

SYNOPSIS

z=**CONJG**(*complex*)

DESCRIPTION

The complex function **CONJG** evaluates $y = x_r - i * x_i$. The argument range is $|x_r|, |x_i| < \mathrm{inf}$. **CONJG** is in-line Cray Fortran code.

EXAMPLE

```
PROGRAM CONTEST
COMPLEX ARG, RESULT
ARG=(3.0,4.0)
RESULT=CONJG(ARG)
PRINT *,RESULT
STOP
END
```

The preceding program gives **RESULT=(3.0,-4.0)**.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

COS, DCOS, CCOS – Computes the cosine (Cray Fortran intrinsic function)

SYNOPSIS

r=COS(*real*)

d=DCOS(*double*)

z=CCOS(*complex*)

DESCRIPTION

COS (generic name) solves for the equation *y*=cos(*x*). The ranges for the real, double-precision, and complex functions are as follows:

For COS:

$|x| < 2^{24}$

For DCOS:

$|x| < 2^{48}$

For CCOS:

$|x_r| < 2^{24}$, $|x_i| < 2^{13} * \ln 2$

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

COSH, DCOSH – Computes the hyperbolic cosine (Cray Fortran intrinsic function)

SYNOPSIS

r=COSH(*real*)

d=DCOSH(*double*)

DESCRIPTION

COSH (generic name) and **DCOSH** solve the equation y=cosh(x). The hyperbolic cosine functions have a real or double-precision argument in the range of $|x| < 2^{13} * \ln 2$.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

COT, DCOT – Computes the cotangent (Cray Fortran intrinsic function)

SYNOPSIS

r=COT(*real*)

d=DCOT(*double*)

DESCRIPTION

COT (generic name) solves for the equation $y = \cot(x)$. The range for the real and double-precision arguments is $|x| < 2^{24}$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

DBLE, DFLOAT – Converts to type double-precision

SYNOPSIS

d=**DBLE**(*arg*)

d=**DFLOAT**(*integer*)

DESCRIPTION

DBLE (type double-precision, Cray Fortran intrinsic function) converts complex, integer, 24-bit integer, Boolean, real, and double-precision arguments into type double-precision. Type conversion routines assign the appropriate type to Boolean arguments without shifting or manipulating the bit patterns they represent. The range for real, double-precision, and Boolean arguments is $|x| <$ inf.

Complex arguments have a range of $|x_r| <$ inf. (for complex arguments $x = x_r + i * x_i$).

DFLOAT converts integer arguments to floating-point double-precision variables.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

DIM, IDIM, DDIM – Positive difference of two numbers (Cray Fortran intrinsic function)

SYNOPSIS

r=**DIM**(*real*,*real*)

i=**IDIM**(*integer*,*integer*)

d=**DDIM**(*double*,*double*)

DESCRIPTION

These functions evaluate two numbers and, depending on their magnitude, subtract them. The result is a positive difference. **DIM** (generic function) solves for

$y=x_1-x_2$ if $x_1>x_2$

$y=0$ if $x_1\leq x_2$

The range for all positive difference functions is $|x_1|,|x_2|<$inf. **DIM** and **IDIM** are in-line code functions.

EXAMPLE

```
      PROGRAM DIMTEST
      INTEGER A,B,C,D,E
      A=77
      B=10
      C=IDIM(A,B)
      WRITE 1,A,B,C
1     FORMAT(I2,'POSITIVE DIFFERENCE ',I2,' EQUALS ', I2)
      D=IDIM(B,A)
      WRITE 2,B,A,D
2     FORMAT(I2,'POSITIVE DIFFERENCE ',I2,' EQUALS ',I2)
      STOP
      END
```

The preceding program gives the following output.

```
77 POSITIVE DIFFERENCE 10 EQUALS 67
10 POSITIVE DIFFERENCE 77 EQUALS  0
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

DPROD – Computes double-precision product of two real numbers

SYNOPSIS

d=**DPROD**(*real*,*real*)

DESCRIPTION

This double-precision function evaluates $y = x_1 * x_2$. The argument range is $|x_1|, |x_2| < \mathrm{inf}$. **DPROD** is an in-line code function.

EXAMPLE

```
PROGRAM DOUBT
REAL X,Y
DOUBLE PRECISION Z
X=5.0
Y=6.0
Z=DPROD(X,Y)
PRINT *, Z
STOP
END
```

The preceding program gives **Z** to be the double-precision number 30.0 (or in Fortran, 30.D0).

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

EQV – Computes the logical equivalence

SYNOPSIS

l=**EQV**(*logical*,*logical*)
b=**EQV**(*arg*,*arg*)

DESCRIPTION

arg Argument of type integer, real, or Boolean

When given two arguments of type logical, **EQV** computes a logical equivalence and returns a logical result. When given two arguments of type integer, real, or Boolean, **EQV** computes a bit-wise logical equivalence and returns a Boolean result. The truth tables below show both the logical equivalence and bit-wise logical equivalence.

Logical 1	Logical 2	(Logical 1) **EQV** (Logical 2)
T	T	T
T	F	F
F	T	F
F	F	T

Bit 1	Bit 2	(Bit 1) **EQV** (Bit 2)
1	1	1
1	0	0
0	1	0
0	0	1

EXAMPLES

The following section of Fortran code shows the **EQV** function used with two arguments of type logical.

```
LOGICAL L1, L2, L3
...
L3 = EQV(L1,L2)
```

The following section of Fortran code shows the **EQV** function used with two arguments of type integer. The bit patterns of the arguments and result are also shown below. For clarity, an 8-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2, I3
...
I3 = EQV(I1,I2)
```

0	0	0	0	1	1	0	0

I1

0	0	0	0	1	0	1	0

I2

1	1	1	1	1	0	0	1

I3

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

EXP, DEXP, CEXP – Computes exponential function (Cray Fortran intrinsic function)

SYNOPSIS

r=**EXP**(*real*)

d=**DEXP**(*double*)

z=**CEXP**(*complex*)

DESCRIPTION

EXP (generic name) evaluates $y=e^x$ with real, double-precision, and complex arguments. The argument ranges are as follows:

For EXP:

$|x| < 2^{13} * \ln(2)$

For DEXP:

$|x| < 2^{13} * \ln(2)$

For CEXP:

$|x_r| < 2^{13} * \ln(2)$

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

INDEX – Determines index location of a character substring within a string (Cray Fortran intrinsic function)

SYNOPSIS

i=**INDEX**(*string,substring*)

DESCRIPTION

The integer function **INDEX** takes Fortran character string arguments and returns an integer index into that string. If *substring* is not located within *string*, a value of 0 is returned. If there is more than one occurrence of *substring*, only the first index is returned. *string* and *substring* can be any legal Fortran character string.

EXAMPLE

```
PROGRAM INDEX1
CHARACTER*23,A
CHARACTER*13,B
A='CRAY X-MP SUPERCOMPUTER'
B='SUPERCOMPUTER'
I=INDEX(A,B)
PRINT *, I
STOP
END
```

The preceding program returns the index number of the substring **SUPERCOMPUTER** as I=11.

```
PROGRAM INDEX2
CHARACTER*20,A
CHARACTER*6,B
A='CRAY-1 SUPERCOMPUTER'
B='CRAY-1'
I=INDEX(A,B)
PRINT *, I
STOP
END
```

The preceding program returns the index number of the substring **CRAY-1** as I=1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

INT, IFIX, IDINT – Converts to type integer (Cray Fortran intrinsic function)

SYNOPSIS

i=**INT**(*argl*)

i=**IFIX**(*real*)
i=**IFIX**(*boolean*)

i=**IDINT**(*double*)

DESCRIPTION

argl Argument of type integer, complex, real, or Boolean

These type integer functions (all are in-line Fortran code) convert specified types to type integer by truncating toward 0 (the fraction is lost). **INT** is the generic name.

The ranges for **INT** are as follows: for complex and real arguments, $|x_r| < 2^{46}$; for 24-bit integer arguments, $|x| < 2^{23}$; and for integer and Boolean arguments $|x| < 2^{63}$. Type conversion routines assign the appropriate type to Boolean arguments without shifting or manipulating the bit patterns they represent.

The range for **IFIX** real and Boolean arguments is $|x_r| < 2^{46}$.

The range for **IDINT** arguments is $|x_r| < 2^{63}$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

INT24, LINT – Converts 64-bit integer to 24-bit integer and vice versa (Cray Fortran intrinsic function)

SYNOPSIS

i24=INT24(*integer*)
i24=INT24(*boolean*)

i24=LINT(*integer*)

DESCRIPTION

INT24 and **LINT** (type integer) are inverse functions. Both functions are CRI extensions to the ANSI standard, and both are in-line code.

INT24 converts an integer argument into a 24-bit integer. **LINT** converts a 24-bit integer back into an integer type. The range for all arguments is $|x| < 2^{23}$. *i24* represents a 24-bit integer result.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

LEADZ – Counts the number of leading 0 bits (Cray Fortran intrinsic function)

SYNOPSIS

i=**LEADZ**(*arg*)

DESCRIPTION

arg Argument of type integer, real, logical, or Boolean

When given an argument of type integer, real, logical, or Boolean, **LEADZ** counts the number of leading 0 bits in the 64-bit representation of the argument.

NOTES

The bit representation of the logical data type is not consistent among Cray machines. See the following manuals for further details: Fortran (CFT) Reference Manual, publication SR-0009; CRAY-2 Fortran (CFT2) Reference Manual, publication SR-2007; CFT77 Reference Manual, publication SR-0018.

LEADZ(0) is equal to 64.

EXAMPLES

The following section of Fortran code shows the **LEADZ** function used with an argument of type integer. The bit pattern of the argument and the value of the result are also shown below. For clarity, a 16-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2 ... I2 = LEADZ(I1)
```

0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	0

I1

The **LEADZ** function returns the value **5** to the integer variable **I2**.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

LEN – Determines the length of a character string (Cray Fortran intrinsic function)

SYNOPSIS

i=**LEN**(*string*)

DESCRIPTION

The integer function **LEN** takes Fortran character string arguments and returns an integer length. *string* can be any valid Fortran character string. **LEN** is an in-line code function.

EXAMPLE

```
PROGRAM LENTEST
I=LEN('l...+....1....+....2....+....3....+..')
PRINT *,I
STOP
END
```

The preceding program returns the length of the character string; I=37.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

LGE, LGT, LLE, LLT – Compares strings lexically (Cray Fortran intrinsic function)

SYNOPSIS

l=LGE(*string1*,*string2*)

l=LGT(*string1*,*string2*)

l=LLE(*string1*,*string2*)

l=LLT(*string1*,*string2*)

DESCRIPTION

Each of the these type logical functions takes two character string arguments and return a logical value. *string1* and *string2* are compared according to the ASCII collating sequence, and the resulting true or false value is returned. Arguments can be any valid character string. If the strings are of different lengths, the function treats the shorter string as though it were blank-filled on the right to the length of the longer string.

The defining equation for each function is as follows:

For LGE, logic = $a_1 \geq a_2$.

For LGT, logic = $a_1 > a_2$.

For LLE, logic = $a_1 \leq a_2$.

For LLT, logic = $a_1 < a_2$

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

MOD, AMOD, DMOD – Computes remainder of x_1/x_2 (Cray Fortran intrinsic function)

SYNOPSIS

i=**MOD**(*integer*,*integer*)

r=**AMOD**(*real*,*real*)

d=**DMOD**(*double*,*double*)

DESCRIPTION

MOD (generic name) evaluates the equation $y=x_1-x_2[x_1/x_2]$. **AMOD** is an in-line code function.

The argument range for each function is as follows:

For **MOD**:

$$|x_1|<2^{63},$$
$$0<|x_2|<2^{63}\ ,2^{-63}<|x_1/x_2|<2^{63}$$

For **AMOD**:

$$|x_1|<2^{47}$$
$$0<|x_2|<2^{47}\ ,2^{-47}<|x_1/x_2|<2^{47}$$

For **DMOD**:

$$|x_1|<2^{95}$$
$$0<|x_2|<2^{95}\ ,2^{-95}<|x_1/x_2|<2^{95}$$

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

NEQV – Computes the logical difference

SYNOPSIS

l=NEQV(*logical,logical*)
b=NEQV(*arg,arg*)

DESCRIPTION

arg Argument of type integer, real, or Boolean

When given two arguments of type logical, **NEQV** computes a logical difference and returns a logical result. When given two arguments of type integer, real, or Boolean, **NEQV** computes a bit-wise logical difference and returns a Boolean result. The truth tables below show both the logical difference and bit-wise logical difference.

Logical 1	Logical 2	(Logical 1) **NEQV** (Logical 2)
T	T	F
T	F	T
F	T	T
F	F	F

Bit 1	Bit 2	(Bit 1) **NEQV** (Bit 2)
1	1	0
1	0	1
0	1	1
0	0	0

EXAMPLES

The following section of Fortran code shows the **NEQV** function used with two arguments of type logical.

```
LOGICAL L1, L2, L3
...
L3 = NEQV(L1,L2)
```

The following section of Fortran code shows the **NEQV** function used with two arguments of type integer. The bit patterns of the arguments and result are also shown below. For clarity, an 8-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2, I3
...
I3 = NEQV(I1,I2)
```

0	0	0	0	1	1	0	0

I1

0	0	0	0	1	0	1	0

I2

0	0	0	0	0	1	1	0

I3

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

NINT, IDNINT – Finds the nearest integer (Cray Fortran intrinsic function)

SYNOPSIS

i=NINT(*real*)

i=IDNINT(*double*)

DESCRIPTION

NINT (generic name) finds the nearest integer for real and double-precision numbers as defined by the following equations.

$y=[x+.5]$ if $x \geq 0$

$y=[x-.5]$ if $x<0$

The argument range for **NINT** is $|x| < 2^{46}$. The range for **IDNINT** is $|x| < 2^{63}$.

NINT and **IDNINT** are both type integer functions. **NINT** is an in-line code function.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

OR – Computes the logical sum (Cray Fortran intrinsic function)

SYNOPSIS

l=**OR**(*logical,logical*)
b=**OR**(*arg,arg*)

DESCRIPTION

arg Argument of type integer, real, or Boolean

When given two arguments of type logical, **OR** computes a logical sum and returns a logical result. When given two arguments of type integer, real, or Boolean, **OR** computes a bit-wise logical sum and returns a Boolean result. The truth tables below show both the logical sum and bit-wise logical sum.

Logical 1	Logical 2	(Logical 1) **OR** (Logical 2)
T	T	T
T	F	T
F	T	T
F	F	F

Bit 1	Bit 2	(Bit 1) **OR** (Bit 2)
1	1	1
1	0	1
0	1	1
0	0	0

EXAMPLES

The following section of Fortran code shows the **OR** function used with two arguments of type logical.

```
LOGICAL L1, L2, L3
...
L3 = OR(L1,L2)
```

The following section of Fortran code shows the OR function used with two arguments of type integer. The bit patterns of the arguments and result are also shown below. For clarity, an 8-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2, I3
...
I3 = OR(I1,I2)
```

0	0	0	0	1	1	0	0

I1

0	0	0	0	1	0	1	0

I2

0	0	0	0	1	1	1	0

I3

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

POPCNT – Counts the number of bits set to 1 (Cray Fortran intrinsic function)

SYNOPSIS

i=**POPCNT**(*arg*)

DESCRIPTION

arg Argument of type integer, real, logical, or Boolean

When given an argument of type integer, real, logical, or Boolean, **POPCNT** counts the number of bits set to 1 in the 64-bit representation of the argument. **POPCNT** is an in-line code function.

NOTES

The bit representation of the logical data type is not consistent among Cray machines. See the following manuals for further details: Fortran (CFT) Reference Manual, publication SR-0009; CRAY-2 Fortran (CFT2) Reference Manual, publication SR-2007; CFT77 Reference Manual, publication SR-0018.

EXAMPLES

The following section of Fortran code shows the **POPCNT** function used with an argument of type integer. The bit pattern of the argument and the value of the result are also shown below. For clarity, a 16-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2
...
I2 = POPCNT(I1)
```

0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	0

I1

The **POPCNT** function returns the value **6** to the integer variable **I2.**

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

POPPAR – Computes the bit population parity (Cray Fortran intrinsic function)

SYNOPSIS

i=**POPPAR**(*arg*)

DESCRIPTION

arg Argument of type integer, real, logical, or Boolean

When given an argument of type integer, real, logical, or Boolean, **POPPAR** returns the value 0 if an even number of bits are set to 1 in the 64-bit representation of the argument or the value 1 if an odd number of bits are set to 1 in the 64-bit representation of the argument. **POPPAR** is an in-line code function.

NOTES

The bit representation of the logical data type is not consistent among Cray machines. See the following manuals for further details: Fortran (CFT) Reference Manual, publication SR-0009; CRAY-2 Fortran (CFT2) Reference Manual, publication SR-2007; CFT77 Reference Manual, publication SR-0018.

EXAMPLES

The following section of Fortran code shows the **POPPAR** function used with an argument of type integer. The bit pattern of the argument and the value of the result are also shown below. For clarity, a 16-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2
...
I2 = POPPAR(I1)
```

0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	0

I1

The **POPPAR** function returns the value **0** to the integer variable **I2**.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

RANF, RANGET, RANSET – Computes pseudo-random numbers (Cray Fortran intrinsic function)

SYNOPSIS

r=**RANF( )**

r=**RANGET**([*integer*])

r=**RANSET**(*integer*)

DESCRIPTION

These CRI extension functions compute pseudo-random numbers and either set or retrieve a seed. **RANF** obtains the first or next in a series of pseudo-random numbers, such that $0<y<1$, in the form of a normalized floating-point number. **RANF** uses a null argument.

RANGET obtains a seed. **RANSET** establishes a seed such that $y=x$. **RANGET** has an optional integer argument and **RANSET** a required integer argument in the range of $| x | <$ inf.

NOTE

When the seed of the random number generator is reset, **RANSET** does not store the supplied argument as the first value in the buffer of the random number seeds.

EXAMPLES

```
      DO 10 I=1,10
10    RANDOM(I)=RANF()
```

```
      CALL RANGET(iseed1)
C            or
      iseed=RANGET()
```

```
      CALL RANSET(ivalue)
C            or
      dummy=RANSET(ivalue)
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

REAL, FLOAT, SNGL – Converts to type real (Cray Fortran intrinsic function)

SYNOPSIS

r=**REAL**(*arg*)

r=**FLOAT**(*integer*)

r=**SNGL**(*double*)
r=**SNGL**(*boolean*)

DESCRIPTION

arg Argument of type complex, integer, or real

REAL (generic name) converts types to type real, such that $y=x$ (or $y=x_r$ for complex arguments). All of these functions are inline Fortran code.

The range for **REAL** complex and real arguments is $|x| < \text{inf}$.

The range for **FLOAT** integer arguments is $|x| < 2^{46}$.

The range for **SNGL** Boolean and double-precision arguments is $|x_r| < \text{inf}$. Type conversion routines assign the appropriate type to Boolean arguments without shifting or maniputlating the bit patterns they represent.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SHIFT – Performs a left circular shift

SYNOPSIS

b=**SHIFT**(*arg1*,*arg2*)

DESCRIPTION

arg1 Argument of type integer, real, logical, or Boolean specifying the value to be shifted

arg2 Argument of type integer specifying the number of bits to shift the value

For *arg2* in the range $0 \le arg2 \le 64$, **SHIFT** performs a left circular shift of the 64-bit representation of *arg1* by *arg2* bits.

For $arg2 \ge 65$, a left circular shift is not performed. Instead, **SHIFT** is defined as follows when $arg2 \ge 65$.

For *arg2* in the range $65 \le arg2 \le 128$, **SHIFT**(*arg1*,*arg2*) is defined as **SHIFTL**(*arg1*,*arg2*–64). See **SHIFTL**(3M).

For *arg2* in the range $129 \le arg2 \le 2^{24}-1$, **SHIFT** returns a value with all bits set to 0.

For *arg2* in the range $2^{24} \le arg2 < \infty$, **SHIFT** returns an undefined result.

NOTES

The bit representation of the logical data type is not consistent among Cray machines. See the following manuals for further details: Fortran (CFT) Reference Manual, publication SR-0009; CRAY-2 Fortran (CFT2) Reference Manual, publication SR-2007; CFT77 Reference Manual, publication SR-0018.

EXAMPLES

The following section of Fortran code shows the **SHIFT** function used in the case where *arg1* is of type integer. For purposes of clarity, a 16-bit word is used instead of the actual 64-bit word. The bit pattern of *arg1* and the bit pattern of the result are also shown below.

```
INTEGER I1, I2, I3
...
I2 = 5
I3 = SHIFT(I1,I2)
```

1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

I1 (*arg1*)

1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1

I3 (result)

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SHIFTL – Performs a left shift with zero fill

SYNOPSIS

b=**SHIFTL**(*arg1*,*arg2*)

DESCRIPTION

arg1 Argument of type integer, real, logical, or Boolean specifying the value to be shifted

arg2 Argument of type integer specifying the number of bits to shift the value

For *arg2* in the range $0 \le arg2 \le 2^{24}-1$, **SHIFTL** performs a left shift with zero fill of the 64-bit representation of *arg1* by *arg2* bits. Note that when *arg2* is in the range $64 \le arg2 \le 2^{24}-1$, **SHIFTL** returns a value with all bits set to 0.

For *arg2* in the range $2^{24} \le arg2 < \infty$, **SHIFTL** returns an undefined result.

NOTES

The bit representation of the logical data type is not consistent among Cray machines. See the following manuals for further details: Fortran (CFT) Reference Manual, publication SR-0009; CRAY-2 Fortran (CFT2) Reference Manual, publication SR-2007; CFT77 Reference Manual, publication SR-0018.

EXAMPLES

The following section of Fortran code shows the **SHIFTL** function used in the case where *arg1* is of type integer. The bit pattern of *arg1* and the bit pattern of the result are also shown below. For purposes of clarity, a 16-bit value is used instead of a 64-bit value.

```
INTEGER I1, I2, I3
...
I2 = 5
I3 = SHIFTL(I1,I2)
```

1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

I1 (*arg1*)

1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

I3 (result)

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SHIFTR – Performs a right shift with zero fill

SYNOPSIS

b=**SHIFTR**(*arg1*,*arg2*)

DESCRIPTION

arg1	Argument of type integer, real, logical, or Boolean
arg2	Argument of type integer

For *arg2* in the range $0 \le arg2 \le 2^{24}-1$, **SHIFTR** performs a right shift with zero fill of the 64-bit representation of *arg1* by *arg2* bits. Note that when *arg2* is in the range $64 \le arg2 \le 2^{24}-1$, **SHIFTR** returns a value with all bits set to 0.

For *arg2* in the range $2^{24} \le arg2 < \infty$, **SHIFTR** returns an undefined result.

NOTES

The bit representation of the logical data type is not consistent among Cray machines. See the following manuals for further details: Fortran (CFT) Reference Manual, publication SR-0009; CRAY-2 Fortran (CFT2) Reference Manual, publication SR-2007; CFT77 Reference Manual, publication SR-0018.

EXAMPLES

The following section of Fortran code shows the **SHIFTR** function used in the case where *arg1* is of type integer. The bit pattern of *arg1* and the bit pattern of the result are also shown below. For purposes of clarity, a 16-bit value is used instead of a 64-bit value.

```
INTEGER I1, I2, I3
...
I2 = 5
I3 = SHIFTR(I1,I2)
```

1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

I1 (*arg1*)

0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0

I3 (result)

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SIGN, ISIGN, DSIGN – Transfers sign of numbers (Cray Fortran intrinsic function)

SYNOPSIS

r=SIGN(*real*,*real*)

i=ISIGN(*integer*,*integer*)

d=DSIGN(*double*,*double*)

DESCRIPTION

SIGN (generic name) evaluates one of the following equations, depending on the sign of the number.

$$y = |x_1| \text{ if } x_2 \geq 0$$
or
$$y = -|x_1| \text{ if } x_2 < 0$$

The argument range for all transfer sign functions is $|x_1|, |x_2| < \text{inf}$. All of these functions are inline Fortran code.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SIN, DSIN, CSIN – Computes the sine (Cray Fortran intrinsic function)

SYNOPSIS

r=SIN(*real*)

d=DSIN(*double*)

z=CSIN(*complex*)

DESCRIPTION

SIN (generic name) solves the equation *y*=sin(*x*). The ranges for the real, double-precision, and complex functions are as follows:

For **SIN**:

$|x| < 2^{24}$

For **DSIN**:

$|x| < 2^{48}$

For **CSIN**:

$|x_r| < 2^{24}, \; |x_i| < 2^{13} * \ln 2$

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SINH, DSINH – Computes the hyperbolic sine (Cray Fortran intrinsic function)

SYNOPSIS

r=**SINH**(*real*)

d=**DSINH**(*double*)

DESCRIPTION

SINH (generic name) solves the equation $y=\sinh(x)$. The hyperbolic sine functions have a real or double-precision argument in the range of $|x| < 2^{13} * \ln 2$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SQRT, DSQRT, CSQRT – Computes the square root (Cray Fortran intrinsic function)

SYNOPSIS

r=**SQRT**(*real*)

d=**DSQRT**(*double*)

z=**CSQRT**(*complex*)

DESCRIPTION

SQRT (generic name) evaluates $y=x^{1/2}$ for real, double-precision, and complex arguments. The range for real and double-precision arguments is $0 \leq x \leq$ inf. The complex argument range is $x_r \geq 0$, $x_i <$ inf.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

TAN, DTAN – Computes the tangent (Cray Fortran intrinsic function)

SYNOPSIS

r=**TAN**(*real*)

d=**DTAN**(*double*)

DESCRIPTION

TAN (generic name) solves for the equation $y=\tan(x)$. The range for the real and double-precision arguments is $|x| < 2^{24}$.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

TANH, DTANH – Computes the hyperbolic tangent (Cray Fortran intrinsic function)

SYNOPSIS

r=**TANH**(*real*)

d=**DTANH**(*double*)

DESCRIPTION

TANH (generic name) solves for the equation y=tanh(x). The hyperbolic tangent functions have a real or double-precision argument in the range of $|x| < 2^{13} * \ln 2$. They solve the equation y=tanh(x).

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

XOR – Computes the logical difference

SYNOPSIS

l=**XOR**(*logical,logical*)
b=**XOR**(*arg,arg*)

DESCRIPTION

arg Argument of type integer, real, or Boolean

When given two arguments of type logical, **XOR** computes a logical difference and returns a logical result. When given two arguments of type integer, real, or Boolean, **XOR** computes a bit-wise logical difference and returns a Boolean result. The truth tables below show both the logical difference and bit-wise logical difference.

Logical 1	Logical 2	(Logical 1) **XOR** (Logical 2)
T	T	F
T	F	T
F	T	T
F	F	F

Bit 1	Bit 2	(Bit 1) **XOR** (Bit 2)
1	1	0
1	0	1
0	1	1
0	0	0

EXAMPLES

The following section of Fortran code shows the **XOR** function used with two arguments of type logical.

```
LOGICAL L1, L2, L3
...
L3 = XOR(L1,L2)
```

The following section of Fortran code shows the **XOR** function used with two arguments of type integer. The bit patterns of the arguments and result are also shown below. For clarity, an 8-bit word is used instead of the actual 64-bit word.

```
INTEGER I1, I2, I3
...
I3 = XOR(I1,I2)
```

0	0	0	0	1	1	0	0

I1

0	0	0	0	1	0	1	0

I2

0	0	0	0	0	1	1	0

I3

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

3. COS DATASET MANAGEMENT SUBPROGRAMS

Dataset management subprograms provide the user with the means of managing COS permanent datasets; creating, staging, and releasing datasets; and changing dataset attributes. These routines are grouped into two subsections:

- COS control statement type subprograms
- COS dataset search type subprograms

IMPLEMENTATION

The dataset management routines are available only under COS.

COS CONTROL STATEMENT TYPE SUBPROGRAMS

A control-statement-type subprogram resembles Cray job control language (JCL) statements in name and purpose. A subprogram, however, can be called from within Fortran or CAL programs while a JCL statement cannot. See the COS Reference Manual, publication SR-0011, for a description of control statements, parameters and keywords, and JCL error codes.

The following is an example of a Fortran call to a control-statement-type subprogram:

```
EXAMPL='EXAMPL'L
IDC='PR'L
CALL ASSIGN(irtc,'DN'L,EXAMPL,'U'L,'MR'L,'DC'L,IDC)
```

Variable *irtc* is an integer that contains a status code upon return. A status code of 0 indicates no errors.

This type of subprogram requires call-by-address subroutine linkage with the following calling sequence:

CALL *SUBROUTINE NAME(stat,key1,key2,...,keyn)*

stat Returned status code

key Keyword/value combinations in one of the following formats (must be entered in uppercase):

'*KEYWORD*'L,'*VALUE*'L
or
'*KEYWORD*'L

When the keyword can accept multiple parameter values, the values must be passed as an array: one parameter per word, terminated by a zero word. For example, the COS control statement MODIFY(DN=DATASET,PAM=R:W) would be coded as follows:

```
INTEGER PAM(3)
DATA PAM/'R'L, 'W'L, 0/
CALL MODIFY(ISTAT, 'DN'L, 'DATASET'L, 'PAM'L, PAM)
```

Permanent Dataset Management routines access the COS Permanent Dataset Manager (PDM) and return the status of the operation in *stat*. The value is 0 if an error condition does not exist and nonzero if an error condition does exist. The nonzero error codes correspond to the PMST codes defined in the COS Reference Manual. The following is a list of the PDM routines and their functions.

Control Statement	Function
ACCESS	Associates a permanent dataset with the job
ADJUST	Expands or contracts a permanent dataset
DELETE	Removes a saved dataset. The dataset remains available to the job until it is released or the job terminates.
MODIFY	Changes the permanent dataset characteristics
PERMIT	Specifies the user access mode to a permanent dataset
SAVE	Makes a dataset permanent and enters the dataset's identification and location into the Dataset Catalog (DSC)

Dataset staging routines stage datasets to or from a front-end processor or to the Cray input queue. The transfer aborts and an error code is returned if an error occurs. The error codes correspond to the PMST codes in the COS Reference Manual. The following is a list of dataset staging routines and their functions.

Control Statement	Function
ACQUIRE	Obtains a front-end resident dataset, stages it to the Cray mainframe, and makes it permanent and available to the job making the request
DISPOSE	Directs a dataset to the specified front-end processor or designates it to a scratch dataset
FETCH	Brings a front-end resident dataset to the Cray mainframe and makes the dataset available to the job
SUBMIT	Places a job dataset into the Cray input queue. When called as an integer function, the value of the function is the job sequence number of the submitted job, if successful.

Definition and control routines allow dataset attributes to be changed and datasets to be created and released. They return the status of the operation in *stat*. The value of the *stat* is 0 if no error condition exists and nonzero if an error condition exists. **ASSIGN** returns a three-digit code that corresponds to log file message codes that begin with SL. Thus, a return code of 020 from **ASSIGN** corresponds to the following log file message:

SL020 - INVALID DATASET NAME OR UNIT NUMBER

All of the SL messages and descriptions of their meanings can be found in the COS Message Manual, publication SR-0039.

The following is a list of definition and control routines.

Control Statement	Function
ASSIGN	Opens a dataset for reading and writing and assigns characteristics to it
OPTION	Changes the user-specified options, such as lines per page and dataset statistics, for a job
RELEASE	Closes a dataset, releases I/O buffer space, and renders it unavailable to the job

COS DATASET SEARCH TYPE SUBPROGRAMS

Dataset search subprograms add information to or return information about a dataset.

The following table contains the purpose, name, and heading of each dataset search type routine.

COS Dataset Search Type Subprograms		
Purpose	Name	Heading
Add a name to the Logical File Table (LFT)	**ADDLFT**	**ADDLFT**
Search for a Dataset Parameter Table (DSP) address	**GETDSP**	**GETDSP**
Determine if a dataset has been accessed or created	**IFDNT**	**IFDNT**
Allow a program to access datasets in the System Directory	**SDACCESS**	**SDACCESS**

NAME

ADDLFT – Adds a name to the Logical File Table (LFT)

SYNOPSIS

CALL ADDLFT(*dn,dsp*)

DESCRIPTION

dn	Name to add to the LFT
dsp	Dataset Parameter Table (DSP) address for the name specified by *dn*

IMPLEMENTATION

This routine is available only to the users of the COS operating system.

NAME

CALLCSP – Executes a COS control statement

SYNOPSIS

CALL CALLCSP(*string*)

DESCRIPTION

string A valid COS JCL statement, either packed into an integer array and terminated by a null byte or specified as a literal string.

The control statement specified in the string is executed as if it had been found next in the job stream. For example, the following call invokes the NOTE utility, which writes HIGH, THEIR! to the $OUT dataset:

```
CALL CALLCSP('NOTE,TEXT="HIGH, THEIR!".')
```

Control does not return from the CALLCSP routine.

NOTE

In general, use CALLCSP instead of LGO.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

GETDSP – Searches for a Dataset Parameter Table (DSP) address

SYNOPSIS

CALL GETDSP(*unit,dsp,ndsp,dn*)

DESCRIPTION

unit	Dataset name or unit number
dsp	DSP address
ndsp	Negative DSP offset relative to the base address of DSPs, or DSP address if the DSP is below JCHLM.
dn	Dataset name (ASCII, left-justified, blank-filled)

GETDSP searches for a DSP address. If none is found, a DSP is created.

IMPLEMENTATION

This routine is available only to the users of the COS operating system.

NAME

IFDNT – Determines if a dataset has been accessed or created

SYNOPSIS

stat=IFDNT(*dn*)

DESCRIPTION

stat	-1 (TRUE) if dataset was accessed or opened; otherwise 0 (FALSE).
dn	Dataset name (ASCII, left-justified, zero-filled)

NOTE

stat must be declared **LOGICAL** in the calling program.

EXAMPLE

```
IF (NOT IFDNT('MYFILE'L)) CALL ACCESS('DN'L,'MYFILE'L)
```

If you access **MYFILE** twice in a program, the system aborts the job. **IFDNT** allows you to test for its having been previously accessed.

IMPLEMENTATION

This routine is available only to the users of the COS operating system.

NAME

SDACCESS – Allows a program to access datasets in the System Directory

SYNOPSIS

CALL SDACCESS(*istat,dn*)

DESCRIPTION

istat An integer variable to receive the completion status (0 or 1).

0 The dataset is a system dataset and has been accessed.
1 The dataset is not a system dataset and has not been accessed.

dn Name of the system dataset to be accessed

This function has no corresponding control statement. Datasets accessed in this manner are automatically released at the end of the job step.

EXAMPLE

```
PROGRAM SDTEST
CHARACTER*7 NAME
INTEGER X
READ*, NAME
X=IFDNT(NAME)
IF (X.EQ.0) THEN
    PRINT*,'***DATASET ',NAME, 'WAS NOT LOCAL***'
    CALL SDACCESS(STAT,NAME)
    IF (STAT.NE.0) THEN
        PRINT*,'***DATASET ',NAME,' NOT AVAILABLE'
        CALL ABORT
    ELSE
        PRINT*,'***DATASET ',NAME,' ACCESSED BY SDTEST'
    ENDIF
ELSE
    PRINT*,'DATASET ',NAME,' ALREADY LOCAL'
ENDIF
END
```

IMPLEMENTATION

This routine is available only to the users of the COS operating system.

4. LINEAR ALGEBRA SUBPROGRAMS

The linear algebra subprograms are written to run optimally on Cray computer systems. These subprograms use call-by-address convention when called by a Fortran or CAL program. (See section 1 for details of the call-by-address convention.)

The linear algebra subprograms include the following:

- Linear algebra subprograms
- Linear recurrence routines
- Matrix inverse and multiplication routines
- Filter routines
- Gather, scatter routines
- LINPACK and EISPACK routines

IMPLEMENTATION

All of the Linear Algebra Subprograms are available to users of both the COS and UNICOS operating systems.

LINEAR ALGEBRA SUBPROGRAMS

The Cray computer user has access to the Basic Linear Algebra Subprograms (BLAS) and a subset of the level 2 BLAS. The level 1 package is described first, and is followed by a description of the level 2 package.

BLAS

The level 1 BLAS is a package of 22 CAL-coded routines, and their extensions. The package includes only the single-precision and complex versions. The following operations are available:

- A constant times a vector plus another vector
- Dot products
- Euclidean norm
- Givens transformations
- Sum of absolute values
- Vector copy and swap
- Vector scaling

Section 6 documents two pivot search routines, **ISAMAX** and **ICAMAX**. These integer functions find the first index of the largest absolute value of the elements of a vector.

Each BLAS routine has a real version and a complex version. There are several frequently used variables that must be declared in your program. The following table lists common variables and their Fortran type declaration and dimensions, in generalized terms.

Linear Algebra Variables		
Variable	Description	Fortran Type and Dimension
SX	Primary real array or vector	**REAL SX(*mx*)**
SY	Secondary real array or vector	**REAL SY(*my*)**
SA	Real scalar	**REAL SA**
CX	Primary complex array or vector	**COMPLEX CX(*mx*)**
CY	Secondary complex array or vector	**COMPLEX CY(*my*)**
CA	Complex scalar	**COMPLEX CA**
INCX	Skip distance between elements in SX or CX	**INTEGER INCX**
INCY	Skip distance between elements in SY or CY	**INTEGER INCY**
N	Number of elements in vector to compute	**INTEGER N**

The dimensions of the above arrays are as follows: *mx*=**N*|INCX|** and *my*=**N*|INCY|**, where N is the array length of the input vectors. In all routines, if N ≤0, inputs and outputs return unchanged.

The variables **C, S, A, B, PARM, D1, D2, B1,** and **B4** are used in the Givens plane rotation routines. Their type must be declared real.

The Fortran type declaration for complex functions is especially important; declare them to avoid type conversion to zero imaginary parts. Fortran type declarations for function names follow:

Type	Function Name
REAL	**SASUM, SCASUM, SDOT, SNRM2, SCNRM2**
COMPLEX	**CDOTC, CDOTU**

Negative incrementation - For routines managing noncontiguous array elements, the parameters *incx and incy* specify skip distances. A skip value of 1 or -1 indicate contiguous elements.

Given an *n*-element array A consisting of A(1), A(2), A(3),...,A(*n*), for positive skip distances (*incx*)>0:

- The managed array elements are **A(1), A(1+*incx*), A(1+2**incx*), A(1+3**incx*),...,A(1+(*p*-1)**incx*)**, where *p* is the number of array elements to be processed.
- For *n* **MODULO** *incx*>0, $p \leq 1 + \frac{n}{incx}$. Otherwise, $p \leq \frac{n}{incx}$.

Given the previous array and a negative skip distance (*incx*<0):

- The managed array elements are **A(1+(*p*-1)*ABS(*incx*)), A(1+(*p*-2)*ABS(*incx*)), A(1+(*p*-3)*ABS(*incx*)), A(1+(*p*-4)*ABS(*incx*)),...,A(1+(*p*-p)*ABS(*incx*)),** where *p* is the number of array elements to be processed.
- For *n* **MODULO** *incx*>0, *p*, ≤ 1+*n*/ABS(*incx*). Otherwise, *p* ≤ *n*/ABS(*incx*).

EXAMPLE - The real function **ISAMAX** returns the relative index of I such that **ABS(A(I)) = MAX ABS(A(1+(J-1)*INCX))** for **J=1,2,3,...,*p*.**

The call from Fortran is as follows:

RELINDEX = ISAMAX(*p,array,incx*)

Assume **A(1)=2.0, A(2)=4.0, A(3)=6.0,...,A(20)=40.0** (the number of elements ***n*=20).**

With a positive skip distance (*incx*=3), the number of elements processed *p*=**7 (since 20 MODULO 3 > 0, *p*=1+*n*/*incx*=1+20/3=1+6=7).**

Therefore, the function is evaluated as follows:

ISAMAX(7,A,3)=
rel. index of **MAX{2.0,8.0,14.0,20.0,26.0,32.0,38.0}**
= relative index of 38.0
= 7

With a negative skip distance *incx*=**-3**, the number of elements processed *p*=**7** (since **20 MODULO ABS(-3)>0,** *p* = **1+*n*/ABS(*incx*) =1 + 20/3 = 1+6 = 7.**

Therefore, the function is evaluated as follows:

ISAMAX(7,A,-3)=
rel. index of **MAX{38.0,32.0,26.0,20.0,14.0,8.0,2.0}**
= relative index of 38.0
= 1

The following table contains the purpose, name, and entry of each linear algebra subprogram.

Level 1 BLAS		
Purpose	Name	Entry
Sum the absolute values of a real or complex vector	**SASUM** **SCASUM**	**SASUM**
Add a scalar multiple of a real or complex vector to another vector	**SAXPY** **CAXPY**	**SAXPY**
Copy a real or complex vector into another vector	**SCOPY** **CCOPY**	**SCOPY**
Apply a complex Givens plane rotation	**CROT**	**CROT**
Compute a complex Givens plane rotation matrix	**CROTG**	**CROTG**
Compute a dot product of two real or complex vectors	**SDOT** **CDOTC** **CDOTU**	**DOT**
Scale a real or complex vector	**SSCAL** **CSSCAL** **CSCAL**	**SCAL**
Compute the product of a vector and a matrix and add to another vector	**SXMPY**	**SXMPY**
Compute the Euclidean norm or l_2 norm of a real or complex vector	**SNRM2** **SCNRM2**	**SNRM2**
Add a scalar multiple of a real vector to an indexed vector	**SPAXPY**	**SPAXPY**
Compute an indexed dot product of two real vectors	**SPDOT**	**SPDOT**
Apply a Givens plane rotation	**SROT**	**SROT**
Construct a Givens plane rotation	**SROTG**	**SROTG**
Apply a modified Givens plane rotation	**SROTM**	**SROTM**
Construct a modified Givens plane rotation	**SROTMG**	**SROTMG**
Sum the elements of a real or complex vector	**SSUM** **CSUM**	**SSUM**
Swap two real or two complex arrays	**SSWAP** **CSWAP**	**SSWAP**

BLAS-2

The Basic Linear Algebra Routines version 2 (BLAS-2) consists of 11 Fortran routines for unpacked data of type real. The following table describes these routines.

Level 2 BLAS		
Purpose	Name	Entry
Multiply a real vector by a real general band matrix	**SGBMV**	**SGBMV**
Multiply a real vector by a real general matrix	**SGEMV**	**SGEMV**
Perform rank 1 update of a real general matrix	**SGER**	**SGER**
Multiply a real vector by a real symmetric band matrix	**SSBMV**	**SSBMV**
Multiply a real vector by a real symmetric matrix	**SSYMV**	**SSYMV**
Perform symmetric rank 1 update of a real symmetric matrix	**SSYR**	**SSYR**
Perform symmetric rank 2 update of a real symmetric matrix	**SSYR2**	**SSYR2**
Multiply a real vector by a real triangular band matrix	**STBMV**	**STBMV**
Solve a real triangular banded system of equations	**STBSV**	**STBSV**
Multiply a real vector by a real triangular matrix	**STRMV**	**STRMV**
Solve a real triangular system of equations	**STRSV**	**STRSV**

LINEAR RECURRENCE SUBROUTINES

Linear recurrence subroutines solve first-order and some second-order linear recurrences. A linear recurrence uses the result of a previous pass through the loop as an operand for subsequent passes through the loop. Such use prevents vectorization. Therefore, these subroutines can be used to optimize Fortran loops containing linear recurrences.

The following table contains the purpose, name, and entry of each linear recurrence subroutine.

Linear Recurrence Subroutines		
Purpose	Name	Entry
Solve first-order linear recurrences	**FOLR** **FOLRP**	**FOLR**
Solve first-order linear recurrences and write the solutions to a new vector	**FOLR2** **FOLR2P**	**FOLR2**
Solve first-order linear recurrences for a specific equation	**FOLRC**	**FOLRC**
Solve for the last term of a first-order linear recurrence using Horner's method	**FOLRN**	**FOLRN**
Solve for the last term of a first-order linear recurrence	**FOLRNP**	**FOLRNP**
Solve second-order linear recurrences	**SOLR** **SOLRN** **SOLR3**	**SOLR**
Solve for a partial products problem	**RECPP**	**RECPP**
Solve for a partial summation problem	**RECPS**	**RECPS**

MATRIX INVERSE AND MULTIPLICATION ROUTINES

The matrix inverse subroutine, MINV, computes the matrix inverse and solves systems of linear equations using the Gauss-Jordan elimination. MXM and MXMA are two optimal matrix multiplication routines. MXV and MXVA are similar to MXM and MXMA; however, MXV and MXVA handle the special case of matrix times vector multiplication.

The following table contains a summary of the matrix inverse and multiplication routines.

Matrix Inverse and Multiplication Routines		
Purpose	Name	Heading
Compute the determinant and inverse of a square matrix	**MINV**	**MINV**
Compute a matrix times a matrix (skip distance of 1)	**MXM**	**MXM**
Compute a matrix times a matrix (arbitrary spacing)	**MXMA**	**MXMA**
Compute a matrix times a vector (skip distance of 1)	**MXV**	**MXV**
Compute a matrix times a vector (arbitrary spacing)	**MXVA**	**MXVA**

FILTER SUBROUTINES

The filter subroutines are intended for filter analysis and design. They also solve more general problems. For detailed descriptions, algorithms, performance statistics, and examples, see Linear Digital Filters for CFT Usage, CRI publication SN-0210.

The following table contains a summary of the filter subroutines.

Filter Subroutines		
Purpose	Name	Heading
Compute a convolution of two vectors	**FILTERG**	**FILTERG**
Compute a convolution of two vectors (assumes that the filter coefficient vector is symmetric)	**FILTERS**	**FILTERS**
Solve the Weiner-Levinson linear equations	**OPFILT**	**OPFILT**

GATHER, SCATTER ROUTINES

The GATHER and SCATTER subroutines allow you to gather a vector from a source vector or to scatter a vector into another vector. A third vector of indexes determines which elements are accessed or changed.

LINPACK AND EISPACK ROUTINES

LINPACK routines solve systems of linear equations and compute the QR, Cholesky, and singular value decompositions. EISPACK routines solve the eigenvalue problem, and they compute and use singular value decomposition.

SINGLE-PRECISION REAL AND COMPLEX LINPACK ROUTINES - LINPACK is a package of Fortran routines that solve systems of linear equations and compute the QR, Cholesky, and singular value decompositions. The original Fortran programs are documented in the LINPACK User's Guide by J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, published by the Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1979, Library of Congress catalog card number 78-78206 (available through Cray Research as publicaton S1-0113).

Each single-precision version of the LINPACK routines has the same name, algorithm, and calling sequence as the original version. Optimization of each routine includes the following:

- Replacement of calls to the BLAS routines **SSCAL, SCOPY, SSWAP, SAXPY,** and **SROT** with in-line Fortran code that the Cray Fortran compilers vectorize
- Removal of Fortran **IF** statements if the result of either branch is the same
- Replacement of **SDOT** to solve triangular systems of linear equations in **SGESL, SPOFA, SPOSL, STRSL,** and **SCHDD** with more vectorizable code

These optimizations affect only the execution order of floating-point operations in modified **DO** loops. See the LINPACK User's Guide for further descriptions. The complex routines have been added without extensive optimization.

SINGLE-PRECISION EISPACK ROUTINES

EISPACK is a package of Fortran routines for solving the eigenvalue problem and for computing and using the singular value decomposition.

The original Fortran versions are doucmented in the Matrix Eigensystem Routines - EISPACK Guide, second edition, by T. B. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, published by Springer-Verlag, New York, 1976, Library of Congress catalog card number 76-2662 (available through Cray Research as publicaton S2-0113); and in the Matrix Eigensystem Routines - EISPACK Guide Extension by B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, published by Springer-Verlag, New York, 1977, Library of Congress catalog card number 77-2802 (available through Cray Research as publicaton S3-0113).

Each SCILIB version of the EISPACK routines has the same name, algorithm, and calling sequence as the original version. Optimization of each routine includes the following:

- Use of the BLAS routines SDOT, SASUM, SNRM2, ISAMAX, and ISMIN when applicable
- Removal of Fortran IF statements if the result of either branch is the same
- Unrolling complicated Fortran DO loops to improve vectorization
- Use of the Fortran compiler directive CDIR IVDEP when no dependencies that prevent vectorization exist

These modifications increase vectorization and, therefore, reduce execution time. Only the order of computations within a loop is changed; the modified version produces the same answers as the original versions unless the problem is sensitive to small changes in the data.

NAME

CROT – Applies the complex plane rotation computed by **CROTG**

SYNOPSIS

CALL CROT(*n,cx,incx,cy,incy,cc,cs***)**

DESCRIPTION

n	Number of elements in the vector
cx	Complex vector to be modified
incx	Skip distance between elements of *cx*
cy	Complex vector to be modified
incy	Skip distance between elements of *cy*
cc	Complex cosine of the following equation
cs	Complex sine of the following equation

CROT performs the following equation:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

where x and y are complex row vectors.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

CROTG

NAME

CROTG – Computes the elements of a complex plane rotation matrix

SYNOPSIS

CALL CROTG(*ca,cb,cc,cs*)

DESCRIPTION

ca	Complex *a* of the following equation
cb	Complex *b* of the following equation
cc	Complex cosine of the following equation
cs	Complex sine of the following equation

CROTG computes the elements of a complex Givens plane rotation matrix as in the following equation:

$$\begin{bmatrix} a' \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

The 2 x 2 matrix is unitary, and *a* and *b* are overwritten.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

CROT

NAME

SDOT, CDOTC, CDOTU – Computes a dot product (inner product) of two real or complex vectors

SYNOPSIS

dot=SDOT(*n*,*sx*,*incx*,*sy*,*incy*)

cdot=CDOTC(*n*,*cx*,*incx*,*cy*,*incy*)

cdot=CDOTU(*n*,*cx*,*incx*,*cy*,*incy*)

DESCRIPTION

n	Number of elements in the vectors
sx	Real vector operand
cx	Complex vector operand
incx	Skip distance between elements of *sx* or *cx*
sy	Real vector operand
cy	Complex vector operand
incy	Skip distance between elements of *sy* or *cy*. For contiguous elements, *incy*=1.

These real and complex functions compute an inner product of two vectors. SDOT computes

$$dot = \sum_{i=1}^{n} x_i y_i$$

where x_i and y_i are elements of real vectors.

CDOTC computes

$$cdot = \sum_{i=1}^{n} \overline{x_i} y_i$$

where x_i and y_i are elements of complex vectors and $\overline{x_i}$ is the complex conjugate of x_i.

CDOTU computes

$$cdot = \sum_{i=1}^{n} x_i y_i$$

where x_i and y_i are elements of complex vectors.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

EISPACK – Single-precision EISPACK routines

DESCRIPTION

EISPACK is a package of Fortran routines for solving the eigenvalue problem and for computing and using the singular value decomposition.

The original Fortran versions are documented in the Matrix Eigensystem Routines /- EISPACK Guide, second edition, by B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, published by Springer-Verlag, New York, 1976, Library of Congress catalog card number 76-2662 (available through Cray Research as publicaton S2-0113); and in the Matrix Eigensystem Routines – EISPACK Guide Extension by B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, published by Springer-Verlag, New York, 1977, Library of Congress catalog card number 77-2802 (available through Cray Research as publicaton S3-0113).

Each scientific library version of the EISPACK routines has the same name, algorithm, and calling sequence as the original version. Optimization of each routine includes the following:

- Use of the BLAS routines **SDOT, SASUM, SNRM2, ISAMAX,** and **ISMIN** when applicable
- Removal of Fortran IF statements if the result of either branch is the same
- Unrolling complicated Fortran DO loops to improve vectorization
- Use of the Fortran compiler directive **CDIR$ IVDEP** when no dependencies exist that prevent vectorization

These modifications increase vectorization and, therefore, reduce execution time. Only the order of computations within a loop is changed; the modified versions produce the same answers as the original versions unless the problem is sensitive to small changes in the data.

The following summary provides a list of the routines giving the name, matrix or decomposition, and the purpose for each routine.

Name	Matrix or Decomposition	Purpose
CG	Complex general	Find eigenvalues and eigenvectors (as desired)
CH	Complex Hermitian	
RG	Real general	
RGG	Real general generalize $A\ x = \lambda B\ x$	
RS	Real symmetric	
RSB	Real symmetric band	
RSG	Real symmetric generalize $A\ x = \lambda B\ x$	
RSGAB	Real symmetric generalize $AB\ x = \lambda x$	
RSGBA	Real symmetric generalize $BA\ x = \lambda x$	
RSP	Real symmetric packed	

Name	Matrix or Decomposition	Purpose
RST	Real symmetric tridiagonal	
RT	Special real tridiagonal	
BALANC	Real general	Balance matrix and isolate eigenvalues whenever possible
CBAL	Complex general	
ELMHES **ORTHES**	Real general	Reduce matrix to upper Hessenberg form
COMHES **CORTH**	Complex general	
ELTRAN **ORTRAN**	Real general	Accumulate transformations used in the reduction to upper Hessenberg form done by **ELMHES**, **ORTHES**
BALBAK **ELMBAK** **ORTBAK**	Real general	Form eigenvectors by back transforming those of the corresponding matrices determined by **BALANC**, **ELMHES**, **ORTHES**, **COMMES**, **CORTH**, and **CBAL**
COMBAK **CORTB** **CBABK2** **REBAK** **REBAKB**	Complex general	
TRED1 **TRED2** **TRED3**	Real symmetric	Reduce to symmetric tridiagonal
TRBAK **TRBAK3**	Real symmetric	Form eigenvectors by back transforming those of the by **TRED1** or **TRED3**
IMTQLV **IMTQL1** **IMTQL2**	Symmetric tridiagonal	Find eigenvalues and/or eigenvectors by implicit QL method
RATQR	Symmetric tridiagonal	Find the smallest or largest eigenvalues by rational QR method with Newton corrections
TQLRAT	Symmetric tridiagonal	Find the eigenvalues by rational QL method

Name	Matrix or Decomposition	Purpose
TQL1 **TQL2**		Find the eigenvalues and/or eignenvectors by the rational QL or QL method
BISECT **RIDIB** **TSTURM** **TINVIT**	Symmetric tridiagonal	Find eigenvalues and/or eigenvectors which lie in a specified interval using bisection and/or inverse iteration
FIGI **FIGI2**	Nonsymmetric tridiagonal	Reduce to symmetric tridiagonal with the same eigenvalues
BAKVEC	Nonsymmetric	Form eigenvectors by back transforming corresponding matrix determined by FIGI
HQR **HQR2** **COMQR** **COMQR2**	Real upper Hessenberg Complex upper Hessenberg	Find eigenvalues and/or eigenvectors by QR method
INVIT	Upper Hessenberg	Find eigenvectors corresponding to specified eigenvalues
CINVIT	Complex upper Hessenberg	
BANDR	Real symmetric banded	Reduce to a symmetric tridiagonal matrix
BANDV	Real symmetric banded	Find those eigenvectors corresponding to specified eigenvalues using inverse iteration
BQR	Real symmetric banded	Find eigenvalues using QR algorithm with shifts of origin
MINFIT	Real rectangular	Determine the singular value decomposition $A = USV^T$, forming U^TB rather than U. Householder bidiagonalization and a variant of the QR algorithm are used.
SVD	Real rectangular	Determine the singular value decomposition $A = USV^T$. Householder bidiagonalization and a variant of the QR algorithm are used.

Name	Matrix or Decomposition	Purpose
HTRIBK **HTRIB3** **HTRIDI** **HTRID3**	Complex Hermitian	All eigenvalues and eigenvectors
QZHES **QZIT** **QZVAL** **QZVEC**	Real generalize eigenproblem $Ax = \lambda Bx$	All eigenvalues and eigenvectors
COMLR **COMLR2**	Complex general	Reduce matrix to upper Hessenberg
REDUC	Real symmetric generalize $Ax = \lambda Bx$	Transform generalize symmetric eigenproblems to standard symmetric eigenproblems
REDUC2	Real symmetric generalize $ABx = \lambda Bx$ or $BAx = \lambda Bx$	

NAME

FILTERG – Computes a convolution of two vectors

SYNOPSIS

CALL FILTERG(*a,m,d,n,o***)**

DESCRIPTION

a	Vector of filter coefficients
m	Number of filter coefficients
d	Input data vector
n	Number of data points
o	Output vector

FILTERG computes a convolution of two vectors. Given

(a_i)	$i=1, \ldots, m$	Filter coefficients
(d_j)	$j=l, \ldots, n$	Data

FILTERG computes the following:

$$o_i = \sum_{j=1}^{m} a_j \, d_{i+j-1} \qquad i=1, \ldots, n-m+1$$

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

FILTERS – Computes convolution of two vectors (symmetric coefficient)

SYNOPSIS

CALL FILTERS(*a*,*m*,*d*,*n*,*o*)

DESCRIPTION

a	Symmetric filter coefficient vector
m	M is formally the length of vector A, but because A is symmetric ($a_i = a_{m-i+1}$; $i = 1, \ldots, m$), only $\frac{m}{2}$ elements of A are ever referenced.
d	Input data vector
n	Number of data points
o	Output vector

FILTERS computes the same convolution as FILTERG except that it assumes the filter coefficient vector is symmetric. Given

(c_i) $i=1, \ldots, \lceil m/2 \rceil$

(d_i) $j=l, \ldots, n$

($\lceil m/2 \rceil = \frac{m}{2}$ for m even and $\frac{(m+1)}{2}$ for m odd, called the ceiling function.)

FILTERS computes the following when m is an odd number:

$$o_i = \sum_{j=1}^{(m-1)/2} a_j \left(d_{i+j-1} + d_{i+m-j} \right) + a \lceil m/2 \rceil \; . \; d_{i + \lceil m/2 \rceil} \quad i=1, \ldots, n-m+1$$

FILTERS computes the following when m is an even number:

$$o_i = \sum_{j=1}^{m/2} a_j \left(d_{i+j-1} + d_{i+m-j} \right) \quad i=1, \ldots, n-m+1$$

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

FILTERG

NAME

FOLR, FOLRP – Solves first-order linear recurrences

SYNOPSIS

CALL FOLR(*n,a,inca,b,incb***)**

CALL FOLRP(*n,a,inca,b,incb***)**

DESCRIPTION

n	Length of linear recurrence
a	Vector of length *n* of equation 1. (A(1) is arbitrary.)
inca	Skip distance between elements of the vector operand **A** or *a*
b	For **FOLR**, vector **b** of equation 1 on input and on output. For **FOLRP**, vector **b** of equation 2 on input on output. In either case, the output overwrites the input.
incb	Skip distance between elements of the vector operand *b* and result **B** or *b*

FOLR solves first-order linear recurrences as in equation 1.

Equation 1:

$$b_1 = b_1$$
$$b_i = b_i - b_{i-1} * a_i \quad \text{for} \quad i = 2, 3 \ldots, n$$

The Fortran equivalent of equation 1 is as follows:

```
      B(1) = B(1)
      DO 10 I = 2, N
10    B(I) = B(I) - B(I-1) * A(I)
```

FOLRP solves first-order linear recurrences as in equation 2:

Equation 2:

$$b_1 = b_1$$
$$b_i = a_i b_{i-1} + b_i \quad \text{for} \quad i = 2, 3 \ldots, n$$

The Fortran equivalent of equation 2 is as follows:

```
      B(1)=B(1)
      DO 10 I = 2, N
10    B(I) = A(I) * B(I-1) + B(I)
```

CAUTIONS

Do not specify *inca* or *incb* as zero; doing so yields unpredictable results.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

FOLR2, FOLR2P – Solves first-order linear recurrences, write new vector

SYNOPSIS

CALL FOLR2(*n,a,inca,b,incb,c,incc***)**

CALL FOLR2P(*n,a,inca,b,incb,c,incc***)**

DESCRIPTION

n	Length of the linear recurrence
a	For **FOLR2**, vector *a* of length *n* of equation 1; for **FOLR2P**, vector *a* of length *n* of equation 2. (See equations 1 and 2 for **FOLR**.) A(1) is arbitrary.
inca	Skip distance between elements of the vector operand *a*
b	For **FOLR2**, vector **b** of equation 1. For **FOLR2P**, vector **b** of equation 2 on input.
incb	For **FOLR2**, skip distance between elements of the vector operand *b* and result **C**; for **FOLR2P**, skip distance between elements of the vector operand *b*.
c	For **FOLR2**, vector *c* of equation 1. For **FOLR2P**, vector *c* of equation 2.
incc	Skip distance between elements of the vector result *c*. For contiguous elements, *incc*=1.

FOLR2 solves first-order linear recurrences as in equation 1. (See the **FOLR** subroutine.) The solution, however, is written to vector *c*, which is different from vector **B** in **FOLR. FOLR2P** is a combination of **FOLRP** and **FOLR2.**

CAUTIONS

Do not specify *inca*, *incb*, or *incc* as zero; doing so yields unpredictable results.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

SEE ALSO

FOLR

NAME

FOLRC – Solves first-order linear recurrence shown

SYNOPSIS

result=FOLRC(*n,x,incx,c,incc,coef*)

DESCRIPTION

n	Length of linear recurrence
x	Vector
incx	Skip distance
c	Vector *c*
incc	Skip distance
coef	Coefficient

FOLRC solves a linear recurrence as in the Fortran equivalent below:

```
      I=1
      J=1
      IF (INCX .LT. 0) THEN
            I = 1-(N-1)*INCX
      ENDIF
      IF (INCC. LT. 0) THEN
            J = 1-(N-1)*INCC
      ENDIF
      X(I) = C(J)
      DO 10 I=1, N
            X(I+INCX) = COEF*X(I) + C(J+INCC)
            J = J + INCC
            I = I + INCX
10    CONTINUE
```

CAUTIONS

Do not specify *incx* or *incc* as zero; doing so yields unpredictable results.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

FOLRN – Solves last term of first-order linear recurrence using Horner's method

SYNOPSIS

result=FOLRN(*n*,*a*,*inca*,*b*,*incb*)

DESCRIPTION

n	Length of the linear recurrence
a	Vector *a* of length *n* of equation 1 (see equation 1 under the FOLR). (A(1) is arbitrary.)
inca	Skip distance between elements of the vector operand **A**
b	Vector *b* of length *n* of equation 1. (The output overwrites the input.)
incb	Skip distance between elements of the vector operand and result *b*

FOLRN solves for r_n of

$$r_1 = b_1$$
$$r_i = -a_i\, r_{i-1} + b_i \quad i = 2, 3, \ldots, n$$

CAUTIONS

Do not specify *inca* or *incb* as zero; doing so yields unpredictable results.

EXAMPLE

FOLRN allows for efficient evaluation of polynomials using Horner's method as follows:

$$\text{Let } p(x) = \sum_{i=0}^{n} b_i\, x^{n-i}$$

then $p(a) = (\ldots((b_0 x + b_1)\, x + b_2)\, x + \ldots b_n)$ by Horner's rule.

The Fortran equivalent is as follows:

```
      PA = B(0)
      DO 10 I = 1, N
         PA = PA * X + B(I)
   10 CONTINUE
```

or

```
      PA=FOLRN(N+1,-X,0,B(0),1)
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems

SEE ALSO

FOLR

NAME

FOLRNP – Solves last term of a first-order linear recurrence

SYNOPSIS

result=FOLRNP(*n,a,inca,b,incb*)

DESCRIPTION

n	Length of the linear recurrence
a	Vector *a* of length *n* of equation 1 (see equation 1 under the **FOLR**). (A(1) is arbitrary.)
inca	Skip distance between elements of the vector operand A
b	Vector *b* of length *n* of equation 1. (The output overwrites the input.)
incb	Skip distance between elements of the vector operand and result *b*

FOLRNP solves a linear recurrence as in the following Fortran equivalent:

```
      I=1
      J=1
      IF (INCX .LT. 0) THEN
                I = 1 - (N-1) * INCX
      ENDIF
      IF (INCC .LT. 0) THEN
             J = 1 - (N-1) * INCC
      ENDIF
      RESULT = B(J)
      DO 10 I = 2, N
             RESULT = A(K+INCA) * RESULT + B(J+INCB)
             J = J + INCB
             K = K + INCA
   10 CONTINUE
```

CAUTIONS

Do not specify *inca*, *incb*, or *incc* as zero; doing so yields unpredictable results.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

GATHER – Gathers a vector from a source vector

SYNOPSIS

CALL GATHER(*n*,*a*,*b*,*index*)

DESCRIPTION

n	Number of elements in vectors *a* and *index*; not *b*
a	Output vector
b	Source vector
index	Vector of indexes

GATHER is defined in the following way:

$$a_i = b_{j_i} \quad \text{where } i = 1, \ldots, n$$

In Fortran:

A(I)=B(INDEX(I))

where I = 1, ..., N

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

LINPACK – Single-precision real and complex LINPACK routines

DESCRIPTION

LINPACK is a package of Fortran routines that solve systems of linear equations and compute the QR, Cholesky, and singular value decompositions. The original Fortran programs are documented in the LINPACK User's Guide by J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, published by the Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1979, Library of Congress catalog card number 78-78206. This guide is available through Cray Research as publicaton S1-0113.

Each single-precision scientific library version of the LINPACK routines has the same name, algorithm, and calling sequence as the original version. Optimization of each routine includes the following:

- Replacement of calls to the BLAS routines **SSCAL, SCOPY, SSWAP, SAXPY,** and **SROT** with in-line Fortran code that the Cray Fortran compilers vectorize
- Removal of Fortran **IF** statements if the result of either branch is the same
- Replacement of **SDOT** to solve triangular systems of linear equations in **SGESL, SPOFA, SPOSL, STRSL,** and **SCHDD** with more vectorizable code

These optimizations affect only the execution order of floating-point operations in DO loops. See the LINPACK User's Guide for further descriptions. The complex routines have been added without much optimization.

The following summary provides a list of the routines giving the name, matrix or decomposition, and the purpose for each routine.

Name	Matrix or Decomposition	Purpose
SGECO	Real general	Factor and estimate condition
SGEFA		Factor
SGESL		Solve
SGEDI		Compute determinant and inverse
CGECO	Complex general	Factor and estimate condition
CGEFA		Factor
CGESL		Solve
CGEDI		**Compute determinant and inverse**
SGBCO	Real general banded	Factor and estimate condition
SGBFA		Factor
SGBSL		Solve
SGBDI		Compute determinant
CGBCO	Complex general banded	Factor and estimate condition
CGBFA		Factor
CGBSL		Solve
CGBDI		Compute determinant

Name	Matrix or Decomposition	Purpose
SPOCO	Real positive definite	Factor and estimate condition
SPOFA		Factor
SPOSL		Solve
SPODI		Compute determinant and inverse
CPOCO	Complex positive	Factor and estimate condition
CPOFA	definite	Factor
CPOSL		Solve
CPODI		Compute determinant and inverse
SPPCO	Real positive definite	Factor and estimate condition
SPPFA	packed	Factor
SPPSL		Solve
SPPDI		Compute determinant and inverse
CPPCO	Complex positive	Factor and estimate condition
CPPFA	definite packed	Factor
CPPSL		Solve
CPPDI		Compute determinant and inverse
SPBCO	Real positive definite	Factor and estimate condition
SPBFA	banded	Factor
SPBSL		Solve
SPBDI		Compute determinant
CPBCO	Complex positive	Factor and estimate condition
CPBFA	definite banded	Factor
CPBSL		Solve
CPBDI		Compute determinant
SSICO	Symmetric indefinite	Factor and estimate condition
SSIFA		Factor
SSISL		Solve
SSIDI		Compute inertia, determinant, and inverse
CHICO	Hermitian indefinite	Factor and estimate condition
CHIFA		Factor
CHISL		Solve
CHIDI		Compute inertia, determinant, and inverse
SSPCO	Symmetric indefinite	Factor and estimate condition
SSPFA	packed	Factor
SSPSL		Solve
SSPDI		Compute inertia, determinant, and inverse

Name	Matrix or Decomposition	Purpose
CSPCO	Complex symmetric	Factor and estimate condition
CSPFA	indefinite packed	Factor
CSPSL		Solve
CSPDI		Compute inertia, determinant, and inverse
CHPCO	Hermitian indefinite	Factor and estimate condition
CHPFA	packed	Factor
CHPSL		Solve
CHPDI		Compute inertia, determinant, and inverse
STRCO	Real triangular	Factor and estimate condition
STRSL		Solve
STRDI		Compute determinant and inverse
CTRCO	Complex triangular	Factor and estimate condition
CTRSL		Solve
CTRDI		Compute determinant and inverse
SGTSL	Real tridiagonal	Solve
CGTSL	Complex tridiagonal	Solve
SPTSL	Real positive definite tridiagonal	Solve
CPTSL	Complex	Solve
SCHDC	Real Cholesky	Decompose
SCHDD	decomposition	Downdate
SCHUD		Update
SCHEX		Exchange
CCHDC	Complex Cholesky	Decompose
CCHDD	decomposition	Downdate
CCHUD		Update
CCHEX		Exchange
SQRDC	Real	Orthogonal factorization
SQRSL		Solve
CQRDC	Complex	Orthogonal factorization
CQRSL		Solve
SSVDC	Real	Singular value decomposition
CSVDC	Complex	

NAME

MINV – Computes the determinant and inverse of a square matrix

SYNOPSIS

CALL MINV(*ab,n,nd,scratch,det,eps,m,mode*)

DESCRIPTION

ab	Augmented matrix of the square matrix *a* and the *n* x *m* matrix *b* of the *m* right-hand sides for each system of equations to solve. The solution overwrites the corresponding right-hand side. In the calling routine, *ab* must be dimensioned *a(nd,n+m)*.
n	Order of matrix *a*
nd	Leading dimension of *ab*
scratch	User-defined working storage array of length at least 2**n*
det	Determinant of matrix *a*
eps	User-defined tolerance for the product of pivot elements
m	If m>0, m is the number of systems of linear equations to solve. If m=0, the determinant of *a* is computed, depending on the value of *mode*.
mode	If *mode*=+1, *a* is overwritten with a^{-1}. If *mode*=0, *a* is not saved and a^{-1} is not computed.

MINV can also be used to solve systems of linear equations with multiple right-hand sides.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

MXM – Computes a matrix times matrix product (*c=ab*) ; skip distance equals 1

SYNOPSIS

CALL MXM(*a,nar,b,nac,c,nbc*)

DESCRIPTION

a	First matrix of product
nar	Number of rows of matrices *a* and *c*
b	Second matrix of product
nac	Number of columns of matrix *a* and the number of rows of matrix *b*
c	Result matrix
nbc	Number of columns of matrices *b* and *c*

MXM can be used only if both matrixes being multiplied are equal to the length at which they are dimensioned. If a submatrix is being multiplied, use the MXMA routine.

IMPLEMENTATION

This routine is available to users of both COS and UNICOS operating systems.

SEE ALSO

MXV is similar to MXM; however, MXV handles the special case of a matrix times a vector.

NAME

MXMA – Computes a matrix times matrix product (*c=ab*) with arbitrary skip distance

SYNOPSIS

CALL MXMA(*a,na,iad,b,nb,ibd,c,nc,icd,nar,nbr,ncc*)

DESCRIPTION

a	First matrix of the product
na	Spacing between column elements of *a*
iad	Spacing between row elements of *a*
b	Second matrix of the product
nb	Spacing between column elements of *b*
ibd	Spacing between row elements of *b*
c	Output matrix
nc	Spacing between column elements of *c*
icd	Spacing between row elements of *c*
nar	Number of rows in the first operand and result
nbr	Number of columns in the first operand and number of rows in the second operand
ncc	Number of columns in the second operand and result

MXMA is recommended for multiplying parts of matrices; that is, a multiplication that does not involve all of the elements dimensioned. If the matrices to be multiplied are equal to their declared dimensions, the MXM routine provides better performance.

EXAMPLES

The dimension of matrix A is 3x3. Consider the 2x3 submatrix *A′* marked by asterisks.

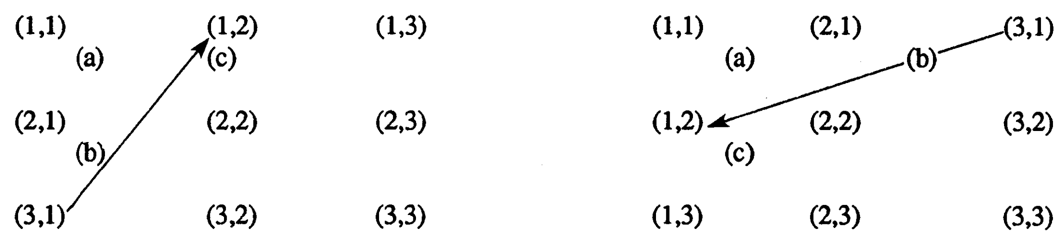
```
(1,1)*                  (1,2)*         (1,3)*
      (a)
(2,1)                   (2,2)          (2,3)
      (b)   (c)
(3,1)*                  (3,2)*         (3,3)*
```

The row spacing of submatrix *A′* (*iad*) is defined as the length of the path through A between two consecutive row elements of *A′*. In this example, the path is (a) through (c) (*iad*=3).

The column spacing of *A′* (*na*) is defined as the path through A between two consecutive column elements of *A′*. In this example, the path is (a) through (b) (*nar*=2); and the number of columns of *A′* is 3 (*nbr*=3).

Example 2:

Consider the following matrices. Let A^T, the transpose of A, equal the first operand of a matrix multiply operand. The transpose of a matrix has as its *i*th column of the original matrix.



Matrix A　　　　Matrix A^T

The length of the path between two consecutive column elements of A^T is the same as the length of the path between two consecutive row elements of A. Refer to paths (a) through (c) of both matrices (*na*=3). The length of the path between two consecutive row elements of A^T is the length of the path between two consecutive column elements of A. This path consists of just (a) (*iad*=1). In this example, *nar*=3 and *nbr*=3.

Therefore, if A is the first operand of a call to MXMA, the following subroutine call is used.

CALL MXMA(A,1,3,..)

If *A 'T* is the first operand of a call to MXMA, the following subroutine call is used

CALL MXMA(A,3,1,...)

IMPLEMENTATION

This routine is available to users of both COS and UNICOS operating systems.

SEE ALSO

MXVA is similar to MXMA; however, MXVA handles the special case of a matrix times a vector.

NAME

MXV – Computes a matrix times a vector, skip distance equals 1

SYNOPSIS

CALL MXV(*a*,*nar*,*b*,*nbr*,*c*)

DESCRIPTION

a	Matrix of the product
nar	Number of rows of matrices *a* and *c*
b	Vector of the product
nbr	Number of elements of vector *b* and the number of columns of matrix *a*
c	Resulting vector

The Fortran equivalent of MXV is as follows:

```
      DO 10 I=1, NAR
   10 C(I)=A(I,1)*B(1)+A(I,2)*B(2)+...+A(I,NBR)*B(NBR)
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

MXM is similar to MXV; however, MXM handles the case of a matrix times a matrix.

NAME

MXVA – Computes a matrix times a vector, arbitrary skip distance

SYNOPSIS

CALL MXVA(*a,na,iad,b,nb,c,nc,nar,nbr*)

DESCRIPTION

a	First matrix of the product
na	Spacing between column elements of *a*
iad	Spacing between row elements of *a*
b	Vector of the product
nb	Spacing between elements of *b*
c	Result vector
nc	Spacing between elements of *c*
nar	Number of rows in the first operand and number of elements in the result
nbr	Number of columns in the first operand and number of elements in the second operand

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

MXMA is similar to MXVA; however, MXMA handles the case of a matrix times a matrix.

NAME

OPFILT – Solves Weiner-Levinson linear equations

SYNOPSIS

CALL OPFILT(*m,a,b,c,r*)

DESCRIPTION

m	Order of system of equations
a	Output vector of filter coefficients
b	Information auto-correlation vector
c	Scratch vector of length 2*m*
r	Signal auto-correlation vector

OPFILT computes the solution to the Weiner-Levinson system of linear equations Ta=b, where T is a Toeplitz matrix in which elements are described by the following:

$$t_{ij} = R(k) \quad \text{for } |j-i|+1=k \quad \text{and} k = 1, \ldots, n$$

NOTE

Although **OPFILT** solves this matrix equation faster than Gaussian elimination can, **OPFILT** does no pivoting; therefore, it is less numerically stable than Gaussian elimination unless the matrix **T** is positive, definite, or diagonally dominant.

EXAMPLE

The following system of linear equations can be solved with the call **OPFILT (3,A,B,C,R)**. The array **C** is one dimension with a length of at least six. (The t_{ij} elements show how the numbers for **R** are obtained.)

$$\begin{bmatrix} R(1) & R(2) & R(3) \\ R(2) & R(1) & R(2) \\ R(3) & R(2) & R(1) \end{bmatrix} \begin{bmatrix} A(1) \\ A(2) \\ A(3) \end{bmatrix} = \begin{bmatrix} B(1) \\ B(2) \\ B(3) \end{bmatrix}$$

$$\begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

RECPP – Solves for a partial products problem

SYNOPSIS

CALL RECPP(*n,x,incx,c,incc*)

DESCRIPTION

n	Length of linear recurrence
x	Vector x
incx	Skip distance of vector x
c	Vector c
incc	Skip distance of vector c

RECPP solves the partial products problem as in the following Fortran equivalent:

```
      I=1
      J=1
      IF (INCX .LT. 0) THEN
            I = 1-(N-1)*INCX
      ENDIF
      IF (INCC. LT. 0) THEN
            J = 1-(N-1)*INCC
      ENDIF
      X(I) = C(J)
      DO 10 I=2,N
            X(I+INCX) = C(J+INCC)*X(I)
            J = J+INCC
            I = I+INCX
 10   CONTINUE
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

RECPS – Solves for the partial summation problem

SYNOPSIS

CALL RECPS(*n,x,incx,c,incc***)**

DESCRIPTION

n	Length of linear recurrence
x	Vector x
incx	Skip distance of vector x
c	Vector c
incc	Skip distance of vector c

RECPS solves the partial summation problem as in the following Fortran equivalent:

```
      I=1
      J=1
      IF (INCX .LT. 0) THEN
              I = 1-(N-1)*INCX
      ENDIF
      IF (INCC .LT. 0) THEN
              J = 1-(N-1)*INCC
      ENDIF
      X(I) = C(J)
      DO 10 I=2,N
              X(I+INCX) = C(J+INCC)+X(I)
              J = J+INCC
              I = I+INCX
10    CONTINUE
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

SASUM, SCASUM – Sums the absolute value of elements of a vector

SYNOPSIS

sum=SASUM(*n*,*sx*,*incx*)

sum=SCASUM(*n*,*cx*,*incx*)

DESCRIPTION

n	Number of elements in the vector to be summed. If $n \le 0$, **SASUM** and **SCASUM** return 0.
sx	Real vector to be summed
cx	Complex vector to be summed
incx	Increment between elements of *sx* or *cx*. For contiguous elements, *incx*=1.

SASUM and **SCASUM** sum the absolute values of the elements of a real or complex vector, respectively. **SASUM** computes

$$sum = \sum_{i=1}^{n} | x_{k_i} | \text{ where } k_i = \begin{cases} 1+(i-1)(incx), & incx>0 \\ 1+(n-i)\,|incx|, & incx<0 \end{cases}$$ and where x_{k_i} is an element of a real vector.

SCASUM computes

$$sum = \sum_{i=1}^{n} \{ |\text{real}(x_{k_i})| + |\text{imag}(x_{k_i})| \}$$ where k_i is as defined for **SASUM**. x_{k_i} is an element of a complex vector.

Note that **SASUM** computes a true l_1 norm, but **SCASUM** does not compute a true l_1 norm.

EXAMPLE

```
      REAL SUM,SUMMER(10)
      SUMMER(1)=0.0
      DO 10 I=2,10
      SUMMER(I)=SUMMER+1.0
10    CONTINUE
      SUM=SASUM(5,SUMMER,2)
      PRINT *,SUM
      STOP
      END
```

In the preceding example, **SUMMER(1)=0.0, SUMMER(2)=1.0, SUMMER(2)=3.0,...SUMMER(10)=9.0.** The printed result of **SUM** equals 20.0.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SAXPY, CAXPY – Adds a scalar multiple of a real or complex vector to another vector

SYNOPSIS

CALL SAXPY(*n,sa,sx,incx,sy,incy*)

CALL CAXPY(*n,ca,cx,incx,cy,incy*)

DESCRIPTION

n	Number of elements in the vectors. If $n \le 0$, SAXPY and CAXPY return without any computation.
sa	Real scalar multiplier
ca	Complex scalar multiplier
sx	Real scaled vector
cx	Complex scaled vector
incx	Increment between elements of *sx* or *cx*; for contiguous elements, $incx \pm 1$.
sy	Real result vector
cy	Complex result vector
incy	Increment between elements of *sy* or *cy*; for contiguous elements, $incy \pm 1$.

These subroutines add a scalar multiple of one vector to another.

SAXPY computes

$$y_{l_i} = ax_{k_i} + y_{l_i}\ ,\quad i=1,\ldots,n \text{ where } k_i = \begin{cases} 1+(i-1)(incx),\ incx>0 \\ 1+(n-i)\,|\,incx\,|,\ incx<0 \end{cases} \text{ and } l_i = \begin{cases} 1+(i-1)(incy),\ incy>0 \\ 1+(n-i)\,|\,incy\,|,\ incy<0 \end{cases}$$

where a is a real scalar multiplier and x_{k_i} and y_{l_i} are elements of real vectors.

CAXPY computes

$y_{l_i} = ax_{k_i} + y_{l_i}$, i=1,..., n and k_i and l_i are as defined for SAXPY.

a is a complex scalar multiplier and x_{k_i} and y_{l_i} are elements of complex vectors.

When $n \le 0$, sa=0, or ca=0+0i, these routines return immediately with no change in their arguments.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SSCAL, CSSCAL, CSCAL – Scales a real or complex vector

SYNOPSIS

CALL SSCAL(*n,sa,sx,incx*)

CALL CSSCAL(*n,sa,cx,incx*)

CALL CSCAL(*n,ca,cx,incx*)

DESCRIPTION

n	Number of elements in the vector. If $n \leq 0$, SSCAL, CSSCAL, and CSCAL return without any computation.
sa	Real scaling factor
ca	Complex scaling factor
sx	Real vector to be scaled
cx	Complex vector to be scaled
incx	Skip distance between elements of *sx* or *cx*

These subroutines scale a vector. SSCAL computes

$$X=aX$$

where a is a real number and X is a real vector. CSSCAL computes

$$X=aX$$

where a is a real number and X is a complex vector. CSCAL computes

$$Y=aY$$

where a is a complex number and Y is a complex vector.

CAUTIONS

Do not specify *incx* as zero; doing so yields unpredictable results.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SCATTER – Scatters a vector into another vector

SYNOPSIS

CALL SCATTER(*n*,*a*,*index*,*b*)

DESCRIPTION

n	Number of elements in vectors *index* and *b*
a	Output vector
index	Vector of indexes
b	Source vector

SCATTER is defined in the following way:

$$a_{j_i} = b_i \qquad \text{where } i = 1, \ldots, n$$

In Fortran:

```
A(INDEX(I))=B(I)
```

where I = 1,..., N

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

SCOPY, CCOPY – Copies a real or complex vector into another vector

SYNOPSIS

CALL SCOPY(*n,sx,incx,sy,incy*)

CALL CCOPY(*n,cx,incx,cy,incy*)

DESCRIPTION

n	Number of elements in the vector to be copied. If $n \le 0$, SCOPY and CCOPY return without any computation.
sx	Real vector to be copied
cx	Complex vector to be copied
incx	Increment between elements of *sx* or *cx*; for contiguous elements, $incx = \pm 1$.
sy	Real result vector
cy	Complex result vector
incy	Increment between elements of *sy* or *cy*; for contiguous elements, $incy = \pm 1$.

These subroutines copy one vector into another.

SCOPY copies a real vector

$y_{l_i} = x_{k_i}$, $i = 1,\ldots,n$ where $k_i = \begin{cases} 1+(i-1)(incx), & incx>0 \\ 1+(n-i)\,|incx|, & incx<0 \end{cases}$ and $l_i = \begin{cases} 1+(i-1)(incy), & incy>0 \\ 1+(n-i)\,|incy|, & incy<0 \end{cases}$ and x_{k_i} and y_{l_i} are elements of real vectors.

CCOPY copies a complex vector

$y_{l_i} = x_{k_i}$, $i = 1,\ldots,n$ where k_i and l_i are as defined above and x_{k_i} and y_{l_i} are elements of complex vectors.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS systems.

NAME

SGBMV – Multiplies a real vector by a real general band matrix

SYNOPSIS

CALL SGBMV(*trans,m,n,kl,ku,alpha,a,lda,x,incx,beta,y,incy*)

DESCRIPTION

SGBMV performs one of the matrix-vector operations

*y:=alpha*a*x+beta*y* or *y:=alpha*a'*x+beta*y*

Arguments *alpha* and *beta* are scalars, *x* and *y* are vectors, *a* is an *m*-by-*n* band matrix, with *kl* sub-diagonals and *ku* super-diagonals, and *a'* denotes the transpose of *a*.

trans Character*1. On entry, *trans* specifies the operation to be performed. If *trans*='N' or 'n', *y* := *alpha*a*x* + *beta*y*. If *trans*='T' or 't', *y* := *alpha*a'*x* + *beta*y*. The *trans* argument is unchanged on exit.

m Integer. On entry, *m* specifies the number of rows of the matrix *a*. *m* must be at least zero. The *m* argument is unchanged on exit.

n Integer. On entry, *n* specifies the number of columns of the matrix *a*. *n* must be at least zero. The *n* argument is unchanged on exit.

kl Integer. On entry, *kl* specifies the number of sub-diagonals of the matrix *a*. *kl* must satisfy 0.LE.*kl*. The *kl* argument is unchanged on exit.

ku Integer. On entry, *ku* specifies the number of super-diagonals of the matrix *a*. *ku* must satisfy 0.LE.*ku*. The *ku* argument is unchanged on exit.

alpha Real. On entry, *alpha* specifies the scalar alpha. The *alpha* argument is unchanged on exit.

a Real array of dimension (*lda,n*). Before entry, the leading (*kl* + *ku* + 1)-by-*n* part of the array *a* must contain the matrix of coefficients, supplied column-by-column, with the leading diagonal of the matrix in row (*ku* + 1) of the array, the first superdiagonal starting at position 2 in row *ku*, the first subdiagonal starting at position 1 in row (*ku* + 2), and so on. Elements in the array *a* that do not correspond to elements in the band matrix (such as the top left *ku*-by-*ku* triangle) are not referenced. The following program segment will transfer a band matrix from conventional full matrix storage to band storage:

```
         DO 20, J=1,N
         K = KU+1-J
         DO 10, I=MAX(1,J-KU), MIN(M,J+KL)
                A(K+I,J) = MATRIX(I,J)
   10       CONTINUE
   20    CONTINUE
```

The *a* argument is unchanged on exit.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling (sub) program. *lda* must be at least (*kl*+*ku*+1). The *lda* argument is unchanged on exit.

x Real array of dimension at least (1+(*n*−1)* abs(*incx*)) when *trans*='N' or 'n' and at least (1+(*m*−1)* abs(*incx*)) otherwise. Before entry, the incremented array *x* must contain the vector *x*. The *x* argument is unchanged on exit.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

beta Real. On entry, *beta* specifies the scalar beta. When *beta* is supplied as zero, *y* need not be set on input. The *beta* argument is unchanged on exit.

y Real array of dimension at least (1+(*m*−1)* abs(*incy*)) when *trans*='N' or 'n' and at least (1+(*n*−1)* abs(*incy*)) otherwise. Before entry, the incremented array *y* must contain the vector *y*. On exit, *y* is overwritten by the updated vector *y*.

incy Integer. On entry, *incy* specifies the increment for the elements of *y*. *incy* must not be zero. The *incy* argument is unchanged on exit.

NOTE

The SGBMV routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

SGEMV – Multiplies a real vector by a real general matrix

SYNOPSIS

CALL SGEMV(*trans,m,n,alpha,a,lda,x,incx,beta,y,incy*)

DESCRIPTION

SGEMV performs one of the matrix-vector operations

*y := alpha*a*x + beta*y*, or *y := alpha*a'*x + beta*y*,

Arguments *alpha* and *beta* are scalars, *x* and *y* are vectors and *a* is an *m*-by-*n* matrix.

trans Character*1. On entry, *trans* specifies the operation to be performed. If *trans*='N' or 'n', *y := alpha*a*x + beta*y*. If *trans*='T' or 't', *y := alpha*a'*x + beta*y*. The *trans* argument is unchanged on exit.

m Integer. On entry, *m* specifies the number of rows of the matrix *a*. *m* must be at least zero. The *m* argument is unchanged on exit.

n Integer. On entry, *n* specifies the number of columns of the matrix *a*. *n* must be at least zero. The *n* argument is unchanged on exit.

alpha Real. On entry, *alpha* specifies the scalar alpha. The *alpha* argument is unchanged on exit.

a Real array of dimension (*lda,n*). Before entry, the leading *m* by n part of the array *a* must contain the matrix of coefficients. The *a* argument is unchanged on exit.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling subprogram. *lda* must be at least **max(1,***m*). The *lda* argument is unchanged on exit.

x Real array of dimension at least (1 + (*n* - 1)***abs**(*incx*)) when *trans*='N' or '**n**' and at least (1 + (*m* - 1)***abs**(*incx*)) otherwise. Before entry, the incremented array *x* must contain the vector *x*. The *x* argument is unchanged on exit.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

beta Real. On entry, *beta* specifies the scalar beta. When *beta* is supplied as zero then *y* need not be set on input. The *beta* argument is unchanged on exit.

y Real array of dimension at least (1 + (*m* - 1)*abs(*incy*)) when trans='N' or '**n**' and at least (1 + (*n* - 1)***abs**(*incy*)) otherwise. Before entry with *beta* nonzero, the incremented array *y* must contain the vector *y*. On exit, *y* is overwritten by the updated vector *y*.

incy Integer. On entry, *incy* specifies the increment for the elements of *y*. *incy* must not be zero. The *incy* argument is unchanged on exit.

NOTE

The SGEMV routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

SGER – Performs the rank 1 update of a real general matrix

SYNOPSIS

CALL SGER(*m,n,alpha,x,incx,y,incy,a,lda*)

DESCRIPTION

SGER performs the rank 1 operation

*a := alpha*x*y' + a*

where *x* is an *m* element vector, *y* is an *n* element vector and *a* is an *m*-by-*n* matrix. **SGER** has the following arguments:

m Integer. On entry, *m* specifies the number of rows of the matrix *a*. *m* must be at least zero. Unchanged on exit.

n Integer. On entry, *n* specifies the number of columns of the matrix *a*. *n* must be at least zero. Unchanged on exit.

alpha Real. On entry, *alpha* specifies the scalar alpha. Unchanged on exit.

x Real. Array of dimension at least (1 + (*m* - 1)*abs(*incx*)). Before entry, the incremented array *x* must contain the *m* element vector *x*. Unchanged on exit.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. Unchanged on exit.

y Real. Array of dimension at least (1 + (*n* - 1)*abs(*incy*)). Before entry, the incremented array *y* must contain the *n* element vector *y*. Unchanged on exit.

incy Integer. On entry, *incy* specifies the increment for the elements of *y*. *incy* must not be zero. Unchanged on exit.

a Real array of dimension (*lda,n*). Before entry, the leading *m*-by-*n* part of the array *a* must contain the matrix of coefficients. On exit, *a* is overwritten by the updated matrix.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling subprogram. *lda* must be at least max(1,*m*). Unchanged on exit.

NOTE

The **SGER** routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

SMXPY – Computes the product of a column vector and a matrix and adds the result to another column vector

SYNOPSIS

CALL SMXPY(*n1*,*y*,*n2*,*ldm*,*x*,*m*)

DESCRIPTION

n1	Number of elements in the vector y
y	Real vector
n2	Number of elements in the vector x
ldm	Leading dimension of matrix m
x	Real vector
m	Matrix

SMXPY executes an operation equivalent to the following Fortran code:

```
      SUBROUTINE SMXPY(N1,Y,N2,LDM,X,M)
      REAL Y(1), X(1), M(LDM,1)
      DO 20 J=1,N2
              DO 20 I=1,N1
                 Y(I)=Y(I) + X(J) * M(I,J)
   20 CONTINUE
      RETURN
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

SNRM2, SCNRM2 – Computes the Euclidean norm of a vector

SYNOPSIS

eucnorm=SNRM2(*n*,*sx*,*incx*)

eucnorm=SCNRM2(*n*,*cx*,*incx*)

DESCRIPTION

n	Number of elements in the vector. If $n \le 0$, SNRM2 and SCNRM2 return without any computation.
sx	Real vector operand
cx	Complex vector operand
incx	Skip distance between elements of *sx* or *cx*

These real functions compute the Euclidean or l_2 norm of a vector.

SNRM2 computes

$$eucnorm = \left[\sum_{i=1}^{n} x_i^2 \right]^{\frac{1}{2}}$$

where x_i is an element of a real vector. SCNRM2 computes

$$eucnorm = \left[\sum_{i=1}^{n} x_i \, \overline{x_i} \right]^{\frac{1}{2}}$$

where x_i is a complex vector and $\overline{x_i}$ is the complex conjugate of x_i.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SOLR, SOLRN, SOLR3 – Solves second-order linear recurrences

SYNOPSIS

CALL SOLR(*n*,*a*,*inca*,*b*,*incb*,*c*,*incc*)

result=SOLRN(*n*,*a*,*inca*,*b*,*incb*,*c*,*incc*)

CALL SOLR3(*n*,*a*,*inca*,*b*,*incb*,*c*,*incc*)

DESCRIPTION

n	Length of linear recurrence. For SOLR and SOLR3, if $n \le 0$, SOLR and SOLR3 return without any computation. For SOLRN, 0 is returned.
a	Vector *a* of length *n* of equation below
inca	Skip distance between elements of the vector operand *a*
b	Vector *b* of length *n* of equation below
incb	Skip distance between elements of the vector operand *b*
c	Vector result *c* of length *n*+2 of equation below
incc	Skip distance between elements of the vector result *c*. C(1) and C(2) are input to this routine; C(3),C(4),...,C(N+2) are output from this routine.

SOLR solves a second-order linear recurrence. SOLRN solves a second-order linear recurrence for the last term only. SOLR3 solves a second-order linear recurrence for three terms.

SOLR solves second-order linear recurrences as in the following equation:

$$c_i = a_{i-2}\, c_{i-1} + b_{i-2}\, c_{i-2} \quad \text{for } i=3,\ldots,n$$

The Fortran equivalent of equation 3 is:

```
      DO 10 I=3,N
10      C(I)=A(I-2)*C(I-1) + B(I-2)*C(I-2)
```

SOLRN, a real function, solves for only the last term of a second-order linear recurrence, that is, *c*(*n*) of SOLR(*n*,*a*,*inca*,*b*,*incb*,*c*,*incc*).

The Fortran loop

```
      R1=C(1)
      R2=C(2)
      DO 10 I=3,N
            TEMP=R2
            R2=A(I-2)*R2+B(I-22)*R1
            R1=TEMP
10    CONTINUE
      RESULT=R2
```

could be solved as follows:

result=SOLRN(n,a,1,b,1,c)

SOLR3 computes a second-order linear recurrence of three terms, that is

$$c_1 = c_1 \; c_2 = c_2 \; c_i = c_i + a_{i-2}\, c_{i-1} + b_{i-2}\, c_{i-2} \quad \text{for } i=3, \ldots n$$

CAUTIONS

Do not specify *inca*, *incb*, or *incc* as zero; doing so yields unpredictable results.

EXAMPLES

Example 1 – **SOLRN**:

SOLRN might be used to find r_n of the calculation

$$\prod_{i=1}^{n-2} \begin{bmatrix} a_i & b_i \\ 1 & 0 \end{bmatrix} \begin{bmatrix} c_2 \\ c_1 \end{bmatrix} = \begin{bmatrix} r_n \\ r_{n-1} \end{bmatrix}$$

with the following call:

r_n =SOLRN(n,a,1,b,1,c,1)

The Fortran equivalent for example 1 is as follows:

```
      R1=C(1)
      R2=C(2)
      DO 10 I=1,N
            TEMP=R2
            R2=A(I)*R2+B(I)*R1
            R1=TEMP
   10 CONTINUE
      RN=R2
```

Example 2 – **SOLR3**:

SOLR3 solves a system of lower bidiagonal linear equations Lx=b.

$$Lx = \begin{bmatrix} 1\,0\,0\,0 \ldots\ldots 0 \\ e_1\,1\,0\,0 \ldots\ldots 0 \\ f_1\,e_2\,1\,0 \ldots\ldots 0 \\ 0\,f_2\,e_3\,1\,0 \ldots\ldots 0 \\ .\,0\,f_3\,e_4\,1\,0 \ldots 0 \\ \ldots\ldots\ldots\ldots 0 \\ \ldots\ldots\ldots\ldots 0 \\ \ldots\ldots\ldots\ldots 0 \\ 0\,0\,0\,.\,f_{n-2}\,e_{n-1}\,1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ . \\ . \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ . \\ . \\ . \\ . \\ b_n \end{bmatrix} = b$$

Then there is

$$x_1 = b_1$$

$$x_2 = b_2 - e_1 x_1$$

$$x_i = b_i - e_{i-1} x_{i-1} - f_{i-2} x_{i-2} \qquad i=3, \ldots, n$$

This problem can be solved with the following Fortran:

```
      DO 10 I=1,N-1
10       E(I)=-E(I)
      DO 20 I=1,N-2
20       F(I)=-F(I)
      B(1)=B(1)
      B(2)=B(2)+E(1)*B(1)
      CALL SOLR3(N,E(2),1,F(1),1,B(1),1)
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SPDOT, SPAXPY – Primitives for the lower upper factorization and solution of sparse linear systems

SYNOPSIS

pdot=SPDOT(*n*,*sy*,*index*,*sx*)

CALL SPAXPY(*n*,*sa*,*sx*,*sy*,*index*)

DESCRIPTION

For SPDOT:

n	Number of elements in the vectors
sy	Real vector operand
index	Vector of indexes ascending order.
sx	Real vector operand

For SPAXPY:

n	Numbers of elements in the vectors
sa	Real scalar multiplier
sx	Real vector operand
sy	Real vector operand
index	Vector of indexes. All values in *index* should be unique and in ascending order.

SPAXPY executes an operation equivalent to the following Fortran code:

```
      DO 10 I=1,N
10    SY(INDEX(I))=SA*SX(I)+SY(INDEX(I))
```

SPDOT executes an operation equivalent to the following Fortran code:

```
      DO 10 I=1,N
10    PDOT=PDOT+SY(INDEX(I))*SX(I)
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SROT – Applies an orthogonal plane rotation

SYNOPSIS

CALL SROT(*n*,*sx*,*incx*,*sy*,*incy*,*c*,*s*)

DESCRIPTION

n	Number of elements in the vector
sx	Real vector to be modified
incx	Skip distance between elements of *sx*
sy	Real vector to be modified
incy	Skip distance between elements of *sy*. For contiguous elements, *incy*=1.
c	Real cosine of equation 1. Normally calculated using **SROTG**.
s	Real sine of equation 1. Normally calculated using **SROTG**.

This subroutine performs a matrix multiplication. If the coefficients c and s satisfy $c * c + s * s = 1.0$, the transformation is a Givens rotation. The coefficients c and s can be calculated from *sx* and *sy* using **SROTG**. **SROT** computes equation 1 on each pair of elements x_i, y_i of real arrays.

Equation 1:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} := \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \bullet \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{for } i=1, \ldots, n$$

SROT returns without modification to any input parameters if c=1 and s=0.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

SROTG

NAME

SROTG – Constructs a Givens plane rotation

SYNOPSIS

CALL SROTG(*a*,*b*,*c*,*s*)

DESCRIPTION

a	Scalar a of equation 2
b	Scalar b of equation 2
c	Scalar cosine of equation 2
s	Scalar sine of equation 2

SROTG computes the elements of a rotation matrix. The previous call calculates the parameters r, z, c, and s from input coordinates a, b as in the following equation:

$$\begin{bmatrix} r \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

Given a and b each of these subroutines computes

$$\sigma = \begin{cases} sgn(a) & \text{if } |a| > |b| \\ sgn(b) & \text{if } |b| \le |a| \end{cases}$$

$$r = \sigma(a^2+b^2)^{1/2}$$

$$c = \begin{cases} 1/r & \text{if } r \ne 0 \\ 1 & \text{if } r = 0 \end{cases}$$

$$s = \begin{cases} b/r & \text{if } r \ne 0 \\ 0 & \text{if } r = 0 \end{cases}$$

σ is not needed in computing a Givens rotation matrix; however, its use permits later reconstruction of c and s from just one number. For this reason parameter z is also calculated as follows:

$$z = \begin{cases} s & \text{if } |a| > |b| \\ 1/c & \text{if } |b| \ge |a| \text{ and } c \ne 0 \\ 1 & \text{if } c = 0 \end{cases}$$

The subroutine uses parameters a and b and returns r, z, c, and s, where r overwrites a, and z overwrites b.

A later reconstruction of c and x from z can be done as follows:

If $z = 1$ set $c = 0$ and $s = 1$

If $|z| < 1$ set $c = (1-z^2)^{½}$ and $s = z$

If $|z| > 1$ set $c = 1/z$ and $s = (1-c^2)^{½}$

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

SROT

NAME

SROTM – Applies a modified Givens plane rotation

SYNOPSIS

CALL SROTM(*n,sx,incx,sy,incy,param*)

DESCRIPTION

SROTM applies the modified Givens plane rotation constructed by SROTMG. It computes

$$\begin{bmatrix} x_i \\ \hline y_i \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ \hline h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ \hline y_i \end{bmatrix} : i=1 , \ldots , n$$

where the parameters H11, H21, H12, and H22 are passed in the array **PARAM** according to the following schedule: **PARAM(1)** is the key parameter having values 1.0, 0.0, -1.0, or -2.0.

Case for which **PARAM(1)=1.0**:

H11=PARAM(2)

H21=-1.0

H12=1.0

H22=PARAM(5)

and **PARAM(3)** and **PARAM(4)** are ignored.

Case for which **PARAM(1)=0.0**:

H11=1.0

H21=PARAM(3)

H12=PARAM(4)

H22=1.0

and **PARAM(2)** and **PARAM(5)** are ignored.

Case for which **PARAM(1)=-1.0** is rescaling case:

H11=PARAM(2)

H21=PARAM(3)

H12=PARAM(4)

H22=PARAM(5)

is a full matrix multiplication.

Case for which **PARAM(1)=2.0** is H=1, namely:

H11=1.0

H21=0.0

H12=0.0

H22=1.0

and **PARAM(2)**, **PARAM(3)**, **PARAM(4)**, and **PARAM(5)** are ignored. If H=1, **SROTM** returns with no operation on input arrays *sx*, *sy*.

If any other value for **PARAM(1)** is read (other than 1., 0, -1., or -2.), **SROTM** aborts the job with the following message appearing in the logfile:

SROTM CALLED WITH INCORRECT PARAMETER KEY

The array **PARAM** must be declared in a dimension statement in the calling program, as follows:

DIMENSION PARAM(5)

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

See the description of **SROTMG** for further details about the modified Givens transformation and the array **PARAM**.

NAME

SROTMG – Constructs a modified Givens plane rotation

SYNOPSIS

CALL SROTMG(d_1 , d_2 , b_1 , b_2 , *param*)

DESCRIPTION

SROTMG computes the elements of a modified Givens plane rotation matrix.

SROTMG sets up parameters *param* from inputs d_1, d_2, b_1, and b_2.

An application of the Givens plane rotation

$$\begin{bmatrix} x' \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = G \begin{bmatrix} x \\ y \end{bmatrix}$$

can be written in a form such that repeated applications require matrix multiplications by matrices containing only two nonunit elements. Row transformations require only 2N rather than 4N multiplications. Scale factors d_1 , d_2 are defined such that

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sqrt{d_1} & 0 \\ 0 & \sqrt{d_2} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = D^{\frac{1}{2}} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

where the scaling upon each application of the G's is updated. Let H be a matrix

$$H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$$

such that

$$G \begin{bmatrix} x \\ y \end{bmatrix} = D'^{\frac{1}{2}} \; H \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

where $D'^{\frac{1}{2}} = diag\left\{\sqrt{d'_1} \; , \; \sqrt{d'_2}\right\}$ contains the updated scale factors; therefore, H is chosen according to equation 3 or 4.

Equation 3:

$$\begin{bmatrix} x' \\ 0 \end{bmatrix} = D'^{\frac{1}{2}} \; H \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Equation 4:

$$\begin{bmatrix} \sqrt{d'_1} * h_{11} & \sqrt{d'_1} * h_{12} \\ \sqrt{d'_2} * h_{21} & \sqrt{d'_2} * h_{22} \end{bmatrix} = \begin{bmatrix} \sqrt{d_1}\, c & d_2\, s \\ -\sqrt{d_1}\, s & d_2\, c \end{bmatrix}$$

Coefficients c and s are determined by equations 5 and 6.

Equation 5:

$$c = \frac{x}{\sqrt{x^2+y^2}} = \frac{\sqrt{d_1}b_1}{\sqrt{d_1b_1^2 + d_2b_2^2}},$$

Equation 6:

$$s = \frac{y}{\sqrt{x^2+y^2}} = \frac{\sqrt{d_2}b_2}{\sqrt{d_1b_1^2 + d_2b_2^2}}$$

Equation 4 shows that the d's are going to be scaled by c or s if two of the h's are to be unity. Two cases, $|c| > |s|$ and $|s| \geq |c|$, are considered so that the d's are scaled down the least upon repeated applications.

Case 1:

If $|c| > |s|$ (which from equations 5 and 6 is the same as $|d_1b_1^2| > |d_2b_2^2|$), the solutions for equation 4 are determined by equation 7.

Equation 7:

$$h_{11} = h_{22} = 1$$

Case 2:

If $|s| \geq |c|$ (which is $|d_2b_2^2| \geq |d_1b_1^2|$), equation 8 is chosen.

Equation 8:

$$h_{12} = -h_{21} = 1$$

Distinguishing the two cases $|c| > \frac{1}{\sqrt{2}}$ or $|s| \geq \frac{1}{\sqrt{2}}$ is the updating factor. Then the complete solutions for $D'^{\frac{1}{2}}$ and H are as follows.

Case 1:

In case 1, where $| c | > | s |$ or $| d_1 b_1^2 | > | d_2 b_2^2 |$, the following solutions for H are chosen:

$$h_{11} = 1 \qquad h_{12} = \frac{d_2 b_2}{d_1 b_1}$$

$$h_{21} = \frac{-b_2}{b_1} \qquad h_{22} = 1$$

and scale factors d_1, d_2 are updated to

$$d'_1 = d_1 / \mathrm{u} = c^2 d_1$$

$$d'_2 = d_2 / \mathrm{u} = c^2 d_2$$

where

$$u = det\ (\mathrm{H}) = \frac{d_2 b_2^2}{d_1 b_1^2}$$

Since $x' = r, y' = 0$, and $b'_1 = x' / \sqrt{d'_1}$, then $b'_1 = b'_1 \cdot u'$ is updated.

Case 2:

In case 2, where $| s | \geq | c |$ or $| d_1 b_1^2 | \leq | d_2 b_2^2 |$, the following solutions for H are chosen:

$$h_{11} = \frac{d_1 b_1}{d_2 b_2} \qquad h_{12} = 1$$

$$h_{21} = -1 \qquad h = \frac{b_1}{b_2}$$

Scale factors d_1 are updated to

$$d'_1 = d_2/u$$

$$d'_2 = d_1/u$$

with

$$u = det\ (\mathrm{H}) = 1 + \frac{d_1\, b_1^2}{d_2\, b_2^2}$$

and the x' factor becomes

$$b'_1 = b_2.u.$$

Case 3:

Let $m = 4096$. Whenever the parameters d_i are updated to be outside the window

$$(m)^{-2} \le |\, d'_1 \,| \le (m)^2$$

which preserves about 36 = 48 - 12 bits or 10 decimal digits of precision, all parameters are rescaled such that the d_i's are within that window. If either of the d_i's is 0, however, no rescaling action is taken.

Underflow:

If $|\, d'_1 \,| < (m)^{-2}$, the following is set:

$$d'_i := d'_i \,.\, (m)^2 \qquad h'_{i1} := h'_{i1} \,.\, (m)^{-1}$$

$$b'_1 := b'_1 \,.\, (m)^{-1} \qquad h'_{i2} := h'_{i2} \,.\, (m)^{-1}$$

Overflow:

If $|\, d'_1 \,| > (m)^2$, the following is set:

$$d'_i := d'_i \,.\, (m)^{-2} \qquad h'_{i1} := h'_{i1} \,.\, (m)$$

$$b'_1 := b'_1 \,.\, (m) \qquad h'_{i2} := h'_{i2} \,.\, (m)$$

SROTMG modifies the input parameters **D1, D2,** and **B1** and returns the array **PARAM** according to the following cases:

Case 4:

If **ABS(D1*B1*B1).GT.ABS(D2*B2*B2)**, then

```
PARAM(1)=0
PARAM(3)=-B2/B1
PARAM(4)=D2*B2/D1*B1
```

and parameters **D1, D2,** and **B1** are written over by

```
D1=D1/U
D2=D2/U
B1=B1*U
```

where

```
U=1.+(D2*B2*B2)/(D1*B1*B1).
```

Case 5:

If **ABS(D2*B2*B2).GE.ABS(D1*B1*B1)**, then

```
PARAM(1)=1.
PARAM(2)=(D1*B1)/(D2*B2)
PARAM(5)=B1/B2
```

and parameters **D1, D2,** and **B1** are written over according to the following sequence:

```
TEMP=D1/U
D1=D2/U
B1=B2*U
```

U=1.+(D1*B1*B1)/(D2*B2*B2)

Case 6:

If, in either case 4 or case 5, the updated parameters **D1** and **D2** have been rescaled below/above the window

(*m*)**(-2).LE.ABS(D1).LE.(*m*)**2
(*m*)**(-2).LE.ABS(D2).LE.(*m*)**2

then the parameters **D1, H11, H12, B1** and **D2, H21, H22**, respectively, are rescaled up/down by factors of *m*. Rescaling occurs as many times as necessary to bring **D1** or **D2** within the preceding window. If **D1** and **D2** are within the window on entry, rescaling occurs only once.

Output parameters are

PARAM(1)=-1.
PARAM(2)=H11
PARAM(3)=H21
PARAM(4)=H12
PARAM(5)=H22

and **D1, D2,** and **B1** are written over by correctly scaled versions of case 5 or case 6.

If **D1<0**, the matrix **H=0** is generated (that is, $h_{11} = h_{12} = h_{21} = h_{22} = 0$). **PARAM(1)=-1**, and the rest of the elements of **PARAM** contain 0.

Case 7:

If **D2*B2=0** on entry, then **H=1**. Output is

PARAM(1)=-2.0

only.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

SROTG

NAME

SSBMV – Multiplies a real vector by a real symmetric band matrix

SYNOPSIS

CALL SSBMV(*uplo,n,k,alpha,a,lda,x,incx,beta,y,incy*)

DESCRIPTION

SSBMV performs the matrix-vector operation

*y:=alpha*a*x+beta*y*

where *alpha* and *beta* are scalars, *x* and *y* are *n* element vectors, and *a* is an *n*-by-*n* symmetric band matrix, with *k* super-diagonals. SSBMV has the following arguments:

uplo Character*1. On entry, *uplo* specifies whether the upper or lower triangular part of the band matrix *a* is being supplied. When *uplo*='U' or 'u', only the upper triangular part of array *a* is to be referenced. When *uplo*='L' or 'l', only the lower triangular part of array *a* is to be referenced. The *uplo* argument is unchanged on exit.

n Integer. On entry, *n* specifies the order of the matrix *a*. The *n* argument must be at least zero. The *n* argument is unchanged on exit.

k Integer. On entry, *k* specifies the number of super-diagonals of the matrix *a*. *k* must satisfy 0.LE.*k*. The *k* argument is unchanged on exit.

alpha Real. On entry, *alpha* specifies the scalar alpha. The *alpha* argument is unchanged on exit.

a Real array of dimension (*lda,n*). Before entry with *uplo*='U' or 'u', the leading (*k*+1)-by-*n* part of the array *a* must contain the upper triangular band part of the symmetric matrix, supplied column-by-column, with the leading diagonal of the matrix in row (*k*+1) of the array, the first super-diagonal starting at position 2 in row *k*, and so on. The top left *k*-by-*k* triangle of the array *a* is not referenced. The following program segment will transfer the upper triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
      DO 20, J=1,N
      M = K+1-J
      DO 10, I=MAX(1,J-K), J
            A(M+I,J) = MATRIX(I,J)
10       CONTINUE
20    CONTINUE
```

Before entry with *uplo*='L' or 'l', the leading (*k*+1)-by-*n* part of the array *a* must contain the lower triangular band part of the symmetric matrix, supplied column-by-column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right *k*-by-*k* triangle of the array *a* is not referenced. The following program segment will transfer the lower triangular part of a symmetric band matrix from conventional full matrix storage to band storage:

```
      DO 20, J=1,N
      M = 1-J
      DO 10, I=J, MIN(N,J+K)
            A(M+I,J) = MATRIX(I,J)
10       CONTINUE
20    CONTINUE
```

The *a* argument is unchanged on exit.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling (sub)program. *lda* must be at least (k + 1). The *lda* argument is unchanged on exit.

x Real array of dimension at least (1+(n−1)* abs(*incx*)). Before entry, the incremented array *x* must contain the vector *x*. The *x* argument is unchanged on exit.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

beta Real. On entry, *beta* specifies the scalar beta. The *beta* argument is unchanged on exit.

y Real. Array of dimension at least (1+(n−1)* abs(*incy*)). Before entry, the incremented array *y* must contain the vector *y*. The *y* argument is unchanged on exit.

incy Integer. On entry, *incy* specifies the increment for the elements of *y*. *incy* must not be zero. The *incy* argument is unchanged on exit.

NOTE

The **SSBMV** routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

SSUM, CSUM – Sums the elements of a real or complex vector

SYNOPSIS

sum=SSUM(*n*,*sx*,*incx*)

sum=CSUM(*n*,*cx*,*incx*)

DESCRIPTION

n	Number of elements in the vector. If $n \le 0$, **SSUM** and **CSUM** return 0.
sx	Real vector to be summed
cx	Complex vector to be summed
incx	Skip distance between elements of *sx* or *cx*

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SSWAP, CSWAP – Swaps two real or complex arrays

SYNOPSIS

CALL SSWAP(*n*,*sx*,*incx*,*sy*,*incy*)

CALL CSWAP(*n*,*cx*,*incx*,*cy*,*incy*)

DESCRIPTION

n	Number of elements in the vector. If $n \leq 0$, SSWAP and CSWAP are returned.
sx	One real vector
cx	One complex vector
incx	Skip distance between elements of *sx* or *cx*
sy	Another real vector
cy	Another complex vector
incy	Skip distance between elements of *sy* or *cy*. For contiguous elements, *incy*=1.

SSWAP exchanges two real vectors. CSWAP exchanges two complex vectors.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SSYMV – Multiplies a real vector by a real symmetric matrix

SYNOPSIS

CALL SSYMV(uplo,n,alpha,a,lda,x,incx,beta,y,incy)

DESCRIPTION

SSYMV performs the matrix-vector operation

$$y := alpha*a*x + beta*y$$

where *alpha* and *beta* are scalars, *x* and *y* are *n* element vectors, and *a* is an *n*-by-*n* symmetric matrix. SSYMV has the following arguments:

uplo Character*1. On entry, *uplo* specifies whether the upper or lower triangular part of the band matrix *a* is being supplied. When *uplo*='U' or 'u', only the upper triangular part of array *a* is to be referenced. When *uplo*='L' or 'l', only the lower triangular part of array *a* is to be referenced. The *uplo* argument is unchanged on exit.

n Integer. On entry, *n* specifies the order of the matrix *a*. The *n* argument must be at least zero. The *n* argument is unchanged on exit.

alpha Real. On entry, *alpha* specifies the scalar alpha. The *alpha* argument is unchanged on exit.

a Real array of dimension (*lda*,*n*). Before entry with *uplo*='U' or 'u', the leading *n*-by-*n* upper triangular part of the array *a* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *a* is not referenced. Before entry with *uplo*='L' or 'l', the leading *n*-by-*n* part of the array *a* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *a* is not referenced. The *a* argument is unchanged on exit.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling subprogram. *lda* must be at least max(1,*n*). The *lda* argument is unchanged on exit.

x Real array of dimension at least $(1+(n-1)*\text{abs}(incx))$. Before entry, the incremented array *x* must contain the *n* element vector *x*. The *x* argument is unchanged on exit.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

beta Real. On entry, *beta* specifies the scalar beta. When *beta* is supplied as zero, *y* need not be set on input. The *beta* argument is unchanged on exit.

y Real. Array of dimension at least $(1+(n-1)*\text{abs}(incy))$. Before entry, the incremented array *y* must contain the *n* element vector *y*. On exit, *y* is overwritten by the updated vector *y*.

incy Integer. On entry, *incy* specifies the increment for the elements of *y*. *incy* must not be zero. The *incy* argument is unchanged on exit.

NOTE

The SSYMV routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

SSYR – Performs symmetric rank 1 update of a real symmetric matrix

SYNOPSIS

CALL SSYR(*uplo,n,alpha,x,incx,a,lda*)

DESCRIPTION

SSYR performs the symmetric rank 1 operation

$$a := alpha*x*x' + a$$

where *alpha* is a real scalar, *x* is an *n* element vector, and *a* is an *n*-by-*n* symmetric matrix. SSYR has the following arguments:

uplo Character*1. On entry, *uplo* specifies whether the upper or lower triangular part of the array *a* is to be referenced. When *uplo*='U' or 'u', only the upper triangular part of array a is to be referenced. When *uplo*='L' or 'l', only the lower triangular part of array a is to be referenced. The *uplo* argument is unchanged on exit.

n Integer. On entry, *n* specifies the number of columns of the matrix *a*. The *n* argument must be at least zero. The *n* argument is unchanged on exit.

alpha Real. On entry, *alpha* specifies the scalar alpha. The *alpha* argument is unchanged on exit.

x Real. Array of dimension at least (1+(*m*−1)* abs(*incx*)). Before entry, the incremented array *x* must contain the *m* element vector *x*. The *x* argument is unchanged on exit.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

a Real array of dimension (*lda,n*). Before entry, the leading *m*-by-*n* part of the array *a* must contain the matrix of coefficients. On exit, *a* is overwritten by the updated matrix.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling subprogram. *lda* must be at least **max**(1,*m*). The *lda* argument is unchanged on exit.

NOTE

The SSYR routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

SSYR2 – Performs symmetric rank 2 update of a real symmetric matrix

SYNOPSIS

CALL SSYR2(*uplo,n,alpha,x,incx,y,incy,a,lda*)

DESCRIPTION

SSYR2 performs the symmetric rank 2 operation

$$a := alpha*x*y' + alpha*y*x' + a$$

where *alpha* is a scalar, *x* and *y* are *n* element vectors, and *a* is an *n*-by-*n* symmetric matrix. **SSYR2** has the following arguments:

uplo Character*1. On entry, *uplo* specifies whether the upper or lower triangular part of the band matrix *a* is being supplied. When *uplo*='U' or 'u', only the upper triangular part of array *a* is to be referenced. When *uplo*='L' or 'l', only the lower triangular part of array *a* is to be referenced. The *uplo* argument is unchanged on exit.

n Integer. On entry, *n* specifies the order of the matrix *a*. The *n* argument must be at least zero. The *n* argument is unchanged on exit.

alpha Real. On entry, *alpha* specifies the scalar alpha. The *alpha* argument is unchanged on exit.

x Real array of dimension at least (1+(*n*−1)* abs(*incx*)). Before entry, the incremented array *x* must contain the *n* element vector *x*. The *x* argument is unchanged on exit.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

y Real. Array of dimension at least (1+(*n*−1)* abs(*incy*)). Before entry, the incremented array *y* must contain the *n* element vector *y*. The *y* argument is unchanged on exit.

incy Integer. On entry, *incy* specifies the increment for the elements of *y*. *incy* must not be zero. The *incy* argument is unchanged on exit.

a Real array of dimension (*lda*,n). Before entry with *uplo*='U' or 'u', the leading *n*-by-*n* upper triangular part of the array *a* must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of *a* is not referenced. On exit, the upper triangular part of the array *a* is overwritten by the upper triangular part of the updated matrix. Before entry with *uplo*='L' or 'l', the leading *n*-by-*n* lower triangular part of the array *a* must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of *a* is not referenced. On exit, the lower triangular part of the array *a* is overwritten by the lower triangular part of the updated matrix.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling (sub)program. *lda* must be at least max(1,*n*). The *lda* argument is unchanged on exit.

NOTE

The **SSYR2** routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

STBMV – Multiplies a real vector by a real triangular band matrix

SYNOPSIS

CALL STBMV(*uplo,trans,diag,n,k,a,lda,x,incx*)

DESCRIPTION

STBMV performs one of the matrix-vector operations

$x := a*x$ or $x := a'*x$

where x is an n element vector, and a is an n-by-n unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals. STBMV has the following arguments:

uplo Character*1. On entry, *uplo* specifies whether matrix is an upper of lower triangular matrix. When *uplo*='U' or 'u', *a* is an upper triangular matrix. When *uplo*='L' or 'l', *a* is a lower triangular matrix. The *uplo* argument is unchanged on exit.

trans Character*1. On entry, *trans* specifies the operation to be performed. If *trans* = 'N' or 'n', $x := a*x$. If *trans* = 'T' or 't', $x := a'*x$. The *trans* argument is unchanged on exit.

diag Character*1. On entry, *diag* specifies whether or not *a* is unit triangular. If *diag* = 'U' or 'u', *a* is assumed to be unit triangular. If *diag* = 'N' or 'n', *a* is not assumed to be unit triangular. The *diag* argument is unchanged on exit.

n Integer. On entry, *n* specifies the order of the matrix *a*. The *n* argument must be at least zero. The *n* argument is unchanged on exit.

k Integer. On entry with *uplo*='U' or 'u', *k* specifies the number of super-diagonals of the matrix *a*. On entry with *uplo*='L' or 'l', *k* specifies the number of sub-diagonals of the matrix *a*. *k* must satisfy 0.LE.*k*. The *k* argument is unchanged on exit.

a Real array of dimension (*lda,n*). Before entry with *uplo*='U' or 'u', the leading (*k*+1)-by-*n* part of the array *a* must contain the upper triangular band part of the matrix of coefficients, supplied column by column, with the leading diagonal of the matrix in row (*k*+1) of the array, the first super-diagonal starting at position 2 in row *k*, and so on. The top left *k*-by-*k* triangle of the array *a* is not referenced. The following program segment will transfer the upper triangular band matrix from conventional full matrix storage to band storage:

```
          DO 20, J=1,N
          M = K+1-J
          DO 10, I=MAX(1,J-K), J
                A(M+I,J) = MATRIX(I,J)
10          CONTINUE
20        CONTINUE
```

Before entry with *uplo*='L' or 'l', the leading (*k*+1)-by-*n* part of the array *a* must contain the lower triangular band part of the matrix of coefficients, supplied column-by-column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right *k*-by-*k* triangle of the array *a* is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
      DO 20, J=1,N
      M = 1-J
      DO 10, I=J, MIN(N,J+K)
            A(M+I,J) = MATRIX(I,J)
10       CONTINUE
20    CONTINUE
```

Note that when *diag*='U' or 'u' the elements of the array *a* corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. The *a* argument is unchanged on exit.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling subprogram. *lda* must be at least (k + 1). The *lda* argument is unchanged on exit.

x Real array of dimension at least (1+(n−1)* abs(*incx*)). Before entry, the incremented array *x* must contain the *n* element vector *x*. On exit, *x* is overwritten with the transformed vector *x*.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

NOTE

The **STBMV** routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

STBSV – Solves a real triangular banded system of linear equations

SYNOPSIS

CALL STBSV(*uplo,trans,diag,n,k,a,lda,x,incx*)

DESCRIPTION

STBSV solves one of the systems of equations

$$a*x = b \quad \text{or} \quad a'*x = b$$

where b and x are n element vectors, and a is an n-by-n unit, or non-unit, upper or lower triangular band matrix, with $(k + 1)$ diagonals.

No test for singularity or near-singularity is included in this routine. Such tests must be performed before calling this routine.

uplo Character*1. On entry, *uplo* specifies whether matrix is an upper of lower triangular matrix. When *uplo*='U' or 'u', *a* is an upper triangular matrix. When *uplo*='L' or 'l', *a* is a lower triangular matrix. The *uplo* argument is unchanged on exit.

trans Character*1. On entry, *trans* specifies the equation to be solved. If *trans*='N' or 'n', $a*x = b$. If *trans*='T' or 't', $a'*x = b$. The *trans* argument is unchanged on exit.

diag Character*1. On entry, *diag* specifies whether or not *a* is unit triangular. If *diag*='U' or 'u', *a* is assumed to be unit triangular. If *diag*='N' or 'n', *a* is not assumed to be unit triangular. The *diag* argument is unchanged on exit.

n Integer. On entry, *n* specifies the order of the matrix *a*. The *n* argument must be at least zero. The *n* argument is unchanged on exit.

k Integer. On entry with *uplo*='U' or 'u', *k* specifies the number of super-diagonals of the matrix *a*. On entry with *uplo*='L' or 'l', *k* specifies the number of sub-diagonals of the matrix *a*. *k* must satisfy 0.LE.*k*. The *k* argument is unchanged on exit.

a Real array of dimension (*lda,n*). Before entry with *uplo*='U' or 'u', the leading (k+1)-by-n part of the array *a* must contain the upper triangular band part of the matrix of coefficients, supplied column-by-column, with the leading diagonal of the matrix in row (k+1) of the array, the first super-diagonal starting at position 2 in row *k*, and so on. The top *k*-by-k traingle of the array *a* is not referenced. The following program segment will transfer an upper triangular band matrix from conventional full matrix storage to band storage:

```
      DO 20, J=1,N
      M = K+1-J
      DO 10, I=MAX(1,J-K), J
             A(M+I,J) = MATRIX(I,J)
10       CONTINUE
20    CONTINUE
```

Before entry with *uplo*='L' or 'l', the leading (k+1)-by-n part of the array *a* must contain the lower triangular band part of the matrix of coefficients, supplied column-by-column, with the leading diagonal of the matrix in row 1 of the array, the first sub-diagonal starting at position 1 in row 2, and so on. The bottom right *k* by *k* triangle of the array *a* is not referenced. The following program segment will transfer a lower triangular band matrix from conventional full matrix storage to band storage:

```
          DO 20, J=1,N
          M = 1-J
          DO 10, I=J, MIN(N,J+K)
                A(M+I,J) = MATRIX(I,J)
 10          CONTINUE
 20       CONTINUE
```

Note that when *diag*='U' or 'u' the elements of the array *a* corresponding to the diagonal elements of the matrix are not referenced, but are assumed to be unity. The *a* argument is unchanged on exit.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling (sub)program. *lda* must be at least (k+1). The *lda* argument is unchanged on exit.

x Real array of dimension at least $(1+(n-1)*\text{abs}(incx))$. Before entry, the incremented array *x* must contain the *n* element right-hand side vector *b*. On exit, *x* is overwritten with the solution vector *x*.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

NOTE

The **STBSV** routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

STRMV – Multiplies a real vector by a real triangular matrix

SYNOPSIS

CALL STRMV(*uplo,trans,diag,n,a,lda,x,incx*)

DESCRIPTION

STRMV solves one of the matrix-vector operations

$x := a^*x$ or $x := a'^*x$

where x is an n element vector, and a is an n-by-n unit, or non-unit, upper or lower triangular band matrix.

uplo Character*1. On entry, *uplo* specifies whether matrix is an upper of lower triangular matrix. When *uplo*='U' or 'u', *a* is an upper triangular matrix. When *uplo*='L' or 'l', *a* is a lower triangular matrix. The *uplo* argument is unchanged on exit.

trans Character*1. On entry, *trans* specifies the equation to solved as follows: If *trans*='N' or 'n' $x := a^*x$. If *trans*='T' or 't' $x := a'^*x$. The *trans* argument is unchanged on exit.

diag Character*1. On entry, *diag* specifies whether or not *a* is unit triangular as follows: If *diag*='U' or 'u' *a* is assumed to be unit triangular. If *diag*='N' or 'n' *a* is not assumed to be unit triangular. The *diag* argument is unchanged on exit.

n Integer. On entry, *n* specifies the order of the matrix *a*. The *n* argument must be at least zero. The *n* argument is unchanged on exit.

a Real array of dimension (*lda,n*). Before entry with *uplo*='U' or 'u', the leading *n* by *n* upper triangular part of the array *a* must contain the upper triangular matrix and the strictly lower triangular part of *a* is not referenced. Before entry with *uplo*='L' or 'l', the leading *n*-by-*n* lower triangular part of the array *a* must contain the lower triangular matrix and the strictly upper triangular part of *a* is not referenced. Note that when *diag*='U' or 'u', the diagonal elements of *a* are not referenced either, but are assumed to be unity. The *a* argument is unchanged on exit.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling (sub)program. *lda* must be at least max(1,*nk*). The *lda* argument is unchanged on exit.

x Real array of dimension at least $(1+(n-1)^*\mathrm{abs}(incx))$. Before entry, the incremented array *x* must contain the *n* element vector *b*. On exit, *x* is overwritten with the transformed vector *x*.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

NOTE

The **STRMV** routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

STRSV – Solves a real triangular system of linear equations

SYNOPSIS

CALL STRSV(*uplo,trans,diag,n,a,lda,x,incx*)

DESCRIPTION

STRSV solves one of the systems of equations

$a*x = b$ or $a'*x = b$

where b and x are n element vectors, and a is an n-by-n unit, or non-unit, upper or lower triangular matrix.

uplo Character*1. On entry, *uplo* specifies whether matrix is an upper of lower triangular matrix. When *uplo*='U' or 'u', *a* is an upper triangular matrix. When *uplo*='L' or 'l', *a* is a lower triangular matrix. The *uplo* argument is unchanged on exit.

trans Character*1. On entry, *trans* specifies the operation to be performed. If *trans*='N' or 'n', $a*x = b$. If *trans*='T' or 't', $a'*x = b$. The *trans* argument is unchanged on exit.

diag Character*1. On entry, *diag* specifies whether or not *a* is unit triangular. If *diag*='U' or 'u', *a* is assumed to be unit triangular. If *diag*='N' or 'n', *a* is not assumed to be unit triangular. The *diag* argument is unchanged on exit.

n Integer. On entry, *n* specifies the order of the matrix *a*. The *n* argument must be at least zero. The *n* argument is unchanged on exit.

a Real array of dimension (*lda,n*). Before entry with *uplo*='U' or 'u', the leading *n*-by-*n* upper triangular part of the array *a* must contain the upper triangular matrix and the strictly lower triangular part of *a* is not referenced. Before entry with *uplo*='L' or 'l', the leading *n*-by-*n* lower triangular part of the array *a* must contain the lower triangular matrix and the strictly upper triangular part of *a* is not referenced. Note that when *diag*='U' or 'u', the diagonal elements of *a* are not referenced either, but are assumed to be unity. The *a* argument is unchanged on exit.

lda Integer. On entry, *lda* specifies the first dimension of *a* as declared in the calling subprogram. *lda* must be at least max(1,*n*). The *lda* argument is unchanged on exit.

x Real array of dimension at least $(1+(n-1)*\text{abs}(incx))$. Before entry, the incremented array *x* must contain the *n* element right-hand side vector *b*. On exit, *x* is overwritten with the solution vector *x*.

incx Integer. On entry, *incx* specifies the increment for the elements of *x*. *incx* must not be zero. The *incx* argument is unchanged on exit.

NOTE

The STRSV routine is a level 2 Basic Linear Algebra Subroutine (BLAS2).

NAME

SXMPY – Computes the product of a row vector and a matrix and adds the result to another row vector

SYNOPSIS

CALL SXMPY(*n1*,*ldy*,*y*,*n2*,*ldx*,*x*,*ldm*,*m*)

DESCRIPTION

n1	Number of columns in matrix *y*
ldy	Leading dimension of matrix *y*
y	Matrix *y*
n2	Number of columns in matrix *x*
ldx	Leading dimension of matrix *x*
x	Matrix *x*
ldm	Leading dimension of matrix *m*
m	Matrix *m*

SXMPY executes an operation equivalent to the following Fortran code:

```
      SUBROUTINE SXMPY(N1,LDY,Y,N2,LDX,X,LDM,M)
      REAL Y(LDY,1), X(LDX,1), M(LDM,1)
      DO 20 J=1,N2
              DO 20 I=1,N1
                 Y(1,I)=Y(1,I) + X(1,J) * M(J,I)
 20   CONTINUE
      RETURN
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

5. FAST FOURIER TRANSFORM ROUTINES

These routines apply a Fast Fourier transform. Each routine can compute either a Fourier analysis or a Fourier synthesis. Detailed descriptions, algorithms, performance statistics, and examples of these routines appear in Complex Fast Fourier Transform Binary Radix Subroutine (CFFT2), CRI publication SN-0203; Real to Complex Fast Fourier Transform Binary Radix Subroutine (**RCFFT2**), CRI publication SN-0204; and Complex to Real Fast Fourier Transform Binary Radix Subroutine (**CRFFT2**), CRI publication SN-0206.

IMPLEMENTATION

All routines in this section are available to users of both the COS and UNICOS operating systems.

INTRODUCTION

Each routine has the same argument list: (*init,ix,n,x,work,y*).

Parameter	Description
init	Initialization flag
ix	Analysis/Synthesis flag
n	Size of transform
x	Input vector
work	Working storage vector
y	Result vector

The routines are called the first time with $init \neq 0$ and n as a power of 2 to initialize the needed sine and cosine tables in the working storage area *work*. Then for each input vector of length n (length $(n/2)+1$ for **CRFFT2**), each routine is called with $init=0$. The sign of *ix* determines whether a Fourier synthesis or a Fourier analysis is computed: if the sign of *ix* is negative, a synthesis is computed; if the sign is positive, an analysis is computed. The following table shows the size and formats of *x*, *y*, and *work* for each routine.

Arguments for Fast Fourier Transform Routines			
Argument	**CFFT2**	**RCFFT2**	**CRFFT2**
x	Complex n	Real n	Complex $(n/2)+1$
work	Complex $(5/2)n$	Complex $(3/2)n+2$	Complex $(3/2)n+2$
y	Complex n	Complex $(n/2)+1$	Real n

The following table contains the purpose, name, and entry of each Fast Fourier transform routine.

Fast Fourier Transform Routines		
Purpose	Name	Entry
Apply a complex Fast Fourier transform	**CFFT2**	**CFFT2**
Apply multiple complex-to-real Fast Fourier transforms	**CFFTMLT**	**CFFTMLT**
Apply a complex to real Fast Fourier transform	**CRFFT2**	**CRFFT2**
Apply a real to complex Fast Fourier transform	**RCFFT2**	**RCFFT2**
Apply multiple real-to-complex Fast Fourier transforms	**RFFTMLT**	**RFFTMLT**

NAME

CFFT2 – Applies a complex Fast Fourier transform

SYNOPSIS

CALL CFFT2(*init,ix,n,x,work,y*)

DESCRIPTION

init	≠0	Generates sine and cosine tables in *work*
	=0	Calculates Fourier transforms using sine and cosine tables of the previous call
ix	>0	Calculates a Fourier analysis
	<0	Calculates a Fourier synthesis

n Size of the Fourier transform; 2^m where $3 \le m$ for the CRAY X-MP computer system and $2 \le m$ for the CRAY-1 computer system.

x Input vector. Vector of *n* complex values. Range: $10^{2466} / n \ge |x(i)| \ge n * (10^{-2466})$ for i=1, *n*. The input vector *x* can be equivalenced to either *y* or *work*; then the input sequence is overwritten.

work Working storage. Vector of $(\frac{5}{2})n$ complex values.

y Result vector. Vector of *n* complex values.

CFFT2 calculates

$$y_k = \sum_{j=0}^{n-1} x_j \exp(\pm \frac{2\pi i}{n} jk)$$

for k=0,1,...,n-1

where x_i i=0,1,...,n-1 are stored in X(I),I=1,N
y_i i=0,1,...,n-1 are stored in Y(I),I=1,N

and the sign of the exponent is determined by SIGN(IX).

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

CFFTMLT – Applies complex-to-complex Fast Fourier transforms on multiple input vectors

SYNOPSIS

CALL CFFTMLT(*ar,ai,work,trigs,ifax,inc,jump,n,lot,isign*)

DESCRIPTION

CFFTMLT applys complex-to-complex Fast Fourier transforms on more than one input vector. The arguments are as follows:

ar Input vector. Vector of *n*lot* real values. It contains the real part of the input data. Result vector. It contains the real part of the transformed data.

ai Input vector. Vector of *n*lot* real values. It contains the imaginary part of the input data. Result vector. It contains the imaginary part of the transformed data.

work Working storage; a work area of size 4**n*lot* real elements.

trigs Input vector of size 2**n*. It must be initialized to contain sine and cosine tables. Vectors *trigs* and *ifax*(*) can be initialized by the following call:

CALL CFTFAX(*n,ifax,trigs*).

ifax Input vector. Vector of size 19 integer elements. It has a previously prepared list of factors of *n*

inc The increment within each data vector.

jump The increment between the start of each data vector. *inc* and *jump* apply to both the real and imaginary data. To obtain best performance *jump* should be an odd number.

n Length of the data vectors *n* must be factorable as: *n* = (2***p*) * (3***q*) * (4***r*) * (5***s*) , where *p*, *q*, *r* and *s* are integers.

lot The number of data vectors

isign +1 for fourier analysis
–1 for fourier synthesis

CFFTMLT computes:

$$(ar(inc*j+1), ai(inc*j+1)) = \sum_{k=0}^{n-1} exp(isign*iota*2*pi*k*j/n)(ar(inc*k+1), ai(inc*k+1))$$

for j = 0,2,...,n-1.

This calculation is performed for each of the n-vectors in the input.

The normalization used by **cfftmlt** is different from that used by **cfft2, crfft2**, and **rcfft2.**

Vectorization is achieved by doing the transforms in parallel, with vector length = *lot.*

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

CRFFT2 – Applies a complex to real Fast Fourier transform

SYNOPSIS

CALL CRFFT2(*init*,*ix*,*n*,*x*,*work*,*y*)

DESCRIPTION

init	≠0	Generates sine and cosine tables in *work*
	=0	Calculates Fourier transforms using sine and cosine tables of the previous call
ix	>0	Calculates a Fourier analysis
	<0	Calculates a Fourier synthesis

n Size of the Fourier transform; 2^m where $3 \le m$

x Input vector. Vector of $(\frac{n}{2})+1$ complex values.
Range: $10^{2466} / n \ge |x(i)| \ge n$ (** (10^{-2466}) for i=1,n.

work Working storage. Vector of $(\frac{3}{2})n+2$ complex values.

y Result vector. Vector of *n* real values.

CRFFT2 calculates the following equation, where the x_i elements are complex and $x_j=\overline{x}_{n-j}$ for $j=0,1, \cdots (\frac{n}{2})$. Only the first $(\frac{n}{2})+1$ elements are stored in x.

$$y_k = \sum_{j=0}^{n-1} x_j \exp(\pm \frac{2\pi i}{n} jk)$$

for k=0,1,...,n-1

where x_j elements are complex and related by $x_j=\overline{x}_{n-j}$

for $j=1,2,3,\ldots,(\frac{n}{2})$

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

RCFFT2 – Applies a real to complex Fast Fourier transform

SYNOPSIS

CALL RCFFT2(*init*,*ix*,*n*,*x*,*work*,*y*)

DESCRIPTION

init $\neq 0$ Generates sine and cosine tables in *work*
$=0$ Calculates Fourier transforms using sine and cosine tables of the previous call

ix >0 Calculates a Fourier analysis
<0 Calculates a Fourier synthesis

n Size of the Fourier transform; 2^m where $3 \leq m$

x Input vector. Vector of *n* complex values. Range:
$10^{2466} / (2 * n) \geq |x(i)| \geq (2 * n) * (10^{-2466})$ for i=1,n.

work Working storage. Vector of $(\frac{3}{2})n+2$ complex values.

y Result vector. Vector of $(\frac{n}{2})+1$ complex values.

RCFFT2 calculates

$$y_k = 2 \sum_{j=0}^{n-1} x_j \exp(\pm \frac{2 \pi i}{n} jk)$$

for $k=0,1,\ldots,(\frac{n}{2})$

where x_i $i=0,1,\ldots,n-1$ are stored in X(I),I=1,N
y_i $i=0,1,\ldots,(\frac{n}{2})$ are stored in $Y(I),I=1,(\frac{N}{2})+1$

and the sign of the exponent is determined by SIGN(IX).

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

RFFTMLT – Applies complex-to-real and real-to-complex Fast Fourier transforms on multiple input vectors

SYNOPSIS

CALL RFFTMLT(*a,work,trigs,ifax,inc,jump,n,lot,isign*)

DESCRIPTION

RFFTMLT applys complex-to-real and real-to-complex Fast Fourier transforms on more than one input vector. The arguments are as follows:

a When *isign* = –1, the *n* real input values for each data vector:
$a(1), a(inc+1), \cdots, a(inc*(n-1)+1)$
should be stored in array *a* with stride=*inc*. The computed output vector is:
$a(2*inc*i+1), a(2*inc*i+inc+1), i=0,1,...,n/2$
The *i*-th fourier coefficient is:
$(a(2*inc*i+1), a(2*inc*i+inc+1))$
When *isign* = +1, the input and output data formats are reversed. It is important to note that for i=1 and $i=n/2$ the imaginary parts of the complex input numbers must be 0.

work Working storage; a work area of size 2**n***lot* real elements.

trigs Input vector of size 2**n*. It must be initialized to contain sine and cosine tables. Vectors *trigs* and *ifax*(*) can be initialized by:
CALL FFTFAX(*n,ifax,trigs)*.

ifax Input vector. Vector of size 19 integer elements. It has a previously prepared list of factors of *n*.

inc The increment within each data vector.

jump The increment between the start of each data vector. *inc* and *jump* apply to both the real and imaginary data. For the best performance, *jump* should be an odd number.

n Length of the data vectors. *n* must be factorable as: $n = (2**p) * (3**q) * (4**r) * (5**s)$, where *p*, *q*, *r* and *s* are integers.

lot The number of data vectors

isign –1 to calculate real-to-complex fourier transform
+1 to calculate complex-to-real fourier transform

For *isign* = –1, RFFTMLT calculates the following:

$$(ar(inc*j+1), ai(inc*j+1)) = \sum_{k=0}^{n-1} exp(-iota*2*pi*k*j/n)*a(inc*k+1)/n$$

for j = 0,1,...,n/2. The numbers on the left side of the equation are complex.

This calculation is performed for each of the n-vectors in the input.

For *isign* = +1, RFFTMLT calculates the following:

$$a(inc*j+1) = \sum_{k=0}^{n-1} exp(iota*2*pi*k*j/n)*(a(2*inc*k+1), a(2*inck+inc+1))$$

for $j = 0,1,...,n$. Each input vector satisfies the relation:

$$(a(2*inc*k+1), a(2*inc*k+inc+1)) = (a(2*inc*(n-k)+1), -a(2*inc*(n-k)+inc+1)),\ k = 0,1,...,n/2.$$

Only the first $n/2+1$ complex numbers are needed.

This calculation is performed for each of the n-vectors in the input.

It is important to note that for *isign* = -1, the division by *n* uses a normalization that is different from the normalization used by CFFT2, CRFFT2, RCFFT2, and CFFTMLT.

Vectorization is achieved by doing the transforms in parallel, with vector length = *lot*.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

6. SEARCH ROUTINES

The following search routines are written to run optimally on Cray computer systems. These subprograms use the call-by-address convention when called by a Fortran or CAL program. See section 1, Introduction, for details of the call-by-address convention.

The subprograms are grouped as follows:

- Maximum/minimum element search routines
- Vector search routines

IMPLEMENTATION

All routines in this section are available to users of both the COS and UNICOS operating systems.

MAXIMUM/MINIMUM ELEMENT SEARCH ROUTINES

The maximum or minimum element search routines find the largest or smallest element of a vector or argument and return either the element or its index.

To return an index - **ISMAX** and **ISMIN** return the index of the maximum or minimum vector element, respectively. **ISAMAX, ICAMAX,** and **ISAMIN** search for maximum or minimum absolute values in a real vector and return the index. **INTMAX** and **INTMIN** are the corresponding maximum and minimum search routines for an integer vector. **INTFLMAX** and **INTFLMIN** return the index of the maximum and minimum value within a table. The type declaration for these routines is integer. For further details regarding type and dimension declarations for variables occurring in these subprograms, see section 4, Linear Algebra Subprograms.

To return an element - The following functions find the maximum or minimum elements of two or more vector arguments: **MAX0, AMAX1, DMAX1, AMAX0, MAX1, MIN0, AMIN1, DMIN1, AMIN0,** and **MIN1.** These functions differ mainly in their types for integer, real, and double-precision arguments. In the description of these functions, the argument type does not always reflect the function type.

The following table contains the purpose, name, and entry of each maximum/minimum element search routine.

Maximum/Minimum Element Search Routines		
Purpose	Name	Entry
Find the first index of the largest absolute value of the elements of a real or complex vector	**ISAMAX** **ICAMAX**	**ISAMAX**
Return the index of the maximum value in a table	**INTFLMAX**	**INTFLMAX**
Return the index of the minimum value in a table	**INTFLMIN**	
Return the index of the integer vector element with maximum value	**INTMAX**	**INTMAX**
Return the index of the integer vector element with minimum value	**INTMIN**	
Return the index of the vector element with maximum value	**ISMAX**	**ISMAX**
Return the index of the vector element with minimum value	**ISMIN**	
Return the index of the vector element with minimum absolute value	**ISAMIN**	
Return the largest of all arguments	**MAX0** **AMAX1** **DMAX1** **AMAX0** **MAX1**	**MAX**
Return the smallest of all arguments	**MIN0** **AMIN1** **DMIN1** **AMIN0** **MIN1**	**MIN**

VECTOR SEARCH ROUTINES

Vector search routines have one of the following functions:

- To return occurrences of an object in a vector
- To search for an object in a vector

To return occurrences of an object in a vector - These integer routines return the number of occurrences of a given relation in a vector. The routines **ILLZ** and **IILZ** find the first occurrence. **ILSUM** counts the number of such occurrences. All three of these functions are described under the heading **IILZ**.

To search for an object in a vector - **ISRCH** routines find the positions of an object in a vector. These include the following: **ISRCHEQ**, **ISRCHNE**, **ISRCHFLT**, **ISRCHFLE**, **ISRCHFGT**, **ISRCHFGE**, **ISRCHILT**, **ISRCHILE**, **ISRCHIGT**, **ISRCHIGE**, **ISRCHMEQ**, **ISRCHMNE**, **ISRCHMLT**, **ISRCHMLE**, **ISRCHMGT**, and **ISRCHMGE**. These functions return the first location in an array that has a true relational value to the target.

The **WHEN** routines are similar to the **ISRCH** routines in that they return the locations of elements in an array that have a true relational value to the target. However, all locations are returned in an indexed array. The **WHEN** routines are **WHENEQ, WHENNE, WHENFLT, WHENFLE, WHENFGT, WHENFGE, WHENILT, WHENILE, WHENIGT, WHENIGE, WHENME, WHENNE, WHENMLT, WHENMLE, WHENMGT** and, **WHENMGE.**

The **CLUS** routines find the index of clusters that have a true relational value to the target. These routines are further divided into integer (**CLUSILT, CLUSILE, CLUSIGT, CLUSIGT**) and real (**CLUSFLT, CLUSFLE, CLUSFGT,** and **CLUSFGE**) routines.

The **OSRCHI** and **OSRCHF** subroutines return the index of the location that would contain the target in an ordered array. This is useful for sorting elements into a new array. Searching always begins at the lowest value in the ordered array. The total number of occurrences of the target in the array can also be returned.

The following table contains the purpose, name, and entry of each **ISRCH, WHEN, CLUS,** and **OSRCH** routine.

ISRCH, WHEN, CLUS, and OSRCH Routines		
Purpose	Name	Entry
Find the index of clusters equal or not equal to the target	CLUSEQ CLUSNE	CLUSEQ
Find the index of clusters of real elements in relation to a target	CLUSFLT CLUSFLE CLUSFGT CLUSFGE	CLUSFLT
Find the index of cluster of integer elements in relation to a target	CLUSILT CLUSILE CLUSIGT CLUSIGE	CLUSILT
Find the first array element that is equal or not equal to the target	ISRCHEQ ISRCHNE	ISRCHEQ
Find the first real array element that is less than, less than or equal to, greater than, or greater than or equal to the real target	ISRCHFLT ISRCHFLE ISRCHFGT ISRCHFGE	ISRCHFLT
Find the first integer array element that is less than, less than or equal to, greater than, or greater than or equal to the integer target	ISRCHILT ISRCHILE ISRCHIGT ISRCHIGE	ISRCHILT
Find the first array element that is equal or not equal to the target within a field	ISRCHMEQ ISRCHMNE	ISRCHMEQ
Find the first array element that is less than, less than or equal to, greater than, or greater than or equal to the target within a field	ISRCHMLT ISRCHMLE ISRCHMGT ISRCHMGE	ISRCHMLT

ISRCH, WHEN, CLUS, and OSRCH Routines (continued)		
Purpose	Name	Entry
Search an ordered integer or real array and return the index of the first location that contains the target	**OSRCHI** **OSRCHF**	**OSRCHI**
Find all array elements that are equal or not equal to the target	**WHENEQ** **WHENNE**	**WHENEQ**
Find all real array elements that are less than, less than or equal to, greater than, or great than or equal to the real target	**WHENFLT** **WHENFLE** **WHENFGT** **WHENFGE**	**WHENFLT**
Find all integer array elements that are less than, less than or equal to, greater than, or greater than or equal to the integer target	**WHENILT** **WHENILE** **WHENIGT** **WHENIGE**	**WHENILT**
Find all array elements that are equal or not equal to the target within a field	**WHENMEQ** **WHENNME**	**WHENMEQ**
Find all array elements that are less than, less than or equal to, greater than, or great than or equal to the target within a field	**WHENMLT** **WHENMLE** **WHENMGT** **WHENMGE**	**WHENMLT**

NAME

CLUSEQ, CLUSNE – Finds index of clusters within a vector

SYNOPSIS

CALL CLUSEQ(*n,array,inc,target,index,nn***)**

CALL CLUSNE(*n,array,inc,target,index,nn***)**

DESCRIPTION

n	Number of elements to be searched; length of the array. Type integer.
array	Real or integer vector to be searched
inc	Skip distance between elements of the searched array; type integer.
target	Scalar to match logically. Type integer or real.
index	Indexes in *array* where the cluster starts and stops (one based); *index* should be dimensioned **INDEX(2,***n***/2)**.
nn	Number of matches found; length of index. Type integer.

These routines find the index of clusters of occurrences equal to or not equal to a scalar within a vector. The Fortran equivalent of the type of logical search performed for **CLUSEQ** and **CLUSNE** follows:

```
ARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).EQ.TARGET

ARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).NE.TARGET
```

NOTE

Searching for the cluster allows vectorization. Before using these routines, you should know that the logical search results in clusters of finds.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

CLUSFLT, CLUSFLE, CLUSFGT, CLUSFGE – Finds real clusters in a vector

SYNOPSIS

CALL CLUSFLT(*n,array,inc,target,index,nn*)

CALL CLUSFLE(*n,array,inc,target,index,nn*)

CALL CLUSFGT(*n,array,inc,target,index,nn*)

CALL CLUSFGE(*n,array,inc,target,index,nn*)

DESCRIPTION

n Number of elements to be searched; length of the array. Type integer.

array Real vector to be searched

inc Skip distance between elements of the searched array. Type integer.

target Scalar to match logically. Type real.

index Indexes in *array* where the cluster starts and stops (one based); *index* should be dimensioned INDEX(2,*n*/2).

nn Number of matches found; length of index. Type integer.

These routines find the index of clusters of real occurrences in relation to a scalar within a vector. The Fortran equivalent of the type of logical search performed for follows:

ARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).LT.TARGET

ARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).LE.TARGET

ARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).GT.TARGET

ARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).GE.TARGET

NOTE

Searching for the cluster allows vectorization. Before using these routines, you should know that the logical search results in clusters of finds.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

CLUSILT, CLUSILE, CLUSIGT, CLUSIGE – Finds integer clusters in a vector

SYNOPSIS

CALL CLUSILT(*n,iarray,inc,itarget,index,nn*)

CALL CLUSILE(*n,iarray,inc,itarget,index,nn*)

CALL CLUSIGT(*n,iarray,inc,itarget,index,nn*)

CALL CLUSIGE(*n,iarray,inc,itarget,index,nn*)

DESCRIPTION

n Number of elements to be searched; length of the array. Type integer.

iarray Integer vector to be searched

inc Skip distance between elements of the searched array. Type integer.

itarget Scalar to match logically. Type integer.

index Indexes in *iarray* where the cluster starts and stops (one based). *index* should be dimensioned **INDEX(2,*n*/2)**.

nn Number of matches found; length of index. Type integer.

These routines find the index of clusters of integer occurrences in relation to a scalar within a vector. The Fortran equivalent of the type of logical search performed for follows:

```
IARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).LT.ITARGET

IARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).LE.ITARGET

IARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).GT.ITARGET

IARRAY(I,I=INDEX(1,J),INDEX(2,J),J=1,NN).GE.ITARGET
```

NOTE

Searching for the cluster allows vectorization. Before using these routines, you should know that the logical search will result in clusters of finds.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

IILZ, ILLZ, ILSUM – Returns number of occurrences of object in a vector

SYNOPSIS

kount=IILZ(*n*,*array*,*incl*)

kount=ILLZ(*n*,*array*,*incl*)

kount=ILSUM(*n*,*array*,*incl*)

DESCRIPTION

n	Number of elements to process in the vector (*n*=vector length if *incl*=1; *n*=vector length/2 if *incl*=2, and so on)
array	Vector operand
incl	Skip distance between elements of the vector operand. For contiguous elements, *incl*=1.

IILZ returns the number of zero values in a vector before the first nonzero value. **ILLZ** returns the number of leading elements of a vector that do not have the sign bit set. **ILSUM** returns the number of **TRUE** values in a vector declared logical.

When scanning backward (*incl* < 0), both **IILZ** and **ILLZ** start at the end of the vector and move backward (L(N),L(N + INCL),L(N + 2*INCL),...).

If *array* is of type logical, **ILLZ** returns the number of **FALSE** values before encountering the first **TRUE** value.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

INTFLMAX, INTFLMIN – Searches for the maximum or minimum value in a table

SYNOPSIS

index=INTFLMAX(*n*,*ix*,*inc*,*mask*,*shift*)

index=INTFLMIN(*n*,*ix*,*inc*,*mask*,*shift*)

DESCRIPTION

index	Index in *ix* where maximum or minimum occurs (one based). Type integer.
n	Number of elements to be searched; length of the array. Type integer.
ix	Table to be searched. Type integer.
inc	Skip distance through *ix*. Type integer.
mask	Right-justified mask used for masking the table vector
shift	Number of bits to right shift the table vector before masking

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

INTMAX, INTMIN – Searches for the maximum or minimum value in a vector

SYNOPSIS

index=INTMAX(*n*,*ix*,*inc*)

index=INTMIN(*n*,*ix*,*inc*)

DESCRIPTION

index	Index in *ix* where maximum or minimum occurs (one based). Type integer.
n	Number of elements to be searched; length of the array. Type integer.
ix	Integer vector to be searched
inc	Skip distance between elements of *ix*. Type integer.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ISAMAX, ICAMAX – Finds first index of largest absolute value in vectors

SYNOPSIS

imax=ISAMAX(*n*,*sx*,*incx*)

imax=ICAMAX(*n*,*cx*,*incx*)

DESCRIPTION

n	Number of elements to process in the vector to be searched (n=vector length if $incx$=1; n=vector length/2 if $incx$=2, and so on). If $n \le 0$, ISAMAX and ICAMAX return 0.
sx	Real vector to be searched
cx	Complex vector to be searched
incx	Skip distance between elements of *sx* or *cx*; for contiguous elements, $incx$=1.

These integer functions find the first index of the largest absolute value of the elements of a vector. ISAMAX returns the first index i such that

$$| x_i | = \max | x_j | : j = 1, \ldots ,n$$

where x_j is an element of a real vector. ICAMAX determines the first index i such that

$$| Real(x_i) | + | Imag(x_i) | = \max \; | Real(x_j) | + | Imag(x_j) | : j = 1, \ldots ,n \; |$$

where x_j is an element of a complex vector.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ISMAX, ISMIN, ISAMIN – Finds maximum, minimum, or minimum absolute value

SYNOPSIS

imax=ISMAX(*n*,*sx*,*incx*)

imin=ISMIN(*n*,*sx*,*incx*)

imin=ISAMIN(*n*,*sx*,*incx*)

DESCRIPTION

n	Number of elements to process in the vector to be searched (*n*=vector length if *incx*=1; *n*=vector length/2 if *incx*=2; and so on). If $n \leq 0$, ISMAX, ISMIN, and ISAMIN return 0.
sx	Real vector to be searched
incx	Skip distance between elements of *sx*. For contiguous elements, *incx*=1.

ISMAX returns the first index i such that

$$|x_i| = \max x_j : j = 1,...,n$$

These routines return the index of the element with maximum, minimum, or minimum absolute value. ISMIN and ISAMIN return the first index i such that

$$|x_i| = \min x_j : j = 1,...,n$$

where x_j is an element of a real vector.

ISMAX, ISMIN, and ISAMIN are integer functions.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ISRCHEQ, ISRCHNE – Finds array element equal or not equal to target

SYNOPSIS

location=**ISRCHEQ**(*n,array,inc,target*)

location=**ISRCHNE**(*n,array,inc,target*)

DESCRIPTION

n Number of elements to be searched. If $n \leq 0$, 0 is returned.

array First element of the real or integer array to be searched

inc Skip distance between elements of the searched array

target Real or integer value searched for in the array. If *target* is not found, the returned value is $n+1$.

ISRCHEQ finds the first real or integer array element that is equal to a real or integer target. **ISRCHNE** returns the first location for which the relational value not equal to is true for real and integer arrays.

The Fortran equivalent code for **ISRCHEQ** is as follows:

```
      FUNCTION ISRCHEQ(N,ARRAY,INC,TARGET)
      DIMENSION ARRAY(N)
      J=1
      IF(INC.LT.0) J=1-(N-1)*INC
      DO 100 I=1,N
            IF(ARRAY(J).EQ.TARGET) GO TO 200
            J=J+INC
  100 CONTINUE
  200 ISRCHEQ=J
      RETURN
      END
```

NOTE

ISRCHEQ replaces the **ISEARCH** routine, but it has an entry point of **ISEARCH** as well as **ISRCHEQ**.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ISRCHFLT, ISRCHFLE, ISRCHFGT, ISRCHFGE – Finds first real array element in relation to a real target

SYNOPSIS

location=**ISRCHFLT**(*n,array,inc,target*)

location=**ISRCHFLE**(*n,array,inc,target*)

location=**ISRCHFGT**(*n,array,inc,target*)

location=**ISRCHFGE**(*n,array,inc,target*)

DESCRIPTION

n	Number of elements to be searched. If $n \leq 0$, 0 is returned.
array	First element of the real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in *array*. If *target* is not found, the returned value is n+1.

These functions return the first location for which the relational operator is true for real arrays.

ISRCHFLT finds the first real array element that is less than the real target. **ISRCHFLE** finds the first real array element that is less than or equal to the real target. **ISRCHFGT** finds the first real array element that is greater than the real target. **ISRCHFGE** finds the first real array element that is greater than or equal to the real target.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ISRCHILT, ISRCHILE, ISRCHIGT, ISRCHIGE – Finds first integer array element in relation to an integer target

SYNOPSIS

location=ISRCHILT(*n*,*iarray*,*inc*,*itarget*)

location=ISRCHILE(*n*,*iarray*,*inc*,*itarget*)

location=ISRCHIGT(*n*,*iarray*,*inc*,*itarget*)

location=ISRCHIGE(*n*,*iarray*,*inc*,*itarget*)

DESCRIPTION

n	Number of elements to be searched. If $n \leq 0$, 0 is returned.
iarray	First element of the integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in *iarray*. If *target* is not found, the returned value is $n+1$.

These functions return the first location for which the relational operator is true for integer arrays.

ISRCHILT finds the first integer array element that is less than the integer target. **ISRCHILE** finds the first integer array element that is less than or equal to the integer target. **ISRCHIGT** finds the first integer array element that is greater than the integer target. **ISRCHIGE** finds the first integer array element that is greater than or equal to the integer target.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ISRCHMEQ, ISRCHMNE – Finds the index of the first occurrence equal or not equal to a scalar within a field of a vector

SYNOPSIS

index=**ISRCHMEQ**(*n*,*array*,*inc*,*target*,*mask*,*right*)

index=**ISRCHMNE**(*n*,*array*,*inc*,*target*,*mask*,*right*)

DESCRIPTION

index	Index in array where first logical match with the target occurred (one based); *index*=*n*+1 if match is not found. Type integer.
n	Number of elements to be searched; length of the array. Type integer.
array	Real or integer vector to be searched
inc	Skip distance between elements of the searched array. Type integer.
target	Scalar to match logically. Type integer or real.
mask	Mask of 1's from the right; the size of the field looked for in the table.
right	Number of bits to shift right so as to right-justify the field searched. Type integer.

The Fortran equivalent of **ISRCHMEQ** and **ISRCHMNE** follows:

```
TABLE(ARRAY(INDEX(I),I=1,NN)).EQ.TARGET

TABLE(ARRAY(INDEX(I),I=1,NN)).NE.TARGET

where TABLE(X)=AND(MASK,SHIFTR(X,RIGHT))
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

ISRCHMLT, ISRCHMLE, ISRCHMGT, ISRCHMGE – Searches vector for logical match

SYNOPSIS

index=**ISRCHMLT**(*n,array,inc,target,mask,right*)

index=**ISRCHMLE**(*n,array,inc,target,mask,right*)

index=**ISRCHMGT**(*n,array,inc,target,mask,right*)

index=**ISRCHMGE**(*n,array,inc,target,mask,right*)

DESCRIPTION

These routines search an array, returning the index of the first element that creates a logical match with the target. **ISRCHMLT** searches for an element less than the target, **ISRCHMLE** for one that is less than or equal to the target, **ISRCHMGT** for one that is greater than the target, and **ISRCHMGE** for one that is greater than or equal to the target.

index Index in array where first logical match with the target occurred (one based); *index*=*n*+1 if match is not found. Type integer.

n Number of elements to be searched; length of the array. Type integer.

array Real or integer vector to be searched

inc Skip distance between elements of the searched array. Type integer.

target Scalar to match logically. Type integer or real.

mask Mask of 1's from the right; the size of the field looked for in the table.

right Number of bits to shift right so as to right justify the field searched (type integer)

The Fortran equivalent of each logical search performed follows:

```
TABLE(ARRAY(INDEX(I),I=1,NN)).LT.TARGET

TABLE(ARRAY(INDEX(I),I=1,NN)).LE.TARGET

TABLE(ARRAY(INDEX(I),I=1,NN)).GT.TARGET

TABLE(ARRAY(INDEX(I),I=1,NN)).GE.TARGET

where TABLE(X)=AND(MASK,SHIFTR(X,RIGHT))
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

MAX0, AMAX1, DMAX1, AMAX0, MAX1 – Returns the largest of all arguments

SYNOPSIS

i=**MAX0** (*integer*$_1$,*integer*$_2$,..., *integer*$_n$)

r=**AMAX1**(*real*$_1$,*real*$_2$,..., *real*$_n$)

d=**DMAX1**(*double*$_1$,*double*$_2$,..., *double*$_n$)

r=**AMAX0**(*integer*$_1$,*integer*$_2$,..., *integer*$_n$)

i=**MAX1**(*real*$_1$,*real*$_2$,..., *real*$_n$)

DESCRIPTION

MAX0, AMAX1, and **DMAX1** use integer, real, and double-precision arguments, respectively, and return the same type of result. Each function is of the same type as its arguments.

AMAX0 (type real) returns a real result from integer arguments. **MAX1** (type integer) returns an integer result from real arguments.

All of the arguments within each function must be of the same type, and the number of arguments n must be in the range $2 \leq n < 64$. Arguments must be in the range $|x| <$inf.

NOTE

MAX is the generic name for the maximum routines **MAX0, AMAX1,** and **DMAX1.** Calls to

i=MAX(*integer*$_1$,*integer*$_2$,...,*integer*$_n$)
r=MAX(*real*$_1$,*real*$_2$,...,*real*$_n$)
d=MAX(*double*$_1$,*double*$_2$,...,*double*$_n$)

will return integer, real, and double-precision results, respectively.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

MIN0, AMIN1, DMIN1, AMIN0, MIN1 – Returns the smallest of all arguments

SYNOPSIS

i=**MIN0**($integer_1$,$integer_2$,..., $integer_n$)

r=**AMIN1**($real_1$,$real_2$,..., $real_n$)

d=**DMIN1**($double_1$,$double_2$,..., $double_n$)

r=**AMIN0**($integer_1$,$integer_2$,..., $integer_n$)

i=**MIN1**($real_1$,$real_2$,...,$real_n$)

DESCRIPTION

MIN0, AMIN1, and **DMIN1** use integer, real, and double-precision arguments, respectively, and return the same type of result. Each of these functions is of the same type as its arguments.

AMIN0 (type real) returns a real result from integer arguments. **MIN1** (type integer) returns an integer result from real arguments.

All of the arguments within each function must be of the same type, and the number of arguments n must be in the range $2 \leq n < 64$. Arguments must be in the range $|x| <$inf.

NOTE

MIN is the generic name for the minimum routines **MIN0, AMIN1,** and **DMIN1.** Calls to

i=MIN($integer_1$,$integer_2$,...,$integer_n$)
r=MIN($real_1$,$real_2$,...,$real_n$)
d=MIN($double_1$,$double_2$,...,$double_n$)

will return integer, real, and double-precision results, respectively.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

OSRCHI, OSRCHF – Searches an ordered array and return index of the first location that contains the target

SYNOPSIS

CALL OSRCHI(*n*,*iarray*,*inc*,*target*,*index*,*iwhere*,*inum*)

CALL OSRCHF(*n*,*array*,*inc*,*target*,*index*,*iwhere*,*inum*)

DESCRIPTION

n Number of elements of the array to be searched

iarray Beginning address of the integer array to be searched

array Beginning address of the real array to be searched

inc A positive skip increment indicates an ascending array and returns the index of the first element encountered, starting at the beginning of the array.

A negative skip increment indicates a descending array and returns the index of the last element encountered, starting at the beginning of the array.

target Integer or real target of the search

index Index of the first location in the searched array that contains the target; exceptional cases are as follows:

If $n < 1$, *index*=0
If no equal array elements, *index*=*n*+1

iwhere Index of the first location in the searched array that would contain the target if it were found in the array. If the target is found, *index*=*iwhere*. There is one exceptional case; if *n* is less than 1, *iwhere*=0.

inum Number of target elements found in the array

OSRCHI searches an ordered integer array and returns the index of the first location that contains the target (type integer). **OSRCHF** searches an ordered real array and returns the index of the first location that contains the target (type real).

Searching always begins at the lowest value in the ordered array. Even if the target is not found, the index of the location that would contain the target is returned. The total number of occurrences of the target in the array (*inum*) can also be returned.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

WHENEQ, WHENNE – Finds all array elements equal to or not equal to the target

SYNOPSIS

CALL WHENEQ(*n,array,inc,target,index,nval***)**

CALL WHENNE(*n,array,inc,target,index,nval***)**

DESCRIPTION

n	Number of elements to be searched
array	First element of the real or integer array to be searched
inc	Skip distance between elements of the searched array
target	Real or integer value searched for in the array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

WHENEQ finds all real or integer array elements that are equal to a real or integer target. **WHENNE** returns all locations for which the relational value not equal to is true for real and integer arrays.

The Fortran equivalent follows:

```
      INA=1
      NVAL=0
      IF(INC .LT. 0) INA=(-INC)*(N-1)+1
      DO 100 I=1,N
         IF(ARRAY(INA) .EQ. TARGET) THEN
            NVAL=NVAL+1
            INDEX(NVAL)=I
         END IF
         INA=INA+INC
  100 CONTINUE
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

WHENFLT, WHENFLE, WHENFGT, WHENFGE – Finds all real array elements in relation to the real target

SYNOPSIS

CALL WHENFLT(*n,array,inc,target,index,nval***)**

CALL WHENFLE(*n,array,inc,target,index,nval***)**

CALL WHENFGT(*n,array,inc,target,index,nval***)**

CALL WHENFGE(*n,array,inc,target,index,nval***)**

DESCRIPTION

n	Number of elements to be searched
array	First element of the real array to be searched
inc	Skip distance between elements of the searched array
target	Real value searched for in the array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

These functions return all locations for which the relational operator is true for real arrays.

WHENFLT finds all real array elements that are less than the real target. **WHENFLE** finds all real array elements that are less than or equal to the real target. **WHENFGT** finds all real array elements that are greater than the real target. **WHENFGE** finds all real array elements that are greater than or equal to the real target.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

WHENILT, WHENILE, WHENIGT, WHENIGE – Finds all integer array elements in relation to the integer target

SYNOPSIS

CALL WHENILT(*n,iarray,inc,itarget,index,nval*)

CALL WHENILE(*n,iarray,inc,itarget,index,nval*)

CALL WHENIGT(*n,iarray,inc,itarget,index,nval*)

CALL WHENIGE(*n,iarray,inc,itarget,index,nval*)

DESCRIPTION

n	Number of elements to be searched
iarray	First element of the integer array to be searched
inc	Skip distance between elements of the searched array
itarget	Integer value searched for in the array
index	Integer array containing the index of the found target in the array
nval	Number of values put in the index array

These functions return all locations for which the relational operator is true for integer arrays.

WHENILT finds all integer array elements that are less than the integer target. **WHENILE** finds all integer array elements that are less than or equal to the integer target. **WHENIGT** finds all integer array elements that are greater than the integer target. **WHENIGE** finds all integer array elements that are greater than or equal to the integer target.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

WHENMEQ, WHENMNE – Finds the index of occurrences equal or not equal to a scalar within a field in a vector

SYNOPSIS

CALL WHENMEQ(*n,array,inc,target,index,nn,mask,right*)

CALL WHENMNE(*n,array,inc,target,index,nn,mask,right*)

DESCRIPTION

n	Number of elements to be searched; length of the array.
array	Vector to be searched
inc	Skip distance between elements of the searched array
target	Scalar to match logically
index	Indexes in *array* where all logical matches with the target occurred (one based)
nn	Number of matches found. Length of index.
mask	Mask of 1's from the right; the size of the field looked for in the table.
right	Number of bits to shift right so as to right-justify the field searched

The Fortran equivalent of **WHENMEQ** and **WHENMNE follows:**

```
TABLE(ARRAY(INDEX(I),I=1,NN)).EQ.TARGET

TABLE(ARRAY(INDEX(I),I=1,NN)).NE.TARGET

where TABLE(X)=AND(MASK,SHIFTR(X,RIGHT))
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

WHENMLT, WHENMLE, WHENMGT, WHENMGE – Finds the index of occurrences in relation to a scalar within a field in a vector

SYNOPSIS

CALL WHENMLT(*n,array,inc,target,index,nn,mask,right*)

CALL WHENMLE(*n,array,inc,target,index,nn,mask,right*)

CALL WHENMGT(*n,array,inc,target,index,nn,mask,right*)

CALL WHENMGE(*n,array,inc,target,index,nn,mask,right*)

DESCRIPTION

n	Number of elements to be searched; length of the array.
array	Vector to be searched
inc	Skip distance between elements of the searched array
target	Scalar to match logically
index	Indexes in *array* where all logical matches with the target occurred (one based)
nn	Number of matches found. Length of index.
mask	Mask of 1's from the right; the size of the field looked for in the table.
right	Number of bits to shift right so as to right-justify the field searched

The Fortran equivalent of logical search performed follows:

```
TABLE(ARRAY(INDEX(I),I=1,NN)).LT.TARGET

TABLE(ARRAY(INDEX(I),I=1,NN)).LE.TARGET

TABLE(ARRAY(INDEX(I),I=1,NN)).GT.TARGET

TABLE(ARRAY(INDEX(I),I=1,NN)).GE.TARGET

where TABLE(X)=AND(MASK,SHIFTR(X,RIGHT))
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

7. SORTING ROUTINES

There are two ways to perform a sort on files: they can be sorted using the **SORT** control statement or the SORT subroutines. The **ORDERS** routine is used to sort memory arrays rather than files.

The **SORT** control statement provides a generalized sort and merge capability. **SORT** accesses multiple input files and permits mixed key types and variable length records. It provides a variety of user-specified random access devices (such as disk, Buffer Memory Resident (BMR), and SSD solid-state storage device) and tuning parameters for performance enhancement.

The **SORT** program provides these capabilities through calls to the **SORT** subroutines. **SORT** subroutines provide all of the above-mentioned options and allow the use of user-supplied subroutines. For more information on **SORT** and its associated subroutines, see the SORT Reference Manual, CRI publication SR-0074.

ORDERS is an internal, fixed-length record sort optimized for Cray computer systems. This section gives the synopsis and description of the **ORDERS** routine, including several examples using **ORDERS**.

NAME

ORDERS – Sorts using internal, fixed-length record sort optimized for Cray computer systems

SYNOPSIS

CALL ORDERS(*mode,iwork,data,index,n,ireclth,ikeylth,iradsiz***)**

DESCRIPTION

ORDERS assumes that the *n* records to be sorted are of length *ireclth* and have been stored in an array *data* that has been dimensioned, as in the following Fortran code:

DIMENSION DATA(*ireclth,n***)**

ORDERS does not move records within *data*, but returns a vector *index* containing pointers to each of the records in ascending order. For example, **DATA(1,INDEX(1))** is the first word of the record with the smallest key.

The **ORDERS** arguments are as follows:

mode Integer flag; describes the type of key and indicates an initial ordering of the records, as follows:

- **0** The key is binary numbers of length 8**ikeylth*. These numbers are considered positive integers in the range 0 to $2^{(8*ireclth)-1}$. (The ordering of ASCII characters is the same as their ordering as positive integers.)
- **1** The key is 64-bit Cray integers. These are twos complement signed integers in the range -2^{63} to $+2^{63}$. (The key length, if specified, must be 8 bytes.)
- **2** The key is 64-bit Cray floating-point numbers. (The key length, if specified, must be 8 bytes.)
- **10** The key is the same as *mode*=0, but the array **INDEX** has an initial ordering of the records (see subsection MULTIPASS SORTING later in this section).
- **11** The key is the same as *mode*=1, but the array **INDEX** has an initial ordering of the records.
- **12** The key is the same as *mode*=2, but the array **INDEX** has an initial ordering of the records.

Upon completion of a call, **ORDERS** returns an error flag in *mode*. A value equal to the input *mode* value indicates no errors. A value less than 0 indicates an error, as follows:

-1 Too few arguments; must be greater than 4.

-2 Too many arguments; must be less than 9.

-3 Number of words per record less than 1 or greater than 2**24

-4 Length of key greater than the record

-5 Radix not equal to 1 or 2

-6 Key less than 1 byte long

-7 Number of records less than 1 or greater than 2**24

-8 Invalid *mode* input values; must be **0, 1, 2, 10, 11**, or **12.**

-9 Key length must be 8 bytes for real or integer sort

iwork User-supplied working storage array of length K, where K=257 if *iradsiz*=1, or K=65537 if *iradsiz*=2

data Array dimensioned *irecIth* by N, containing N records of length *irecIth* each. The key in each record starts at the left of the first word of the record and continues *ikeylth* bytes into successive words as necessary. (By offsetting this address, any word within the record may be used as a key. See subsection EXAMPLES later in this section.)

index Integer array of length *n* containing pointers to the records. In *mode*=**10, 11**, or **12**, *index* contains an initial ordering of the records (see subsection MULTIPASS SORTING later in this section). On output, *index* contains the ordering of the records; that is, **DATA(1,INDEX(I))** is the first word of the record with the smallest key, and **DATA(1,INDEX(N))** is the first word of the record with the largest key.

n Number of records to be sorted. Must be ≥1.

irecIth Length of each record as a number of 64-bit words. Default is 1. *irecIth* is used as a skip for vector loads and stores; therefore, *irecIth* should be chosen to avoid bank conflicts.

ikeylth Length of each key as a number of 8-bit bytes. Default is 8 bytes (1 word).

iradsiz Radix of the sort. *iradsiz* is the number of bytes processed per pass over the records. Default is 1. See subsection of LARGE RADIX SORTING for *iradsiz*=2.

METHOD

ORDERS uses the radix sort, more commonly known as a bucket or pocket sort. For this type of sort, the length of the key in bytes determines the number of passes made through all of the records. The method has a linear work factor and is stable, in that the original order of records with equal keys is preserved.

ORDERS has the option of processing 1 or 2 bytes of the key per pass through the records. This process halves the number of passes through the record, but at the expense of increased working storage and overhead per pass. **ORDERS** can sort on several keys within a record by using its multipass capability. The first 8 bytes of the keys use a radix sort. If the key length is greater than 8 bytes and any records have the first 8 bytes equal, these records are sorted using a simple bubble sort. Using the bubble sort with many records is time-consuming; therefore, the multipass option should be used.

ORDERS has been optimized in CAL to make efficient use of the vector registers and functional units at each step of a pass through the data. Keys are read into vector registers with a skip through memory of *ireclth*; therefore, *ireclth* should be chosen to avoid bank conflicts.

LARGE RADIX SORTING

The number of times the key of each record is read from memory is proportional to *ikeylth*/*iradsiz*. Using **ORDERS** with *iradsiz*=2 halves this ratio because 2 bytes instead of 1 are processed each time the key is read. The disadvantage of halving the number of passes is that the user-supplied working storage array goes from 257 words to 65,537 words. This favors a 1-byte pass for sorting up to approximately 5000 records. For more than 5000 records, however, a 2-byte pass is faster.

MULTIPASS SORTING

Because the array **INDEX** can define an ordering of the records, several calls can be made to **ORDERS** where the order of the records is that of the previous call. *mode*=**10, 11,** or **12** specifies that the array **INDEX** contains an ordering from a previous call to **ORDERS**. This specification allows sorting of text keys that extend over more than 1 word or keys involving double-precision numbers. (See the subsection EXAMPLES later in this section.)

Although the length of the key is limited only by the length of the record, up to 8 bytes are sorted with the radix sort. The remaining key is sorted using a bubble sort, but only in those records whose keys are equal for the first 8 bytes. Therefore, a uniformly-distributed key over the first 8 bytes of length greater than 8 bytes might be sorted faster using a single call with a large *ikeylth* rather than a multipass call. When using the multipass capability, sort the least significant word first.

IMPLEMENTATION

ORDERS is available to users of both the COS and UNICOS operating systems.

EXAMPLES

Example 1:

This example performs a sort on an array of random numbers, 20 records long, with a key length of 8 bytes (1 word).

```
      PROGRAM ORDWAY
      DIMENSION DATA(1,20)
      DIMENSION INDEX(20)
      DIMENSION IWORK(257)
C
C     Place random numbers into the array DATA
C
      DO 1 I=1,20
1      DATA(1,I)=2*RANF( )
C
      N=20
      MODE=0
C
      CALL ORDERS(MODE,IWORK,DATA,INDEX,N,1,8,1)
C
C     Print out the sorted records in increasing order
C
      DO 2 K=1,20
2      PRINT *, DATA(1,INDEX(K))
      STOP
      END
```

Example 2:

This program uses two calls to **ORDERS** to completely sort an array of double-precision numbers. The sign bit of the first word is used to change the second word into a text key that preserves the ordering. A sort is done on these 6 bytes of the second word. (The changes made to the second word are reversed after the call.) Last, a sort is done on the first word as a real key using the initial ordering from the previous call.

```
      PROGRAM SORT2
      DOUBLE PRECISION DATA(100)
      INTEGER IATA(200)
      EQUIVALENCE(IATA, DATA)
      INTEGER INDEX(100), IWORK(257)
      N=12
      DO 5 I=1, N
          DATA(I)=(-1.D0)**10.D0**(-20)*DBLE(RANF())
5     CONTINUE
```

```
C
C        First the second word key is changed
C
         DO 10 I=2, 2*N, 2
               IF(DATA(I/2).LE.0.D0) THEN
                IATA(I)=COMPL(IATA(I))
               ELSE
                IATA(I)=IATA(I)
               ENDIF
   10    CONTINUE
C
C        Sort on second word
C
         MODE=0
         CALL ORDERS(MODE,IWORK,IATA(2),INDEX,N, 2, 6, 1)
C
C        Restore second word to original form
C
         DO 20 I=2, 2*N, 2
               IF(DATA(I/2).LE.0.D0) THEN
                IATA(I)=COMPL(IATA(I))
               ELSE
                IATA(I)=IATA(I)
               ENDIF
   20    CONTINUE
C
C        Sort on the first word using the initial ordering
C
         MODE=12
         CALL ORDERS(MODE,SORT,DATA,INDEX,N,2,8,1)
         DO 50 I=1,N
               WRITE(6, 900)I, INDEX(I), DATA(INDEX(I))
   50    CONTINUE
  900    FORMAT(1X, 2I5, 2X, D40.30)
         END
```

8. CONVERSION SUBPROGRAMS

These Fortran-callable subroutines perform conversion of data residing in Cray memory. Conversion subprograms are listed under the following types of routines:

- Foreign data conversion
- Numeric conversion
- ASCII conversion
- Other conversion

For more information regarding foreign data conversion, see the Foreign Data Conversion on CRAY-1 and CRAY X-MP Computer Systems technical note, publication SN-0236.

FOREIGN DATA CONVERSION ROUTINES

The foreign data conversion routines allow data translation between Cray internal representations and other vendors' data types. These include IBM, CDC, and VAX data conversion routines.

The following tables convert values from Cray data types to IBM, VAX/VMS, and CDC data types. Routines that are inverses of each other (that is, convert from Cray data types to IBM and IBM to Cray) are generally listed under a single entry. Routine descriptions follow later in this section, listed alphabetically by entry name.

The following table lists routines that convert foreign types to Cray types.

Convert Foreign Types to Cray Types			
Convert to	Foreign types		
Convert from	IBM	CDC	VAX/VMS
Foreign single-precision to Cray single-precision	**USSCTC**	**FP6064**	**VXSCTC**
Foreign double-precision to Cray single-precision	**USDCTC**	---	**VXDCTC** **VSGCTC**
Foreign integer to Cray integer	**USICTC**	**INT6064**	**VXICTC**
Foreign logical to Cray logical	**USLCTC**	---	**VXLCTC**
Foreign character to ASCII	**USCCTC**	**DSASC**	---
VAX 64-bit complex to Cray single-precision	---	---	**VXZCTC**
IBM packed decimal field to Cray integer	**USPCTC**	---	---

The following table lists routines that convert Cray types to foreign types.

Convert Cray Types to Foreign Types			
Convert To	Foreign Types		
Convert From	IBM	CDC	VAX/VMS
Cray single-precision to foreign single-precision	**USSCTI**	**FP6460**	**VXSCTI**
Cray single-precision to foreign double-precision	**USDCTI**	---	**VXDCTI**
Cray integer to foreign integer	**USICTI**	**INT6460**	**VXICTI**
Cray logical to foreign logical	**USLCTI**	---	**VXLCTI**
ASCII character to foreign character	**USCCTI**	**ASCDC**	---
Cray complex to foreign complex	---	---	**VXZCTI**
Cray integer to foreign packed-decimal field	**USICTP**	---	---

NUMERIC CONVERSION ROUTINES

Numeric conversion routines convert a character to a numeric format or a number to a character format.

The following table contains the purpose, names, and entry of each numeric conversion routine.

Numeric Conversion Routines		
Purpose	Name	Entry
Convert decimal ASCII numerals to an integer value	**CHCONV**	**CHCONV**
Convert an integer to a decimal ASCII string	**BICONV**	**BICONV**
Convert an integer to a decimal ASCII string (zero-filled, right-justified)	**BICONZ**	

ASCII CONVERSION FUNCTIONS

The ASCII conversion functions convert binary integers to or from 1-word ASCII strings (not Fortran character variables). Fortran-callable entry points (in the form *xxx*) use the call-by-address sequence; CAL-callable entry points (in the form *xxx%*) use the call-by-value sequence.

NOTE - The ASCII conversion functions are not intrinsic to Fortran. Their default type is real, even though their results are generally used as integers.

IMPLEMENTATION - The ASCII conversion functions are available to users of both the COS and UNICOS operating systems.

The ASCII conversion routines use one type integer argument. The **DTB/DTB%** and **OTB/OTB%** routines can also use a second optional argument as an error code. The resulting error codes (0 if no error; –1 if there are errors) are returned in the second argument for Fortran calls and in register S0 for CAL calls. If no error code argument is included in Fortran calls, the routine aborts upon encountering an error.

The following calls show how the ASCII conversion routines are used. These Fortran calls convert a binary number to decimal ASCII, then convert back from ASCII to binary:

result=BTD(*integer*)

result Decimal ASCII result (right-justified, blank-filled)

integer Integer argument

result=DTB(*arg,errcode*)

result Integer value

arg Decimal ASCII (left-justified, zero-filled)

errcode 0 if conversion successful; –1 if error.

ASCII Conversion Routines			
Purpose	Name	Argument Range	Result
Binary to decimal ASCII (right-justified, blank-filled)	**BTD** **BTD%**	$0 \le x \le 99999999$	One-word ASCII string (right-justified, blank-filled)
Binary to decimal ASCII (left-justified, zero-filled)	**BTDL** **BTDL%**	$0 \le x \le 99999999$	One-word ASCII string (left-justified, zero-filled)
Binary to decimal ASCII (right-justified, zero-filled)	**BTDR** **BTDR%**	$0 \le x \le 99999999$	One-word ASCII string (right-justified, zero-filled)
Binary to octal ASCII (right-justified, blank-filled)	**BTO** **BTO%**	$0 \le x \le 77777777_8$	One-word ASCII string (right-justified, blank-filled)
Binary to octal ASCII (left-justified, zero-filled)	**BTOL** **BTOL%**	$0 \le x \le 77777777_8$	One-word ASCII string (left-justified, (zero-filled)
Binary to octal ASCII (right-justified, zero-filled)	**BTOR** **BTOR%**	$0 \le x \le 77777777_8$	One-word ASCII string (right-justified, zero-filled)
Decimal ASCII to binary	**DTB** **DTB%**	Decimal ASCII (left-justified, zero-filled)	One word containing decimal equivalent of ASCII string
Octal ASCII to binary	**OTB** **OTB%**	Octal ASCII (left-justified, zero-filled)	One word containing octal equivalent of ASCII string

OTHER CONVERSION ROUTINES

These routines place the octal ASCII representation of a Cray word into a character area, convert trailing blanks to nulls or trailing nulls to blanks, and translate a string from one code to another, using a translation table.

The following table contains the purpose, name, and entry of these conversion routines.

<table>
<tr><th colspan="3">Other Conversion Routines</th></tr>
<tr><th>Purpose</th><th>Name</th><th>Entry</th></tr>
<tr><td>Place an octal ASCII representation of a Cray word into a character area</td><td>B2OCT</td><td>B2OCT</td></tr>
<tr><td>Convert trailing blanks to nulls</td><td>RBN</td><td rowspan="2">RBN</td></tr>
<tr><td>Convert trailing nulls to blanks</td><td>RNB</td></tr>
<tr><td>Translate a string from one code to another, using a translation table</td><td>TR</td><td>TR</td></tr>
</table>

NAME

B2OCT – Places an octal ASCII representation of a Cray word into a character area

SYNOPSIS

CALL B2OCT(*s*,*j*,*k*,*v*,*n*)

DESCRIPTION

s First word of an array where the ASCII representation is to be placed

j Byte offset within array *s* where the first character of the octal representation is to be placed. A value of 1 indicates that the destination begins with the first (leftmost) byte of the first word of *s*. *j* must be greater than 0.

k Number of characters used in the ASCII representation; *k* must be greater than 0. *k* indicates the size of the total area to be filled, and the area is blank-filled if necessary.

v Value to be converted. The low-order *n* bits of word *v* are used to form the ASCII representation. *v* must be less than or equal to $2^{63}-1$.

n Number of low-order bits of *v* to convert to ASCII character representation ($1 \leq n \leq 64$). If insufficient character space is available ($3k<n$), the character region is automatically filled with asterisks (*).

B2OCT places the ASCII representation of the low-order *n* bits of a full Cray word into a specified character area.

The *k* characters in array *s*, pointed to by *j*, are first set to blanks. The low-order *n* bits of *v* are then converted to octal ASCII, using leading zeros if necessary. The converted value (*n*/3 characters, rounded up) is right-justified into the blanked-out destination character region.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

BICONV, BICONZ – Converts a specified ASCII string representing the integer

SYNOPSIS

CALL BICONV(*int,dest,isb,len*)

CALL BICONZ(*int,dest,isb,len*)

DESCRIPTION

int	Integer variable, expression, or constant to be converted
dest	Variable or array of any type or length to contain the ASCII result
isb	Starting byte count to generate the output string. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

BICONV converts a specified integer to an ASCII string. The string generated by **BICONV** is blank-filled, right-justified, and has a maximum width of 256 bytes. If the specified field width is not long enough to hold the converted integer number, left digits are truncated and no indication of overflow is given. If the number to be converted is negative, a minus sign is positioned in the output field to the left of the first significant digit.

BICONZ is the same as **BICONV** except that the output string generated is zero-filled, right-justified. (A minus sign, if any, appears in the leftmost character position of the field.)

IMPLEMENTATION

These routines are available only to users of the COS operating system.

NAME

CHCONV – Converts decimal ASCII numerals to an integer value

SYNOPSIS

CALL CHCONV(*src*,*isb*,*num*,*ir*)

DESCRIPTION

src	Variable or array of type Hollerith containing ASCII data or blanks
isb	Starting character in the *src* string. Specify an integer variable, expression, or constant. Characters are numbered from 1, beginning at the leftmost character position of *src*.
num	Number of ASCII characters to convert. Specify an integer variable, expression, or constant.
ir	Integer result

Blanks in the input field are treated as zeros. A minus sign encountered anywhere in the input field produces a negative result. Input characters other than blank, digits 0 through 9, a minus sign, or more than one minus sign produce a fatal error.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

DSASC, ASCDC – Converts CDC display code character to ASCII character and vice versa

SYNOPSIS

CALL DSASC(*src,sc,dest,num***)**

CALL ASCDC(*src,sc,dest,num***)**

DESCRIPTION

src For **DSASC**, a variable or array of any type or length containing CDC display code characters (64-character set), left-justified in a 64-bit Cray word. Contains a maximum of 10 display code characters per word. For **ASCDC**, a variable or array of any type or length containing ASCII data.

sc Display code or ASCII character position to begin the conversion. Leftmost position is 1.

dest For **DSASC**, a variable or array of any type or length to contain the converted ASCII data. Results are packed 8 characters per word. For **ASCDC**, a variable or array of any type or length to contain the converted CDC display code characters (64-character set). Results are packed into continuous strings without regard to word boundaries.

num Number of CDC display code or ASCII characters to convert. Specify an integer variable, expression, or constant.

DSASC converts CDC display code characters to ASCII character.

ASCDC converts ASCII characters to CDC display code characters.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

FP6064, FP6460 – Converts CDC 60-bit single-precision numbers to Cray 64-bit single-precision numbers and vice versa

SYNOPSIS

CALL FP6064(*fpn,dest,num*)

CALL FP6460(*fpn,dest,num*)

DESCRIPTION

fpn For **FP6064**, a variable or array of any type or length containing CDC 60-bit, single-precision numbers, left-justified in a Cray 64-bit word. For **FP6460**, a variable or array of any length and of type real containing Cray single-precision numbers.

dest Variable or array of type real to contain the converted Cray 64-bit, single-precision or CDC 60-bit single-precision numbers. (In **FP6460**, each floating-point number is left-justified in a 64-bit word.)

num Number of CDC or Cray single-precision numbers to convert. Specify an integer variable, expression, or constant.

FP6064 converts CDC 60-bit single-precision numbers to Cray 64-bit single-precision numbers.

FP6460 converts Cray 64-bit single-precision numbers to CDC 60-bit single-precision numbers.

IMPLEMENTATION

These routines are available to users of the both the COS and UNICOS operating systems.

NAME

INT6064 – Converts CDC 60-bit integers to Cray 64-bit integers

SYNOPSIS

CALL INT6064(*src,idest,num***)**

DESCRIPTION

src Variable or array of any type or length containing CDC 60-bit integers, left-justified in a Cray 64-bit word

idest Variable or array of type integer to contain the converted values. Each such integer is left-justified and zero-filled.

num Number of CDC integers to convert. Specify an integer variable, expression, or constant.

INT6064 converts CDC 60-bit integer numbers to Cray 64-bit integer numbers.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

INT6460 is the inverse of this routine

NAME

INT6460 – Converts Cray 64-bit integers to CDC 60-bit integers

SYNOPSIS

CALL INT6460(*in,idest,num*)

DESCRIPTION

in Variable or array of any length and of type integer containing Cray integer numbers

idest Variable or array of type integer to contain the converted values or CDC integer numbers. Each such integer is left-justified and zero-filled.

num Number of Cray integers to convert. Specify an integer variable, expression, or constant.

INT6460 converts Cray 64-bit integer numbers to CDC 60-bit integer numbers.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

INT6064 is the inverse of this routine

NAME

RBN, RNB – Converts trailing blanks to nulls and vice versa

SYNOPSIS

noblanks=**RBN**(*blanks*)

blanks=**RNB**(*noblanks*)

DESCRIPTION

blanks For **RBN**, the argument to be converted. For **RNB**, the argument after conversion.

noblanks For **RBN**, the argument after conversion. For **RNB**, the argument to be converted.

RBN converts trailing blanks to nulls. **RNB** converts trailing nulls to blanks.

NOTE

Fortran programs using **RBN** or **RNB** must declare the function to be type integer.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

TR – Translates a string from one code to another using a translation table

SYNOPSIS

CALL TR(*s*,*j*,*k*,*table*)

DESCRIPTION

s	First word of an array containing the characters to be translated
j	Byte offset within array *s* where the first character to be translated occurs
k	Number of characters to be translated
table	Translation table

TR translates a string in place from one character code to another using a user-supplied translation table. The routine assumes 8-bit characters.

The translation table must be considered a string of 256 bytes (32 words). As each character to be translated is fetched, it is used as an index into the translation table. The new value of the character is the content of the translation-table byte addressed by the old value. (The first byte of the translation table is considered to be byte 0.)

NOTE

Several translation tables are available as comdecks in **UTLIBPL**. You may want to make use of these predefined tranlations. Each comdeck contains a Fortran declaration of the comdeck name as a 32-word integer array and contains data statements for each word of the array. The available translations are:

Comdeck	Translation
TRUPPER	ASCII to ASCII, but converting lower-case letters to upper-case
TRLOWER	ASCII to ASCII, but converting upper-case letters to lower-case
TRASCII	EBCDIC to ASCII. This differs from the translation provided by **USSCTC** in that every byte value has a unique translation. (Use a subsequent translation with **TRNPC** to remove nonprinting characters).
TREBCDIC	ASCII to EBCDIC. This differs from the translation provided by **USSCTI** in that every byte value has a unique translation.
TRNPC	ASCII to ASCII, but converting nonprinting characters to periods ('.')

Note that **TRASCII** and **TREBCDIC** are inverses of each other.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

TRR1 – Translates characters stored one character per word

SYNOPSIS

CALL TRR1(*s,k,table***)**

DESCRIPTION

s	Array containing the characters to be translated
k	Number of characters to be translated
table	Translation table

TRR1 translates *k* characters, stored one character per word, right-justified, zero-filled, in array *s* using the translation table *table*.

table is a 256-word array (dimensioned (0:255)) containing the translation for each character in the entry for the character viewed as an integer.

TRR1 leaves *s*(I) unchanged if *s*(I) is not in the range 0,...,255.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

USCCTC, USCCTI – Converts IBM EBCDIC data to ASCII data and vice versa

SYNOPSIS

CALL USCCTC(*src,isb,dest,num,npw*[*,val*])

CALL USCCTI(*src,dest,isb,num,npw*[*,val*])

DESCRIPTION

src — Variable or array of any type or length containing IBM EBCDIC data or ASCII data, left-justified, in Cray words, to convert

isb — For USCCTC, a byte number to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*. For USCCTI, a byte number at which to begin generating EBCDIC characters in *dest*.

dest — Variable or array of any type or length to contain the IBM EBCDIC or ASCII data

num — Number of IBM EBCDIC or ASCII characters to convert. Specify an integer variable, expression, or constant.

npw — Number of characters per word generated in *dest* (or selected from *src* in USCCTI). The *npw* characters are left-justified and blank-filled in each word of *dest*. Specify an integer variable, expression, or constant. Value must be from 1 to 8.

val — A value of nonzero specifies lowercase characters (a through z) that are to be translated to uppercase. A value of 0 results in no case translation. This is an optional parameter specified as an integer variable, expression, or constant. The default is no case translation.

USCCTC converts IBM EBCDIC data to ASCII data. The same array can be specified for output as for input only if *isb* = 1 and *npw* = 8.

USCCTI converts ASCII data to IBM EBCDIC data. All unprintable characters are converted to blanks. The same array can be specified for output as for input only if *isb* = 1 and *npw* = 8.

NOTE

You may also find routine **TR** (described in this section) useful. It provides somewhat more control over the specific translation used, although it does require the translation to be done in place.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

USDCTC – Converts IBM 64-bit floating-point numbers to Cray 64-bit single-precision numbers

SYNOPSIS

CALL USDCTC(*dpn*,*isb*,*dest*,*num*[,*inc*])

DESCRIPTION

dpn Variable or array of any type or length containing IBM 64-bit floating-point numbers to convert

isb Byte number to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *fpn* or *dpn*.

dest Variable or array of type real to contain the converted values

num Number of IBM 64-bit floating-point numbers to convert. Specify an integer variable, expression, or constant.

inc Memory increment for storing the conversion results in *dest*. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

USDCTI is the inverse of this routine.

NAME

USDCTI – Converts Cray 64-bit single-precision, floating-point numbers to IBM 64-bit double precision numbers

SYNOPSIS

CALL USDCTI(*fpn*,*dest*,*isb*,*num*,*ier*[,*inc*])

DESCRIPTION

fpn Variable or array of any length and type real, containing Cray 64-bit single-precision, floating-point numbers to convert

dest Variable or array of type real to contain the converted values

isb Byte number at which to begin storing the converted results. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

num Number of Cray floating-point numbers to convert. Specify an integer variable, expression, or constant.

ier Overflow indicator of type integer. Value is 0 if all Cray values convert to IBM values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.

inc Memory increment for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

USDCTI converts Cray 64-bit single-precision, floating-point numbers to IBM 64-bit double-precision, floating-point numbers. Precision is extended by introducing 8 more bits into the rightmost byte of the fraction from the Cray number being converted. Numbers that produce an underflow when converted to IBM format are converted to 64 binary 0s. Numbers that produce an overflow when converted to IBM format are converted to the largest IBM floating-point representation with the sign bit set if negative. An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow. No such indication is given for underflow.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

USDCTC is the inverse of this routine.

NAME

USICTC, USICTI – Converts IBM INTEGER*2 and INTEGER*4 numbers to Cray 64-bit integer numbers, and vice versa

SYNOPSIS

CALL USICTC(*in*,*isb*,*dest*,*num*,*len*[,*inc*])

CALL USICTI(*in*,*dest*,*isb*,*num*,*len*,*ier*[,*inc*])

DESCRIPTION

in — Variable or array of any type or length containing IBM INTEGER*2 or INTEGER*4 numbers or Cray 64-bit integers to convert

isb — Byte number at which to begin the conversion or at which to begin storing the converted results. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *in* (*dest* in USICTI).

dest — Variable or array of type integer to contain the converted values

num — Number of IBM numbers or Cray integers to convert. Specify an integer variable, expression, or constant.

len — Size of the IBM numbers to convert or of IBM result numbers. These values must be 2 or 4. A value of 2 indicates that input or output integers are INTEGER*2 (16-bit). A value of 4 indicates that input or output integers are INTEGER*4 (32-bit). Specify an integer variable, expression, or constant.

inc — Memory increment for storing the conversion results in *dest* or for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

ier — Overflow indicator of type integer. The value is zero if all Cray values converted to IBM values without overflow. The value is not zero if one or more Cray values overflowed in the conversion.

USICTC converts IBM INTEGER*2 and INTEGER*4 numbers to Cray 64-bit integer numbers.

USICTI converts Cray 64-bit integer numbers to IBM INTEGER*2 or INTEGER*4 numbers.

Numbers that produce an overflow when converted to IBM format are converted to the largest IBM integer representation, with the sign bit set if negative. An error parameter returns nonzero to indicate that one or more of the numbers converted produced an overflow.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

USICTP – Converts a Cray 64-bit integer to IBM packed-decimal field

SYNOPSIS

CALL USICTP(*ian,dest,isb,num*)

DESCRIPTION

ian Cray integer to be converted to an IBM packed-decimal field. Specify an integer variable, expression, or constant.

dest Variable or array of any type or length to contain the packed field generated

isb Byte number within *dest* specifying the beginning location for storage. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

num Number of bytes to be stored. Specify an integer variable, expression, or constant.

If the input value contains more digits than can be stored in *num* bytes, the leftmost digits are not converted.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

USPCTC is the inverse of this routine.

NAME

USLCTC, USLCTI – Converts IBM LOGICAL*1 and LOGICAL*4 values into Cray 64-bit logical values, and vice versa

SYNOPSIS

CALL USLCTC(*src,isb,dest,num,len*[*,inc*])

CALL USLCTI(*src,dest,isb,num,len*[*,inc*])

DESCRIPTION

src Variable or array of any type (type logical in USLCTI) and any length containing IBM LOGICAL*1, LOGICAL*4, or Cray logical values to convert.

isb Byte number to begin the conversion or, in USLCTI, specifying the beginning location for storage. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.

dest Variable or array of any type or length to contain the converted values

num Number of IBM or Cray logical values to be converted. Specify an integer variable, expression, or constant.

len Size of the IBM logical values to convert or of the logical result value. These values must be 1 or 4. A value of 1 indicates that input or output logical values are LOGICAL*1 (8-bit). A value of 4 indicates that input or output logical values are LOGICAL*4 (32-bit). Specify an integer variable, expression, or constant.

inc Memory increment for storing the conversion results in *dest* or for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

USLCTC converts IBM LOGICAL*1 and LOGICAL*4 values to Cray 64-bit logical values.

USLCTI converts Cray logical values to IBM LOGICAL*1 or LOGICAL*4 values.

All arguments must be entered in the same order in which they appear in the synopsis.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

USPCTC – Converts a specified number of bytes of an IBM packed-decimal field to a 64-bit integer field

SYNOPSIS

CALL USPCTC(*src,isb,num,ian*)

DESCRIPTION

src	Variable or array of any type or length containing a valid IBM packed-decimal field
isb	Byte number to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.
num	Number of bytes to convert. Specify an integer variable, expression, or constant.
ian	Returned integer result

The input field must be a valid packed-decimal number less than 16 bytes long, of which only the rightmost 15 digits are converted.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

USICTP is the inverse of this routine.

NAME

USSCTC – Converts IBM 32-bit floating-point numbers to Cray 64-bit single-precision numbers

SYNOPSIS

CALL USSCTC(*fpn*,*isb*,*dest*,*num*[,*inc*])

DESCRIPTION

fpn Variable or array of any type or length containing IBM 32-bit floating-point numbers to convert

isb Byte number to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *fpn* or *dpn*.

dest Variable or array of type real to contain the converted values

num Number of IBM 32-bit floating-point numbers to convert. Specify an integer variable, expression, or constant.

inc Memory increment for storing the conversion results in *dest*. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

USSCTI is the inverse of this routine.

NAME

USSCTI – Converts Cray 64-bit single-precision, floating-point numbers to IBM 32-bit single-precision numbers

SYNOPSIS

CALL USSCTI(*fpn*,*dest*,*isb*,*num*,*ier*[,*inc*])

DESCRIPTION

fpn Variable or array of any length and type real, containing Cray 64-bit single-precision, floating-point numbers to convert

dest Variable or array of type real to contain the converted values

isb Byte number at which to begin storing the converted results. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

num Number of Cray floating-point numbers to convert. Specify an integer variable, expression, or constant.

ier Overflow indicator of type integer. Value is 0 if all Cray values convert to IBM values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.

inc Memory increment for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

USSCTI converts Cray 64-bit single-precision, floating-point numbers to IBM 32-bit single-precision, floating-point numbers. Numbers that produce an underflow when converted to IBM format are converted to 32 binary 0s. Numbers that produce an overflow when converted to IBM format are converted to the largest IBM floating-point representation, with the sign bit set if negative.

An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow. No such indication is given for underflow.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

USSCTC is the inverse of this routine.

NAME

VXDCTC – Converts VAX 64-bit D format numbers to Cray single-precision numbers

SYNOPSIS

CALL VXDCTC(*dpn,isb,dest,num,*[*inc*])

DESCRIPTION

dpn Variable or array of any type or length containing VAX D format numbers to convert

isb Byte number within *dpn* at which to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte of *dpn.*

dest Variable or array of type real to contain the converted values

num Number of VAX D format numbers to convert. Specify an integer variable, expression, or constant.

inc Memory increment for storing the conversion results in *dest.* This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXDCTI is the inverse of this routine.

NAME

VXDCTI – Converts Cray 64-bit single-precision, floating-point numbers to VAX D format single-precision, floating-point numbers

SYNOPSIS

CALL VXDCTI(*fpn*,*dest*,*isb*,*num*,*ier*,[*inc*])

DESCRIPTION

fpn Variable or array of any length and type real containing Cray 64-bit single-precision, floating-point numbers to convert

dest Variable or array of type real to contain the converted values

isb Byte number at which to begin storing the converted results. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

num Number of Cray floating-point numbers to convert. Specify an integer variable, expression, or constant.

ier Overflow indicator of type integer. Value is 0 if all Cray values convert to VAX values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.

inc Memory increment for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant.

Numbers that produce an underflow when converted to VAX format are converted to 32 binary 0s. Numbers that are in overflow on the Cray computer system are converted to a "reserved" floating-point representation, with the sign bit set if negative. Numbers that are valid on the Cray computer system but overflow on the VAX are converted to the most positive possible number or most negative possible number, depending on the sign.

An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow. (Deferred implementation; at present, you must supply the parameter, which is always returned as 0.) No such indication is given for underflow.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXDCTC is the inverse of this routine.

NAME

VXGCTC – Converts VAX 64-bit G format numbers to Cray single-precision numbers

SYNOPSIS

CALL VXGCTC(*dpn*,*isb*,*dest*,*num*,[*inc*])

DESCRIPTION

dpn	Variable or array of any type or length containing VAX G format numbers to convert
isb	Byte number within *dpn* at which to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte of *dpn*.
dest	Variable or array of type real to contain the converted values
num	Number of VAX G format numbers to convert. Specify an integer variable, expression, or constant.
inc	Memory increment for storing the conversion results in *dest*. This is an optional parameter specified as an integer variable, expression, or conxtant. The default value is 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXGCTI is the inverse of this routine.

NAME

VXGCTI – Converts Cray 64-bit single-precision, floating-point numbers to VAX G format single-precision, floating-point numbers

SYNOPSIS

CALL VXGCTI(*fpn*,*dest*,*isb*,*num*,*ier*,[*inc*])

DESCRIPTION

fpn Variable or array of any length and type real, containing Cray 64-bit single-precision, floating-point numbers to convert

dest Variable or array of type real to contain the converted values

isb Byte number at which to begin storing the converted results. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

num Number of Cray floating-point numbers to convert. Specify an integer variable, expression, or constant.

ier Overflow indicator of type integer. Value is 0 if all Cray values convert to VAX values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.

inc Memory increment for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

VXGCTI converts Cray 64-bit single-precision, floating-point numbers to VAX G format single-precision, floating-point numbers.

Numbers that produce an underflow when converted to VAX format are converted to 32 binary zeros. Numbers that are in overflow on the Cray computer system are converted to a "reserved" floating-point representation, with the sign bit set if negative. Numbers that are valid on the Cray computer system but overflow on the VAX are converted to the most positive possible number or most negative possible number, depending on the sign.

An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow (Deferred implementation. At present, you must supply the parameter, which is always as 0.) No such indication is given for underflow.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXGCTC is the inverse of this routine.

NAME

VXICTC – Converts VAX INTEGER*2 or INTEGER*4 to Cray 64-bit integers

SYNOPSIS

CALL VXICTC(*in,isb,dest,num,len,*[*inc*])

DESCRIPTION

in	Variable or array of any type or length containing VAX 16- or 32-bit integers
isb	Byte number at which to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *in*.
dest	Variable or array of type integer to contain the converted values
num	Number of VAX integers to convert. Specify an integer variable, expression, or constant.
len	Size of the VAX numbers to convert. This value must be 2 or 4. A value of 2 indicates that input integers are 16-bit. A value of 4 indicates that input integers are 32-bit. Specify an integer variable, expression, or constant.
inc	Memory increment for storing conversion results in *dest*. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXICTI is the inverse of this routine.

NAME

VXICTI – Converts Cray 64-bit integers to either VAX INTEGER*2 or INTEGER*4 numbers

SYNOPSIS

CALL VXICTI(*in*,*dest*,*isb*,*num*,*len*,*ier*,[*inc*])

DESCRIPTION

in Variable or array of any length and type integer, containing Cray integers to convert

dest Variable or array of type integer to contain the converted values

isb Byte number at which to begin storing the converted results. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

num Number of Cray integers to convert. Specify an integer variable, expression, or constant.

len Size of the VAX result numbers. This value must be 2 or 4. A value of 2 indicates that output integers are INTEGER*2 (16-bit). A value of 4 indicates that output integers are INTEGER*4 (32-bit). Specify an integer variable, expression, or constant.

ier Overflow indicator of type integer. Value is 0 if all Cray values are converted to VAX values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.

inc Memory increment for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

Numbers that produce an overflow when converted to VAX format are converted to the largest VAX integer representation, with the sign bit set if negative.

An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow. (Deferred implementation; at present, you must supply the parameter, which is always returned as 0.) No such indication is given for underflow.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXICTC is the inverse of this routine.

NAME

VXLCTC – Converts VAX logical values to Cray 64-bit logical values

SYNOPSIS

CALL VXLCTC(*src,isb,dest,num,len,*[*inc*])

DESCRIPTION

src Variable or array of any type or length containing VAX logical values to convert

isb Byte number at which to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.

dest Variable or array of type logical to contain the converted values

num Number of VAX logical values to be converted. Specify an integer variable, expression, or constant.

len Size of the VAX logical values to convert. At present, this parameter must be set to 4, indicating that 32-bit logical values are to be converted. Specify an integer variable, expression, or constant.

inc Memory increment for storing the conversion results in *dest*. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

VXSCTC – Converts VAX 32-bit floating-point numbers to Cray 64-bit single-precision numbers

SYNOPSIS

CALL VXSCTC(*fpn*,*isb*,*dest*,*num*,[*inc*])

DESCRIPTION

fpn Variable or array of any type containing VAX 32-bit floating-point numbers to convert

isb Byte number at which to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *fpn*.

dest Variable or array of type real to contain the converted values

num Number of VAX floating-point numbers to convert. Specify an integer variable, expression, or constant.

inc Memory increment for storing the conversion results in *dest*. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXSCTI is the inverse of this routine.

NAME

VXSCTI – Converts Cray 64-bit single-precision, floating-point to VAX F format single-precision, floating-point

SYNOPSIS

CALL VXSCTI(*fpn*,*dest*,*isb*,*num*,*ier*,[*inc*])

DESCRIPTION

fpn Variable or array of any length and type real, containing Cray 64-bit single-precision, floating-point numbers to convert

dest Variable or array of type real to contain the converted values

isb Byte number at which to begin storing the converted results. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.

num Number of Cray floating-point numbers to convert. Specify an integer variable, expression, or constant.

ier Overflow indicator of type integer. Value is 0 if all Cray values convert to VAX values without overflow. Value is nonzero conversion.

inc Memory increment for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

Numbers that produce an underflow when converted to VAX format are converted to 32 binary 0s. Numbers that are in overflow on the Cray computer system are converted to a "reserved" floating-point representation, with the sign bit set if negative. Numbers that are valid on the Cray computer system but overflow on the VAX are converted to the most positive possible number or most negative possible number, depending on the sign.

An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow (Deferred implementation. At present you must supply the parameter, which is always returned as 0.) No such indication is given for underflow.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXSCTC is the inverse of this routine.

NAME

VXZCTC – Converts VAX 64-bit complex numbers to Cray complex numbers

SYNOPSIS

CALL VXZCTC(*dpn*,*isb*,*dest*,*num*,[*inc*])

DESCRIPTION

dpn Variable or array of any type or length containing complex numbers to convert

isb Byte number within *dpn* at which to begin the conversion. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte of *dpn*.

dest Variable or array of type complex to contain the converted values

num Number of complex numbers to convert. Specify an integer variable, expression, or constant.

inc Memory increment for storing the conversion results in *dest*. This is an optional parameter specified as an integer variable, expression, or constant. Default value is 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXZCTI is the inverse of this routine.

NAME

VXZCTI – Converts Cray complex numbers to VAX complex numbers

SYNOPSIS

CALL VXZCTI(*fpn*,*dest*,*isb*,*num*,*ier*,[*inc*])

DESCRIPTION

fpn	Variable or array of any length and type complex, containing Cray complex numbers to convert
dest	Variable or array of any type to contain the converted values
isb	Byte number at which to begin storing the converted results. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *dest*.
num	Number of Cray floating-point numbers to convert. Specify an integer variable, expression, or constant.
ier	Overflow indicator of type integer. Value is 0 if all Cray values convert to VAX values without overflow. Value is nonzero if one or more Cray values overflowed in the conversion.
inc	Memory increment for fetching the number to be converted. This is an optional parameter specified as an integer variable, expression, or constant. The default value is 1.

Numbers that produce an underflow when converted to VAX format are converted to 32 binary zero. Numbers that are in overflow on the Cray computer system are converted to a "reserved" floating-point representation, with the sign bit set if negative. Numbers that are valid on the Cray computer system but overflow on the VAX are converted to the most positive possible number or most negative possible number, depending on the sign.

An error parameter returns nonzero to indicate that one or more numbers converted produced an overflow (Deferred implementation. At present, you must supply the parameter, which is always returned as 0.) No such indication is given for underflow.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

VXZCTC is the inverse of this routine.

9. PACKING ROUTINES

The packing routines provide alternative ways to pack and unpack data into or out of Cray words. The following table contains the purpose, name, and entry of each packing routine.

<table>
<tr><th colspan="3">Packing Routines</th></tr>
<tr><th>Purpose</th><th>Name</th><th>Entry</th></tr>
<tr><td>Pack 32-bit words into Cray 64-bit words</td><td>P32</td><td rowspan="2">P32</td></tr>
<tr><td>Unpack 32-bit words from Cray 64-bit words</td><td>U32</td></tr>
<tr><td>Pack 60-bit words into Cray 64-bit words</td><td>P6460</td><td rowspan="2">P6460</td></tr>
<tr><td>Unpack 60-bit words from Cray 64-bit words</td><td>U6064</td></tr>
<tr><td>Compress stored data</td><td>PACK</td><td>PACK</td></tr>
<tr><td>Expand stored data</td><td>UNPACK</td><td>UNPACK</td></tr>
</table>

NAME

PACK – Compresses stored data

SYNOPSIS

CALL PACK(*p,nbits,u,nw*)

DESCRIPTION

p	On exit, vector of packed data
nbits	Number of rightmost bits of data in each partial word; must be 1, 2, 4, 8, 16, or 32.
u	Vector of partial words to be compressed
nw	Number of partial words to be compressed

PACK takes the 1, 2, 4, 8, 16, or 32 rightmost bits of several partial words and concatenates them into full 64-bit words. The following equation gives the number of full words:

$$n = \frac{(nw \cdot nbits)}{64}$$

n	Number of resulting full words
nw	Number of partial words
nbits	Number of rightmost bits of each partial word that contain useful data

This equation restricts $nw \cdot nbits$ to a multiple of 64.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

UNPACK

NAME

P32, U32 – Packs/unpacks 32-bit words into or from Cray 64-bit words

SYNOPSIS

CALL P32(*src,dest,num***)**

CALL U32(*src,dest,num***)**

DESCRIPTION

src For **P32**, a variable or array of any type or length containing 32-bit words, left-justified in a Cray 64-bit word. For **U32**, a variable or array of any type or length containing 32-bit words as a continuous stream of data. Unpacking always starts with the leftmost bit of *src*.

dest For **P32**, a destination array of any type to contain the packed 32-bit words as a continuous stream of data. For **U32**, a destination array of any type to contain the unpacked 32-bit words, left-justified and zero-filled in a Cray 64-bit word.

num Number of 32-bit words to pack or unpack. Reads this many elements of *src* or generates this many elements of *dest*. Specify an integer variable, expression, or constant.

P32 packs 32-bit words into Cray 64-bit words. **U32** unpacks 32-bit words from Cray 64-bit words.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

P6460, U6064 – Packs/unpacks 60-bit words into or from Cray 64-bit words

SYNOPSIS

CALL P6460(*src,dest,isb,num***)**

CALL U6064(*src,isb,dest,num***)**

DESCRIPTION

src Variable or array of any type or length containing 60-bit words, left-justified in a Cray 64-bit word (for **U6064**, words are contained as a continuous stream of data)

dest For **P6460**, a destination array of any type to contain the packed 60-bit words as a continuous stream of data. For **U6064**, a destination array of any type to contain the unpacked 60-bit words, left-justified and zero-filled in a Cray 64-bit word.

isb Bit location that is the leftmost storage location for the 60-bit words. Bit position is counted from the left to right, with the leftmost bit 0. Specify an integer variable, expression, or constant.

num Number of 60-bit words to pack or unpack. Reads this many elements of *src* or generates this many elements of *dest*. Specify an integer variable, expression, or constant.

P6460 packs 60-bit words into Cray 64-bit words. **U6064** unpacks 60-bit words from Cray 64-bit words. Parameter arguments must be addressed in the same order in which they appear in the synopsis above.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

UNPACK – Expands stored data

SYNOPSIS

CALL UNPACK(*p,nbits,u,nw*)

DESCRIPTION

p	Vector of full 64-bit words to be expanded
nbits	Number of rightmost bits of data in each partial word; must be 1, 2, 4, 8, 16, or 32.
u	On exit, vector of unpacked data
nw	Number of resulting partial words

UNPACK reverses the action of **PACK** and expands full words of data into a larger number of right-justified partial words. This routine assumes *nw* * *nbits* to be a multiple of 64.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

PACK

10. BYTE AND BIT MANIPULATION ROUTINES

Byte and bit manipulation routines move bytes and bits between variables and arrays, compare bytes, perform searches with a byte count as a search argument, and perform conversion on bytes.

The following table contains the purpose, name, and entry of each byte and bit manipulation routine.

<table>
<tr><th colspan="3">Byte and Bit Manipulation Routines</th></tr>
<tr><th>Purpose</th><th>Name</th><th>Entry</th></tr>
<tr><td>Replace a byte in a variable or an array with a specified value</td><td>PUTBYT</td><td rowspan="2">BYT</td></tr>
<tr><td>Extract a byte from a variable</td><td>IGTBYT</td></tr>
<tr><td>Search a variable or an array for an occurrence of a character string</td><td>FINDCH</td><td>FINDCH</td></tr>
<tr><td>Compare bytes between variables or arrays</td><td>KOMSTR</td><td>KOMSTR</td></tr>
<tr><td>Move bytes between variables or arrays</td><td>STRMOV</td><td rowspan="2">MOV</td></tr>
<tr><td>Move bits between variables or arrays</td><td>MOVBIT</td></tr>
<tr><td>Move characters between memory areas</td><td>MVC</td><td>MVC</td></tr>
</table>

NAME

PUTBYT, IGTBYT – Replaces a byte in a variable or an array

SYNOPSIS

value=**PUTBYT**(*string*,*position*,*value*)

byte=**IGTBYT**(*string*,*position*)

DESCRIPTION

string The address of a variable or an array. The variable or array may be of any type except character.

position The number of the byte to be replaced or extracted. This parameter must be an integer ≥ 1. If *position* is ≤ 0, no change is made to the destination string; *value* returned is -1. For **IGTBYT**, if *position* is ≥ 0, *value* is an integer between 0 and 255.

value The new value to be stored into the byte. This parameter must be an integer with a value between 0 and 255.

PUTBYT replaces a specified byte in a variable or an array with a specified value. **IGTBYT** extracts a specified byte from a variable or an array.

If **PUTBYT** is called as an integer function (having been properly declared in the user program), the value of the function is the value of the byte stored.

The high-order 8 bits of the first word of the variable or array are called byte 1.

The value of the byte returned by **IGTBYT** is an integer value between 0 and 255.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

FINDCH – Searches a variable or an array for an occurrence of a character string

SYNOPSIS

CALL FINDCH(*chrs,len,str,ls,nb,ifnd***)**

DESCRIPTION

chrs	Variable or array of any type or length containing the search string
len	Length of the search string in bytes (must be from 1 to 256). Specify an integer variable, expression, or constant.
str	Variable or array of any type or length that is searched for a match with *chrs*
ls	Starting byte in the *str* string. Specify an integer variable, expression, or constant. Bytes are numbered from 1, beginning at the leftmost byte position of *str*.
nb	Number of bytes to be searched. Specify an integer variable, expression, or constant.
ifnd	Type integer result

The result of this subroutine search is equal to the 1-based byte index into the variable or array where the matching string was found, or equal to 0 if no matching string was found.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

KOMSTR – Compares specified bytes between variables or arrays

SYNOPSIS

result=**KOMSTR**(*str1,byte1,num,str2,byte2*)

DESCRIPTION

result Type integer result indicating results of the comparison:
= 0 *str1* = *str2*
= 1 *str1* > *str2*
=-1 *str1* < *str2*

str1 Variable or array of any type or length containing the byte string to compare against the byte string in *str2*

byte1 Starting byte of *str1*. Specify an integer variable, expression, or constant greater than 0. In a Cray word, bytes are numbered from 1 to 8, from the leftmost byte to the rightmost byte.

num An integer variable, expression, or constant that contains the number of bytes to compare; must be greater than 0.

str2 Variable or array of any type or length containing the byte string to compare against the byte string in *str1*

byte2 Starting byte of *str2*. Specify an integer variable, expression, or constant greater than 0. In a Cray word, bytes are numbered from 1 to 8, from the leftmost byte to the rightmost byte.

KOMSTR performs an unsigned, twos complement compare of a specified number of bytes from one variable or array with a specified number of bytes from another variable or array.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

STRMOV, MOVBIT – Moves bytes or bits from one variable or array to another

SNYOPSIS

CALL STRMOV(*src,isb,num,dest,idb***)**

CALL MOVBIT(*src,isb,num,dest,idb***)**

DESCRIPTION

src Variable or array of any type or length containing the bytes or string of bits to be moved. Bytes are numbered from 1, beginning at the leftmost byte position of *src*.

isb Starting byte or bit in the *src* string. Specify an integer variable, expression, or constant greater than 0. Bytes and bits are numbered from 1, beginning at the leftmost byte or bit position of *src*.

num An integer variable, expression, or constant that contains the number of bytes or bits to be moved; must be greater than 0.

dest Variable or array of any type or length that contains the starting byte or bit to receive the data. Bytes and bits are numbered from 1, beginning at the leftmost byte or bit position of *dest*.

idb An integer variable, expression, or constant that contains the starting byte or bit to receive the data; must be greater than 0. Bytes and bits are numbered from 1, beginning at the left-most byte or bit position of *dest*.

STRMOV moves bytes from one variable or array to another. **MOVBIT** moves bits from one variable or array to another.

CAUTION

The argument *dest* must be declared long enough to hold *num* bytes, or a spill occurs and data is destroyed.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

MVC – Moves characters from one memory area to another

SYNOPSIS

CALL MVC(s_1,j_1,s_2,j_2,k)

DESCRIPTION

s_1	Word address of the source string
j_1	Byte offset from the source string word address of the first byte of the source string (the high-order byte of the first word of the source string is byte 1)
s_2	Word address of the destination string
j_2	Byte offset from the destination string word address of the first byte of the destination string (the high-order byte of the first word of the destination string is byte 1)
k	Number of bytes to be moved

Source and destination strings can occur on any byte boundary. The move is performed 1 character at a time from left to right. The destination string can overlap the source string.

EXAMPLE

To copy the first byte of an array throughout the array, invoke the routine as follows:

```
CALL MVC(ARRAY,1,ARRAY,2,K-1)
```

where K is the length of the array in bytes.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

TRIMLEN – Returns the number of characters in a string

SYNOPSIS

INTEGER TRIMLEN
num = **TRIMLEN**(*string*)

DESCRIPTION

num An integer variable giving the number of characters, excluding trailing blanks, in *string*

string A string variable

This function is intended for use with **WRITE** statements or with the concatenation operator. If you use it on the right-hand side of an assignment statement, any trailing blanks are put back as they were.

EXAMPLE

The following are examples of typical use:

```
      WRITE(6,901) STRING(1:TRIMLEN(STRING))
901   FORMAT(' The string is >',A,'<')
```

This example writes the string with the < character against the last nonblank character in string **A**.

```
      NEW = STRING(1:TRIMLEN(STRING)) // '<The end'
```

In this example, the < is again butted up against the last significant character in **STRING** even though **STRING** may have trailing blanks.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

11. HEAP MANAGEMENT AND TABLE MANAGEMENT ROUTINES

These routines allow you to manage a block of memory (the heap) within your job area and to manipulate tables.

The management routines are divided into two categories: heap management and table management. Corresponding CAL routines are found in the in the System Library Reference Manual, publication SM-0114.

IMPLEMENTATION

The heap management and table management routines are available to users of both the COS and UNICOS operating systems.

HEAP MANAGEMENT ROUTINES

Heap management routines provide dynamic storage allocations by managing a block of memory, called the heap, within your job area. Each job has its own heap. The functions of the heap management routines include allocating a block of memory, returning a block of memory to the heap's list of available space, and changing the length of a block of memory. Heap managment routines may also move a heap block to a new location if there is no room to extend it, return part of the heap to the operating system, check the integrity of the heap, and report heap statistics. See the COS Reference Manual, publication SR-0011, and the Segment Loader (SEGLDR) Reference Manual, publication SR-0066, for the location of the heap and a description of the parameters on the **LDR** control statement or the SEGLDR directive that affect the heap.

The heap management routines keep various statistics on the use of the heap. These include values used to tune heap parameters specified on the **LDR** control statement or the SEGLDR directive and information used in debugging.

The following table contains the purpose, name, and entry of each heap management routine.

Heap Management Routines		
Purpose	Name	Entry
Allocate a block of memory from the heap	**HPALLOC**	**HPALLOC**
Check the integrity of the heap	**HPCHECK**	**HPCHECK**
Extend a block or copy block contents into a larger block	**HPCLMOVE**	**HPCLMOVE**
Return a block of memory to the heap	**HPDEALLC**	**HPDEALLC**
Dump the address and size of each heap block	**HPDUMP**	**HPDUMP**
Change the size of an allocated heap block	**HPNEWLEN**	**HPNEWLEN**
Return an unused portion of the heap to the operating system	**HPSHRINK**	**HPSHRINK**
Return the length of a heap block	**IHPLEN**	**IHPLEN**
Return statistics about the heap	**IHPSTAT**	**IHPSTAT**

TABLE MANAGEMENT ROUTINES

The following table contains the purpose, name, and entry of each Fortran-callable table management routine.

Table Management Routines		
Purpose	Name	Heading
Add a word to a table	**TMADW**	**TMADW**
Report table management operation statistics	**TMAMU**	**TMAMU**
Allocate table space	**TMATS**	**TMATS**
Request additional memory	**TMMEM**	**TMMEM**
Search the table with a mask to locate a field within an entry	**TMMSC**	**TMMSC**
Move memory	**TMMVE**	**TMMVE**
Preset table space	**TMPTS**	**TMPTS**
Search the table with or without a mask to locate a field within an entry and an offset	**TMSRC**	**TMSRC**
Search a vector table for the search argument	**TMVSC**	**TMVSC**

The Job Communication Block (JCB) field JCHLM (COS only) defines the beginning address of the table area.

You must provide two control information tables with corresponding CAL **ENTRY** pseudo-ops: the Table Base Address (BTAB) and Table Length Table (LTAB). Their formats are listed in the System Library Reference Manual, publication SM-0114. The Fortran-callable versions of these routines use default BTAB and LTAB definitions from a common area in the library.

TMINIT initializes the table descriptor vector, BTAB, and zeros all elements of the table length vector, LTAB. You must preset each element of BTAB to contain the desired interspace value for the corresponding table; for instance, *s1* in the following example determines the interspace value for table 1. Interspace values determine how many words are added to a table when more room is needed for that table or for any table with a lower number.

```
INTEGER BTAB(n), LTAB(n)
DATA BTAB /s1,s2,s3,...,sn/
 .
 .
 .
CALL TMINIT
```

After the call to **TMINIT**, BTAB should not be changed. The interspace values have been shifted 48 bits to the left, bits 16 through 39 contain the current size of each table, and the rightmost 24 bits contain the absolute address of each table's first word. LTAB is used only to pass new table lengths from the user to the Table Manager.

You can use statements such as the following to access each table. In this example, **TABLEi** is accessed.

```
EQUIVALENCE (BTAB(i), PTRi)
INTEGER PTRi, TABLEi (0:0)
POINTER (PTRi, TABLEi)
 .
 .
 .
TABLEi (subscript) = ...
```

TM COMMON BLOCK - The common block name **TM** is reserved for use by the Table Manager and must always contain 64 LTAB words.

```
COMMON /TM/ BTAB(64), LTAB(64)
```

ACCESSING TABLE MANAGER TABLES (ALTERNATE METHOD) - Blank common can be used in the customary way, but the last entry in it should be for a one-dimensional array declared to contain just 1 word. The name of this array is then used to access the tables, beginning immediately after the end of blank common.

```
COMMON // TABLES(1)
```

WARNING

Under COS, the heap management and table management subroutines cannot be used in the same application, unless the heap is of fixed size and placed before blank common. This restriction does not apply to UNICOS.

The following statement function extracts the rightmost 24 bits from a BTAB word and changes that value from an absolute address to a relative address or offset within the table area. Thus the result of **BASE(N)** is an index into **TABLES(1)**, pointing to the first word currently allocated to table **N**.

```
BASE(N) = (BTAB(N) .AND. 77777777B) - LOC(TABLES(1))

      WRITE(6,101) TABN
101   FORMAT ('0 Dump of table ',I2,/)
      OFFSET = 0
102   CONTINUE
         DO 103 I=1,4
            INTABLE = OFFSET .LT. LTAB(TABN)
            IF (INTABLE) THEN
            OCTAL(I) = TABLES(1+BASE(TABN) + OFFSET)
            ALPHA(I)=TABLES(1+BASE(TABN) + OFFSET)
            ELSE
            OCTAL(I) = 0
            ALPHA(I) = '        '
            ENDIF
            OFFSET = OFFSET+1
103      CONTINUE
      WRITE (6,104) OFFSET-4, OCTAL, ALPHA
104   FORMAT (I6,2X,4(022,1X),4A8)

      INTABLE = OFFSET .LT. LTAB(TABN)
      IF (INTABLE) GO TO 102
      WRITE (6,105)
105   FORMAT (/)
      RETURN
      END
```

NAME

HPALLOC – Allocates a block of memory from the heap

SYNOPSIS

CALL HPALLOC(*addr,length,errcode,abort***)**

DESCRIPTION

addr First word address of the allocated block (output)

length Number of words of memory requested (input)

errcode Error code. 0 if no error was detected; otherwise, a negative integer code for the type of error (output).

abort Abort code; nonzero requests abort on error; 0 requests an error code (input).

Allocate routines search the linked list of available space for a block greater than or equal to the size requested.

The length of an allocated block can be greater than the requested length because blocks smaller than the managed memory epsilon specified on the LDR control statement (or in a SEGLDR directive) are never left on the free space list.

Error conditions are as follows:

Error Code	Condition
-1	Length is not an integer greater than 0
-2	No more memory is available from the system (checked if the the request cannot be satisfied from the available blocks on the heap)

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

HPCHECK – Checks the integrity of the heap

SYNOPSIS

CALL HPCHECK(*errcode*)

DESCRIPTION

errcode Error code. 0 if no error was detected; otherwise, a negative integer code for the type of error (output).

Each control word is examined to ensure that it has not been overwritten.

Error conditions are as follows:

Error Code	Condition
-5	Bad control word for the allocated block
-6	Bad control word for the free block

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

HPCLMOVE – Extends a block or copies block contents into a larger block

SYNOPSIS

CALL HPCLMOVE(*addr,length,status,abort***)**

DESCRIPTION

addr On entry, first word address of the block to change; on exit, the new address of the block if it was moved.

length Requested new total length (input)

status Status. 0 if the block was extended in place; 1 if it was moved; a negative integer for the type of error detected (output).

abort Abort code. Nonzero requests abort on error; 0 requests an error code (input).

Change length and move routines extend a block if it is followed by a large enough free block or copy the contents of the existing block to a larger block and return a status code indicating that the block has been moved. These routines can also reduce the size of a block if the new length is less than the old length. In this case, they have the same effect as the change length routines.

The new length of the block can be greater than the requested length because blocks smaller than the managed memory epsilon specified on the **LDR** control statement are never left on the free space list.

Error conditions are as follows:

Error Code	Condition
-1	Length is not an integer greater than 0
-2	No more memory is available from the system (checked if the block cannot be extended and the free space list does not include a large enough block)
-3	Address is outside the bounds of the heap
-4	Block is already free
-5	Address is not at the beginning of the block
-7	Control word for the next block has been overwritten

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

HPDEALLC – Returns a block of memory to the list of available space (the heap)

SYNOPSIS

CALL HPDEALLC(*addr,errcode,abort***)**

DESCRIPTION

addr	First word address of the block to deallocate (input)
errcode	Error code. 0 if no error was detected; otherwise, a negative integer code for the type of error (output).
abort	Abort code. Nonzero requests abort on error; 0 requests an error code (input).

Error conditions are as follows:

Error Code	Condition
-3	Address is outside the bounds of the heap
-4	Block is already free
-5	Address is not at the beginning of the block
-7	Control word for the next block has been overwritten

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

HPDUMP – Dumps the address and size of each heap block

SYNOPSIS

CALL HPDUMP(*code,dsname***)**

DESCRIPTION

code Code for the type of dump requested, as follows:

Code	Meaning
0	Print heap statistics
1	Dump all heap blocks in storage order
2	Dump free blocks; follow **NEXT** links.
3	Dump free blocks; follow **PREV** links.

dsname Name of the dataset to which the dump is to be written. *dsname* must be in left-justified, Hollerith form.

Three types of dump are available: a dump of all heap blocks; a dump of free blocks that traces the links to the next block on the free list; and a dump of free blocks that traces the links to the previous block on the free list. The dump stops if a recognizably invalid value is found in a field needed to continue the dump.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

HPNEWLEN – Changes the size of an allocated heap block

SYNOPSIS

CALL HPNEWLEN(*addr,length,status,abort***)**

DESCRIPTION

addr First word address of the block to change (input)

length Requested new total length of the block (input)

status Status. 0 if the change in length was successful; 1 if the block could not be extended in place; a negative integer for the type of error detected (output).

abort Abort code. Nonzero requests abort on error; 0 requests an error code (input).

Set new length routines change the size of an allocated heap block. If the new length is less than the allocated length, the portion starting at ADDR+LENGTH is returned to the heap. If the new length is greater than the allocated length, the block is extended if it is followed by a free block. A status is returned, telling whether the change was successful.

The new length of the block can be greater than the requested length because blocks smaller than the managed memory epsilon specified on the **LDR** or **SEGLDR** control statement are never left on the free space list.

Error conditions are as follows:

Error Code	Condition
-1	Length is not an integer greater than 0
-3	Address is outside the bounds of the heap
-4	Block is already free
-5	Address is not at the beginning of the block
-7	Control word for the next block has been overwritten

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

HPSHRINK – Returns an unused portion of heap to the operating system

SYNOPSIS

CALL HPSHRINK

DESCRIPTION

The unused portion of the heap is returned to the operating system only if the blocks closest to **HLM** (COS only) are free; no allocated blocks are moved. The minimum amount of memory to be returned is the managed memory increment specified on the **LDR** or **SEGLDR** control statement. These routines are called only from the user program.

IMPLEMENTATION

This routine is available only to the users of the COS operating system.

NAME

IHPLEN – Returns the length of a heap block

SYNOPSIS

length=**IHPLEN** (*addr,errcode,abort*)

DESCRIPTION

length	Length of the block starting at *addr* (output)
addr	First word address of the block (input)
errcode	Error code. 0 if no error was detected; otherwise, a negative integer code for the type of error (output).
abort	Abort code. Nonzero requests abort on error; 0 requests an error code (input).

The length of the block can be greater than the amount requested because of the managed memory *epsilon.*

Error conditions are as follows:

Error Code	Condition
-3	Address is outside the bounds of the heap
-4	Block is already free
-5	Address is not at the beginning of the block
-7	Control word for the next block has been overwritten

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

IHPSTAT – Returns statistics about the heap

SYNOPSIS

value=**IHPSTAT**(*code*)

DESCRIPTION

value Requested information

code Code for the type of information requested, as follows:

Code	Meaning
1	Current heap length
2	Largest size of the heap so far
3	Smallest size of the heap so far
4	Number of allocated blocks
5	Number of times the heap has grown
6	Number of times the heap has shrunk
7	Last routine that changed the heap
8	Caller of the last routine that changed the heap
9	First word address of the heap area changed last
10	Size of the largest free block
11	Amount by which the heap can shrink
12	Amount by which the heap can grow
13	First word address of the heap
14	Last word address of the heap

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

TMADW – Adds a word to a table

SYNOPSIS

index=**TMADW**(*number*,*entry*)

DESCRIPTION

index	Index of the added word
number	Table number
entry	Entry for the table

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

NAME

TMAMU – Reports table management operation statistics

SYNOPSIS

CALL TMAMU(*len,tabnum,tabmov,tabmar,nword*)

DESCRIPTION

len	Allocated length of the table
tabnum	Number of tables used
tabmov	Number of table moves
tabmar	Maximum amount of memory used throughout the Table Manager
nword	Number of words moved

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

NAME

TMATS – Allocates table space

SYNOPSIS

index=TMATS(*number*,*incre*)

DESCRIPTION

index	Index of the specified change
number	Table number
incre	Table increment

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

NAME

TMMEM – Requests additional memory

SYNOPSIS

CALL TMMEM(*mem*)

DESCRIPTION

mem Length of memory requested

Upon exit, memory is extended by the requested amount. No value is returned.

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

NAME

TMMSC – Searches the table with a mask to locate a specific field within an entry

SYNOPSIS

index=TMMSC(*tabnum*,*mask*,*sword*,*nword*)

DESCRIPTION

index	Table index of the match, if found; -1 if no match is found.
tabnum	Table number
mask	Mask defining a field within a word
sword	Search word
nword	Number of words per entry group

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

NAME

TMMVE – Moves memory (words)

SYNOPSIS

CALL TMMVE(*from,to,count*)

DESCRIPTION

from	Address from which words are to be moved
to	Address of the location to which words are to be moved
count	Number of words to be moved

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

NAME

TMPTS – Presets table space

SYNOPSIS

CALL TMPTS(*start*,*len*,*preset*)

DESCRIPTION

start	Starting address
len	Length to preset
preset	Preset value; default is 0.

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

NAME

TMSRC – Searches the table with an optional mask to locate a specific field within an entry and an offset

SYNOPSIS

index=**TMSRC**(*tabnum*,*arg*,*nword*,*offset*,*mask*)

DESCRIPTION

index	Table index of the match, if a match is found; -1 if no match is found.
tabnum	Table number to search
arg	Search argument or key
nword	Number of words per entry
offset	Offset into the entry group
mask	Field being searched for within an entry

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

NAME

TMVSC – Searches a vector table for the search argument

SYNOPSIS

index=TMVSC(*tabnum*,*arg*,*nword*)

DESCRIPTION

index	Table index of the match, if found; -1 if no match is found.
tabnum	Table number
arg	Search argument
nword	Number of words per entry group

IMPLEMENTATION

This routine is available to the users of both the COS and UNICOS operating systems.

12. I/O ROUTINES

The I/O routines include the following:

- Dataset positioning routines
- Auxiliary NAMELIST routines
- Logical record I/O routines
- Random access dataset I/O routines
- Asynchronous queued I/O routines
- Output suppression routines
- Fortran-callable tape routines involving beginning- and end-of-volume processing

DATASET POSITIONING ROUTINES

Dataset positioning routines change or indicate the position of the current dataset. These routines set the current positioning direction to input (read). If the previous processing direction is output (write), end-of-data is written on a blocked sequential dataset, and the buffer is flushed. On a random dataset, the buffer is flushed.

The following table contains the name, purpose, and entry of each dataset positioning routine.

<table>
<tr><th colspan="3">Dataset Positioning Routines</th></tr>
<tr><th>Purpose</th><th>Name</th><th>Entry</th></tr>
<tr><td>Receive position information about an opened tape dataset</td><td>GETTP</td><td>GETTP</td></tr>
<tr><td>Position a specified tape dataset at a tape block</td><td>SETTP</td><td>SETTP</td></tr>
<tr><td>Synchronize the specified program and an opened tape dataset</td><td>SYNCH</td><td>SYNCH</td></tr>
<tr><td>Return current position of an interchange tape or mass storage dataset</td><td>GETPOS</td><td rowspan="2">GETPOS</td></tr>
<tr><td>Return to the position retained from the GETPOS request</td><td>SETPOS</td></tr>
</table>

AUXILIARY NAMELIST ROUTINES

NAMELIST routines allow you to control input and output defaults and are accessed by call-by-address subprogram linkage. No arguments are returned. For a more complete description of the NAMELIST feature, see the Fortran (CFT) Reference Manual, publication SR-0009 or the CFT77 Reference Manual, publication SR-0018.

IMPLEMENTATION - The auxiliary NAMELIST routines are available to users of both the COS and UNICOS operating systems.

The following table contains the purpose, name, and entry of each auxiliary NAMELIST routine.

Auxiliary NAMELIST Routines

Purpose	Name	Entry
Delete or add a trailing comment indicator	**RNLCOMM**	**RNL**
Delete or add a delimiting character	**RNLDELM**	
Delete or add an echo character	**RNLFLAG**	
Delete or add a replacement character	**RNLREP**	
Delete or add a separator character	**RNLSEP**	
Specify the output unit for error messages and echo lines	**RNLECHO**	**RNLECHO**
Take action when an undesired NAMELIST group is encountered	**RNLSKIP**	**RNLSKIP**
Determine the action if a type mismatch occurs across the equal sign on an input record	**RNLTYPE**	**RNLTYPE**
Define an ASCII NAMELIST delimiter	**WNLDELM**	**WNL**
Indicate the first ASCII character of the first line	**WNLFLAG**	
Define ASCII NAMELIST replacement character	**WNLREP**	
Define ASCII NAMELIST separator	**WNLSEP**	
Allow each NAMELIST variable to begin on a new line	**WNLLINE**	**WNLLINE**
Indicate output line length	**WNLLONG**	**WNLLONG**

LOGICAL RECORD I/O ROUTINES

The logical record I/O routines are divided into read routines, write routines, and bad data error recovery routines. The following table contains the purpose, name, and entry of each logical record I/O routine.

Logical Record I/O Routines		
Purpose	Name	Entry
Read words, full record mode	**READ**	**READ**
Read words, partial record mode	**READP**	
Read characters, full record mode	**READC**	**READC**
Read characters, partial record mode	**READCP**	
Read two IBM 32-bit floating-point words from each Cray 64-bit word	**READIBM**	**READIBM**
Write words, full record mode	**WRITE**	**WRITE**
Write words, partial record mode	**WRITEP**	
Write characters, full record mode	**WRITEC**	**WRITEC**
Write characters, partial record mode	**WRITECP**	
Write two IBM 32-bit floating-point words from each Cray 64-bit word	**WRITIBM**	**WRITIBM**
Skip bad data	**SKIPBAD**	**SKIPBAD**
Make bad data available	**ACPTBAD**	**ACPTBAD**

READ ROUTINES - Read routines transfer partial or full records of data from the I/O buffer to the user data area. Depending on the read request issued, the data is placed in the user data area either 1 character per word or in full words. (Blank decompression occurs only when data is being read 1 character per word.) In partial mode, the dataset maintains its position after the read is executed. In record mode, the dataset position is maintained after the end-of-record (EOR) that terminates the current record.

WRITE ROUTINES - Write routines transfer partial or full records of data from the user data area to the I/O buffer. Depending on the write operation requested, data either is taken from the user data area 1 character per word and packed 8 characters per word or is transferred in full words. In partial mode, no end-of-record (EOR) is inserted in the I/O buffer in the word following the data that terminates the record.

BAD DATA ERROR RECOVERY ROUTINES - Bad data error recovery routines enable a user program to continue processing a dataset when bad data is encountered. "Bad data" refers to an unrecovered error encountered while the dataset was being read. Skipping the data forces the dataset to a position past the bad data, so that no data is transferred to the user-specified buffer. Accepting the data causes the bad data to be transferred to a user-specified buffer. The dataset is then positioned immediately following the bad data.

When an unrecovered data error is encountered, continue processing by calling either the **SKIPBAD** or the **ACPTBAD** routine.

RANDOM ACCESS DATASET I/O ROUTINES

Sequentially accessed datasets are used for applications that read input only once during a process and write output only once during a process. However, when large numbers of intermediate results are used randomly as input at different stages of jobs, a random access dataset capability is more efficient than sequential access. A random access dataset consists of records that are accessed and changed. Random access of data eliminates the slow processing and inconvenience of sequential access when the order of reading and writing records differs in various applications.

Random access dataset I/O routines allow you to specify how records of a dataset are to be changed, without the usual limitations of sequential access. Choose specific routines based on performance requirements and the type of access needed.

Random access datasets can be created and accessed by the record-addressable, random access dataset routines (**READMS/WRITMS**, and **READDR/WRITDR**) or the word-addressable, random access dataset routines (**GETWA/PUTWA**).

NOTE - Generally, random access dataset I/O routines used in a program with overlays or segments should reside in the first overlay or root segment. However, if all I/O is done within one overlay or segment, the routines can reside in that overlay. If all I/O is done in an overlay's successor, the routines can reside in the successor overlay.

IMPLEMENTATION - The random access dataset I/O routines are available to users of both the COS and UNICOS operating systems.

RECORD-ADDRESSABLE, RANDOM ACCESS DATASET I/O ROUTINES - Record-addressable, random access dataset I/O routines allow you to generate datasets containing variable-length, individually addressable records. These records can be read and rewritten at your discretion. The library routines update indexes and pointers. The random access dataset information is stored in two places: in an array in user memory and at the end of the random access dataset.

When a random access dataset is opened, an array in user memory contains the master index to the records of the dataset. This master index contains the pointers to and, optionally, the names of the records within the dataset. Although you provide this storage area, it must be modified only by the random access dataset I/O routines.

When a random access dataset is closed and optionally saved, the storage area containing the master index is mapped to the end of the random access dataset, thus recording changes to the contents of the dataset.

The following Fortran-callable routines can change or access a record-addressable, random access dataset: **OPENMS**, **WRITMS**, **READMS**, **CLOSMS**, **FINDMS**, **CHECKMS**, **WAITMS**, **ASYNCMS**, **SYNCMS**, **OPENDR**, **WRITDR**, **READDR**, **CLOSDR**, **STINDR**, **CHECKDR**, **WAITDR**, **ASYNCDR**, **SYNCDR**, and **STINDX**.

The **READDR/WRITDR** random access I/O routines are direct-to-disk versions of **READMS/WRITMS**. All input or output goes directly between the user data area and the mass storage dataset without passing through a system-maintained buffer. Because mass storage can only be addressed in 512-word blocks, all record lengths are rounded up to the next multiple of 512 words.

You can intermix **READMS/WRITMS** and **READDR/WRITDR** datasets in the same program, but you must not use the same file in both packages simultaneously.

OPENMS/OPENDR opens a local dataset and specifies the dataset as a random access dataset that can be accessed or changed by the record-addressable, random access dataset I/O routines. If the dataset does not exist, the master index contains zeros; if the dataset does exist, the master index is read from the dataset. The master index contains the current index to the dataset. The current index is updated when the dataset is closed using **CLOSMS/CLOSDR**.

A single job can use up to 40 active **READMS/WRITMS** files and 20 **READDR/WRITDR** files.

The following table contains the name, purpose, and entry of each record-addressable, random access dataset I/O routine.

Record-addressable, Random Access Dataset I/O Routines		
Purpose	Name	Entry
Set the I/O mode to be asynchronous	**ASYNCMS** **ASYNCDR**	**ASYNCMS**
Check the status of an asynchronous I/O operation	**CHECKMS** **CHECKDR**	**CHECKMS**
Close a random access dataset and write the master index	**CLOSMS** **CLOSDR**	**CLOSMS**
Read records into data buffers used by random access dataset routines	**FINDMS**	**FINDMS**
Open a local dataset as a random access dataset	**OPENMS** **OPENDR**	**OPENMS**
Allow an index to be used as the current index by creating a subindex	**STINDX** **STINDR**	**STINDX**
Set the I/O mode to be synchronous	**SYNCMS** **SYNCDR**	**SYNCMS**
Wait for completion of an asynchronous I/O operation	**WAITMS** **WAITDR**	**WAITMS**
Write data from user memory to a random access dataset and update the index	**WRITMS** **WRITDR**	**WRITMS**

WORD-ADDRESSABLE, RANDOM ACCESS DATASET I/O ROUTINES - A word-addressable, random access dataset consists of an adjustable number of contiguous words. You can access any word or contiguous sequence of words from a word-addressable, random access dataset by using the associated routines. These datasets and their I/O routines are similar to the record-addressable, random access datasets and their routines. The Fortran-callable, word-addressable random access I/O routines are **WOPEN**, **WCLOSE**, **PUTWA**, **APUTWA**, **GETWA**, and **SEEK**. **WOPEN** opens a dataset and specifies it as a word-addressable, random access dataset that can be accessed or changed with the word-addressable routines. The **WOPEN** call is optional. If a call to **GETWA** or **PUTWA** is executed first, the dataset is opened for you with the default number of blocks (16), and *istats* is turned on.

The following table contains the purpose, name, and entry of each word-addressable, random access dataset I/O routine.

Word-addressable, Random Access Dataset I/O Routines		
Purpose	Name	Entry
Synchronously read words from the dataset into user memory	**GETWA**	**GETWA**
Asynchronously read data into dataset buffers	**SEEK**	
Synchronously write words from memory to the dataset	**PUTWA**	**PUTWA**
Asynchronously write words from memory to the dataset	**APUTWA**	
Finalize additions and changes and close the dataset	**WCLOSE**	**WCLOSE**
Open a dataset and specify it as word-addressable, random access	**WOPEN**	**WOPEN**

ASYNCHRONOUS QUEUED I/O ROUTINES

Asynchronous queued I/O routines initiate a transfer of data and allow the subsequent execution sequence to proceed concurrently with the actual transfer.

The following table contains the purpose, name, and entry of each asynchronous queued I/O routine.

<table>
<tr><th colspan="3">Asynchronous Queued I/O Routines</th></tr>
<tr><th>Purpose</th><th>Name</th><th>Entry</th></tr>
<tr><td>Close an asynchronous queued I/O dataset or file</td><td>AQCLOSE</td><td>AQCLOSE</td></tr>
<tr><td>Open a dataset or file for asynchronous queued I/O</td><td>AQOPEN</td><td>AQOPEN</td></tr>
<tr><td>Queue a simple asynchronous I/O read request</td><td>AQREAD</td><td rowspan="4">AQREAD</td></tr>
<tr><td>Queue a compound asynchronous I/O read request</td><td>AQREADC</td></tr>
<tr><td>Queue a compound read request with the ignore bit set</td><td>AQREADCI</td></tr>
<tr><td>Queue a simple read request with the ignore bit set</td><td>AQREADI</td></tr>
<tr><td>Prevent a segment of I/O and part of the program from executing concurrently (used with AQRIR)</td><td>AQRECALL</td><td rowspan="2">AQRECALL</td></tr>
<tr><td>Designate point in I/O at which concurrent processing can resume (used with AQRECALL)</td><td>AQRIR</td></tr>
<tr><td>Check the status of asynchronous queued I/O requests</td><td>AQSTAT</td><td>AQSTAT</td></tr>
<tr><td>Queue a stop request in the asyncronous queued I/O buffer</td><td>AQSTOP</td><td>AQSTOP</td></tr>
<tr><td>Queue a synchronization request in the asynchronous queued I/O buffer</td><td>AQSYNC</td><td>AQSYNC</td></tr>
<tr><td>Wait for completion of asynchronous queued I/O requests</td><td>AQWAIT</td><td>AQWAIT</td></tr>
<tr><td>Queue a simple asynchronous I/O write request</td><td>AQWRITE</td><td rowspan="4">AQWRITE</td></tr>
<tr><td>Queue a compound asynchronous I/O write request</td><td>AQWRITEC</td></tr>
<tr><td>Queue a compound write request with bit set</td><td>AQWRITEC</td></tr>
<tr><td>Queue a write request with the ignore bit set</td><td>AQWRITEI</td></tr>
</table>

OUTPUT SUPPRESSION ROUTINES

Output suppression routines are special-purpose routines designed to output blank values in Fortran programs.

FSUP and **FSUPC** turn suppression on and off for the following Fortran edit descriptors: F-type, G-type, and E-type.

ISUP and **ISUPC** turn suppression on and off for the Fortran edit descriptor I-type.

All of these routines are described under the **FSUP** entry.

BOV/EOV FORTRAN-CALLABLE ROUTINES

Fortran-callable routines are designed to perform special functions on a tape dataset, such as beginning-of-volume (BOV) and end-of-volume (EOV) processing.

The following tables contain the purpose, name, and entry of each BOV/EOV Fortran-callable routine. Cray Research highly recommends using the first set of routines, **STARTSP, SETSP, CLOSEV,** and **ENDSP.**

BOV/EOV Fortran-callable Routines (New Routines)		
Purpose	Name	Entry
Switch tape volumes	**CLOSEV**	**CLOSEV**
End special EOV/BOV processing	**ENDSP**	**ENDSP**
Request notification at end of tape volume	**SETSP**	**SETSP**
Begin tape BOV/EOV processing	**STARTSP**	**STARTSP**

BOV/EOV Fortran-callable Routines (Obsolete Routines)		
Purpose	Name	Entry
Check tape I/O status	**CHECKTP**	**CHECKTP**
Continue normal I/O operation	**CONTPIO**	**CONTPIO**
Begin special processing at BOV	**PROCBOV**	**PROCBOV**
Begin special processing at EOV	**PROCEOV**	**PROCEOV**
Switch tape volume	**SWITCHV**	**SWITCHV**
Initialize/terminate special BOV/EOV processing	**SVOLPRC**	**SVOLPRC**

NAME

ACPTBAD – Makes bad data available

SYNOPSIS

CALL ACPTBAD(*dn,uda,wrdcnt,termcnd,ubcnt*)

DESCRIPTION

dn Dataset name or unit number

uda User data area to receive the bad data

wrdcnt On exit, number of words transferred

termcnd On exit, address of termination condition
- =0 Positioned at end-of-block
- =1 Positioned at end-of-file
- =2 Positioned at end-of-data
- <0 Not positioned at end-of-block

ubcnt On exit, address of unused bit count. Only defined if *termcnd* is 0, and *wrdcnt* is nonzero.

ACPTBAD makes bad data available to you by transferring it to the user-specified buffer. UNICOS does not support bad data recovery on transparent format tapes.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

EXAMPLE

```
C
      PROGRAM EXAMPLE1
      IMPLICIT INTEGER(A-Z)
      REAL UNIT, UNITSTAT
      PARAMETER(NBYTES=400000,NDIM=NBYTES/8,DN=99)
      DIMENSION BUFFER(1:NDIM)
      DIMENSION UDA(1:512)

 2000 CONTINUE

      NWORDS = NDIM
      CALL READ(DN,BUFFER,NWORDS,STATUS)

      UNITSTAT = UNIT(DN)

      IF(STATUS.EQ.4 .OR. UNITSTAT.GT.0.0) THEN !Parity error
 3000   CONTINUE

      CALL ACPTBAD(DN,UDA,WC,TERMCND,UBCNT)

C---->Build up user record:
      IX = 0
      DO 3500 I=(NWORDS + 1), (NWORDS + WC), 1
        IX = IX + 1
        BUFFER(I) = UDA(IX)
 3500   CONTINUE
```

ACPTBAD (3IO)

```
      IF(TERMCND.LT.0) THEN
          GO TO 3000
        ENDIF
      ENDIF

      STOP 'COMPLETE'
      END
```

SEE ALSO

SKIPBAD

NAME

AQCLOSE – Closes an asynchronous queued I/O dataset or file

SYNOPSIS

CALL AQCLOSE(*aqp,status*)

DESCRIPTION

aqp Type INTEGER array. The name of the array in the user's program that contains the asynchronous queued I/O parameter block. This must be the same array specified in the **AQOPEN** request.

status Type INTEGER variable. Status code; *status* returns any errors or status information to the user. On output from **AQCLOSE**, *status* has one of the following values:
>0 Information only
=0 No error detected
<0 Error detected

Status Codes	
0	No errors detected
+1	The asynchronous queued I/O parameter block is full
+2	No I/O is active on the asynchronous queued I/O dataset or file
+3	Asynchronous queued I/O request is stuck
+4	The asynchronous request is queued for I/O
-1	Illegal *aqpsize* on the **AQOPEN** request. Minimum size is equal to 32 + 8*n*, where *n* = 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

AQOPEN, AQREAD, AQREADC, AQSTAT, AQWAIT, AQWRITE, AQWRITEC
The AQIO User's Guide, publication SN-0247

NAME

AQOPEN – Opens a dataset or file for asynchronous queued I/O

SYNOPSIS

CALL AQOPEN(*aqp*,*aqpsize*,*dn*,*status*)

DESCRIPTION

aqp Type **INTEGER** array. The name of the array in the user's program that will contain the asynchronous queued I/O.

aqpsize Type **INTEGER** variable, expression, or constant. The length of the asynchronous queued I/O parameter block. Each queued I/O entry in the parameter block is 8 words long. The array *aqp* must contain at least 1 entry plus 32 words for dataset definitions. Therefore, *aqpsize* should be 32 + 8*n*; *n* is the number of user-specified asynchronous queued I/O entries in the parameter block.

dn Type **INTEGER** variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset.

status Type **INTEGER** variable. Status code *status* returns any errors or status information to the user. On output from **AQOPEN**, *status* has one of the following values:

>0 Information only
=0 No errors detected
<0 Error detected

Status Codes	
0	No errors detected
+1	The asynchronous queued I/O parameter block is full
+2	No I/O is active on the asynchronous queued I/O dataset or file
+3	The asynchronous queued I/O request is stuck
+4	The asynchronous request is queued for I/O
-1	Illegal *aqpsize* on the **AQOPEN** request. Minimum size is equal to 32 + 8*n*, where *n* = 1.

Asynchronous queued I/O provides a method of random access to or from mass storage into buffers in user memory.

NOTES

A file opened using **AQOPEN** should only be closed by **AQCLOSE** or, under COS, by job step advance. If you close the file in some other way, the subsequent behavior of the program is unpredictable. Among these other ways are explicit methods of closing the file (for example, **CLOS** and **CALL RELEASE**) and implicit methods (such as **CALL SAVE**).

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

AQREAD, AQREADC, AQWRITE, AQWRITEC, AQCLOSE, AQWAIT, AQSTAT
The AQIO User's Guide, SN-0247

NAME

AQREAD, AQREADC, AQREADI, ACREADCI – Queues a simple or compound asynchronous I/O read request

SYNOPSIS

CALL AQREAD(*aqp,cpuadd,blknum,blocks,reqid,queue,status***)**

CALL AQREADC(*aqp,cpuadd,mstride,blknum,blocks,dstride,incs,reqid,queue,status***)**

CALL AQREADI(*aqp,cpuadd,blknum,blocks,reqid,queue,status***)**

CALL AQREADCI(*aqp,cpuadd,mstride,blknum,blocks,dstride,incs,reqid,queue,status***)**

DESCRIPTION

aqp Type INTEGER array. The name of the array in the user's program that contains the asynchronous queued I/O parameter block. Must be the same array as specified in the **AQOPEN** request.

cpuadd Type determined by user. Starting memory address; the location where the first word of data is placed.

mstride Type INTEGER variable, expression, or constant. Data buffer stride; the number of memory words to skip between the base addresses of consecutive transfers. The stride value may be positive (to skip forward), negative (to skip backward), or 0. This parameter is valid for compound read requests only.

blknum Type INTEGER variable, expression, or constant. Starting block number. The block number of the first block to be read on this request.

blocks Type INTEGER variable, expression, or constant. The number of 512-word blocks to be read.

dstride Type INTEGER variable, expression, or constant. Disk stride; the number of disk blocks to skip between the base addresses of consecutive transfers. The stride value may be positive (to skip forward), negative (to skip backward), or 0. This parameter is valid for compound requests only.

incs Type INTEGER variable, expression, or constant. The number of simple requests minus 1 that comprise a compound request. Zero (0) implies a simple request. This parameter is valid for compound requests only.

reqid Type INTEGER variable, expression, or constant. A user-supplied value for identifying a particular request.

queue Type INTEGER variable, expression, or constant. Queue flag. If 0, I/O is initiated provided that I/O on the dataset or file is not already active. If the queue flag is set to nonzero, the request is added to the queue but no attempt is made to start I/O.

status Type INTEGER variable. Status code *status* returns any errors to the user. On output from these routines, *status* has one of the following values:

>0 Information only
=0 No error detected
<0 Error detected

Status Codes	
0	No errors detected
+1	The asynchronous queued I/O parameter block is full
+2	No I/O is active on the asynchronous queued I/O dataset or file
+3	The asynchronous queued I/O request is stuck
+4	The asynchronous request is queued for I/O
-1	Illegal *apqsize* on the **AQOPEN** request. Minimum size is equal to 32 + 8*n*, where *n* = 1.

AQREAD, AQREADC, AQREADI, and **AQREADCI** transfer data between the data buffer and the device on which the dataset or file resides. Requests may be simple (**AQREAD** and **AQREADI**) or compound (**AQREADC** and **AQREADCI**). A simple request is one in which data from consecutive sectors on the disk is read into one buffer. A compound request is one in which a number of simple requests are separated by a constant number of sectors on disk, or a constant number of memory words for buffers, or both.

AQREADI and **AQREADCI** (both COS only) operate in the same fashion as **AQREAD** and **AQREADC**, respectively, except the ignore bit is set. The ignore bit tells the operating system not to change from write mode to process this read request. As an example, setting the ignore bit might be helpful on a system with two high-speed SSD channels. A series of **AQWRITE** calls followed by an **AQREADI** call would not force a wait by the operating system as would a normal read.

IMPLEMENTATION

AQREAD and **AQREADC** are available to users of both the COS and UNICOS operating systems. **AQREADI** and **AQREADCI** are available only under the COS operating system.

SEE ALSO

AQWRITE, AQWRITEC, AQCLOSE, AQWAIT, AQSTAT
The AQIO User's Guide, SN-0247

NAME

AQRECALL, AQRIR – Delays program execution during a queued I/O sequence

SYNOPSIS

CALL AQRECALL(*aqp,status*)

CALL AQRIR(*aqp,reqid,queue,status*)

DESCRIPTION

aqp Type **INTEGER** array. The name of the array in the user's program that will contain the asynchronous queued I/O.

reqid Type **INTEGER** variable, expression, or constant. A user-supplied value for identifying a particular request.

queue Type **INTEGER** variable, expression, or constant. Queue flag. If 0, I/O is initiated provided that I/O on the dataset is not already active. If the queue flag is set to nonzero, the request is added to the queue but no attempt is made to start I/O.

status Type **INTEGER** variable. Status code *status* returns any errors or status information to the user. On output from **AQOPEN**, *status* has one of the following values:

>0 Information only
=0 No errors detected
<0 Error detected

Status Codes	
0	No errors detected
+1	The asynchronous queued I/O parameter block is full
+3	The asynchronous queued I/O request is stuck

AQRECALL and **AQRIR** work together to let you suspend the execution of your program during part of an asynchronous queued I/O process. **AQRIR** marks the point in the I/O process up to which program execution is delayed, while **AQRECALL** marks the point in the program beyond which execution should not proceed until the specified I/O is complete.

EXAMPLE

```
      J = 1
      DO I = 1,10
     IF(I.EQ10) J = 0
     CALL AQREAD(AQP,A,IBLOCK,10,I,J,ISTAT)
     IBLOCK = IBLOCK + 10
1     CONTINUE
      CALL AQRIR(AQP,0 0,ISTAT1)
      J = 1
      DO 2 I = 11,30
     IF(I.EQ.30) J = 0
     CALL AQREAD(AQP,A,IBLOCK,10,I,J,ISTAT2)
     IBLOCK = IBLOCK + 10
2     CONTINUE
      CALL AQRECALL(AQP,ISTAT3)
```

In the above example, 10 asynchronous reads are queued up, followed by an **AQRIR**. Any code beyond the **AQRECALL** call does not execute until the **AQRIR** request is encountered in the queue. When it is encountered, execution beyond **AQRECALL** continues. The following illustrates the queue containing the **AQREAD** requests and the **AQRIR** request.

1	AQREAD
2	AQREAD
.	.
.	.
.	.
10	AQREAD
11	AQRIR

IMPLEMENTATION

These routines are available only to users of the COS operating system.

SEE ALSO

AQREAD, AQREADC, AQWRITE, AQWRITEC, AQCLOSE, AQWAIT, AQSTAT
The AQIO User's Guide, SN-0247

NAME

AQSTAT – Checks the status of asynchronous queued I/O requests

SYNOPSIS

CALL AQSTAT(*aqp,reply,reqid,status***)**

DESCRIPTION

aqp Type INTEGER array. The name of the array in the user's program that contains the asynchronous queued I/O parameter block. This must be the same array specified in the **AQOPEN** request.

reply Type INTEGER variable

reqid Type INTEGER variable, expression, or constant. If *reqid* is 0, **AQSTAT** returns the request ID of the next queued I/O request to be done. If *reqid* is nonzero, status information about the specified request ID will be returned.

status Type INTEGER variable. Status code, *status* returns any errors or status information to the user. On output from **AQSTAT**:

>0 Information only
=0 No errors detected
<0 Error detected

Status Codes	
0	No errors detected
+1	The asynchronous queued I/O parameter block is full
+2	No I/O is active on the asynchronous queued I/O dataset or file
+3	The asynchronous queued I/O request is stuck
+4	The asynchronous request is queued for I/O
-1	Illegal *aqpsize* on the **AQOPEN** request. Minimum size is equal to 32 + 8*n*, where *n* = 1.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

AQOPEN, AQREAD, AQREADC, AQWRITE, AQWRITEC, AQCLOSE, AQWAIT
The AQIO User's Guide, SN-0247

NAME

AQSTOP – Stops the processing of asynchronous queued I/O requests

SYNOPSIS

CALL AQSTOP (*aqp,reqid,queue,status*)

DESCRIPTION

aqp Type **INTEGER** array. The name of the array in the user's program that will contain the asynchronous queued I/O.

reqid Type **INTEGER** variable, expression, or constant. A user-supplied value for identifying a particular request.

queue Type **INTEGER** variable, expression, or constant. Queue flag. If 0, I/O is initiated provided that I/O on the dataset is not already active. If the queue flag is set to nonzero, the request is added to the queue but no attempt is made to start I/O.

status Type **INTEGER** variable. Status code *status* returns any errors or status information to the user. On output from **AQOPEN**, *status* has one of the following values:

>0 Information only
=0 No errors detected
<0 Error detected

Status Codes	
0	No errors detected
+1	The asynchronous queued I/O parameter block is full
+2	No I/O is active on the asynchronous queued I/O dataset or file
+3	The asynchronous queued I/O request is stuck
+4	The asynchronous request is queued for I/O
-1	Illegal *aqpsize* on the **AQOPEN** request. Minimum size is equal to $32 + 8n$, where $n = 1$.

The **AQSTOP** routine stops the processing of a list of asynchronous I/O requests when it is encountered in the queue.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

AQREAD, AQWRITE, AQCLOSE, AQWAIT, AQSTAT, AQRECALL, AQSYNC
The AQIO User's Guide, SN-0247

NAME

AQWAIT – Waits on a completion of asynchronous queued I/O requests

SYNOPSIS

CALL AQWAIT(*aqp*,*status*)

DESCRIPTION

aqp Type INTEGER array. The name of the array in the user's program that contains the asynchronous queued I/O parameter block. This must be the same array specified in the **AQOPEN** request.

status Type INTEGER variable. Status code *status* returns any errors or status information to the user. On output from **AQWAIT** *status* has one of the following values:

>0 Information only
=0 No errors detected
<0 Error detected

Status Codes	
0	No errors detected
+1	The asynchronous queued I/O parameter block is full
+2	No I/O is active on the asynchronous queued I/O dataset or file
+3	The asynchronous queued I/O request is stuck
+4	The asynchronous request is queued for I/O
-1	Illegal *aqpsize* on the **AQOPEN** request. Minimum size is equal to 32 + 8*n*, where *n* = 1.

AQWAIT leaves the job suspended until the entire request list is exhausted.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

AQOPEN, AQREAD, AQREADC, AQWRITE, AQWRITEC, AQCLOSE, AQSTAT
The AQIO User's Guide, SN-0247

NAME

AQWRITE, AQWRITEC, AQWRITEI, AQWRTECI – Queues a simple or compound asynchronous I/O write request

SYNOPSIS

CALL AQWRITE(*aqp,cpuadd,blknum,blocks,reqid,queue,status*)

CALL AQWRITEC(*aqp,cpuadd,mstride,blknum,blocks,dstride,incs,reqid,queue,status*)

CALL AQWRITEI(*aqp,cpuadd,blknum,blocks,reqid,queue,status*)

CALL AQWRTECI(*aqp,cpuadd,mstride,blknum,blocks,dstride,incs,reqid,queue,status*)

DESCRIPTION

aqp Type INTEGER array. The name of the array in the user's program that contains the asynchronous queued I/O parameter block. Must be the same array specified in the **AQOPEN** request.

cpuadd Type determined by user. Starting memory address; the location of the first word in the user's program to be written.

mstride Type INTEGER variable, expression, or constant. Data buffer stride; the number of memory words to skip between the base addresses of consecutive transfers. The stride value may be positive (to skip forward), negative (to skip backward), or 0. This parameter is valid for compound write requests only.

blknum Type INTEGER variable, expression, or constant. Starting block number; the block number of the first block to be written on this request.

blocks Type INTEGER variable, expression, or constant. The number of 512-word blocks to be written.

dstride Type INTEGER variable, expression, or constant. Disk stride; the number of disk blocks to skip between the base addresses of consecutive transfers. The stride value may be positive (to skip forward), negative (to skip backward), or 0. This parameter is valid for compound requests only.

incs Type INTEGER variable, expression, or constant. The number of simple requests minus 1 that comprise a compound request. Zero (0) implies a simple request. This parameter is valid for compound requests only.

reqid Type INTEGER variable, expression, or constant. A user-supplied value for identifying a particular request.

queue Type INTEGER variable, expression, or constant. Queue flag. If 0, I/O is initiated provided that I/O on the dataset or file is not already active. If the queue flag is set to nonzero, the request is added to the queue but no attempt is made to start I/O.

status Type INTEGER variable. Status code *status* returns any errors to the user. On output from these routines, *status* has one of the following values:

>0 Information only
=0 No error detected
<0 Error detected

Status Codes	
0	No errors detected
+1	The asynchronous queued I/O parameter block is full
+2	No I/O is active on the asynchronous queued I/O dataset or file
+3	The asynchronous queued I/O request is stuck
+4	The asynchronous request is queued for I/O
-1	Illegal *aqpsize* on the **AQOPEN** request. Minimum size is equal to 32 + 8n, where n = 1.

AQWRITE, **AQWRITEC**, **AQWRITEI**, and **AQWRTECI** transfer data between the device on which the dataset or file resides and the data buffer. Requests may be simple (**AQWRITE** and **AQWRITEI**) or compound (**AQWRITEC** and **AQWRTECI**). A simple request is one in which data from one buffer is written to consecutive sectors on disk. A compound request is one in which a number of simple requests are separated by a constant number of sectors on disk, a constant number of memory words for buffers, or both.

AQWRITEI and **AQWRTECI** (both COS only) operate in the same fashion as **AQWRITE** and **AQWRITEC**, respectively, except the ignore bit is set. The ignore bit tells the operating system not to change from read mode to process this write request. As an example, setting the ignore bit might be helpful on a system with two high-speed SSD channels. A series of **AQREAD** calls followed by an **AQWRITEI** call would not force a wait by the operating system as would a normal write.

IMPLEMENTATION

AQWRITE and **AQWRITEC** are available to users of both the COS and UNICOS operating systems. **AQWRITEI** and **AQWRTECI** are only available under COS.

SEE ALSO

AQOPEN, AQREAD, AQREADC, AQCLOSE, AQWAIT, AQSTAT
The AQIO User's Guide, SN-0247

NAME

ASYNCMS, ASYNCDR – Set I/O mode for random access routines to asynchronous

SYNOPSIS

CALL ASYNCMS(*dn*[,*ierr*])

CALL ASYNCDR(*dn*[,*ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *dn*=7 corresponds to dataset **FT07**). Hollerith constant dataset names must be from 1 to 7 uppercase characters. Specify a type integer variable, expression, or constant.

ierr Error control and code. Specify a type integer variable. If *ierr* is supplied on the call to **ASYNCMS/ASYNCDR**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the log file. Otherwise, an error code is returned, and the message is added to the job's log file. On output from **ASYNCMS/ASYNCDR**:

ierr=0 No errors detected
<0 Error detected. *ierr* contains one of the error codes described in the following table:

Error Codes	
-1	The dataset name or unit number is illegal
-15	**OPENMS/OPENDR** was not called on this dataset

As **ASYNCMS/ASYNCDR** sets the I/O mode for the random access routines to be asynchronous, I/O operations can be initiated, and subsequent execution can proceed simultaneously with the actual data transfer. If you use **READMS**, precede asynchronous reads with calls to **FINDMS**.

IMPLEMENTATION

This routine is available to users of the both the COS and UNICOS operating systems.

SEE ALSO

OPENMS, WRITMS, READMS, CLOSMS, FINDMS, CHECKMS, WAITMS, SYNCMS, OPENDR, WRITDR, READDR, CLOSDR, STINDR, CHECKDR, WAITDR, SYNCDR, STINDX

NAME

CHECKMS, CHECKDR – Checks status of asynchronous random access I/O operation

SYNOPSIS

CALL CHECKMS(*dn*,*istat*[,*ierr*])

CALL CHECKDR(*dn*,*istat*[,*ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset. (For example, dn=7 corresponds to dataset **FT07**.) Hollerith constant dataset names must be from 1 to 7 uppercase characters. Specify a type integer variable, expression, or constant.

istat Dataset I/O Activity flag. Specify a type integer variable.

=0 No I/O activity on the specified dataset
=1 I/O activity on the specified dataset

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **CHECKMS/CHECKDR**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the log file. Otherwise, an error code is returned, and the message is added to the job's log file. On output from **CHECKMS/CHECKDR**:

ierr=0 No error detected
ierr<0 Error detected. *ierr* contains one of the error codes in the following table:

ERROR CODES	
-1	The dataset name or unit number is illegal
-15	OPENMS/OPENDR was not called on this dataset.

A status flag is returned to you, indicating whether the specified dataset is active.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

OPENMS, WRITMS, READMS, CLOSMS, FINDMS, WAITMS, ASYNCMS, SYNCMS, OPENDR, WRITDR, READDR, CLOSDR, STINDR, WAITDR, ASYNCDR, SYNCDR, STINDX

NAME

CHECKTP – Checks tape I/O status

SYNOPSIS

CALL CHECKTP *(dn,istat,icbuf)*

DESCRIPTION

dn Type INTEGER variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset.

istat Type INTEGER variable

=-1 No status
= 0 EOV
= 1 Tape off reel
= 2 Tape mark detected
= 3 Blank tape detected

icbuf Type INTEGER variable. Circular I/O buffer status.

= 0 Circular I/O buffer empty
= 1 Circular I/O buffer not empty

The user program can use **CHECKTP** to check on a tape dataset's condition following normal Fortran I/O requests.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

CONTPIO, PROCBOV, PROCEOV, SWITCHV, SVOLPRC

TAPE

NAME

CLOSEV – Begins user EOV and BOV processing

SYNOPSIS

CLOSEV(*dn*[,*trailer*])

DESCRIPTION

A user program uses the **CLOSEV** subroutine to switch to the next tape volume at any time. **CLOSEV** writes an end-of-volume (EOV) trailer label to a mounted output tape before switching tapes. **CLOSEV** applies only to magnetic tape datasets.

If the tape is an input tape, you have the option of writing an EOV trailer label. An output tape job is aborted if the output buffer is not empty.

In special EOV processing, the user program must execute the **CLOSEV** subprogram to switch to the next tape and perform special beginning-of-volume (BOV) processing. After the **CLOSEV** macro is executed, the next tape is at the beginning of the volume. The user program is permitted BOV processing at this time. After the BOV processing is completed, the user program must execute the **ENDSP** subprogram to inform the operating system that special processing is complete and to continue normal processing.

dn Dataset name or unit number

trailer A logical constant, variable, or expression. If a value of .TRUE. is specified, a trailer EOV label is written.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

CLOSMS, CLOSDR – Writes master index and closes random access dataset

SYNOPSIS

CALL CLOSMS(*dn*[,*ierr*])

CALL CLOSDR(*dn*[,*ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset. (For example, dn=7 corresponds to dataset **FT07**.) Hollerith constant dataset names must be from 1 to 7 uppercase characters. Specify a type integer variable, expression, or constant.

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **CLOSMS/CLOSDR**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the log file. Otherwise, an error code is returned, and the message is added to the job's log file. On output from **CLOSMS/CLOSDR**:

ierr=0 No error detected
ierr<0 Error detected. *ierr* contains one of the error codes in the following table:

ERROR CODES	
-1	The dataset name or unit number is illegal
-15	**OPENMS/OPENDR** was not called on this dataset.

CLOSMS/CLOSDR writes the master index specified in **OPENMS/OPENDR** from the user program area to the random access dataset and then closes the dataset. Statistics on the activity of the random access dataset and written to dataset **$STATS** (see table **CLOSMS** Statistics following). After the random access dataset has been closed by **CLOSMS/CLOSDR**, the statistics can be written to **$OUT** using the following control statements or their equivalent (COS only):

```
REWIND,DN=$STATS.
COPYF,I=$STATS,O=$OUT.
```

Under UNICOS, statistics are written to **stderr**. Under COS, **CLOSMS/CLOSDR** write a message to **$LOG** upon closing the dataset, whether or not you have requested that error messages be written to the logfile.

CAUTION

If a job step terminates without closing the random access dataset with **CLOSMS/CLOSDR**, dataset integrity is questionable.

CLOSMS Statistics	
Message	Description
TOTAL ACCESSES =	Number of accesses
READS =	Number of reads
WRITES =	Number of writes
SEQUENTIAL READS =	Number of sequential reads
SEQUENTIAL WRITES =	Number of sequential writes
REWRITES IN PLACE =	Number of rewrites in place
WRITES TO EOI =	Number of writes to EOI
TOTAL WORDS MOVED =	Number of words moved
MINIMUM RECORD =	Minimum record size
MAXIMUM RECORD =	Maximum record size
TOTAL ACCESS TIME =	Total access time
AVERAGE ACCESS TIME =	Average access time

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

CONTPIO – Continues normal I/O operations (obsolete)

SYNOPSIS

CALL CONTPIO (*dn,iprc*)

DESCRIPTION

dn Type INTEGER variable, expression, or constant. The name of the dataset as a Hollerith constant or the unit number of the dataset.

iprc Type INTEGER variable
- = 2 Continue normal I/O
- =-1 End-of-data (close tape dataset)

The user program can use **CONTPIO** to inform COS that it intends to continue normal I/O operations. This routine may also be used to close the tape dataset.

NOTE

Cray Research discourages the use of the **CONTPIO, PROCBOV, PROCEOV, SWITCHV,** and **SVOL-PROC** routines. Instead, use **CLOSEV, SETSP, STARTSP,** and **ENDSP** when creating special tape processing routines to handle end-of-volume conditions.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

CHECKTP, PROCBOV, PROCEOV, SWITCHV, SVOLPRC

TAPÉ

NAME

ENDSP – Requests notification at the end of a tape volume

SYNOPSIS

CALL ENDSP(*dn*)

DESCRIPTION

ENDSP indicates to COS that special end-of-volume (EOV) and beginning-of-volumen (BOV) processing is complete.

ENDSP does not switch volumes; when the user program wants to switch to the next tape, it must execute **CLOSEV**. Furthermore, for output datasets, data in the I/O Processor (IOP) buffer is not written to tape until **ENDSP** is executed at the beginning of the next tape. When the BOV processing is done, the user program must execute **ENDSP** to terminate special processing. After executing **ENDSP**, the user program can continue to process the tape dataset.

dn Dataset name or unit number

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

FINDMS – Reads record into data buffers used by random access routines

SYNOPSIS

CALL FINDMS(*dn*,*n*,*irec*[,*ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *dn*=7 corresponds to dataset **FT07**. Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.

n The number of words to be read, as in **READMS** or **WRITMS**. Type integer variable, expression, or constant.

irec As in **READMS** or **WRITMS**, the record name or number to be read into the data buffers. Specify a type integer variable, expression, or constant.

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **FINDMS**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the log file. Otherwise, an error code is returned, and the message is added to the job's log file.

On output from **FINDMS**:
ierr=0 No errors detected
ierr<0 Error detected. *ierr* contains one of the error codes in following table:

Error Codes	
-6	The user-supplied named index is invalid
-8	The index number is greater than the maximum on the dataset
-10	The named record was not found is the index array
-15	**OPENMS/OPENDR** was not called on this dataset
-17	The index entry is less than or equal to 0 in the users index array
-18	The user-supplied word count is less than or equal to 0
-19	The user-supplied index number is less than or equal to 0

FINDMS asynchronously reads the desired record into the data buffers used by the random access dataset routines for the specified dataset. The next **READMS** or **WRITMS** call waits for the read to complete and transfers data appropriately.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

OPENMS, WRITMS, READMS, CLOSMS, CHECKMS, WAITMS, ASYNCMS, SYNCMS, OPENDR, WRITDR, READDR, CLOSDR, STINDR, CHECKDR, WAITDR, ASYNCDR, SYNCDR, STINDX

NAME

FSUP, ISUP – Output a value in an argument as blank in Fortran format
FSUPC, ISUPC – Invalidate the function obtained by calling **FSUP** or **ISUP**, returning to ordinary I/O

SYNOPSIS

CALL FSUP(*fvalue*)

CALL ISUP(*ivalue*)

CALL FSUPC

CALL ISUPC

DESCRIPTION

fvalue and *ivalue* are real and integer arguments, respectively. If **FSUP** is not called, F-type, G-type, and E-type values are output as for ordinary Fortran I/O. When **FSUP** is called, all values equal to *fvalue* are output as blanks whenever they are encountered in a formatted I/O operation. **FSUP** may be called again to redefine itself.

FSUPC invalidates the call from **FSUP**, and all types are output as ordinary Fortran I/O.

ISUP and **ISUPC** are the integer equivalents of **FSUP** and **FSUPC**. **ISUP** acts only upon I-type values.

IMPLEMENTATION

These routine are available to users of both the COS and UNICOS operating system.

NAME

GETPOS, SETPOS - Returns the current position of interchange tape or mass storage dataset or file; returns to position retained from **GETPOS** request.

SYNOPSIS

CALL GETPOS(*dn,len,pa*[*,stat*])

CALL SETPOS(*dn,len,pa*[*,stat*])

DESCRIPTION

GETPOS returns the current position of the specified interchange tape or mass storage dataset to the Fortran user. **GETPOS** does not alter the dataset's position, but it captures information that you can use later to recover the current position.

SETPOS lets you return to the position retained from the **GETPOS** request. **SETPOS**, like **GETPOS**, can be used on interchange tape or mass storage datasets.

dn Dataset name, file name, or unit number

len Length in Cray words of the position array. This parameter determines the maximum number of position values to return or process. For **SETPOS**, this parameter allows for the addition of more information fields while ensuring that existing codes continue to run. Possible values for *len* are:
1 For disk datasets
2 For tape datasets
3 For disk or tape datasets recorded as a foreign dataset (valid only under COS)

pa Position array. On exit, *pa* contains the current position information. For **GETPOS**, you should not modify this information. It should be retained to be passed on to **SETPOS**. For **SETPOS**, *pa* contains the desired position information from the **GETPOS** call. The format of the position information is as follows:

- For a disk dataset, one word that contains the current position.
- For a tape dataset, two words; word 0 contains the volume serial number of the current tape reel, and word 1 contains the block number before which the tape unit is positioned.
- For a foreign tape dataset (COS only), three words; word 0 contains the block number before which the tape unit is positioned, word 1 contains the volume serial number of the current tape reel, and word 2 contains the block length.

stat Return conditions. This optional parameter returns errors and warnings from the position information routine, as follows:

=0 For **GETPOS**, indicates position information successfully returned. For **SETPOS**, indicates dataset successfully positioned.

≠0 Error or warning encountered during request. Error message number; see coded $IOLIB messages in the COS Message Manual, publication SR-0039.

To set the position of a mass storage dataset, the position must be at a record boundary; that is, at the beginning-of-dataset (BOD), following an end-of-record (EOR) or end-of-file (EOF), or before an end-of-dataset (EOD). A dataset cannot be positioned beyond the current EOD.

SETPOS positions to a logical record when processing a foreign file (COS only) with the library data conversion support (**FD** parameter on the **ACCESS** and **ASSIGN** control statements). This same capability also exists for mass storage files that have been assigned foreign dataset characteristics.

If foreign dataset conversion has not been requested, the physical tape block and volume position is determined.

For interchange tape dataset, **SETPOS** must synchronize before the dataset can be positioned. Thus, for input datasets, the dataset must be positioned at a Cray EOR. An EOR is added to the EOD before the synchronization if the dataset is an output dataset and the end of the tape block was not already written.

NOTE

For disk files only, **GETPOS** and **SETPOS** also support calls of the following form:

> *pv* = **GETPOS**(*dn*)
> **CALL SETPOS**(*dn*,*pv*)

where *dn* is the dataset or file name or number, and *pv* is the position value. in words

IMPLEMENTATION

These routines are available to users of both the UNICOS and COS operating systems. UNICOS does not support the positioning of blocked files or tapes or of foreign files (those in a non-Cray format).

SEE ALSO

GETTP, SETTP, SYNCH (COS only)

NAME

GETTP - Receives position information about an opened tape dataset or file

SYNOPSIS

CALL GETTP(*dn,len,pa,synch,istat*)

DESCRIPTION

dn Name of the dataset, file, or unit number to get the position information. Must be an integer variable, or an array element containing Hollerith data of not more than 7 characters. This parameter should be of the form '*dn*'**L**.

len Length in Cray words of the position array *pa*. **GETTP** uses this parameter to determine the maximum number of position values to return. This parameter allows for the addition of more information fields while ensuring that existing codes continue to run. Currently, 15 words are used.

pa Position array. On exit, *pa* contains the current position information, as follows:

- *pa*(1) Volume Identifier of last block processed
- *pa*(2) Characters 1 through 8 of permanent dataset name or file name
- *pa*(3) Characters 9 through 16 of permanent dataset name or file name
- *pa*(4) Characters 17 through 24 of permanent dataset name or file name
- *pa*(5) Characters 25 through 32 of permanent dataset name or file name
- *pa*(6) Characters 33 through 40 of permanent dataset name or file name
- *pa*(7) Characters 41 through 44 of permanent dataset name or file name
- *pa*(8) File section number
- *pa*(9) File sequence number
- *pa*(10) Block number
- *pa*(11) Number of blocks in the circular buffer. On output, blocks not sent to I/O Processor (IOP); on input, always 0.
- *pa*(12) Number of blocks in the IOP buffer
- *pa*(13) Device ID (unit number)
- *pa*(14) Device identifier (name)
- *pa*(15) Generic device name

synch Synchronize tape dataset or file. **GETTP** uses this parameter to determine whether to synchronize the program and an opened tape dataset or file before obtaining position information. Synchronization, if requested, is done according to the current positioning direction.

- =0 Do not synchronize tape dataset or file
- =1 Synchronize tape dataset or file before obtaining position information

istat Return conditions. This parameter returns errors and warnings from the position routine.

- =0 Dataset or file position information successfully returned
- ≠0 Error or warning encountered during request

The **GETTP** routine lets you receive information about an opened tape dataset or file. The information returned by **GETTP** refers to the last block processed if the dataset is an input dataset. For output datasets, the information returned by **GETTP** can be meaningless unless the tape dataset or file has been synchronized.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

SETTP, GETPOS, SYNCH (COS only)

NAME

GETWA, SEEK – Synchronously and asynchronously reads data from the word-addressable, random access dataset

SYNOPSIS

CALL GETWA(*dn,result,addr,count*[,*ierr*])

CALL SEEK(*dn,addr,count*[,*ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *dn*=7 corresponds to FT07). Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.

result Variable or array of any type. The location in the user program where the first word is placed.

addr For **GETWA**, the word location of the dataset from which the first word is transferred. For **SEEK**, the word address of the next read. Specify a type integer variable, expression, or constant.

count For **GETWA**, the number of words from result written from the dataset into user memory. For **SEEK**, the number of words of the next read. Specify a type integer variable, expression, or constant.

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **GETWA** or **SEEK**, *ierr* returns any error codes to you. If *ierr* is not supplied, an error aborts the job.

On output from **GETWA**:
ierr=0 No errors detected
<0 Error detected. *ierr* contains one of the error codes in the following table:

Error Codes	
-1	Illegal unit number
-2	The number of datasets has exceeded memory or size availability
-3	User attempt to read past end-of-data (EOD)
-4	The user-supplied word address less than or equal to 0
-5	User-requested word count greater than maximum allowed
-6	Illegal dataset name
-7	User word count less than or equal to 0

The **SEEK** and **GETWA** calls are used together. The **SEEK** call reads the data asynchronously; the **GETWA** call waits for I/O to complete and then transfers the data. The **SEEK** call moves the last write operation pages from memory to disk, loading the user-requested word addresses to the front of the I/O buffers. You can load as much data as fits into the dataset buffers. Subsequent **GETWA** and **PUTWA** calls that reference word addresses in the same range do not cause any disk I/O.

NOTE

Most of the routines in the run-time libraries are reentrant or have internal locks to ensure that they are single threaded. Some library routines, however, must be locked at the user level if they are used by more than one task.

GETWA is not internally locked. You must lock each call to **GETWA** if it is called from more than one task.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

EXAMPLE

Assume you want to use a routine that reads word addresses 1,000,000 to 1,051,200. A dataset is opened with 101 blocks of buffer space, and **CALL SEEK(*dn*,1000000,51200,*ierr*)** is used before calling the routine. Subsequent **GETWA** or **PUTWA** calls with word addresses in the range of 1,000,000 to 1,051,200 do not trigger any disk I/O.

SEE ALSO

WOPEN, WCLOSE, PUTWA, APUTWA

NAME

OPENMS, OPENDR – Opens a local dataset as a random access dataset that can be accessed or changed by the record-addressable, random access dataset I/O routines

SYNOPSIS

CALL OPENMS(*dn,index,length,it*[*,ierr*]**)**

CALL OPENDR(*dn,index,length,it*[*,ierr*]**)**

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *dn*=7 corresponds to dataset **FT07**. Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.

index The name of the array in the user program that is going to contain the master index to the records of the dataset. Specify a type integer array. This array must be changed only by the random access dataset I/O routines. *index* should be a multiple of 512 words.

length The length of the index array. Specify a type integer variable, expression, or constant. The length of *index* depends upon the number of records on or to be written to the dataset using the master index and upon the type of master index. The *length* specification must be at least 2**nrec* if *it*=1 or 3, or *nrec* if *it*=0 or 2. *nrec* is the number of records in or to be written to the dataset using the master index.

it Flag indicating the type of master index. Specify a type integer variable, expression, or constant.

- *it*=0 Records synchronously referenced with a number between 1 and *length*
- *it*=1 Records synchronously referenced with an alphanumeric name of 8 or fewer characters
- *it*=2 Records asynchronously referenced with a number between 1 and *length*
- *it*=3 Records asynchronously referenced with an alphanumeric name of 8 or fewer characters

For a named index, odd-numbered elements of the index array contain the record name, and even-numbered elements of the index array contain the pointers to the location of the record within the dataset. For a numbered index, a given index array element contains the pointers to the location of the corresponding record within the dataset.

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **OPENMS/OPENDR**, *ierr* returns any error codes to you. If *ierr* is not supplied, an error aborts the job.

If you set *ierr*>0 on input to **OPENMS/OPENDR**, error messages are not placed in the logfile. Otherwise, an error code is returned, and the error message is added to the job's logfile. **OPENMS/OPENDR** writes an open message to the logfile whether or not the value of *ierr* selects log messages.

On output from **OPENMS/OPENDR:**

ierr=0 No errors detected

<0 Error detected. *ierr* contains one of the following error codes:

Error Codes	
-1	The dataset name or unit number is illegal
-2	The user-supplied index length is less than or equal to 0
-3	The number of datasets has exceeded memory or size availability
-4	The dataset index length read from the dataset is greater than the user-supplied index length (nonfatal message)
-5	The user-supplied index length is greater than the index length read from the dataset (nonfatal message)
-11	The index word address read from the dataset is less than or equal to 0
-12	The index length read from the dataset is less than 0
-13	The dataset has a checksum error
-14	**OPENMS** has already opened the dataset
-20	Dataset created by WRITDR/WRITMS

NOTES

A file opened with **OPENMS** should only be closed by **CLOSMS**. If you close the file in some other way, the future behavior of the program is unpredictable.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

WRITMS, READMS, CLOSMS, FINDMS, CHECKMS, WAITMS, ASYNCMS, SYNCMS, WRITDR, READDR, CLOSDR, STINDR, CHECKDR, WAITDR, ASYNCDR, SYNCDR, STINDX

NAME

PROCBOV – Allows special processing at beginning-of-volume (BOV) (obsolete)

SYNOPSIS

CALL PROCBOV(*dn*,*iprc*)

DESCRIPTION

dn Type **INTEGER** variable, expression, or constant. The name of the dataset as a Hollerith constant or unit number of the dataset.

iprc Type INTEGER variable

= 1 Special processing at BOV
= 2 Continue normal I/O
=-1 End-of-data (close tape dataset)

The user program can use **PROCBOV** to inform COS that it intends to reposition or perform special I/O processing to the tape. This routine assumes that the tape dataset is positioned at BOV. **PROCBOV** allows special processing at beginning-of-volume. This routine may also be used to continue normal I/O or close the tape dataset.

NOTE

Cray Research discourages the use of the **CONTPIO**, **PROCBOV**, **PROCEOV**, **SWITCHV**, and **SVOLPROC** routines. Instead, use **CLOSEV**, **SETSP**, **STARTSP**, and **ENDSP** when creating special tape processing routines to handle end-of-volume conditions.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

CHECKTP, CONTPIO, PROCEOV, SWITCHV, SVOLPRC

NAME

PROCEOV – Begins special processing at end-of-volume (EOV) (obsolete)

SYNOPSIS

CALL PROCEOV(*dn*,*iprc*)

DESCRIPTION

dn Type **INTEGER** variable, expression, or constant. The name of the dataset as a Hollerith constant or unit number of the dataset.

iprc Type **INTEGER** variable.

= 0 Special processing at EOV
= 1 Special processing at BOV
= 2 Continue normal I/O
=-1 End-of-data (close tape dataset)

The user program can use **PROCEOV** to inform COS that it intends to reposition or perform special I/O processing to the tape. This routine assumes that the tape dataset is positioned at EOV. **PROCEOV** allows special processing at BOV EOV. This routine may also be used to continue normal I/O or to close the tape dataset.

NOTE

Cray Research discourages the use of the **CONTPIO**, **PROCBOV**, **SWITCHV**, **PROCEOV**, and **SVOLPROC** routines. Instead, use **CLOSEV**, **SETSP**, **STARTSP**, and **ENDSP** when creating special tape processing routines to handle end-of-volume conditions.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

CHECKTP, **CONTPIO**, **PROCBOV**, **SWITCHV**, **SVOLPRC**

NAME

PUTWA, APUTWA – Writes to a word-addressable, random-access dataset

SYNOPSIS

CALL PUTWA(*dn*,*source*,*addr*,*count*[,*ierr*])

CALL APUTWA(*dn*,*source*,*addr*,*count*[,*ierr*])

DESCRIPTION

dn — Name of the dataset as a Hollerith constant or the unit number of the dataset. Specify a type integer variable, expression, or constant.

source — Variable or array of any type. The location of the first word in the user program to be written to the dataset.

addr — The word location of the dataset that is to receive the first word from the user program. *addr*=1 indicates beginning of file. Specify a type integer variable, expression, or constant.

count — The number of words from source to be written. Specify a type integer variable, expression, or constant.

ierr — Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **PUTWA**, *ierr* returns any error codes to you. If *ierr* is not supplied, an error causes the job to abort.

On output from **PUTWA/APUTWA**:

ierr=0 No errors detected
-1 Invalid unit number
-2 Number of datasets has exceeded memory size availability
-4 User-supplied word address less than or equal to 0
-5 User-requested word count greater than maximum allowed
-6 Invalid dataset name
-7 User word count less than or equal to 0

PUTWA synchronously writes a number of words from memory to a word-addressable, random-access dataset. **APUTWA** asynchronously writes a number of words from memory to a word-addressable, random-access dataset.

NOTE

Most of the routines in the run-time libraries are reentrant or have internal locks to ensure that they are single threaded. Some library routines, however, must be locked at the user level if they are used by more than one task.

PUTWA is not internally locked. You must lock each call to **PUTWA** if it is called from more than one task.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

SEE ALSO

WOPEN, WCLOSE, GETWA, SEEK

NAME

READ, READP – Reads words, full or partial record modes

SYNOPSIS

CALL READ(*dn,word,count,status,ubc***)**

CALL READP(*dn,word,count,status,ubc***)**

DESCRIPTION

dn Unit number or file name as a Hollerith in seven characters or less (**'MYFILE'**)

word Word-receiving data area, such as a variable or array

count On entry, the number of words requested. (Do not specify a constant.) On exit, the number of words actually transferred.

status On exit, *status* has one of the following values:
=-1 Words remain in record
= 0 EOR
= 1 Null record
= 2 End-of-file (EOF)
= 3 End-of-data (EOD)
= 4 Hardware error

ubc Optional unused bit count. Number of unused bits contained in the last word of the record.

READ and **READP** move words of data from disk to a user's variable or array. They are intended to read COS blocked datasets, under both COS and UNICOS. After reading less than a full record from disk, **READ** leaves the file positioned at the beginning of the next record, while **READP** leaves the file positioned at the next item in the record just read.

EXAMPLE

The following example reads the first two words of two consecutive records:

```
INTEGER REC(10)
NUM = 2
CALL READ(DN=15, REC, NUM)
NUM = 2
CALL READ(DN=15, REC, NUM)
STOP
```

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

SEE ALSO

READC, READCP, READIBM, WRITE, WRITEP, WRITEC, WRITECP, WRITIBM, SKIPBAD, ACPTBAD

NAME

READC, READCP – Reads characters, full or partial record mode

SYNOPSIS

CALL READC(*dn,char,count,status***)**

CALL READCP(*dn,char,count,status***)**

DESCRIPTION

dn Unit number

char Character-receiving data area

count On entry, the number of characters requested. On exit, the number of characters actually transferred.

status On exit, *status* has one of the following values:
- =-1 Characters remain in record
- = 0 End-of-record (EOR)
- = 1 Null record
- = 2 End-of-file (EOF)

Read character routines unpack characters from the I/O buffer and insert them into the user data area beginning at the first word address. Characters are placed into the data area one character per word, right-justified. This process continues until the count is satisfied or an EOR is encountered. If an EOR is encountered first, the remainder of the field specified by the character count is filled with blanks. Blank expansion is performed on the characters read from the buffer to the data area.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

SEE ALSO

READ, READP, READIBM, WRITE, WRITEP, WRITEC, WRITECP, WRITIBM, SKIPBAD, ACPTBAD

NAME

READIBM – Reads two IBM 32-bit floating-point words from each Cray 64-bit word

SYNOPSIS

CALL READIBM(*dn,fwa,word,increment*)

DESCRIPTION

dn	Dataset name or unit number
fwa	First word address (FWA) of the user data area
word	Number of words needed
increment	Increment of the IBM words read

On exit, the IBM 32-bit format is converted to the equivalent Cray 64-bit value. The Cray 64-bit words are stored in the user data area.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

READ, READP, READC, READCP, WRITE, WRITEP, WRITEC, WRITECP, WRITIBM, SKIPBAD, ACPTBAD

NAME

READMS, READDR – Reads a record from a random access dataset

SYNOPSIS

CALL READMS(*dn,ubuff,n,irec*[*,ierr*]**)**

CALL READDR(*dn,ubuff,n,irec*[*,ierr*]**)**

DESCRIPTION

READMS and **READDR** read records from a random access dataset to a contiguous memory area in the user's program.

dn The name of the dataset as a Hollerith constant or the unit number of the dataset. Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.

ubuff The location in your program where the first word of the record is placed. User-specified type.

n The number of words to be read. Specify a type integer variable, expression, or constant. *n* words are read from the random access record *irec* and placed contiguously in memory, beginning at *ubuff*. If necessary, **READDR** rounds *n* up to the next multiple of 512 words. If the file is in synchronous mode, the data is saved and restored after the read.

irec The record number or record name of the record to be read. Specify a type integer variable, expression, or constant. A record name is limited to a maximum of 8 characters. For a numbered index, *irec* must be between 1 and the length of the index declared in the **OPENMS/OPENDR** call, inclusive. For a named index, *irec* is any 64-bit entity you specify.

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **READMS/READDR**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned, and the message is added to the job's logfile.

On output from **READMS/READDR**:
ierr=0 No errors detected
<0 Error detected. *ierr* contains one of the error codes in the following table:

Error Codes	
-1	The dataset name or unit number is invalid
-6	The user-supplied named index is invalid
-7	The named record index array is full
-8	The index number is greater than the maximum on the dataset
-9	Rewrite record exceeds the original
-10	The named record was not found is the index array
-15	OPENMS/OPENDR was not called on this dataset
-17	The index entry is less than or equal to 0 in the users index array
-18	The user-supplied word count is less than or equal to 0
-19	The user-supplied index number is less than or equal to 0

WARNING

If you are using **READDR** in asynchronous mode, and the record size is not a multiple of 512 words, user data can be overwritten and not restored. With **SYNCDR**, the dataset can be switched to read synchronously, causing data to be copied out and restored after the read has completed.

NOTE

Most of the routines in the run-time libraries are reentrant or have internal locks to ensure that they are single threaded. Some library routines, however, must be locked at the user level if they are used by more than one task.

READMS and **READDR** are not internally locked. You must lock each call to these routines if they are called from more than one task.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

OPENMS, WRITMS, CLOSMS, FINDMS, CHECKMS, WAITMS, ASYNCMS, SYNCMS, OPENDR, WRITDR, CLOSDR, STINDR, CHECKDR, WAITDR, ASYNCDR, SYNCDR, STINDX

NAME

RNLFLAG, RNLDELM, RNLSEP, RNLREP, RNLCOMM – Adds or deletes characters from the set of characters recognized by the **NAMELIST** input routine

SYNOPSIS

CALL RNLFLAG(*char,mode*)

CALL RNLDELM(*char,mode*)

CALL RNLSEP(*char,mode*)

CALL RNLREP(*char,mode*)

CALL RNLCOMM(*char,mode*)

DESCRIPTION

char For **RNLFLAG**, an echo character. Default is 'E'.
For **RNLDELM**, a delimiting character. Default is '$' and '&'.
For **RNLSEP**, a separator character. Default is ','.
For **RNLREP**, a replacement character. Default is '='.
For **RNLCOMM**, a trailing comment indicator. Defaults are ':' and ';'.

mode =0 Delete character
≠0 Add character

In each of these user-control subroutine argument lists, *char* is a character specified as 1L*x* or 1R*x*.

RNLFLAG adds or removes *char* from the set of characters that, if found in column 1, initiates echoing of the input lines to $OUT (under COS) or **stdout** (under UNICOS).

RNLDELM adds or removes *char* from the set of characters that precede the **NAMELIST** group name and signal end-of-input.

RNLSEP adds or removes *char* from the set of characters that must follow each constant to act as a separator.

RNLREP adds or removes *char* from the set of characters that occur between the variable name and the value.

RNLCOMM adds or removes *char* from the set of characters that initiate trailing comments on a line.

No checks are make to determine the reasonableness, usefulness, or consistency of these changes.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

SEE ALSO

RNLSKIP, RNLECHO, RNLTYPE WNL, WNLLONG, WNLLINE

NAME

RNLECHO – Specifies output unit for NAMELIST error messages and echo lines

SYNOPSIS

CALL RNLECHO(*unit*)

DESCRIPTION

unit Output unit to which error messages and echo lines are sent. If *unit*=0, error messages and lines echoed because of an **E** in column 1 go to $OUT (under COS) or **stdout** (under UNICOS) (default).

If *unit* ≠0, error messages and input lines are echoed to *unit*, regardless of any echo flags present. If *unit*=6 or *unit*=101, $OUT (under COS) or **stdout** (under UNICOS) is implied.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

RNL, RNLSKIP, RNLTYPE WNL, WNLLONG, WNLLINE

NAME

RNLSKIP – Takes appropriate action when an undesired NAMELIST group is encountered

SYNOPSIS

CALL RNLSKIP(*mode*)

DESCRIPTION

mode	<0	Skips the record and issues a logfile message (default)
	=0	Skips the record
	>0	Aborts the job or goes to the optional ERR= branch

RNLSKIP determines action if the **NAMELIST** group encountered is not the desired group.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

RNL, RNLSKIP, RNLECHO WNL, WNLLONG, WNLLINE

NAME

RNLTYPE – Determines action if a type mismatch occurs across the equal sign on an input record

SYNOPSIS

CALL RNLTYPE(*mode*)

DESCRIPTION

mode	≠0	Converts the constant to the type of the variable (default)
	=0	Aborts the job or goes to the optional ERR= branch

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

RNL, RNLSKIP, RNLECHO WNL, WNLLONG, WNLLINE

NAME

SETSP – Requests notification at the end of a tape volume

SYNOPSIS

CALL SETSP(*dn*,*on*)

DESCRIPTION

SETSP informs the operating system that you wish to perform extra processing when the end of a tape volume is reached. You must call **SYNCH** to ensure all data is written to tape before calling **SETSP**.

After the user program has called **SETSP**, the end-of volume (EOV) condition is set when the tape is positioned after the last data block. For an input dataset, this occurs after the system has read the last data block on the volume. For an output dataset, this occurs when end-of-tape (EOT) status is detected.

Automatic volume switching is not done by COS following the successful execution of **SETSP** with the *on* parameter non-zero. If you want to switch volumes, call **CLOSEV**.

dn Dataset name or unit number

on Type LOGICAL variable, expression, or constant. A value of .FALSE. turns off special processing. A value of .TRUE. turns on special processing.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

STARTSP, ENDSP, CLOSEV

TAPE

NAME

SETTP – Positions a tape dataset or file at a tape block of the dataset or file

SYNOPSIS

CALL SETTP(*dn,nbs,nb,nvs,nv,vi,synch,istat*)

DESCRIPTION

dn Name of the dataset or file or unit number to be positioned. Must be an integer variable, or an array element containing Hollerith data of not more than 7 characters. This parameter should be of the form '*dn*'**L**.

nbs Block number request sign. This parameter must be set to either '+'L, '-'L, or ' 'L. See the block number parameter (*nb*) for usage detail.

nb Block number or number of blocks to forward space or backspace from the current position. The direction of the positioning is specified by the block number request sign parameter *nbs*.

- *+nb* Specifies the number of blocks to forward space from the current position. The *nbs* parameter should be set to '+'L when forward block positioning is desired. The + sign is invalid if either *nv* or *vi* is requested.
- *-nb* Spcified the number of blocks to backspace from the current position. The *nbs* parameter should be set to '-'L when backward block positioning is desired. The - sign is invalid if either *nv* or *vi* is requested.
- *nb* Specifies the absolute block number to be positioned to. The *nbs* parameter should be set to a blank (' 'L) when absolute block positioning is desired. This option is not supported under UNICOS.

nvs Volume number request sign. This parameter must be set to '+'L, '-'L, or ' 'L. See the volume number parameter (*nv*) for usage details.

nv Volume number or number of volumes to forward space or backspace from the current position. This parameter should be set equal to a binary volume number or number of volumes to forward space or backspace. This direction of the positioning is specified by the volume number request sign parameter *nvs*. This parameter is invalid if *vi* is also requested.

- *+nv* Specifies the number of volumes to forward space from the current volume. The *nvs* parameter should be set to **An** *nb* request must not be specified with + or - signs.
- *-nv* Specifies the number of volumes to backspace from the current volume. The *nvs* parameter should be set to **A** *nb* request must not be specified with + or - signs.
- *nv* Specifies the absolute volume number to be positioned to. The *nvs* parameter should be set to

vi Volume identifier to be mounted. This parameter is invalid if *nv* is also requested. Also, *nb* must not be specified without + or - signs. The volume identifier must be left-justified, zero-filled.

synch Synchronize tape dataset. **SETTP** uses this parameter to determine whether to synchronize the program and an opened tape dataset before positioning. Synchronization, if requested, is done according to the current positioning direction.

=0 Do not synchronize tape dataset or file

=1 Synchronize tape dataset or file before positioning

istat Return conditions. This parameter is used to return errors and warnings from the position routine.

=0 Dataset or file successfully positioned

≠0 Error or warning encountered during request

SETTP allows you to position a tape dataset at a particular tape block of the dataset. Data blocks on the tape are numbered so that block number 1 is the first data block on a tape. Before a tape dataset is positioned with **SETTP**, the dataset must be synchronized with the **SYNCH** routine (COS only) or with the synchronization parameter on the **SETTP** request.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

GETTP, **SYNCH** (COS only), **GETPOS**

NAME

SKIPBAD – Skips bad data

SYNOPSIS

CALL SKIPBAD(*dn,blocks,termcnd*)

DESCRIPTION

dn Dataset name or unit number

blocks On exit, under COS, contains the number of blocks skipped. Under UNICOS, this is the number of physical tape blocks skipped.

termcnd On exit, termination condition.

<0 Not positioned at end-of-block
=0 Positioned at end-of-block
=1 If 1, positioned at end-of-file

SKIPBAD allows you to skip bad data so that no bad data is sent to the user-specified buffer. UNICOS does not support bad data recovery on transparent tapes.

EXAMPLE

```
      PROGRAM EXAMPLE2
      IMPLICIT INTEGER(A - Z)
      REAL UNIT, UNITSTAT
      PARAMETER(NBYTES=400000,NDIM=NBYTES/8,DN=99)
      DIMENSION BUFFER(1:NDIM)
 2000 CONTINUE
      NWORDS = NDIM
      CALL READ(DN,BUFFER,NWORDS,STATUS)
      UNITSTAT = UNIT(DN)
      IF(STATUS.EQ.4 .OR. UNITSTAT.GT.0.0) THEN !Parity error
        CALL SKIPBAD(DN,BLOCKS,TERCND)
        IF(TERMCND.LT.0) THEN
          CALL ABORT("SKIPBAD should position tape at EOR/EOF
            ENDIF
      STOP 'COMPLETE'
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and the UNICOS operating systems.

SEE ALSO

ACPTBAD

NAME

STARTSP – Begins user EOV and BOV processing

SYNOPSIS

CALL STARTSP(*dn*)

DESCRIPTION

STARTSP starts special end-of-volume (EOV) and beginning-of-volume processing. No special-processing I/O to the tape occurs until this routine (or the implementing macro) has been executed. The user program must inform COS that it intends to reposition or perform special I/O to the tape by executing the **STARTSP** routine.

After executing the **STARTSP** routine, the user program can issue **READ, WRITE**, and **SETTP** requests. When processing is done, the user program must execute **ENDSP** to inform COS that special processing is complete. **STARTSP** does not switch volumes; when the user program wants to switch to the next tape, you must invoke **CLOSEV**. Moreover, after you execute **STARTSP** and before you execute **ENDSP**, the **CLOSEV** call is the only method to perform volume switching for the user program.

Call **SYNCH** before executing **STARTSP**. For output datasets, the data in the IOP buffer is not written to tape until the **ENDSP** call at the beginning of the next tape.

dn Dataset name or unit number

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

STINDX, STINDR – Allows an index to be used as the current index by creating a subindex

SYNOPSIS

CALL STINDX(*dn,index,length,it*[*,ierr*])

CALL STINDR(*dn,index,length,it*[*,ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the file. Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.

index The user-supplied array used for the subindex or new current index. Specify a type integer array. If *index* is a subindex, it must be a storage area that does not overlap the area used in **OPENMS/OPENDR** to store the master index.

length The length of the index array. Specify a type integer variable, expression, or constant. The length of *index* depends upon the number of records on or to be written to the dataset using the master index and upon the type of master index. If *it*=1, *length* must be at least twice the number of records on or to be written to the dataset using *index*. If *it*=0, *length* must be at least the number of records on or to be written to the dataset using *index*.

it A flag to indicate the type of index. Specify a type integer variable, expression, or constant. When *it*=0, the records are referenced with a number between 1 and *length*. When *it*=1, the records are referenced with an alphanumeric name of 8 or fewer characters. For a named index, odd-numbered elements of the index array contain the record name, and even-numbered elements of the index array contain pointers to the location of the record within the dataset. For a numbered index, a given index array element contains pointers to the location of the corresponding record within the dataset. The index type defined by **STINDX/STINDR** must be the same as that used by **OPENMS/OPENDR**.

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **STINDX/STINDR**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the log file. Otherwise, an error code is returned, and the message is added to the job's log file.

On output from **STINDX/STINDR**:

ierr=0 No errors detected

<0 Error detected. *ierr* contains one of the error codes described in the following table:

Error Codes	
-1	The dataset name or unit number is invalid
-15	OPENMS/OPENDR was not called on this dataset
-16	A STINDX/STINDR

STINDX/STINDR reduce the amount of memory needed by a dataset containing a large number of records. It also maintains a dataset containing records logically related to each other. Records in the dataset, rather than records in the master index area, hold secondary pointers to records in the dataset.

STINDX/STINDR allow more than one index to manipulate the dataset. Generally, **STINDX/STINDR** toggle the index between the master index (maintained by **OPENMS/OPENDR** and **CLOSMS/CLOSDR**) and a subindex (supplied and maintained by you).

You must maintain and update subindex records stored in the dataset. Records in the dataset can be accessed and changed only by using the current index.

After a **STINDX/STINDR** call, subsequent calls to **READMS/READDR** and **WRITMS/WRITDR** use and alter the current index array specified in the **STINDX/STINDR** call. You can save the subindex by calling **STINDX/STINDR** with the master index array, then writing the subindex array to the dataset using **WRITMS/WRITDR.** Retrieve the subindex by calling **READMS/READDR** on the record containing the subindex information. Thus, **STINDX/STINDR** allow logically infinite index trees into the dataset and reduces the amount of memory needed for a random access dataset containing many records.

CAUTION

When generating a new subindex (for example, building a database), set the array or memory area used for the subindex to 0. If the subindex storage is not set to 0, unpredictable results occur.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

OPENMS, WRITMS, READMS, CLOSMS, FINDMS, CHECKMS, WAITMS, ASYNCMS, SYNCMS, OPENDR, WRITDR, READDR, CLOSDR, CHECKDR, WAITDR, ASYNCDR, SYNCDR

NAME

SVOLPRC – Initializes/terminates special BOV/EOV processing (obsolete)

SYNOPSIS

CALL SVOLPRC(*dn*,*iflag*)

DESCRIPTION

dn Type **INTEGER** variable, expression, or constant. The name of the dataset as a Hollerith constant or unit number of the dataset.

iflag Type **INTEGER** variable

=1 Turn BOV/EOV processing ON
=0 Turn BOV/EOV processing OFF

SVOLPRC should be called to inform the operating system that you wish to perform extra processing when the end of a tape volume is reached. Calling **SVOLPRC** with the OFF flag indicates that the user program no longer needs to be notified of EOV conditions. COS does not perform automatic volume switching following an **SVOLPRC** call with the ON flag set.

NOTE

Cray Research discourages the use of the **CONTPIO**, **PROCBOV**, **PROCEOV**, **SWITCHV**, and **SVOLPROC** routines. Instead, use **CLOSEV**, **SETSP**, **STARTSP**, and **ENDSP** when creating special tape processing routines to handle end-of-volume conditions.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

CHECKTP, **CONTPIO**, **PROCBOV**, **PROCEOV**, **SWITCHV**

TAPE

NAME

SWITCHV – Switches tape volume

SYNOPSIS

CALL SWITCHV(*dn,iprc,istat,icbuf*)

DESCRIPTION

dn Type **INTEGER** variable, expression, or constant. The name of the dataset as a Hollerith constant or unit number of the dataset.

iprc Type **INTEGER** variable. Processing option at EOV.

= 1 Continue processing at EOV
= 0 Stop at EOV and return tape status information

istat Type **INTEGER** variable

=-1 No status
= 0 EOV
= 1 Tape off reel
= 2 Tape mark detected
= 3 Blank tape detected

icbuf Type **INTEGER** variable. Circular I/O buffer status.

= 0 Circular I/O buffer empty
= 1 Circular I/O buffer not empty

The user program can use **SWITCHV** to switch to the next tape volume and to check on a tape dataset's condition.

NOTE

Cray Research discourages the use of the **CONTPIO**, **PROCBOV**, **PROCEOV**, **SWITCHV**, and **SVOLPROC** routines. Instead, use **CLOSEV**, **SETSP**, **STARTSP**, and **ENDSP** when creating special tape processing routines to handle end-of-volume conditions.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

CHECKTP, CONTPIO, PROCBOV, PROCEOV, SVOLPRC

TAPE

NAME

SYNCH – Synchronizes the program and an opened tape dataset

SYNOPSIS

CALL SYNCH(*dn*,*pd*,*istat*)

DESCRIPTION

dn Name of the dataset or unit number to be synchronized. Must be a type integer variable or an array element containing Hollerith data of not more than 7 characters. This parameter should be of the form '*dn*'**L**.

pd Processing direction:

- =0 Input dataset
- ≠0 Output dataset

istat Return conditions. This parameter returns errors and warnings from the position routine.

- =0 Dataset successfully synchronized
- ≠0 Error or warning encountered during request, as follows:
  - =1 Execution error
  - =2 Dataset is not a tape dataset.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

GETTP, SETTP, GETPOS, SETPOS

NAME

SYNCMS, SYNCDR – Sets I/O mode for random access routines to synchronous

SYNOPSIS

CALL SYNCMS(*dni*[,*ierr*])

CALL SYNCDR(*dni*[,*ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset. Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **SYNCMS/SYNCDR**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned, and the message is added to the job's logfile.

On output from **SYNCMS/SYNCDR**:
ierr=0 No errors detected
<0 Error detected. *ierr* contains one of the following error codes:

Error Codes	
-1	The dataset name or unit number is invalid
-15	OPENMS/OPENDR was not called on this dataset

All I/O operations wait for completion.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

OPENMS, WRITMS, READMS, CLOSMS, FINDMS, CHECKMS, WAITMS, ASYNCMS, OPENDR, WRITDR, READDR, CLOSDR, STINDR, CHECKDR, WAITDR, ASYNCDR, STINDX

NAME

WAITMS, WAITDR – Waits for completion of an asynchronous I/O operation

SYNOPSIS

CALL WAITMS(*dn,istat*[,*ierr*])

CALL WAITDR(*dn,istat*[,*ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset. Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.

istat Dataset Error flag. Specify a type integer variable.
istat=0 No error occurred during the asynchronous I/O operation
=1 Error occurred during the asynchronous I/O operation

ierr Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **WAITMS/WAITDR**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the logfile. Otherwise, an error code is returned, and the message is added to the job's logfile.

On output from **WAITMS/WAITDR**:
ierr=0 No errors detected
<0 Error detected. *ierr* contains one of the error codes described in the following table:

Error Codes	
-1	The dataset name or unit number is invalid
-15	OPENMS/OPENDR was not called on this dataset

A status flag is returned to you, indicating whether or not the I/O on the specified dataset was completed without error.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

OPENMS, WRITMS, READMS, CLOSMS, FINDMS, CHECKMS, ASYNCMS, SYNCMS, OPENDR, WRITDR, READDR, CLOSDR, STINDR, CHECKDR, ASYNCDR, SYNCDR, STINDX

NAME

WCLOSE – Closes a word-addressable, random access dataset

SYNOPSIS

CALL WCLOSE(*dn*[,*ierr*])

DESCRIPTION

dn The name of the dataset as a Hollerith constant or the unit number of the dataset. Specify a type integer variable, expression, or constant.

ierr Error control and code. Specify a type integer variable, expression, or constant. If you supply *ierr* on the call to **WCLOSE**, *ierr* returns any error codes to you. If *ierr* is not supplied, an error aborts the job.

On output from **WCLOSE**:
ierr=0 No errors detected
=-1 Invalid unit number
=-6 Invalid dataset name

WCLOSE finalizes the additions and changes to the word-addressable, random access dataset and closes the dataset.

SYNOPSIS

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

WOPEN, PUTWA, APUTWA, GETWA, SEEK

NAME

WNLFLAG, WNLDELM, WNLSEP, WNLREP – Provides user control of output format

SYNOPSIS

CALL WNLFLAG(*char*)

CALL WNLDELM(*char*)

CALL WNLSEP(*char*)

CALL WNLREP(*char*)

DESCRIPTION

char For **WNLFLAG**, the first ASCII character of the first line. Default is blank.
For **WNLDELM**, a **NAMELIST** delimiter. Default is '**&**'.
For **WNLSEP**, a **NAMELIST** separator. Default is ','.
For **WNLREP**, a **NAMELIST** replacement character. Default is '='.

WNLFLAG changes the character written in column 1 of the first line from blank to *char*. Typically, *char* is used for carriage control if the output is to be listed, or for forcing echoing if the output is to be used as input for **NAMELIST** reads.

WNLDELM changes the character preceding the group name and **END** from '**&**' to *char*.

WNLSEP changes the separator character immediately following each value from ',' to *char*.

WNLREP changes the replacement operator that comes between *name* and *value* from '=' to *char*.

In each of these subroutines, *char* can be any ASCII character specified by 1L*x* or 1R*x*. No checks are made to determine if *char* is reasonable, useful, or consistent with other characters. If the default characters are changed, use of the output line as **NAMELIST** input might not be possible.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

SEE ALSO

RNL, RNLECHO, RNLSKIP, RNLTYPE WNLLINE, WNLLONG

NAME

WNLLINE – Allows each **NAMELIST** variable to begin on a new line

SYNOPSIS

CALL WNLLINE(*value*)

DESCRIPTION

value	=0	No new line
	=1	New line for each variable

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

RNL, RNLECHO, RNLSKIP, RNLTYPE WNL, WNLLONG

NAME

WNLLONG – Indicates output line length

SYNOPSIS

CALL WNLLONG(*length*)

DESCRIPTION

length Output line length; 8<*length*<161 or *length*=-1 (-1 specifies default of 133 unless the unit is 102 or **$PUNCH**, in which case the default is 80).

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

RNL, RNLECHO, RNLSKIP, RNLTYPE WNL, WNLLINE

NAME

WOPEN – Opens a word-addressable, random access dataset

SYNOPSIS

CALL WOPEN(*dn,blocks,istats*[*,ierr*])

DESCRIPTION

dn	The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, 7 corresponds to **FT07**). Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.
blocks	The maximum number of 512-word blocks that the word-addressable package can use for a buffer. Specify a type integer variable, expression, or constant.
istats	Specify a type integer variable, expression, or constant. If *istats* is nonzero, statistics about the changes and accesses to the dataset *dn* are collected. (See the following table for information about the statistics that are collected.) Under COS, these statistics are written to dataset **$STATS** and can be written to **$OUT** by using the following control statements or their equivalents after the dataset has been closed by **WCLOSE**. REWIND,DN=$STATS. COPYD,I=$STATS,O=$OUT. Under UNICOS, statistics are written to **stderr**.
ierr	Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **WOPEN**, *ierr* returns any error codes to you. If *ierr* is not supplied, an error aborts the job. On output from **WOPEN**: *ierri*=**0** No errors detected **-1** Invalid unit number **-2** Number of datasets has exceeded memory size availability **-6** Invalid dataset name

WOPEN opens a dataset and specifies it as a word-addressable, random access dataset that can be accessed or changed with the word-addressable I/O routines. The **WOPEN** call is optional.

NOTES

A file opened using **WOPEN** should only by closed by **WOPEN** or, under COS, job step advance. If you close the file in some other way, the subsequent behavior of the program is unpredictable. These other ways of closing a file include explicit methods (for example, **CLOSE** and **CALL RELEASE**) and implicit methods (such as **CALL SAVE**).

If you bypass **WCLOSE**, the internal tables maintained by the word-addressable I/O package are not updated, leaving dangling pointers in future computation.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

WCLOSE, PUTWA, APUTWA, GETWA, SEEK

MESSAGES

WOPEN Statistics	
Message	Description
BUFFERS USED =	Number of 512-word buffers used by this dataset
TOTAL ACCESSES =	Number of accesses. This is the sum of the **GETWA** and **PUTWA** calls.
GETS =	Number of times the user called **GETWA**
PUTS =	Number of times the user called **PUTWA**
FINDS =	Number of times the user called **SEEK**
HITS =	Number of times word addresses desired were resident in memory
MISSES =	Number of times no word addresses desired were resident in memory
PARTIAL HITS =	Number of times that some but not all of the word addresses desired were in memory
DISK READS =	Number of physical disk reads done
DISK WRITES =	Number of times a physical disk was written to
BUFFER FLUSHES =	Number of times buffers were flushed
WORDS READ =	Number of words moved from buffers to user
WORDS WRITTEN =	Number of words moved from user to buffers
TOTAL WORDS =	Sum of **WORDS READ** and **WORDS WRITTEN**
TOTAL ACCESS TIME =	Real time spent in disk transfers
AVER ACCESS TIME =	**TOTAL ACCESS TIME** divided by the sum of **DISK READS** and **DISK WRITES**
EOD BLOCK NUMBER =	Number of the last block of the dataset
DISK WORDS READ =	Count of number of words moved from disk to buffers
DISK WDS WRITTEN =	Count of number of words moved from buffers to disk
TOTAL DISK XFERS =	Sum of **DISK WORDS READ**
BUFFER BONUS % =	**TOTAL WORDS** divided by value **TOTAL DISK XFERS** multiplied by 100

NAME

WRITE, WRITEP – Writes words, full or partial record mode

SYNOPSIS

CALL WRITE(*dn,word,count,ubc*)

CALL WRITEP(*dn,word,count,ubc*)

DESCRIPTION

dn	Unit number or file name, seven characters or less and specified as a Hollerith
word	Data area containing words
count	Word count. For **WRITE**, a value of 0 causes an end-of record (EOR) record control word to be written.
ubc	Optional unused bit count. Number of unused bits contained in the last word of the record.

In routines where words are written, the number of words specified by the count are transmitted from the area beginning at the first word address and are written in the I/O buffer. These routines are intended to write to COS blocked datasets under both COS and UNICOS.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

SEE ALSO

READ, READP, READC, READCP, READIBM, WRITEC, WRITECP, WRITIBM, SKIPBAD, ACPTBAD

NAME

WRITEC, WRITECP – Writes characters, full or partial record mode

SYNOPSIS

CALL WRITEC(*dn,char,count***)**

CALL WRITECP(*dn,char,count***)**

DESCRIPTION

dn	Dataset name or unit number
char	Data area containing characters
count	Character count

Write character routines pack characters into the I/O buffer for the dataset. The count specifies the number of characters packed. These characters originate from the user area defined at the first word address, which is 1 character per source word (right-justified). Blank compression is performed on the characters written out.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

SEE ALSO

READ, READP, READC, READCP, READIBM, WRITE, WRITEP, WRITIBM, SKIPBAD, ACPTBAD

NAME

WRITIBM – Writes two IBM 32-bit floating-point words from each Cray 64-bit word

SYNOPSIS

CALL WRITIBM(*dn,fwa,value,increment*)

DESCRIPTION

dn	Dataset name or unit number
fwa	First word address (FWA) of the user data area
value	Number of values to be written
increment	Increment of the source (Cray) words written

On exit, IBM 32-bit words are written to the unit.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

READ, READP, READC, READCP, READIBM, WRITE, WRITEP, WRITEC, WRITECP, SKIPBAD, ACPTBAD

NAME

WRITMS, WRITDR – Writes to a random access dataset on disk

SYNOPSIS

CALL WRITMS(*dn,ubuff,n,irec,rrflag,s*[*,ierr*])

CALL WRITDR(*dn,ubuff,n,irec,rrflag,s*[*,ierr*])

DESCRIPTION

dn — The name of the dataset as a Hollerith constant or the unit number of the dataset (for example, *dn*=7 corresponds to dataset **FT07**). Hollerith constant dataset names must be from 1 to 7 characters. Specify a type integer variable, expression, or constant.

ubuff — The location of the first word in the user program to be written to the record. User-specified type.

n — The number of words to be written to the record. Specify a type integer variable, expression, or constant. *n* contiguous words from memory, beginning at *ubuff*, are written to the dataset record. Since COS unblocked-dataset I/O is in multiples of 512 words, it is recommended that *n* be a multiple of 512 words when speed is important. However, the random access dataset I/O routines support record lengths other than multiples of 512 words. **WRITDR** rounds *n* up to the next multiple of 512 words, if necessary.

irec — The record number or record name of the record to be written. Specify a type integer variable, expression, or constant. A record name is limited to a maximum of 8 characters. For a numbered index, *irec* must be between 1 and the length of the index declared in the **OPENMS/OPENDR** call. For a named index, *irec* is any 64-bit entity you specify.

rrflag — A flag indicating record rewrite control. Specify a type integer variable, expression, or constant. *rrflag* can be one of the following codes:

- 0 — Write the record at EOD.
- 1 — If the record already exists, and the new record length is less than or equal to the old record length, rewrite the record over the old record. If the new record length is greater than the old, abort the job step or return the error code in *ierr*. If the record does not exist, the job aborts or the error code is returned in *ierr*.
- -1 — If the record exists, and its new length does not exceed the old length, write the record over the old record. Otherwise, write the record at EOD.

s — A sub-index flag. Specify a type integer variable, expression, or constant. (The implementation of this parameter has been deferred.)

ierr — Error control and code. Specify a type integer variable. If you supply *ierr* on the call to **WRITMS/WRITDR**, *ierr* returns any error codes to you. If *ierr*>0, no error messages are put into the log file. Otherwise, an error code is returned, and the message is added to the job's log file.

On output from **WRITMS/WRITDR**:

ierr=0 No errors detected

<0 Error detected. *ierr* contains one of the error codes described in the following table:

Error Codes	
-1	The dataset name or unit number is invalid
-6	The user-supplied named index is invalid
-7	The named record index array is full
-8	The index number is greater than the maximum on the dataset
-9	Rewrite record exceeds the original
-15	**OPENMS/OPENDR** was not called on this dataset
-17	The index entry is less than or equal to 0 in the users index array
-18	The user-supplied word count is less than or equal to 0
-19	The user-supplied index number is less than or equal to 0

WRITMS and **WRITDR** write data from user memory to a record in a random access dataset on disk and updates the current index.

NOTE

Most of the routines in the run-time libraries are reentrant or have internal locks to ensure that they are single threaded. Some library routines, however, must be locked at the user level if they are used by more than one task.

WRITMS and **WRITDR** are not internally locked. You must lock each call to these routines if they are called from more than one task.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

EXAMPLES

The following examples show some of the features and uses of random access dataset routines.

Example 1 - In the program **SORT**, a sequence of records is read in and then printed out as a sorted sequence of records.

```
      PROGRAM SORT
      INTEGER IARRAY (512)
      INTEGER INDEX (512), KEYS (100)
      CALL OPENMS ('SORT',INDEX,255,1)
      N=50
C READ IN RANDOM ACCESS RECORDS FROM UNIT "SORT"
      DO 21 I=1,N
      READ(5,1000) (IARRAY(J),J=1,512)
      NAME=IARRAY(1)
      KEYS(I)=IARRAY(1)
      CALL WRITMS ('SORT',IARRAY,512,NAME,0)
   21 CONTINUE
C SORT KEYS ALPHABETICALLY IN ASCENDING ORDER USING
C EXCHANGE SORT
      DO 23 I=1,N-1
      MIN=I
```

```
14      J=I+1
15      DO 22 K=J,N
16         IF (KEYS(K).LT.KEYS(MIN)) MIN=K
     22 CONTINUE
18      IB=KEYS(I)
19      KEYS(I)=KEYS(MIN)
20      KEYS(MIN)=IB
     23 CONTINUE
   C WRITE OUT RANDOM ACCESS RECORDS IN ASCENDING
   C ALPHABETICAL ORDER
22      DO 24 I=1,N
23      NAME=KEYS(I)
24      CALL READMS ('SORT',IARRAY,512,NAME)
25      WRITE(6,5120) (IARRAY(J),J=1,512)
     24 CONTINUE
   1000 FORMAT ("......")
   5120 FORMAT (1X,"......")
29      CALL CLOSMS ('SORT')
30      STOP
31      END
```

In this example, the random access dataset is initialized as shown in line 4. Lines 6 through 11 show that a record is read from unit 5 into array **IARRAY** and then written as a record to the random access dataset **SORT**. The first word of each record is assumed to contain an 8-character name to be used as the name of the record.

Lines 12 through 21 show that the names of the records are sorted in the array **KEYS**. Lines 22 through 26 show that the records are read in and then printed out in alphabetical order.

Example 2 - The programs **INITIAL** and **UPDATE** show how the random access dataset might be updated without the usual search and positioning of a sequential access dataset.

Program **INITIAL:**

```
1   PROGRAM INITIAL
2   INTEGER IARRAY(512)
3   INTEGER INDEX (512)
C
C OPEN RANDOM ACCESS DATASET
C THIS INITIALIZES THE RECORD KEY "INDEX"
C
4   CALL OPENMS ('MASTER',INDEX,101,1)
C
C READ IN RECORDS FROM UNIT 6 AND
C WRITE THEM TO THE DATASET "MASTER"
C
5   DO 10 I=1,50
6   READ(6,600) (IARRAY(J),J=1,512)
7   NAME=IARRAY(1)
8   CALL WRITMS ('MASTER',IARRAY,512,NAME,0,0)
 10 CONTINUE
```

```
C
C CLOSE "MASTER" AND SAVE RECORDS FOR UPDATING
C
      CALL CLOSMS ('MASTER')
  600 FORMAT (1X,'.....')
      STOP
      END
```

Program **UPDATE**:

```
      PROGRAM UPDATE
      INTEGER INEWRCD(512)
      INTEGER INDX (512)
C
C OPEN RANDOM ACCESS DATASET CREATED IN THE
C PREVIOUS PROGRAM "INITIAL"
C
C INDX WILL BE WRITTEN OVER THE OLD RECORD KEY
C
      CALL OPENMS ('MASTER',INDX,101,1)
C
C READ IN NUMBER OF RECORDS TO BE UPDATED
C
      READ (6,610) N
C
C READ IN NEW RECORDS FROM UNIT 6 AND
C WRITE THEM IN PLACE OF THE OLD RECORD THAT HAS
C THAT NAME
C
      DO 10 I=1,N
      READ(6,600) (INEWRCD(J),J=1,512)
      NAME=INEWRCD(1)
      CALL WRITMS ('MASTER',INEWRCD,512,NAME,1,0)
   10 CONTINUE
C
C CLOSE "MASTER" AND SAVE NEWLY UPDATED RECORDS
C FOR FURTHER UPDATING
C
      CALL CLOSMS ("MASTER")
  600 FORMAT (1X,"......")
  610 FORMAT (1X,"......")
      STOP
      END
```

In the preceding example, program **INITIAL** creates a random access dataset on unit **MASTER**; program **UPDATE** then replaces particular records of this dataset without changing the remainder of the records.

Line 10 shows that the call to **CLOSMS** at the end of **INITIAL** caused the contents of **INDEX** to be written to the random access dataset.

Line 4 shows that the call to **OPENMS** at the beginning of **UPDATE** has caused the record key of the random access dataset to be written to **INDX**. The random access dataset and **INDX** are now the same as the random access dataset and **INDEX** at the end of **INITIAL**.

Lines 6 through 10 show that certain records are replaced.

Example 3 - The program **SNDYMS** is an example of the use of the secondary index capability, using **STINDX**. In this example, dummy information is written to the random access dataset.

```
      PROGRAM SNDYMS
      IMPLICIT INTEGER (A-Y)
      DIMENSION PINDEX(20),SINDEX(30),ZBUFFR(50)
      DATA PLEN,SLEN,RLEN /20,30,50/
C OPEN THE DATASET.
      CALL OPENMS (1,PINDEX,PLEN,0,ERR)
      IF (ERR.NE.0) THEN
        PRINT*,' Error on OPENMS, err=',ERR
        STOP 1
      ENDIF
C LOOP OVER THE 20 PRIMARY INDICES. EACH TIME
C A SECONDARY INDEX IS FULL, WRITE THE
C SECONDARY INDEX ARRAY TO THE DATASET.
      DO 40 K=1,PLEN
C ZERO OUT THE SECONDARY INDEX ARRAY.
      DO 10 I=1,SLEN
   10 SINDEX(I)=0
C CALL STINDX TO CHANGE INDEX TO SINDEX.
      CALL STINDX (1,SINDEX,SLEN,0,ERR)
      IF (ERR.NE.0) THEN
        PRINT*,' Error on STINDX, err=',ERR
        STOP 2
      ENDIF
C WRITE SLEN RECORDS.
      DO 30 J=1,SLEN
C GENERATE A RECORD LENGTH BETWEEN 1 AND RLEN.
      TRLEN=MAX0(IFIX(RANF(0)*FLOAT(RLEN)),1)
C FILL THE "DATA" ARRAY WITH RANDOM FLOATING POINT
C NUMBERS.
      DO 20 I=1,TRLEN
   20 ZBUFFR(I)=(J+SIN(FLOAT(I)))**(1.+RANF(0))
      CALL WRITMS (1,ZBUFFR,TRLEN,J,-1,DUMMY,ERR)
      IF (ERR.NE.0) THEN
        PRINT*,' Error on WRITMS, err=',ERR
        STOP 3
      ENDIF
   30 CONTINUE
```

```
C "TOGGLE" THE INDEX BACK TO THE MASTER AND
C WRITE THE SECONDARY INDEX TO THE DATASET.
   CALL STINDX (1,PINDEX,PLEN,0)
C NOTE THE ABOVE STINDX CALL DOES NOT USE THE
C OPTIONAL ERROR PARAMETER, AND WILL ABORT
C IF STINDX DETECTS AN ERROR.
   CALL WRITMS (1,SINDEX,SLEN,K,-1,DUMMY,ERR)
   IF (ERR.NE.0) THEN
     PRINT*,' Error on STINDX, err=',ERR
     STOP 4
   ENDIF
40 CONTINUE
C CLOSE THE DATASET.
   CALL CLOSMS (1,ERR)
   IF (ERR.NE.0) THEN
     PRINT*,' Error on CLOSMS, err=',ERR
     STOP 5
   ENDIF
   STOP 'Normal'
   END
```

13. DATASET UTILITY ROUTINES

The dataset utility routines manipulate datasets for use by a program unit. The following routines are ANSI standard Fortran routines (except **LENGTH** and **UNIT**, which are CFT extensions) and are described in the Fortran (CFT) Reference Manual, publication SR-0009 and the CFT77 Reference Manual, publication SR-0018.

Routine	Description
OPEN	Connects a dataset to a unit
CLOSE	Terminates the connection of a dataset to a unit
INQUIRE	Returns status of a unit or a dataset
BACKSPACE	Positions a dataset after the previous end-of-record (EOR)
REWIND	Rewinds a dataset
ENDFILE	Writes end-of-file (EOF) on a file
UNIT	Returns I/O status upon completion of an I/O operation
LENGTH	Returns the number of Cray words transferred

IMPLEMENTATION

The preceding ANSI standard Fortran routines are available to users of both the COS and UNICOS operating systems.

The following routine types are described by entries in this section: copy, skip, dataset positioning, termination, and I/O status routines.

Copy routines copy a specified number of records or files from one dataset to another, copy one dataset to another, and copy a specified number of sectors or all data to end-of-data (EOD).

Skip routines direct the system either to bypass a specified number of records, files, sectors, or all data from the current position of a named dataset, or to position a blocked dataset at EOD.

The termination routine **EODW** terminates a dataset by writing EOF, EOR, and EOD. It also clears the uncleared End-of-file flag (**UEOF**) in the Dataset Parameter Table (DSP).

The last group of dataset utility routines return I/O information.

The following table contains the name, purpose, and entry for each dataset utility routine.

Dataset Utility Routines		
Purpose	Name	Entry
Position a dataset after the previous EOF and clear the **UEOF** flag in the DSP	**BACKFILE**	**BACKFILE**
Copy records from one dataset to another	**COPYR** **COPYSR**	**COPYR**
Copy files from one dataset to another	**COPYF** **COPYSF**	
Copy one dataset to another	**COPYD** **COPYSD**	
Copy sectors or all data to EOD	**COPYU**	**COPYU**
Terminate a dataset by writing EOD, EOF, and EOR and clear the **UEOF** flag in the DSP	**EODW**	**EODW**
Return the real value EOF status and clear the **UEOF** flag in the DSP	**EOF**	**EOF**
Return the integer value EOF status and clear the **UEOF** flag in the DSP	**IEOF**	
Return EOF and EOD status	**IOSTAT**	**IOSTAT**
Return the current size of a dataset in 512-word blocks	**NUMBLKS**	**NUMBLKS**
Skip records	**SKIPR**	**SKIPR**
Skip files	**SKIPF**	
Position a blocked dataset at EOD	**SKIPD**	**SKIPD**
Skip sectors in a dataset	**SKIPU**	**SKIPU**

NAME

BACKFILE – Positions a dataset after the previous EOF

SNYOPSIS

CALL BACKFILE(*dn*)

DESCRIPTION

dn Dataset name or unit number of the dataset to be repositioned

BACKFILE positions a dataset after the previous end-of-file (EOF) and then clears the UEOF flag in the Dataset Parameter Table (DSP).

This function is nonoperational if the dataset is at beginning-of-data (BOD).

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

COPYR, COPYF, COPYD – Copies records, files, or a dataset from one dataset to another

SYNOPSIS

CALL COPYR(*idn,odn,record*[,*istat*])
CALL COPYSR(*idn,odn,record,scount*[,*istat*])

CALL COPYF(*idn,odn,file*[,*istat*])
CALL COPYSF(*idn,odn,file,scount*[,*istat*])

CALL COPYD(*idn,odn*)
CALL COPYSD(*idn,odn,scount*)

DESCRIPTION

idn Dataset name or unit number of the dataset to be copied

odn Dataset name or unit number of the dataset to receive the copy

record Number of records to be copied

file Number of files to be copied

scount Number of ASCII blanks to be inserted at the beginning of each line of text

istat A two-element integer array that returns the number of records copied in the first element and the number of files copied in the second element. (For **COPYR**, the number of files copied is always 0.) *istat* is an optional parameter. If present, only fatal messages are written to the log file.

COPYR and **COPYF** copy a specified number of records or files from one dataset to another, starting at the current dataset position. Following the copy, the datasets are positioned after the EOR or EOF for the last record or file copied.

COPYD copies one dataset to another, starting at their current positions. Following the copy, both datasets are positioned after the EOF of the last file copied. The EOD is not written to the output dataset.

COPYSR, COPYSF, and **COPYSD** are the same as **COPYR, COPYF,** and **COPYD**, respectively, except that the copied data is preceded by *scount* blanks.

CAUTION

These routines are not intended for use with foreign dataset translation. When foreign dataset record boundaries coincide with Cray dataset record boundaries, proper results may be expected. However, it is difficult in general to determine when such coincidences occur. Use of these routines with foreign datasets is discouraged.

IMPLEMENTATION

These routines are available only to users of the COS operating system.

SEE ALSO

COPYU, SKIPR, SKIPD, SKIPU

NAME

COPYU – Copies either specified sectors or all data to EOD

SYNOPSIS

CALL COPYU(*idn*,*odn*,*ns*[,*istat*])

DESCRIPTION

idn Name of the unblocked dataset to be copied

odn Name of the unblocked dataset to receive the copy

ns Decimal number of sectors to copy. If the unblocked dataset contains fewer than *ns* sectors, the copy terminates at EOD. The entire dataset is copied if -1 is specified. If COPYU is called with only two parameters, only one sector is copied.

istat An integer array or variable that returns the number of sectors copied. *istat* is an optional parameter. If *istat* is present, only fatal messages are written to the log file.

Copying begins at the current position on both datasets. Following the copy, the datasets are positioned after the last sector copied.

CAUTION

This routine is not intended for use with foreign dataset translation.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

COPYR, SKIPU

NAME

EODW – Terminates a dataset by writing EOD, EOF, and EOR

SYNOPSIS

CALL EODW(*dn*)

DESCRIPTION

dn Dataset name or unit number of the dataset to be terminated

EODW writes an EOD, and, if necessary, an EOF and an EOR. The **UEOF** flag in the DSP is cleared.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

EOF, IEOF – Returns real or integer value EOF status

SYNOPSIS

rexit=**EOF**(*dn*)

iexit=**IEOF**(*dn*)

DESCRIPTION

rexit

-1.0	EOD on the last operation
0.0	Neither EOD nor EOF on the last operation
+1.0	EOF on the last operation

iexit

-1	EOD on the last operation
0	Neither EOD nor EOF on the last operation
+1	EOF on the last operation

dn Dataset name or unit number

EOF returns one of the above real values when checking the EOF status. **IEOF** returns one of the above integer values when checking the EOF status. Under COS, both routines clear the **UEOF** flag in the DSP.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

IOSTAT – Returns EOF and EOD status

SYNOPSIS

iexit=IOSTAT(*dn*)

DESCRIPTION

iexit	0	No error
	1	Dataset at EOF (UEOF cleared)
	2	Dataset at EOD (UEOF cleared)
dn	Dataset name or unit number	

IMPLEMENTATION

This routine is only available to users of the COS operating system.

NAME

NUMBLKS – Returns the current size of a dataset in 512-word blocks

SYNOPSIS

val=**NUMBLKS**(*dn*)

DESCRIPTION

val Number of blocks returned as an integer value. The value returned reflects only the data actually written to disk and does not take into account data still in the buffers. If the dataset is not local to the job, or has never been written to, a function value of 0 is returned. A negative value indicates that the underlying system call failed.

dn Dataset name or unit number

IMPLEMENTATION

This routine is available to users of the both the COS and UNICOS operating systems.

NAME

SKIPD – Positions a blocked dataset at EOD

SYNOPSIS

CALL SKIPD(*dn*[,*istat*])

DESCRIPTION

dn Dataset name or unit number to be skipped. Must be a character constant, an integer variable, or an array element containing Hollerith data of not more than 7 characters.

istat A two-element integer array that returns the number of records skipped in the first element and the number of files skipped in the second element. *istat* is an optional parameter. If it is present, only fatal messages are written to the log file.

SKIPD directs the system to position a blocked dataset at EOD, that is, after the last EOF of the dataset. If the specified dataset is empty or is already at EOD, the call has no effect.

CAUTION

This routine is not intended for use with foreign dataset translation.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

COPYR, SKIPR, SKIPU

NAME

SKIPR, SKIPF – Skip records or files

SYNOPSIS

CALL SKIPR(*dn,record*[*,istat*])

CALL SKIPF(*dn,file*[*,istat*])

DESCRIPTION

dn Dataset name or unit number that contains the record or file to be skipped. Must be a character constant, an integer variable, or an array element containing Hollerith data of not more than 7 characters. If *dn* is opened before **SKIPR** or **SKIPF** is called, *dn* must be opened to allow read or read/write access.

record Decimal number of records to be skipped. The default is 1. If *record* is negative, **SKIPR** skips backward on *dn*.

file Decimal number of files to be skipped. The default is 1. If *file* is negative, **SKIPR** skips backward on *dn*. If *dn* is positioned midfile, the partial file skipped counts as one file.

istat A two-element integer array that returns the number of records skipped in the first element and the number of files skipped in the second element. (For **SKIPR**, the number of files skipped is always 0.) *istat* is an optional parameter. If it is present, only fatal messages are written to the log file.

SKIPR directs the system to bypass a specified number of records from the current position of the named blocked dataset.

SKIPR does not bypass EOF or beginning-of-data (BOD). If an EOF or BOD is encountered before *record* records have been bypassed when skipping backward, the dataset is positioned after the EOF or BOD. When skipping forward, the dataset is positioned after the last EOR of the current file.

SKIPF directs the system to skip a specified number of files from the current position of the named blocked dataset.

SKIPF does not skip EOD or BOD. If a BOD is encountered before *file* files have been skipped when skipping backward, the dataset is positioned after the BOD. When skipping forward, the dataset is positioned before the EOD of the current file.

CAUTION

These routines are not intended for use with foreign dataset translation. When foreign dataset record boundaries coincide with Cray dataset record boundaries, proper results may be expected. However, it is difficult in general to determine when such coincidences occur. Use of these routines with foreign datasets is discouraged.

EXAMPLE

If the dataset connected to unit **FT07** is positioned just after an EOF, the following Fortran call positions the dataset after the previous EOF. If the dataset is positioned midfile, it is positioned at the beginning of that file.

```
CALL SKIPF('FT07',-1)
```

IMPLEMENTATION

These routines are available only to users of the COS operating system.

SEE ALSO

COPYR, SKIPD, SKIPU

NAME

SKIPU – Skips a specified number of sectors in a dataset

SYNOPSIS

CALL SKIPU(*dn*,*ns*[,*istat*])

DESCRIPTION

dn Dataset name or unit number of the unblocked dataset to be bypassed. Must be an integer variable or an array element containing ASCII data of not more than 7 characters.

ns Decimal number of sectors to bypass. The default value is 1. If *ns* is negative, SKIPU skips backward on *dn*.

istat An integer array or variable that returns the number of sectors skipped. *istat* is an optional parameter. If it is present, only fatal messages are written to the logfile.

SKIPU directs the system to bypass a specified number of sectors or all data from the current position of the named unblocked dataset.

CAUTION

This routine is not intended for use with foreign dataset translation.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

COPYU, SKIPR, SKIPD

14. MULTITASKING ROUTINES

Multitasking routines create and synchronize parallel tasks within programs. They are grouped in the following categories:

- Task routines
- Lock routines
- Event routines
- History trace buffer routines
- Barrier routines

For further information on using these subprograms in a multitasking environment, see the CRAY Y-MP and CRAY X-MP Multitasking Programmer's Manual, publication SR-0222.

TASK ROUTINES

Task routines handle tasks and task-related information.

TASK CONTROL ARRAY - Each user-created task is represented by an integer task control array, constructed by the user program. At a minimum, the array must consist of 2 Cray words; however, a third word can be included. The three words composing the array contain the following information:

LENGTH — Length of the array in Cray words. The length must be set to a value of 2 or 3, depending on the optional presence of the task value field. Set the LENGTH field before creating the task.

TASK ID — A task identifier assigned by the multitasking library when a task is created. This identifier is unique among active tasks within the job step. The multitasking library uses this field for task identification, but the task identifier is of limited use to the user program.

TASK VALUE (optional field)
This field can be set to any value before the task is created. If TASK VALUE is used, LENGTH must be set to a value of 3. The task value can be used for any purpose. Suggested values include a programmer-generated task name or identifier or a pointer to a task local-storage area. During execution, a task can retrieve this value with the TSKVALUE subroutine.

The following example sets parameters for the task control array TASKARY:

```
      PROGRAM MULTI
      INTEGER TASKARY(3)
C
C     SET TASKARY PARAMETERS
      TASKARY(1)=3
      TASKARY(3)='TASK 1'
C     ...
      END
```

TASK SUBROUTINES - The following table contains the purpose, name, and entry of each task routine.

Task Routines		
Purpose	Name	Entry
Initiate a task	**TSKSTART**	**TSKSTART**
Indicate whether a task exists	**TSKTEST**	**TSKTEST**
Modify tuning parameters within the library scheduler	**TSKTUNE**	**TSKTUNE**
Wait for a task to complete execution	**TSKWAIT**	**TSKWAIT**
Retrieve the user identifier specified in the task control array	**TSKVALUE**	**TSKVALUE**

LOCK ROUTINES

Lock routines protect critical regions of code and shared memory.

The following table contains the purpose, name, and entry of each lock routine.

Lock Routines		
Purpose	Name	Entry
Identify an integer variable to be used as a lock	**LOCKASGN**	**LOCKASGN**
Set a lock and return control to the calling task	**LOCKON**	**LOCKON**
Clear a lock and return control to the calling task	**LOCKOFF**	**LOCKOFF**
Release the identifier assigned to a lock	**LOCKREL**	**LOCKREL**
Test a lock to determine its state (locked or unlocked)	**LOCKTEST**	**LOCKTEST**

EVENT ROUTINES

Event routines signal and synchronize between tasks.

The following table contains the purpose, name, and entry of each event routine.

Event Routines		
Purpose	Name	Entry
Post an event and return control to the calling task	**EVPOST**	**EVPOST**
Clear an event and return control to the calling task	**EVCLEAR**	**EVCLEAR**
Identify a variable to be used as an event	**EVASGN**	**EVASGN**
Release the identifier assigned to a task	**EVREL**	**EVREL**
Test an event to determine its posted state	**EVTEST**	**EVTEST**
Delay the calling task until an event is posted	**EVWAIT**	**EVWAIT**

MULTITASKING HISTORY TRACE BUFFER ROUTINES

The user-level routines for the multitasking history trace buffer can be called from a user program to control what is recorded in the buffer and to dump the contents of the buffer to a dataset.

The following table contains the purpose, name, and entry of each multitasking history trace buffer routine.

Multitasking History Trace Buffer Routines		
Purpose	Name	Entry
Modify parameters used to control which multitasking actions are recorded in the history trace buffer	**BUFTUNE**	**BUFTUNE**
Write a formatted dump of the history trace buffer to a dataset	**BUFPRINT**	**BUFPRINT**
Write an unformatted dump of the history trace buffer to a dataset	**BUFDUMP**	**BUFDUMP**
Add entries to the history trace buffer	**BUFUSER**	**BUFUSER**

BARRIER ROUTINES

A barrier is a synchronization point in an application, beyond which no task will proceed until a specified number of tasks have reached the barrier.

The following table contains the purpose, name, and entry of each barrier routine.

Barrier Routines		
Purpose	Name	Entry
Identify an integer variable to use as a barrier	**BARASGN**	**BARASGN**
Register the arrival of a task as a barrier	**BARSYNC**	**BARSYNC**
Release the identifier assigned to a barrier	**BARREL**	**BARREL**

NAME

BARASGN – Identifies an integer variable to use as a barrier

SYNOPSIS

CALL BARASGN(*name*,*value*)

DESCRIPTION

name Integer variable to be used as a barrier. The library stores an identifier into this variable. Do not modify the variable after the call to **BARASGN** unless a call to **BARREL** first releases the variable.

value The integer number of tasks, between 1 and 31 inclusive, must call **BARSYNC** with *name* before the barrier is opened and the waiting tasks allowed to proceed.

Before an integer variable can be used as an argument to any of the other barrier routines, it must first be identified as a barrier variable by **BARASGN**.

IMPLEMENTATION

This routine is available both to users of the COS and UNICOS operating systems.

NAME

BARREL – Releases the identifier assigned to a barrier

SYNOPSIS

CALL BARREL(*name*)

DESCRIPTION

name Integer variable used as a barrier

IMPLEMENTATION

This routine is available both to users of the COS and UNICOS operating systems.

NAME

BARSYNC – Registers the arrival of a task at a barrier

SYNOPSIS

CALL BARSYNC(*name*)

DESCRIPTION

name Integer variable used as a barrier

IMPLEMENTATION

This routine is available both to users of the COS and UNICOS operating systems.

NAME

BUFDUMP – Unformatted dump of multitasking history trace buffer

SYNOPSIS

CALL BUFDUMP(*empty,dn*)

DESCRIPTION

empty On entry, an integer flag that is 0 if the buffer pointers are to be left unchanged, nonzero if the buffer is to be emptied after its contents are dumped

dn Name of the dataset to which an unformatted dump of the contents of the multitasking history trace buffer is to be written. If 0, the dataset passed to **BUFTUNE** is used; if no dataset was specified through **BUFTUNE**, the request is ignored.

BUFDUMP writes an unformatted dump of the contents of the multitasking history trace buffer to a specified dataset. *dn* can later be used by **MTDUMP** to examine the dataset and provide formatted reports of its contents. Actions are reported in chronological order. A special entry is added if the buffer has overflowed and entries have been lost.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

BUFPRINT – Formatted dump of multitasking history trace buffer to a specified dataset

SYNOPSIS

CALL BUFPRINT(*empty*[,*dn*])

DESCRIPTION

empty On entry, an integer flag that is 0 if the buffer pointers are to be left unchanged or nonzero if the buffer is to be emptied after its contents are printed

dn Name of the dataset or file to which a formatted dump is to be written. If none is specified, **$OUT** (under COS) or **stdout** (under UNICOS) is used.

BUFPRINT writes a formatted dump of the contents of the multitasking history trace buffer to a specified dataset. Actions are reported in chronological order.

EXAMPLE

This example of **BUFPRINT** leaves the buffer unchanged after its output to **$OUT**:

```
IEMPTY = 0
CALL BUFPRINT(IEMPTY)
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

BUFDUMP

NAME

BUFTUNE – Tune parameters controlling multitasking history trace buffer

SYNOPSIS

CALL BUFTUNE(*keyword,value*[,*string*])

DESCRIPTION

keyword	ASCII string, left-justified, blank-filled (see keywords following)
value	Either an integer or an ASCII string (left-justified, blank-filled), depending on the keyword
string	A 24-character string (left-justified, blank-filled) used only with the keyword **INFO**

Valid keywords and their associated functions and meanings are as follows:

Keyword	Description
DN	The value of the **DN** keyword is the dataset which you specify to receive a dump of the multitasking history trace buffer. **DN** itself directs this dump of the buffer to the dataset. If **BUFTUNE** is called without the **DN** keyword, the multitasking history trace buffer is not dumped to any dataset.
FLUSH	The minimum-allowed integer number of unused entries in the multitasking history trace buffer. When the number of unused entries falls below this level, the buffer is automatically flushed; that is, it is written to the dataset specified by the **DN** option. If **DN** is specified, the default **FLUSH** value is 40.
ACTIONS	Value is a 128-element integer array with a flag for each action that can be recorded in the multitasking history trace buffer. If the array element corresponding to a particular action is nonzero, that action is recorded; if the array element is 0, the action is ignored. The array indexes (action codes) corresponding to each action follow:

Action Code	Action
1	Start task
2	Complete task
3	**TSKWAIT**, no wait
4	Begin wait for task
5	Run after wait for task
6	Test task
7	Assign lock
8	Release lock
9	Set lock
10	Begin wait to set lock
11	Run after wait for lock
12	Clear lock
13	Test lock

Action Code	Action
14	Assign event
15	Release event
16	Post event
17	Clear event
18	**EVWAIT**, no wait
19	Begin wait for event
20	Run after wait for event
21	Test event
22	Attach to logical CPU
23	Detach from logical CPU
24,25	Request a logical CPU (Note that these actions require two action codes, the second containing internal information.)
26	Acquire a logical CPU
27,28	Delete a logical CPU (Note that these actions require two action codes, the second containing internal information.)
29,30	Suspend a logical CPU (Note that these actions require two action codes, the second containing internal information.)
31,32	Activate a logical CPU (Note that these actions require two action codes, the second containing internal information.)
33	Begin spin-wait for a logical CPU
34	Assign barrier
35	Release barrier
36	Call **BARSYNC**, no wait
37	Begin wait at barrier
38	Run after wait for barrier
39-64	Reserved for future use
65-128	Reserved for user access (see **BUFUSER**)

INFO The value for this parameter is the integer user action code (65 through 128).

string is a 24-character information string, unique to each action, that you enter and is printed for each user action code that is dumped.

BUFUSER allows you to add entries to the multitasking history trace buffer. When the multitasking history trace buffer is dumped using **DEBUG**, **BUFPRINT**, or **MTDUMP**, this 24-character information string is dumped along with each action. This information must be available early in the program so that the strings can be written to the dump dataset for processing by **MTDUMP**. The **INFO** keyword does not turn these actions on to be recorded. They are normally on by default, but if you have previously turned them off, you may reactivate them using the **ACTIONS** or **USERS** keyword in a **BUFTUNE** call.

Keyword	Description
TASKS	If value='ON'H, the actions numbered 1 through 6 are recorded; if value='OFF'H, those actions are ignored. The default is 'ON'H.
LOCKS	If value='ON'H, the actions numbered 7 through 13 are recorded; if value='OFF'H, those actions are ignored. The default is 'ON'H.
EVENTS	If value='ON'H, the actions numbered 14 through 21 are recorded; if value='OFF'H, those actions are ignored. The default is 'ON'H.
CPUS	If value='ON'H, the actions numbered 22 through 33 are recorded; if value='OFF'H, those actions are ignored. The default is 'ON'H.
USERS	If value='ON'H, the actions numbered 65 through 128 are recorded; if value='OFF'H, those actions are ignored. The default is value='ON'H.
FIOLK	If value='ON'H, actions affecting the Fortran I/O lock are recorded; if value='OFF'H they are ignored. Library routines that handle Fortran reads and writes use this lock. The default is 'OFF'H.

BUFTUNE can be called any number of times. If it is not called, or before it is called for the first time, default parameter values are used.

Before **BUFTUNE** is called, all actions involving tasks, locks, events, logical CPUs, and users are recorded except for actions involving the Fortran I/O lock, which are ignored. A call to **BUFTUNE** with the **TASKS**, **LOCKS**, **EVENTS**, **CPUS**, or **USERS** keyword affects only the actions associated with that keyword. The **ACTIONS** option overrides what has been requested through **TASKS**, **LOCKS**, **EVENTS**, **CPUS**, or **USERS**.

EXAMPLES

The following **BUFTUNE** examples turn on task actions and turn everything else off:

```
*  Example #1
   INTEGER ACTION (64)
   DATA ACTION(6*1,58*0)
   CALL BUFTUNE ('DN'L,'DMPFILE'L)

*  Example #2
   CALL BUFTUNE ('DN'L,'DMPFILE'L)
   CALL BUFTUNE ('TASKS'L,'ON'L)
   CALL BUFTUNE ('LOCKS'L,'OFF'L)
   CALL BUFTUNE ('EVENTS'L,'OFF'L)
   CALL BUFTUNE ('CPUS'L,'OFF'L)
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

BUFUSER – Adds entries to the multitasking history trace buffer

SYNOPSIS

CALL BUFUSER(*action,data***)**

DESCRIPTION

action On entry, code for the type of action (see action codes in **MTDUMP**). This value is compared against the bit of the same number in the mask in global variable **G@BUFMSK**, set up by **BUFTUNE**. If the mask bit is set, an entry is added to the buffer. This value becomes the third word of the buffer entry.

data Values added to the multitasking history trace buffer in addition to the internal task identifier and the current time. These actions-dependent data codes can be user-defined task values, a logical CPU number, a lock or event address, or the task identifier of the waited-upon task. The only restriction on these values is that they should be a single word. If an entry is added to the buffer, this value becomes the fourth word of the entry.

These entries are added unconditionally.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

EVASGN – Identifies an integer variable to be used as an event

SYNOPSIS

CALL EVASGN(*name*[,*value*])

DESCRIPTION

name Name of an integer variable to be used as an event. The library stores an identifier into this variable; you should not modify this variable.

value The initial integer value of the event variable. An identifier should be stored into the variable only if it contains the value. If *value* is not specified, an identifier is stored into the variable unconditionally.

Before an integer variable can be used as an argument to any of the other event routines, it must first be identified as an event variable by EVASGN.

EXAMPLE

```
      PROGRAM MULTI
      INTEGER EVSTART,EVDONE
      COMMON /EVENTS/ EVSTART,EVDONE
C     ...
      CALL EVASGN (EVSTART)
      CALL EVASGN (EVDONE)
C     ...
      END
      SUBROUTINE SUB1
      INTEGER EVENT1
      COMMON /EVENT1/ EVENT1
      DATA EVENT1 /-1/
C     ...
      CALL EVASGN (EVENT1,-1)
C     ...
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

EVCLEAR – Clears an event and returns control to the calling task

SYNOPSIS

CALL EVCLEAR(*name*)

DESCRIPTION

name Name of an integer variable used as an event

EVCLEAR clears an event and returns control to the calling task. When the posting of a single event is required (a simple signal), **EVCLEAR** should be called immediately after **EVWAIT** to note that the posting of the event has been detected.

EXAMPLE

```
      PROGRAM MULTI
      INTEGER EVSTART,EVDONE
      COMMON /EVENTS/ EVSTART,EVDONE
C     ...
      CALL EVASGN (EVSTART)
      CALL EVASGN (EVDONE)
C     ...
      CALL EVPOST (EVSTART)
      END

      SUBROUTINE MULTI2
      INTEGER EVSTART,EVDONE
      COMMON /EVENTS/ EVSTART,EVDONE
C     ...
      CALL EVWAIT (EVSTART)
      CALL EVCLEAR (EVSTART)
C     ...
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

EVPOST – Posts an event and returns control to the calling task

SYNOPSIS

CALL EVPOST(*name*)

DESCRIPTION

name Name of an integer variable used as an event

EVPOST posts an event and returns control to the calling task. Posting the event allows any other tasks waiting on that event to resume execution, but this is transparent to the task calling **EVPOST**.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

EVREL – Releases the identifier assigned to the task

SYNOPSIS

CALL EVREL(*name*)

DESCRIPTION

name Name of an integer variable used as an event

If tasks are currently waiting for this event to be posted, an error results. This subroutine detects erroneous uses of the event beyond the specified region. The event variable can be reused following another call to **EVASGN**.

EXAMPLE

```
      PROGRAM MULTI
      INTEGER EVSTART,EVDONE
      COMMON /EVENTS/ EVSTART,EVDONE
C     ...
      CALL EVASGN (EVSTART)
      CALL EVASGN (EVDONE)
C     ...
      CALL EVPOST (EVSTART)
C     ...
C     EVSTART WILL NOT BE USED FROM NOW ON
      CALL EVREL (EVSTART)
C     ...
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

EVTEST – Tests an event to determine its posted state

SYNOPSIS

LOGICAL EVTEST
return=**EVTEST**(*name*)

DESCRIPTION

return	A logical **.TRUE.** if the event is posted. A logical **the event is not posted.**
name	Name of an integer variable used as an event

NOTE

EVTEST and *return* must be declared as type **LOGICAL** in the calling module.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

EVWAIT – Delays the calling task until the specified event is posted

SYNOPSIS

CALL EVWAIT(*name*)

DESCRIPTION

name Name of an integer variable used as an event

If the event is already posted, the task resumes execution without waiting.

EXAMPLE

```
      SUBROUTINE MULTI2
      INTEGER EVSTART,EVDONE
      COMMON /EVENTS/ EVSTART,EVDONE
C     ...
      CALL EVWAIT (EVSTART)
C     ...
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

JCCYCL – Returns machine cycle time

SYNOPSIS

INTEGER JCCYCL
integer = **JCCYCL()**

DESCRIPTION

integer Integer representing the cycle time of the machine in picoseconds.

JCCYCL returns the contents of the Job Control Block (JCB) field JCCYCL. For a CRAY X-MP computer system with a clock period of 8.5 nanoseconds, JCCYCL returns the integer 8,500.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

LOCKASGN – Identifies an integer variable intended for use as a lock

SYNOPSIS

CALL LOCKASGN(*name*[,*value*])

DESCRIPTION

name Name of an integer variable to be used as a lock. The library stores an identifier into this variable; you should not modify this variable.

value The initial integer value of the lock variable. An identifier should be stored into the variable only if it contains the value. If *value* is not specified, an identifier is stored into the variable unconditionally.

Before an integer variable can be used as an argument to any of the other lock routines, it must first be identified as a lock variable by LOCKASGN.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

LOCKOFF – Clears a lock and returns control to the calling task

SYNOPSIS

CALL LOCKOFF(*name*)

DESCRIPTION

name Name of an integer variable used as a lock

LOCKOFF clears a lock and returns control to the calling task.

Clearing the lock may allow another task to resume execution, but this is transparent to the task calling **LOCKOFF.**

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

LOCKON – Sets a lock and returns control to the calling task

SYNOPSIS

CALL LOCKON(*name*)

DESCRIPTION

name Name of an integer variable used as a lock

LOCKON sets a lock and returns control to the calling task.

If the lock is already set when **LOCKON** is called, the task is suspended until the lock is cleared by another task and can be set by this one. In either case, the lock will have been set by the task when it next resumes execution.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

LOCKREL – Releases the identifier assigned to a lock

SYNOPSIS

CALL LOCKREL(*name*)

DESCRIPTION

name Name of an integer variable used as a lock

If the lock is set when **LOCKREL** is called, an error results. This subroutine detects some errors that arise when a task is waiting for a lock that is never cleared. The lock variable can be reused following another call to **LOCKASGN**.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

LOCKTEST – Tests a lock to determine its state (locked or unlocked)

SYNOPSIS

LOGICAL LOCKTEST
return=**LOCKTEST**(*name*)

DESCRIPTION

return A logical **.TRUE.** if the lock was originally in the locked state. A logical **.FALSE.** if the lock was originally in the unlocked state, but has now been set.

name Name of an integer variable used as a lock

Unlike **LOCKON**, the task does not wait. A task using **LOCKTEST** must always test the return value before continuing.

NOTE

LOCKTEST and *return* must be declared type **LOGICAL** in the calling module.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

MAXLCPUS – Returns the maximum number of logical CPUs that can be attached at one time to your job

SYNOPSIS

INTEGER MAXLCPUS
integer = MAXLCPUS()

DESCRIPTION

integer Integer value for the maximum number of CPUs that can be attached at one time to your job.

MAXLCPUS returns the contents of the Job Control Block (JCB) field JCMCP.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

TSECND – Returns elapsed CPU time for a calling task during a multitasked program

SYNOPSIS

second=**TSECND**([*result*])

CALL TSECND(*second*)

DESCRIPTION

second Result; elapsed CPU time (in floating-point seconds)

result Same as above (optional for function call)

TSECND returns the elapsed CPU time (in floating-point seconds) of a calling process since the start of that process. than subsequent calls due to certain initializations performed by the routine. If the cost of calling **TSECND** is important, ignore the initial call when computing **TSECND**'s time.

EXAMPLE

The following example calculates how much of the total execution time for a multitasked program is accumulated by the calling process.

```
BEFORE = SECOND()
TBEFORE = TSECND()
CALL DOWORK()              ! The subroutine DOWORK or
AFTER = SECOND()           ! something it calls may be
TAFTER = TSECND()          ! multitasked.
CPU = (AFTER - BEFORE)
TCPU = (TAFTER - TBEFORE)
MYPORTION = TCPU/CPU
```

IMPLEMENTATION

This routine is available only to users of the UNICOS operating system.

SEE ALSO

SECOND(3U)

NAME

TSKSTART – Initiates a task

SYNOPSIS

CALL TSKSTART(*task-array*,*name*[,*list*])

DESCRIPTION

task-array Task control array used for this task. Word 1 must be set. Word 3, if used, must also be set. On return, word 2 is set to a unique task identifier that the program must not change.

name External entry point at which task execution begins. Declare this name **EXTERNAL** in the program or subroutine making the call to **TSKSTART**. (Fortran does not allow a program unit to use its own name in this parameter.)

list List of arguments being passed to the new task when it is entered. This list can be of any length. See the CRAY Y-MP and CRAY X-MP Multitasking Programmer's Manual, publication SR-0222, for restrictions on arguments included in *list* (optional parameter).

EXAMPLE

```
      PROGRAM MULTI
      INTEGER TASK1ARY(3),TASK2ARY(3)
      EXTERNAL PLLEL
      REAL DATA(40000)
C
C     LOAD DATA ARRAY FROM SOME OUTSIDE SOURCE
C     ...
C
C     CREATE TASK TO EXECUTE FIRST HALF OF THE DATA
C
      TASK1ARY(1)=3
      TASK1ARY(3)='TASK 1'
C
      CALL TSKSTART(TASK1ARY,PLLEL,DATA(1),20000)
C
C     CREATE TASK TO EXECUTE SECOND HALF OF THE DATA
C
      TASK2ARY(1)=3
      TASK2ARY(3)='TASK 2'
C
      CALL TSKSTART(TASK2ARY,FLLEL,DATA(20001),20000)
C     ...
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

TSKTEST – Returns a value indicating whether the indicated task exists

SYNOPSIS

LOGICAL TSKTEST
return=TSKTEST(*task-array*)

DESCRIPTION

return A logical **.TRUE.** if the indicated task exists. A logical **.FALSE.** if the task was never created or has completed execution.

task-array Task control array **TSKTEST** and *return* must be declared type **LOGICAL** in the calling module.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

TSKTUNE – Modifies tuning parameters within the library scheduler

SYNOPSIS

CALL TSKTUNE(*keyword*$_1$,*value*$_1$,*keyword*$_2$, *value*$_2$,...)

DESCRIPTION

Each keyword is a Fortran constant or variable of type **CHARACTER**. Each value is an integer. The parameters must be specified in pairs, but the pairs can occur in any order. Legal keywords are as follows:

Keyword	Description
MAXCPU	Maximum number of COS logical CPUs allowed for the job
DBRELEAS	Deadband for release of logical CPUs
DBACTIVE	Deadband for activation or acquisition of logical CPU
HOLDTIME	Number of clock periods to hold a CPU, waiting for tasks to become ready, before releasing it to the operating system
SAMPLE	Number of clock periods between checks of the ready queue

Each parameter has a default setting within the library and can be modified at any time to another valid setting.

For more information about using this routine, see the CRAY Y-MP and CRAY X-MP Multitasking Programmer's Manual, publication SR-0222.

NOTE

This routine should not be used when multitasking on a CRAY-1 computer system. Because of variability between and during runs, the effects of this routine are not reliably measurable in a batch environment.

EXAMPLE

```
CALL TSKTUNE('DBACTIVE',1,'MAXCPU',2)
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

TSKVALUE – Retrieves user identifier specified in task control array

SYNOPSIS

CALL TSKVALUE(*return*)

DESCRIPTION

return Integer value that was in word 3 of the task control array when the calling task was created. A 0 is returned if the task control array length is less than 3 or if the task is the initial task.

TSKVALUE retrieves the user identifier (if any) specified in the task control array used to create the executing task.

EXAMPLE

```
      SUBROUTINE PLLEL(DATA,SIZE)
      REAL DATA(SIZE)
C
C     DETERMINE WHICH OUTPUT FILE TO USE
C
      CALL TSKVALUE(IVALUE)
      IF(IVALUE .EQ. 'TASK 1')THEN
         IUNITNO=3
      ELSEIF(IVALUE .EQ. 'TASK 2')THEN
         IUNITNO=4
      ELSE
         STOP        !Error condition; do not continue.
      ENDIF
C     ...
      END
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

TSKWAIT – Waits for the indicated task to complete execution

SYNOPSIS

CALL TSKWAIT(*task-array***)**

DESCRIPTION

task-array Task control array

EXAMPLE

```
      PROGRAM MULTI
      INTEGER TASK1ARY(3),TASK2ARY(3)
      EXTERNAL PLLEL
      REAL DATA(40000)
C
C     LOAD DATA ARRAY FROM SOME OUTSIDE SOURCE
C     ...
C
C     CREATE TASK TO EXECUTE FIRST HALF OF THE DATA
C
      TASK1ARY(1)=3
      TASK1ARY(3)='TASK 1'
C
      CALL TSKSTART(TASK1ARY,PLLEL,DATA(1),20000)
C
C     CREATE TASK TO EXECUTE SECOND HALF OF THE DATA
C
      TASK2ARY(1)=3
      TASK2ARY(3)='TASK 2'
C
      CALL TSKSTART(TASK2ARY,PLLEL,DATA(20001),20000)
C     ...
C     NOW WAIT FOR BOTH TO FINISH
C
      CALL TSKWAIT(TASK1ARY)
      CALL TSKWAIT(TASK2ARY)
C
C     AND PERFORM SOME POST-EXECUTION CLEANUP
C     ...
C
      END
```

In the preceding example, **TSKSTART** is called once for each of two tasks. As an alternative, the second **TSKSTART** could be replaced by a call to **PLLEL**, and the **TSKWAIT** removed. This alternate approach reduces the overhead of the additional task but can make understanding the program structure more difficult. The two approaches, however, produce the same results.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

15. TIMING ROUTINES

The timing routines are grouped as follows:

- Time-stamp routines
- Time and date routines

TIME-STAMP ROUTINES

System accounting programs use these routines to convert between various representations of time. Time-stamps can be used to measure from one point in time to another. Cray time-stamps are defined relative to an initial date of January 1, 1973.

The following table contains the purpose, name, and entry for each time-stamp routine.

Time-stamp Routines		
Purpose	Name	Entry
Convert from date and time to time-stamp	**DTTS**	**DTTS**
Convert time-stamps into ASCII date and time strings	**TSDT**	**TSDT**
Convert time-stamp to real-time clock value	**TSMT**	**TSMT**
Convert real-time clock value to time-stamp	**MTTS**	
Return time-stamp units in standard time units	**UNITTS**	**UNITTS**

TIME AND DATE ROUTINES

Time and date routines produce the time and/or date in specified forms. These routines can be called as Fortran functions or routines. All of the routines are called by address.

The following table contains the purpose, name, and entry for each time and date routine.

Time and Date Routines		
Purpose	Name	Heading
Return the current system clock time	**CLOCK**	**CLOCK**
Return the current date	**DATE**	**DATE**
Return the current Julian date	**JDATE**	
Return real-time clock values	**RTC** **IRTC**	**RTC**
Return the elapsed CPU time (in floating-point seconds) since the start of a job	**SECOND**	**SECOND**
Return the elapsed wall-clock time since the initial call to **TIMEF**	**TIMEF**	**TIMEF**
Return the CPU time (in floating-point seconds) remaining for a job	**TREMAIN**	**TREMAIN**

NAME

CLOCK – Returns the current system-clock time

SYNOPSIS

time=CLOCK( )
CALL CLOCK(*time*)

DESCRIPTION

time Time in *hh:mm:ss* format (type integer)

CLOCK returns the current system-clock time in ASCII *hh:mm:ss* format.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

DATE, JDATE – Returns the current date and the current Julian date

SYNOPSIS

date=**DATE()**

CALL DATE(*date*)

date=**JDATE()**

CALL JDATE(*date*)

DESCRIPTION

date For **DATE**, today's date in *mm/dd/yy* format (type integer). For **JDATE**, today's Julian date in *yyddd* format.

DATE returns today's date in *mm/dd/yy* format.

JDATE returns today's Julian (ordinal) date in *yyddd* format, left-justified, blank-filled.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

DTTS – Converts ASCII date and time to time-stamp

SYNOPSIS

ts=**DTTS**(*date*,*time*,*ts*)

DESCRIPTION

ts	Time-stamp corresponding to *date* and *time* (type integer). On return, if *ts*=0, an incorrect parameter was passed to **DTTS**.
date	On entry, ASCII date in *mm/dd/yy* format
time	On entry, ASCII time in *hh:mm:ss* format

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

RTC, IRTC – Return real-time clock values

SYNOPSIS

time=**RTC( )**
CALL RTC(*time***)**

time=**IRTC( )**
CALL IRTC(*time***)**

DESCRIPTION

time For **RTC**, the low-order 46 bits of the clock register expressed as a floating-point integer (real type). For **IRTC**, the current clock register content expressed as an integer.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

SECOND – Returns elapsed CPU time

SYNOPSIS

second=**SECOND**([*result*])

CALL SECOND(*second*)

DESCRIPTION

second Result; CPU time (in floating-point seconds) accumulated by all processes in a program.

result Same as above (optional for function call)

SECOND returns the elapsed CPU time (in floating-point seconds) since the start of a program, including time accumulated by all processes in a multitasking program.

Under COS, all programs run as job steps of a job, and **SECOND** returns the total execution time for all job steps since the job started. Under UNICOS, **SECOND** returns execution time for the current program. For example, a job (COS or UNICOS) runs a 50-second program 10 times. In COS, if you make a **SECOND** call at the end of the 10th run, **SECOND** will return 500 seconds. In UNICOS, a **SECOND** call at the end of the 10th run (or first or third or seventh) will return 50 seconds.

NOTE

The initial call to **SECOND** may take longer than subsequent calls due to certain initializations performed by the routine. If the cost of calling **SECOND** is important, ignore the initial call when computing **SECOND**'s time. The assignment to **JUNK** in the second example below serves this purpose.

EXAMPLE

```
BEFORE = SECOND()
CALL DOWORK()
AFTER = SECOND()
CPUTIME = AFTER - BEFORE
```

This example calculates the CPU time used in **DOWORK**. If the CPU time is small enough that the overhead for calling **SECOND** may be significant, the following example is more accurate:

```
JUNK = SECOND()
T0 = SECOND()
OVERHEAD = SECOND() - T0
BEFORE = SECOND()
CALL DOWORK()
AFTER = SECOND()
CPUTIME = (AFTER - BEFORE) - OVERHEAD
```

IMPLEMENTATION

This routine is available to users of both the UNICOS and COS operating systems.

SEE ALSO

TSECND(3U)

NAME

TIMEF – Returns elapsed wall-clock time since the call to **TIMEF**

SYNOPSIS

timef=TIMEF([*result*])
CALL TIMEF(*timef*)

DESCRIPTION

timef Elapsed wall-clock time (in floating-point milliseconds) since the initial call to **TIMEF**. Type real. The initial call to **TIMEF** returns 0.

result Same as *timef*

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

TREMAIN – Returns the CPU time (in floating-point seconds) remaining for job

SYNOPSIS

CALL TREMAIN(*result*)

DESCRIPTION

result Calculated CPU time remaining; stored in *result*. Type real.

NOTE

The time remaining is the time specified on the COS **JOB** statement, minus the time elapsed so far.

The value returned by **TREMAIN** may not always be updated between calls. For instance, the values for **X** and **Y** may be the same in the following code:

```
      CALL TREMAIN(X)
      DO 10 I = 1, 1000000
10       T(I) = FLOAT(I)
      CALL TREMAIN(Y)
```

The value that **TREMAIN** uses is only updated when a program is exchanged out of memory. If calls to **TREMAIN** occur during the same time slice (that is, the job has not been exchanged), the values will be the same. If more accurate times are required, use the routine **SECOND and subtract the value from your job's time limit.**

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

TSDT – Converts time-stamps to ASCII date and time strings

SYNOPSIS

CALL TSDT(*ts,date,hhmmss,ssss***)**

DESCRIPTION

ts	Time-stamp on entry (type integer)
date	Word to receive ASCII date in *mm/dd/yy* format
hhmmss	Word to receive ASCII time in *hh:mm:ss* format
ssss	Word to receive ASCII fractional seconds in *.ssssnnn* format

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

TSMT, MTTS – Converts time-stamp to a corresponding real-time value, and vice versa

SYNOPSIS

irtc=**TSMT**(*ts*[,cptype,cpcycle])
ts=**MTTS**(*irtc[,cptype,cpcycle]*)

DESCRIPTION

irtc For **TSMT**, real-time clock value corresponding to specified time-stamp. For **MTTS**, real-time clock value to be converted.

ts For **TSMT**, time-stamp to be converted (type integer). For **MTTS**, time-stamp corresponding to real-time clock value (type integer).

cptype CPU type. This is an optional argument specifying the CPU type. Valid values are as follows:

1 CRAY-1, models A and B
2 CRAY-1, model S
3 CRAY X-MP
4 CRAY-1, model M

The default is the CPU of the host machine. The *cptype* is necessary when doing a conversion for a machine type other than the host machine. The real-time clock value is different on, for instance, a CRAY X-MP computer system than on a CRAY-1 computer system because of the difference in cycle time. For **TSMT** to generate a correct result and for **MTTS** to correctly interpret its argument, they must know the correct machine type.

cpcycle CPU cycle time in picoseconds; for instance, a CRAY X-MP computer system with a cycle time of 8.5 nanoseconds would be specified as 8500. The default is the cycle time of the host machine.

TSMT converts a time-stamp to a corresponding real-time value. **MTTS** converts a real-time clock value to its corresponding time-stamp.

IMPLEMENTATION

These routines are available only to users of the COS operating system.

NAME

UNITTS – Returns time-stamp units in specified standard time units

SYNOPSIS

ts=UNITTS(*periods*,*units*)

DESCRIPTION

ts Number of time-stamp units in *periods* and *units* (type integer)

periods Number of time-stamp units to be returned in standard time units (that is, number of seconds, minutes, and so on); type integer.

units Specification for the units in which *periods* is expressed. The following values are accepted: **'DAYS'H, 'HOURS'H, 'MINUTES'H, 'SECONDS'H, 'MSEC'H** (milliseconds), **'USEC'H** (microseconds), **'USEC100'H** (100s of microseconds). Left-justified, blank-filled, Hollerith. UNITTS must be declared type integer.

EXAMPLE

ts=UNITTS(2,'DAYS'H)

ts Number of time-stamp units in 2 days

IMPLEMENTATION

This routine is available only to users of the COS operating system.

16. PROGRAMMING AID ROUTINES

Programming aids consist of the following types of routines:

- Flowtrace routines
- Traceback routines
- Dump routines
- Exchange Package processing routines
- Hardware performance monitor interface routine

FLOWTRACE ROUTINES

Flowtrace routines process the CFT flowtrace option (ON=F). The Cray Fortran compiler automatically inserts calls to these routines (see the Fortran (CFT) Reference Manual, or the CFT77 Reference Manual for details on flowtracing). Flowtrace routines are called by address. For more information on flow trace calls from CAL, see the System Library Reference Manual, publication SM-0114, the UNICOS Performance Utilities Reference Manual, publication SR-2040, and the COS Performance Utilities Reference Manual, publication SR-0146.

NOTE

Many of the flowtrace subroutines begin with the characters "FLOW0". You should avoid using names with this prefix.

IMPLEMENTATION - The flowtrace routines are available to users of both the COS and UNICOS operating systems.

The following table contains the purpose, name, and call to each flow trace routine.

Flowtrace Routines	
Purpose	Name and Call
Process entry to a subroutine	**CALL FLOWENTR**
Process **RETURN** execution	**CALL FLOWEXIT**
Process a **STOP** statement	**CALL FLOWSTOP**
Initiate a detailed tracing of every call and return	**SETPLIMQ(*lines*)** *lines* Cycle count when execution of caller ceased
Print the final report	**CALL FLOW0STP(*outdev*)** *outdev* Device to which the report is written
Return the cycles charged to a job	*integer*=**IGETSEC()**
Return the cycle time in picoseconds (value of field JCCYCL in the JCB)	*integer*=**JCCYCL()**

PERFTRACE ROUTINES

The perftrace routines output detailed information from the Hardware Performance Monitor Interface for individual segments of a Fortran Program. These routines have the same interfaces as the flowtrace routines, which are described in the UNICOS Performance Utilities Reference Manual, publication SR-2040.

IMPLEMENTATION - The perftrace subroutines are available only to the users of the UNICOS operating system.

TRACEBACK ROUTINES

The traceback routines list all subroutines active in the current calling sequence (**TRBK**) and return information for the current level of the calling sequence (**TRBKLVL**). Traceback routines return unpredictable results when subroutine linkage does not use CRI standard calling sequences.

DUMP ROUTINES

Dump routines produce a memory image and are called by address.

The following table contains the purpose, name, and entry of each dump routine.

Dump Routines		
Purpose	Name	Entry
Print a memory dump to a dataset	**CRAYDUMP**	**CRAYDUMP**
Dump memory to $OUT and abort the job	**DUMP**	**DUMP**
Dump memory to $OUT and return control to the calling program	**PDUMP**	
Create an unblocked dataset containing the user job area image	**DUMPJOB**	**DUMPJOB**
Copy current register contents to $OUT	**SNAP**	**SNAP**
Produce a symbolic dump	**SYMDEBUG**	**SYMDEBUG**
Produce a snapshot dump of a running program	**SYMDUMP**	**SYMDUMP**

EXCHANGE PACKAGE PROCESSING ROUTINES

Exchange Package processing routines (**XPFMT** and **FXP**) switch execution from one program to another. An Exchange Package is a 16-word block of memory associated with a particular program.

HARDWARE PERFORMANCE MONITOR INTERFACE ROUTINE

PERF provides an interface to the hardware performance monitor feature on CRAY X-MP computer systems.

NAME

CRAYDUMP – Prints a memory dump to a specified dataset

SYNOPSIS

CALL CRAYDUMP(*fwa,lwa,dn*)

DESCRIPTION

fwa	First word to be dumped
lwa	Last word to be dumped
dn	Name or unit number of the dataset to receive the dump output

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

DUMP, PDUMP – Dumps memory to $OUT and either abort or return to the calling program

SYNOPSIS

CALL DUMP(*fwa,lwa,type***)**
CALL PDUMP(*fwa,lwa,type***)**

DESCRIPTION

fwa	First word to be dumped
lwa	Last word to be dumped
type	Dump type code, as follows:

0 or 3	Octal dump
1	Floating-point dump
2	Integer dump

DUMP dumps memory to **$OUT** and aborts the job. **PDUMP** dumps memory to **$OUT** and returns control to the calling program.

NOTES

If 4 is added to the dump type code, the first word and last word addresses specified are then addresses of addresses (indirect addressing).

First word/last word/dump type address sets can be repeated up to 19 times.

IMPLEMENTATION

These routines are available only to users of the COS operating system.

NAME

DUMPJOB – Creates an unblocked dataset containing the user job area image

SYNOPSIS

CALL DUMPJOB(*dn*)

DESCRIPTION

dn Fortran unit number or Hollerith unit name. If no parameter is supplied, **$DUMP** is used by default.

DUMPJOB creates an unblocked dataset containing the user job area image, including register states and the Job Table Area. This data is suitable for input to the **DUMP** or **DEBUG** programs.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

DUMP, SYMDEBUG

NAME

FXP – Formats and writes the contents of the Exchange Package to an output dataset

SYNOPSIS

CALL FXP(*dsp*,*xp*,*vm*,*ret*)

DESCRIPTION

dsp	Output Dataset Parameter Table address
xp	Exchange Package address
vm	Vector mask (VM) to be formatted
ret	Contents of B0 register to be formatted

FXP formats and writes to the output dataset the contents of the Exchange Package, the contents of the vector mask (VM), and the contents of the B0 register. This routine complements the user reprieve processing.

IMPLEMENTATION

This routine is available only to users of the COS operating system. **FXP** formats and writes to the output dataset the contents

NAME

PERF – Provides an interface to the hardware performance monitor feature on the CRAY X-MP mainframe

SYNOPSIS

CALL PERF(*func,group,buffer,bufl*)

DESCRIPTION

func Performance monitor function. Either an integer function number or one of the following ASCII strings, left-justified, and zero-filled.

'ON'L	Enable performance monitoring
'OFF'L	Disable performance monitoring
'REPORT'L	Report current performance monitor statistics
'RESET'L	Report current statistics, then clear performance monitor tables

group Performance monitor group number (type integer). See the Performance Counter Group Description table for group numbers and their corresponding counters and counter contents.

buffer First word address of a performance monitor request buffer

bufl Number of words in the buffer array

Thirty-two counters are available, arranged into four groups of eight counters each. Only one group can be accessed at a time.

The **PERF** request block format contains a fixed header and a variable number of subblocks following the header. The first 3 words of the header are set in subroutine **PERF** before calling the system, while the remaining words in the header are returned by the system.

The words in the block header allow you to analyze the information returned in the subblocks without the use of constants. This allows programs to continue executing correctly when the contents of the header or the subblocks change.

The block header format is as follows:

Field	Word	Description
HMRSF	0	Subfunction (0 through 3)
HMRGN	1	Group number (0 through 3) for PM$ON
HMRNW	2	Length of the request block
HMRNU	3	Number of words used
HMRBH	4	Number of words in the block header
HMRTS	5	Set to nonzero if the block is too small
HMRCT	6	Offset to the first group counter in the subblock
HMRCP	7	Offset to the first group accounted CPU cycles
HMRGE	8	Length of the counter group entry in subblock
HMRNC	9	Number of counters in each group entry
HMRNG	10	Number of groups in each subblock
HMRLE	11	Length of subblock entries

Timing subblocks are returned for every **REPORT** and **RESET** call. Each subblock contains hardware performance monitor data from a single COS user task.

The address of the first timing subblock is at (BLOCK FWA) +(contents of block header field **HMRBH**), with the next following (contents of block header field **HMRLE**) word after the first. Subblocks end when the offset to the next block would start after (contents of block header field **HMRNU**) words.

Each subblock contains a 2-word header, with fields **HMTN** and **HMGRP**. **HMTN** is the COS user task number associated with the subblock. **HMGRP** is the last hardware performance monitor group number active for the subblock.

Within the subblock, there are (contents of block header field **HMRNG**) performance monitor groups reported. Each group report consists of two fields: counters associated with the group, and the number of CPU cycles that were accounted for while the specified monitor was active. The offset to the first group counter is (contents of block header field **HMRCT**) words into the subblock; there are (contents of block header field **HMRNC**) counters for each performance monitor group. The offset to the first group's accounted CPU cycle is at (contents of block header field **HMRCP**).

Timing groups within a subblock follow each other by (contents of block header field **HMRGE**) words. The subblock format follows:

Field	Word	Description
HMTN	0	User task number
HMGRP	1	Latest performance monitor group number
HMCNT0	2-9	Group 0, counter 0 through 7
HMCCY0	10	Group 0, accounted CPU cycles
HMCNT1	11-18	Group 1, counter 0 through 7
HMCCY1	19	Group 1, accounted CPU cycles
HMCNT2	20-27	Group 2, counter 0 through 7
HMCCY2	28	Group 2, accounted CPU cycles
HMCNT3	29-36	Group 3, counter 0 through 7
HMCCY3	37	Group 3, accounted CPU cycles

The performance counter group descriptions are listed below in the following table.

Performance Counter Group Descriptions		
Group	Performance Counter	Description
0		Number of:
	0	Instructions issued
	1	Clock periods holding issue
	2	Fetches
	3	I/O references
	4	CPU references
	5	Floating-point add operations
	6	Floating-point multiply operations
	7	Floating-point reciprocal operations
1		Hold issue conditions:
	0	Semaphores
	1	Shared registers
	2	A registers and functional units
	3	S registers and functional units
	4	V registers
	5	V functional units
	6	Scalar memory
	7	Block memory
2		Number of:
	0	Fetches
	1	Scalar references
	2	Scalar conflicts
	3	I/O references
	4	I/O conflicts
	5	Block references
	6	Block conflicts
	7	Vector memory references
3		Number of:
	0	000 – 017 instructions
	1	020 – 137 instructions
	2	140 – 157, 175 instructions
	3	160 – 174 instructions
	4	176, 177 instructions
	5	Vector integer operations
	6	Vector floating-point operations
	7	Vector memory references

IMPLEMENTATION

This routine is only available to users of the COS operating system.

NAME

SNAP – Copies current register contents to $OUT

SYNOPSIS

CALL SNAP(*regs,control,form*)

DESCRIPTION

regs Code indicating registers to be copied, as follows:

1	B registers
2	T registers
3	B and T registers
4	V registers
5	B and V registers
6	T and V registers
7	B, T, and V registers

control Control word (currently unused)

form Code indicating the format of the dump. Dumps from registers S, T, and V are controlled by the following type codes:

0	Octal
1	Floating-point
2	Decimal
3	Hexadecimal

Dumps from registers A and B are in octal format.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

SYMDEBUG – Produces a symbolic dump

SYNOPSIS

CALL SYMDEBUG('*param*{,*param*}.')

DESCRIPTION

param **SYMDEBUG** parameters. Under COS, *param* must be in uppercase; under UNICOS, *param* may be entered in either uppercase or lowercase.

Some **SYMDEBUG** parameters allow you to specify a value along with the parameter. In these cases, *param=value* substitutes for *param*.

SYMDEBUG uses the following parameters:

S=*sdn* *sdn* names the dataset or file containing the debug symbol tables. Under COS, the default is **$DEBUG**. Under UNICOS, the default is **a.out**. The symbol file is **SYMBOLS**.

L=*ldn* *ldn* names the dataset or file to receive the listing output from the symbolic debug routine. Under COS, the default is **$OUT**. Under UNICOS, output goes to standard output.

CALLS=*n* Number of routine levels to be looked at in a symbolic dump. For each task reported, **SYMDEBUG** traces back through the active subprograms the number of levels specified by *n*. Routines for which no symbol table information is available are not counted for purposes of the **CALLS** count. If this parameter is omitted, or if **CALLS** is specified without a value, the default is **50**.

MAXDIM=*dim*{*:dim*}*fR*
Maximum number of elements from each dimension of the arrays to be dumped. **MAXDIM** allows you to sample the contents of arrays without creating huge amounts of output. When **MAXDIM** is specified, arrays are dumped in storage order (row, column for Pascal; column, row for Fortran). **MAXDIM** applies to all blocks dumped. The default is **MAXDIM=20:5:2:1:1:1:1**. No more than seven dimensions can be specified.

BLOCKS=*blk*{*:blk*}
List of common blocks to be included from the symbolic dump. A maximum of 20 blocks can be specified. Separate the *blks* with colons. All symbols (qualified by the **SYMS** and **NOTSYMS** parameters) in the named blocks are dumped. Default is no common blocks dumped; if you specify **BLOCKS** without any *blks*, all common blocks declared in routines to be dumped are included in the symbolic dump.

NOTBLKS=*nblk*{*:nblk*}
List of common blocks to be excluded in the symbolic dump. A maximum of 20 blocks can be specified. Separate the *nblks* with colons. This parameter is used in conjunction with **BLOCKS** and takes precedence over the **BLOCKS** parameter.

RPTBLKS Repeat blocks; when this option is used, the contents of common blocks specified with the **BLOCKS** and **NOTBLKS** parameters are displayed for each subroutine in which they are declared. The default displays common blocks only once.

PAGES=*np* Page limit for the symbolic dump routine. Under UNICOS, **SYMDEBUG** does not format output in pages. However, this parameter can still be used to regulate the amount of output that **SYMDEBUG** generates. Every page is worth 45 lines of output from **SYMDEBUG**. The default *np* is **70**.

EXAMPLE

The following are example calls from Fortran to **SYMDEBUG**:

```
CALL SYMDEBUG('CALLS=40,RPTBLKS.')

CALL SYMDEBUG('BLOCKS=AA:BB:CC.')
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

SEE ALSO

The Symbolic Debugging Package Reference Manual, publication SR-0112

NAME

SYMDUMP – Produces a snapshot dump of a running program

SYNOPSIS

CALL SYMDUMP ('**-b** *blklist* **-B** **-c** *calls* **-d** *dimlist* **-l** *lfile* **-r** **-s** *symfile* **-V** **-y** *symlist* **-Y**', *abort_flag*)

DESCRIPTION

SYMDUMP is a library routine that produces the same sort of output as **DEBUG**. It accepts C character descriptors, Fortran hollerith strings, and Pascal packed character arrays.

The method of calling library routines differs from language processor to language processor, but **SYMDUMP** accepts the same arguments regardless of the language processor. The argument string, if provided, must be enclosed in parentheses, and the options (excluding the abort flag) must be enclosed in quotation marks. When calling **SYMDUMP** from Fortran or Pascal, the quotation marks must be single; when calling from C, the quotation marks must be double. All arguments are optional.

The options indicate the type and extent of information to be dumped by **SYMDUMP**. The options string is passed to **SYMDUMP** in one of the following forms:

- As a character descriptor, produced by Fortran and C for defined characters strings
- As an address of a null terminated string, such as an integer, Hollerith, or Pascal packed character array

The argument string can contain a maximum of 4,096 characters. All options are optional, and they may appear in any order.

Unlike command lines, **SYMDUMP** option-arguments may not be grouped after one hyphen on the **SYMDUMP** call. That is, **SYMDUMP('-V -r')** is permitted, but **SYMDUMP('-Vr')** is not permitted. The following are valid options and arguments:

-b *blklist*

-B These options control the displaying of common block symbols. The symbols to be displayed from any particular common block will depend upon the use of the **-Y** and **-y** *symlist* options.

If neither option is specified, no common blocks are included in the symbolic dump. This is the default. If **-B** is specified, all common blocks are included in the symbolic dump. If **-b** *blklist* is specified, only the common blocks named in *blklist* are included in the symbolic dump. If both options are specified, all common blocks are included in the symbolic dump except those in *blklist*.

blklist may have up to 20 common blocks named. There is no limit on the length of a common block name. The common blocks named in *blklist* must be separated by commas (for example: **-b c,d**).

Enter the common blocks named in *blklist* in the case in which they appear in the symbol table. Names may not always appear in the symbol table in the same way they appear in your program. The UNICOS Symbolic Debugging Package Reference Manual, publication SR-0112, describes how symbol names appear in the symbol table.

-c *calls* *calls* is an integer that specifies the number of routine levels to be displayed in the symbolic dump. For each task reported, **SYMDUMP** traces back through active routines the number of levels specified by calls. Routines for which no symbol table information is available are not counted for purposes of the routine level count. The default is 50.

-d *dimlist* *dimlist* is an integer that specifies the maximum number of elements from each dimension of the arrays to be dumped. **SYMDUMP** can dump array elements from up to seven dimensions. The dimensions must be specified by integer values, and the values must be separated by commas (example: **-d 4,6**)

This option allows you to sample the contents of an array without creating huge amounts of output. *dimlist* applies to all blocks dumped, and the arrays are dumped in storage order. The default is **-d 20,5,2,1,1,1,1**.

-l *lfile* *lfile* names an output file. Specifying **-l** *file* directs **SYMDUMP** to write output to the specified file. If you call **SYMDUMP** more than once, and you specify **-l** with the same file each time, **SYMDUMP** output will be appended to the file each time. By default, **SYMDUMP** output is written to **stdout**.

-r Repeat blocks. When this option is used, **SYMDUMP** displays the contents of common blocks specified with the **-B** and **-b** *blklist* for each subroutine in which they are declared. The default displays common blocks only once.

-s *symfile*

symfile names a file containing the Debug symbol tables. There is no limit on the length of the symfile file name, and it may include a pathname to the desired file. **SEGLDR** puts both the symbol table information and the executable binary in the same file. By default, Debug symbol tables are written to **a.out**.

-V With **-V** specified, **SYMDUMP** generates **SYMDUMP** release statistics.

-y *symlist*

-Y These options may occur anywhere in the option string in any order. Use one of the following methods to control the way symbols are displayed:

If neither option is specified, all symbols are displayed. Default.

If only the **-Y** option is specified, no symbols are displayed.

If only the **-y** option is specified, all symbols except those named in *symlist* are displayed.

If both options are specified, only the symbols named in *symlist* are displayed.

symlist may contain up to 20 named symbols, and there is no limit to the length of the symbol names. The symbols named in *symlist* must be separated by commas (example: **-y a, b**)

Enter the symbols in the same case in which they appear in the symbol table. Names may not always appear in the symbol table in the same way they appear in your program.

abort_flag

An optional *abort_flag* indicates to **SYMDUMP** whether or not to abort if it finds an error when parsing the **SYMDUMP** statement. An *abort_flag* with a value of zero indicates no abort; an *abort_flag* with a value other than zero indicates abort.

You cannot enter an *abort_flag* if you have not entered any options.

By default, **SYMDUMP** examines all options, reports errors found, and generates a dump based on the options it could understand; the program does not abort.

Note that the *abort_flag* is not allowed when options contains a Pascal variant array.

NOTES

Use **SEGLDR** or **ld(1)** to load programs that call **SYMDUMP**. When using **SEGLDR**, specify library **libdb.a**, which contains **SYMDUMP**, on the **-l** option.

The following three examples show how to load programs that call **SYMDUMP**.

Example 1:
If you are not expanding blank common and do not need to specify a **SEGLDR HEAP** directive on the SEGLDR command line for any other reason, you do not need to specify a **SEGLDR HEAP** or **STACK** directive. The following example shows a **SEGLDR** command line without **HEAP** or **STACK** directives:

```
segldr -l libdb.a *.o
```

Example 2:
If you are expanding blank common, you need to specify **SEGLDR STACK** and **HEAP** directives. The following example shows a **SEGLDR** command line that can be used if the program expands blank common.

```
segldr -l libdb.a -D "STACK=3000+0;HEAP=10000+0" *.o
```

This example shows settings that should provide enough stack and heap space for **SYMDUMP** to run, assuming that your program is an average large application that has as many as 1000 blocks. For applications with more blocks, 6 to 7 words per block over 1000 should be added to the heap setting. Optimal heap settings depend on the specific application.

If running the application causes **SYMDUMP** to exit with the following error message, the value on the **HEAP** directive is too small:

```
HPALLOC failed; return status = i
```

Example 3:
If a **SEGLDR DYNAMIC** directive is used, the stack and heap cannot expand, so a **SEGLDR STACK** or **HEAP** directive may also be needed. Refer to the previous example for information about expanding the stack and heap. To load the heap prior to blank common, use **DYNAMIC=//** on **SEGLDR**'s **-D** option, as shown in the following example:

```
segldr -l libdb.a -D "DYNAMIC=//" *.o
```

For more information on **SEGLDR**, see the Segment Loader (SEGLDR) Reference Manual, publication SR-0066.

EXAMPLES

The following example shows how to call **SYMDUMP** from a Fortran program when passing a character descriptor:

```
character*30 string
integer abtfl
.
.
string = '-s test -B -b STRING'
abtfl = 1
.
```

```
      .
      C  CHARACTER VARIABLE
       call symdump (string, abtfl)
      .
      .
      C  CHARACTER CONSTANT
       call symdump ('-l outfile -V')
```

The following example shows how to call **SYMDUMP** from C:

```
extern void SYMDUMP();

int abt_flag = 1;
char *string;

string = "-s a.out -V";
SYMDUMP (string, &abt flag);
```

The following example shows how to call **SYMDUMP** from Pascal when passing a conformant array:

```
type
  string_type = packed array [1..30] of char;
var
  abort_flag: boolean;

procedure symdump (var string: string_type; var flag: boolean);
imported (SYMDUMP);

abort_flag := true;
string [1..20] := '-s test -y STRING -Y';
string [21] := chr (0);          (* must null terminate the string *)
symdump (string, abort_flag);
```

IMPLEMENTATION

This routine is available only to users of the UNICOS operating system.

NAME

TRBK – Lists all subroutines active in the current calling sequence

SYNOPSIS

CALL TRBK[(*arg*)]

DESCRIPTION

arg Address of dataset name or unit number

TRBK prints a list of all subroutines active in the current calling sequence from the currently active subprogram. It also identifies the address of the reference. You can specify a unit (*arg*) to receive the list. If you do not specify a unit, the list is printed to the user logfile or message log.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

TRBKLVL – Returns information on current level of calling sequence

SYNOPSIS

CALL TRBKLVL(*trbktab,arglist,status,name,calladr,entpnt,seqnum,numarg*)

DESCRIPTION

trbktab	Current level's Traceback Table address. On exit, current level's caller's Traceback Table address. Zero if the current level is a main-level routine.
arglist	Current level's argument list address. On exit, current level's caller's argument list address. Zero if the current level is a main-level routine.
status	<0 if error =0 if no error >0 if no error and the current level is the main level
name	Current level's name (ASCII, left-justified, blank-filled)
calladr	Parcel address from which the call to the current level was made
entpnt	Parcel address of the current level's entry point
seqnum	Line sequence number corresponding to the call address (0 indicates none)
numarg	Number of arguments or registers passed to the current level

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

XPFMT – Produces a printable image of an Exchange Package

SYNOPSIS

CALL XPFMT(*address,in,out,mode***)**

DESCRIPTION

address The nominal location of the Exchange Package to be printed as the starting Exchange Package address. This is not the address of the 16-word buffer containing the Exchange Package to be formatted.

in A 16-word integer array containing the binary representation of the Exchange Package

out An integer array, dimensioned (8,0:23), into which the character representation of the Exchange Package is stored. Line 0 is a ruler for debugging and is not usually printed.

The first word of each line is an address and need not always be printed.

mode An integer word indicating the mode in which the Exchange Package is to be printed. **'Y'L** forces the Exchange Package to be formatted as a CRAY Y-MP Exchange Package; **'X'L** forces the Exchange Package to be formatted as a CRAY X-MP Exchange Package; **'S'L** forces the Exchange Package to be formatted as a CRAY-1 Exchange Package; 0 means that the subprogram is to use the Exchange Package contents to deduce the machine type.

XPFMT produces a printable image of an Exchange Package in a user-supplied buffer. A and S registers appear in the buffer in both octal and character form; in the character form, the contents of the register are copied unchanged to the printable buffer. The calling program is responsible for proper translation of unprintable characters. Parcel addresses have a lowercase a, b, c, or d suffixed to the memory address.

You can specify that the Exchange Package be formatted as a CRAY X-MP or CRAY-1 Exchange Package, or you can allow **XPFMT** to determine which format to use based on the values in the Exchange Package. Values within the Exchange Package determine the Exchange Package format. **XPFMT** assumes that the Exchange Package was produced by or for a CRAY X-MP computer system if either the data base address or the data limit address is nonzero. Otherwise, it assumes that the Exchange Package was produced by or for a CRAY-1 computer system.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

17. SYSTEM INTERFACE ROUTINES

System interface routines are grouped into the following categories:

- Job control language (JCL) symbol routines
- Control statement processing routines
- Job control routines
- Floating-point interrupt routines
- Bidirectional memory transfer routines
- Special purpose interface routines

JOB CONTROL LANGUAGE SYMBOL ROUTINES

The JCL symbol routines manipulate JCL symbols for conditional JCL statements.

JSYMSET changes a value for a JCL symbol. **JSYMGET** allows a user program to retrieve JCL symbols.

CONTROL STATEMENT PROCESSING ROUTINES

Control statement processing routines place control statement elements in appropriate memory locations to perform the specified operations. These routines, **CRACK**, **PPL**, and **CEXPR**, can also process directives obtained from some source other than the control statement file ($CS).

Control statement cracking routines take the uncracked image from the JCCCI field and crack it into the JCCPR field. The Job Communication Block (JCB) contains the control image in JCCCI. JCDLIT is a flag indicating whether or not literal delimiters are to be retained in the string.

The following table contains the purpose, name, and entry of each control statement processing and cracking routine.

Control Statement Processing and Cracking Routines		
Purpose	Name	Entry
Crack a control statement	**CCS**	**CCS**
Process control statement parameter values	**GETPARAM**	**GETPARAM**
Crack a directive	**CRACK**	**CRACK**
Process a parameter list	**PPL**	**PPL**
Crack an expression	**CEXPR**	**CEXPR**

JOB CONTROL ROUTINES

Job control routines perform functions relating to job step termination, either causing a termination or instructing the system on how to handle a termination. Unless otherwise specified, these routines are called by address. No arguments are returned.

The following table contains the purpose, name, and entry of each job control routine.

Job Control Routines		
Purpose	Name	Entry
Request abort with traceback	**ABORT**	**ABORT**
Terminate a job step and advance	**END**	**END**
Continue exit processing after a reprievable condition	**ENDPRV**	
Exit from a Fortran program	**EXIT**	**EXIT**
Request abort	**ERREXIT**	**ERREXIT**
Declare a job rerunnable or not rerunnable	**RERUN**	**RERUN**
Instruct the system to begin or cease monitoring jobs for functions affecting rerunnability	**NORERUN**	
Conditionally transfer control to a specified routine	**SETRPV**	**SETRPV**

FLOATING-POINT INTERRUPT ROUTINES

Floating-point interrupt routines allow you to test, set, and clear the Floating-point Interrupt Mode flag. Subroutine linkage is call-by-address.

The following table contains the purpose, name, and entry of each floating-point interrupt routine.

Floating-point Interrupt Routines		
Purpose	Name	Entry
Temporarily prohibit floating-point interrupts	**CLEARFI**	**CLEARFI**
Temporarily permit floating-point interrupts	**SETFI**	
Temporarily prohibit floating-point interrupts for a job	**CLEARFIS**	**CLEARFIS**
Temporarily enable floating-point interrupts for a job	**SETFIS**	
Determine whether floating-point interrupts are permitted or prohibited	**SENSEFI**	**SENSEFI**

BIDIRECTIONAL MEMORY TRANSFER ROUTINES

Bidirectional memory transfer routines test, set, and clear the Bidirectional Memory Transfer Mode flag. Subroutine linkage is call-by-address.

NOTE

These routines are only effective on CRAY Y-MP and CRAY X-MP computer systems, which have hardware support for bidirectional memory transfer. They are no-ops on other mainframe types.

The following table contains the purpose, name, and entry of each bidirectional memory transfer routine.

<table>
<tr><th colspan="3">Bidirectional Memory Transfer Routines</th></tr>
<tr><th>Purpose</th><th>Name</th><th>Entry</th></tr>
<tr><td>Temporarily disable bidirectional memory transfers</td><td>CLEARBT</td><td rowspan="2">CLEARBT</td></tr>
<tr><td>Temporarily enable bidirectional memory transfers</td><td>SETBT</td></tr>
<tr><td>Permanently disable bidirectional memory transfers</td><td>CLEARBTS</td><td rowspan="2">CLEARBTS</td></tr>
<tr><td>Permanently enable bidirectional memory transfers</td><td>SETBTS</td></tr>
<tr><td>Determine current memory transfer mode</td><td>SENSEBT</td><td>SENSEBT</td></tr>
</table>

SPECIAL-PURPOSE INTERFACE ROUTINES

The following table contains the purpose, name, and entry of each special-purpose interface routine.

Special-purpose Interface Routines		
Purpose	Name	Entry
Return the Job Accounting Table	**ACTTABLE**	**ACTTABLE**
Program a Cray channel on an IOS	**DRIVER**	**DRIVER**
Turn on or off the class of messages to the user logfile	**ECHO**	**ECHO**
Allow a job to suspend itself	**ERECALL**	**ERECALL**
Return lines per page	**GETLLP**	**GETLLP**
Return the integer ceiling of a rational number formed by two integer parameters	**ICEIL**	**ICEIL**
Allow a job to communicate with another job	**IJCOM**	**IJCOM**
Return the job name	**JNAME**	**JNAME**
Load an absolute program from a dataset containing a binary image	**LGO**	**LGO**
Return the memory address of a variable or an array	**LOC**	**LOC**
Manipulate a job's memory allocation and/or mode of field length reduction	**MEMORY**	**MEMORY**
Return the edition for a previously accessed permanent dataset	**NACSED**	**NACSED**
Load an overlay and transfer control to the overlay entry point	**OVERLAY**	**OVERLAY**
Enter a message (preceded by a message prefix) in the user and system logfiles	**REMARK**	**REMARK**
Enter a message in the user and system logfiles	**REMARK2**	
Enter a formatted message in the user and system logfiles	**REMARKF**	**REMARKF**
Return Cray machine constants (machine epsilon; smallest and largest normalized numbers.)	**SMACH** **CMACH**	**SMACH**
Test the sense switch	**SSWITCH**	**SSWITCH**
Make requests of the operating system	**SYSTEM**	**SYSTEM**

NAME

ABORT – Requests abort with traceback

SYNOPSIS

CALL ABORT[(*log*)]

DESCRIPTION

log Log file message

ABORT requests abort with traceback and provides an optional log file message. The optional user-supplied log file message is written to both user and system log files. The message is written in the same format in which it was sent.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

ACTTABLE – Returns the Job Accounting Table (JAT)

SYNOPSIS

CALL ACTTABLE(*array*,*count*[,*tac*,*tasz*,*gut*,*gusz*,*fut*,*fusz*])

DESCRIPTION

array An array in which to write a copy of the JAT

count Count; the first *count* words of the JAT are returned in the array. If *count* is greater than the size of the JAT, the array is padded with minus ones.

tac Address in which to write a copy of the Task Accounting Table

tasz Length of the task accounting information to copy in words. No more than *tasz* words are returned.

gut Address in which to write a copy of the Generic Resource Table

gusz Length of the Generic Resource Table information in words. No more than *gusz* words are returned.

fut Address in which to write a copy of the Fast Secondary Storage (FSS) device utilization information

fusz Length of the FSS device utilization information area in words. No more than *fusz* words are returned.

You can specify *array* and *count* without requesting any of the optional information with the other parameters. However, to request any of the optional information, you must enter values for all six of the optional parameters, entering a zero length for those you do not want.

EXAMPLE

The call to **ACTTABLE** in the following example returns information from the JAT and six words from the Task Accounting Table. Since the size parameters (**GUSZ** and **FUSZ**) are set to zero, no FSS or Generic Resource Table information is returned.

```
PROGRAM ACTTAB

IMPLICIT INTEGER (A-Z)

PARAMETER (COUNT = 10)
PARAMETER (TASZ = 6)
PARAMETER (GUSZ = 0)
PARAMETER (FUSZ = 0)
DIMENSION ARRAY(60), TAC(6)

CALL ACTTABLE(ARRAY,COUNT,TAC,TASZ,JUNK,GUSZ,JUNK,FUSZ)
STOP
END
```

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

CCS – Cracks a control statement

SYNOPSIS

CALL CCS

DESCRIPTION

No parameters. **CCS** aborts the job if errors are encountered.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

CEXPR – Cracks an expression

SYNOPSIS

CALL CEXPR(*char*,*out*,*lmt*,*size*)

DESCRIPTION

char	Expression character-string array (terminated by a 0 byte)
out	Reverse Polish Table array for output
lmt	Upper limit to the size of the Reverse Polish Table
size	Actual size of the Reverse Polish Table on return

CEXPR transforms an expression character string (1 right-justified character per word) to a Reverse Polish Table.

An expression can contain a mixture of symbols, literals, numeric values, and operators. Expressions handled by this routine resemble Fortran in syntax.

Operator hierarchy follows Fortran rules and does parenthesis nesting. Symbols are defined as 1- to 8-character strings having unknown value to **CEXPR. CEXPR** simply flags the strings for the caller. The first character cannot be numeric. Literals are 1- to 15-character strings enclosed by double quotes (").

A character string consisting of numeric digits is taken as a 64-bit integer. A trailing B signifies an octal number.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

CLEARBT, SETBT – Temporarily disables/enables bidirectional memory transfers

SYNOPSIS

CALL CLEARBT
CALL SETBT

DESCRIPTION

CLEARBT temporarily disables bidirectional memory transfers. **SETBT** temporarily enables bidirectional memory transfers.

These routines are local to the current job step. The system restores the most recent mode setting at the start of the next job step. No arguments are required or returned.

IMPLEMENTATION

These routines are available only to users of the COS operating system.

NAME

CLEARBTS, SETBTS – Permanently disables/enables bidirectional memory transfers

SYNOPSIS

CALL CLEARBTS
CALL SETBTS

DESCRIPTION

CLEARBTS permanently disables bidirectional memory transfers. **SETBTS** permanently enables bidirectional memory transfers.

The results of these routines are permanent and are propagated through job steps. The system does not alter the mode setting unless another bidirectional memory transfer control subroutine is called or a **MODE** control statement is executed. No arguments are required or returned.

IMPLEMENTATION

These routines are available only to users of the COS operating system.

NAME

CLEARFI, SETFI – Temporarily prohibits/permits floating-point interrupts

SYNOPSIS

CALL CLEARFI
CALL SETFI

DESCRIPTION

CLEARFI temporarily prohibits floating-point interrupts. **SETFI** temporarily permits floating-point interrupts.

These routines are local to the current job step. The system restores the most recent mode setting at the start of the next job step. No arguments are required or returned.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

CLEARFIS, SETFIS – Temporarily prohibits/permits floating-point interrupts for a job

SYNOPSIS

CALL CLEARFIS
CALL SETFIS

DESCRIPTION

CLEARFIS prohibits floating-point interrupts for a job until they are enabled or until the job terminates.

SETFIS enables floating-point interrupts until they are explicitly disabled or until the job terminates.

The results of these routines are propagated through job steps. The system does not alter the mode setting until another floating-point interrupt control subroutine is called or a **MODE** control statement is executed. No arguments are required or returned.

IMPLEMENTATION

These routines are available only to users of the COS operating system.

NAME

CRACK – Cracks a directive

SYNOPSIS

CALL CRACK(*ibuf,ilen,cbuf,clen,flag*[,*dflag*])

DESCRIPTION

ibuf Image of the statement to be cracked

ilen Integer length (in words) of the statement image to be cracked. Maximum value is 10 words.

cbuf Array to receive the cracked image

clen Integer length in words of the array *cbuf*

flag Integer variable to receive completion status. The Return Value flag has the following meanings:

0 Normal termination
1 No error; continuation character encountered.
2 Invalid character encountered
3 Premature end-of-input line
4 CRACK buffer overflow
5 Unbalanced parentheses
6 Input buffer too large

dflag Integer flag indicating that literal string delimiters are to be preserved in the cracked image. If set to 0 or omitted, quotes are not included in the cracked string. If set to 1, all quotes are included in the string.

CRACK reformats (parses) a user-supplied string into verb, separators, keywords, and values. The cracked directive is placed in a user-supplied buffer and returns the status of the crack to the caller. CRACK can be called repeatedly to process a control statement across several records.

NOTES

Each keyword or positional parameter should be assigned a separate word. Keywords or positional parameters of more than 8 characters must be assigned 1 word for each 8 characters plus 1 for any remaining characters if the length is not a multiple of 8 characters. Each separator must also be assigned a separate word.

flag should be set to 0 before the first call to CRACK and should not be changed (except by CRACK) until after the last call to CRACK.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

DELAY – Do nothing for a fixed period of time

SYNOPSIS

CALL DELAY(*mstime*)

DESCRIPTION

mstime Delay time in milliseconds. *mstime* must be in the range 0 to $2^{24}-1$.

DELAY requests that the executing task not be rescheduled to a CPU until *mstime* milliseconds have elapsed.

IMPLEMENTATION

This routine is only available to users of the COS operating system.

NAME

DRIVER – Programs a Cray channel on an I/O Subsystem (IOS)

SYNOPSIS

CALL DRIVER(*array,lentry,status*)

DESCRIPTION

array First element of the integer parameter block array. The array is *lentry* words long. In all cases, **FUNC**, **PLEN**, and **LN** are required in the parameter block, and **COSS** is returned in the User Driver Parameter Block (DRPB) (see the COS Reference Manual, publication SR-0011, for more information on DRPB). **DP** is always sent to the driver and returned to you. See individual driver specifications for the use of the word and other field requirements.

For the Fortran user, **FUNC**, **DIR**, and **COSS** are literal strings. (For example, set **FUNC** to **'CFN$OPE'** and **DIR** to **'DIR$INP'** to open an input channel. **'DRS$RSV'** in **COSS** means the channel is reserved for another job.)

The **'CFN$OPE'** subfunction opens a channel; a job cannot access a channel until it opens the channel. **DRNM**, **TO**, **DIR**, and **OPD** are required.

The **'CFN$CLS'** subfunction closes a channel. Any open channels are closed during termination. **DIR** is required.

The **'CFN$RD'**, **'CFN$RDH'**, and **'CFN$RDD'** subfunctions read data. **BAD** and **DLN** are required; **TLN** is returned. For read, either the channel is read to Central Memory or data is moved from IOS Buffer Memory to Central Memory (if a read/hold was done prior to this read). For read/hold, a second read is performed, and the data is held in Buffer Memory for a subsequent read. For read/read, a second read to Central Memory is done.

The **'CFN$WT'**, **'CFN$WTH'**, and **'CFN$WTD'** subfunctions write data. **BAD** and **LN** are required; **TLN** is returned. For write, data is written to the channel from Central Memory or Buffer Memory (if a write/hold was done prior to this request). For write/hold, a second buffer of data is moved to and held in Buffer Memory for a subsequent write. For write/write a second write is performed from Central Memory.

The **'CFN$DMIN'–'CFN$DMAX'** subfunctions are defined by the driver. **DFP** and **DIR** are required.

lentry Length of the parameter block entry in *array*; user-specified integer variable.

status Status; integer variable set by the system. On return, *status* is 0 if no errors have occurred, and the job must poll **COMS** for nonzero. When **COMS** is nonzero, the driver has completed the request and the driver status is in **DRS**. See the individual driver specifications for driver status. If *status* is nonzero on return, **COSS** contains the error code and the request is not sent to the driver.

If no errors have occurred, and if *status* is nonzero on return, **COSS** contains the error code.

This capability is available only with devices connected to the Master I/O Processor (MIOP). This is a privileged function available to all single-tasked job steps. It is prohibited to multitasking job steps.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

ECHO – Turns on and off the classes of messages to the user logfile

SYNOPSIS

CALL ECHO('ON'L[,*param-array*],'OFF'L[,*param-array*])

DESCRIPTION

param-array Optional array of message class names or 'ALL'. Message class names are defined in the COS Reference Manual, publication SR-0011.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

END, ENDRPV – Terminates a job step

SYNOPSIS

END
CALL ENDRPV

DESCRIPTION

END terminates a job step and advances to the next job step.

ENDRPV continues normal exit processing after a reprievable condition has been processed. This exit processing can be the result of normal termination or abort processing.

IMPLEMENTATION

END is available to users of both the COS and UNICOS operating systems.

NAME

ERECALL – Allows a job to suspend itself until selected events occur

SYNOPSIS

CALL ERECALL(*func,status,sevents,to,oevents,levents*)

DESCRIPTION

func User-specified integer variable to define the requested information or action

'DISABLE'	Disables event monitoring. All other words are ignored.
'ENABLE'	Enables event monitoring or changes the events to be monitored. *levents* and *sevents* are required. If *levents* is 0, time-out is the only enabled event; time-out is enabled to prevent a job remaining indefinitely in recall. *levents* and *oevents* are returned by the system. *to* is ignored.
'RECALL'	Places the job in recall. An error is returned in *status* if monitoring is disabled. *to* is required; *sevents* is ignored. *levents* and *oevents* are set by the system. If *to* is 0, an installation-defined default, **I@TODEF**, is used. If *to* is specified, but less than the installation-defined minimum, **I@TOMIN**, the installation minimum is used with no notification. If *levents* is 0 on return, time-out is the only event that occurred.
'RETURN'	Requests that *levents* and *oevents* be set by the system; all other words are ignored. An error is returned in *status* if monitoring is disabled.

status Status; an integer variable set by the system. *Status* is 0 if no errors occurred; otherwise, see the Event Recall Parameter Block (ERPB) definition in the COS Reference Manual, publication SR-0011, for error codes. The codes are returned as blank-filled literal strings (for example, **ERER$BFN** is returned as **'ERER$BFN'**).

sevents User-specified integer array containing the events to be monitored. *levents* is the number of events specified in *sevents*. The events can be selected from the following:

'IJ'	Interjob message received
'UO'	Unsolicited operator message received (Deferred implementation)
'OR'	Operator reply received (Deferred implementation)

The following events are privileged:

'CH'	Channel driver done
'IQ'	SDT placed in input queue (Deferred implementation)
'OQ'	SDT placed in output queue (Deferred implementation)

to Time-out duration in milliseconds (rightmost 24 bits); user-specified integer variable.

oevents Integer array set by the system to the occurred events. *levents* is the number of event words that have been placed in *oevents* by the system. See *sevents* for possible values.

levents Integer value specifying the number of events in either *sevents* or *oevents*. For **ENABLE**, set *levents* to the number of event words that you have placed in *sevents*. On return from **ENABLE, RECALL,** and **RETURN**, *levents* is the number of event words that the system has placed in *oevents*.

ERECALL allows a job to suspend itself until one or more selected events occur.

NOTE

This routine is available to all single-tasking job steps; it is prohibited to multitasking job steps.

When event monitoring is enabled, the system monitors selected events for a job, keeping track of which ones have occurred. Monitoring is disabled at the beginning of each job step and can be enabled by making a system request, specifying the events to monitor. Once monitoring is enabled, a job can make a system request to change the events that are to be monitored, get a map indicating which of the monitored events occurred, go into event recall until one of the selected events occurs, or disable monitoring.

When monitoring is enabled, a map of occurred events is returned to you and discarded by the system. If monitoring was disabled when the enable occurred, the map is 0.

When the events to be monitored are changed, a map of occurred events is returned to you and discarded by the system.

When a map of occurred events is requested, the map is returned to you and discarded by the system.

When recall is requested and the map of occurred events is 0, the job is suspended for an event until one of the events occurs. If the map is nonzero, the map is returned to you immediately and discarded by the system.

When recall is disabled, the map of occurred events is discarded by the system.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

The COS Reference Manual, publication SR-0011

NAME

ERREXIT – Requests abort

SYNOPSIS

CALL ERREXIT

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

EXIT – Exits from a Fortran program

SYNOPSIS

CALL EXIT

DESCRIPTION

EXIT ends the execution of a Fortran program and writes a message to the log file (COS) or **stdout** (UNICOS). Under COS, the message is as follows:

UT003 – EXIT CALLED BY *routine name*

The UNICOS message is as follows:

EXIT (called by *routine name*, line *n*)

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

GETARG – Return Fortran command-line argument

SYNOPSIS

ichars = **GETARG**(*i*,*c*)
ichars= **GETARG**(*i*,*c*,*size*)

DESCRIPTION

ichars	Number of non-null characters in the string returned
i	Number of the argument to return
c	Character variable or integer array in which to return the command-line argument
size	If *c* is an array, the number of elements in that array

GETARG returns the *i*-th command-line argument of the current process. Thus, if a program is invoked with the following command line, **GETARG(2,C)** returns the string **arg2** in the character variable **C**:

foo arg1 arg2 arg3

SEE ALSO

GETOPT(3C)

IMPLEMENTATION

This routine is available only to users of the UNICOS operating system.

NAME

GETLPP – Returns lines per page

SYNOPSIS

lpp=**GETLPP( )**

DESCRIPTION

lpp Lines per page (type integer)

GETLPP returns the lines per page from field JCLPP of the Job Control Block (JCB) in register S1.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

GETPARAM – Gets parameters

SYNOPSIS

CALL GETPARAM(*table,number,param*)

DESCRIPTION

table The Parameter Control Table (PCT), dimensioned (5,*number*) and containing the following in each 5-element row:

1 A left-justified, zero-filled keyword
2 A default value for use if the keyword is missing
3 A default value for use if the keyword is present but not assigned a value
4 Subscript of *param* into which the first parameter value is stored
5 Index of the last word the the *param* array to be used for storing the parameter value

If item 2 is negative, **GETPARAM** requires the keyword to be on the control statement.

If item 3 is negative, **GETPARAM** does not allow the use of the keyword alone (as in "...,keyword,...").

Either item 2 or 3 can be 0; **GETPARAM** does not distinguish between 0s and any other positive values such as character strings, but the caller can test them after **GETPARAM** returns.

If items 2 and 3 are 0 and 1, or 1 and 0, respectively, **GETPARAM** does not allow the keyword to be followed by an '='. The keyword must be simply absent or present.

If item 1 is a 64-bit mask (that is, 177777 7777 7777 7777 7777B), the value given as the keyword is returned in the control table. When an entry of this type is specified in the control table, the number of parameters is limited to one.

If item 1 is given a value of 0, the entry describes a positional parameter. Entries of this nature must be described in positional order.

If bit 2 in item 4 (that is, 020000 0000 0000 0000 0000B) is set, the parameters following the keyword are defined to be secure and are edited out before the statement is echoed to the user's logfile. If bit 3 is set, it indicates that a NULL character in the first word of a parameter value should be considered a string terminator.

number The number of parameters described in the control table. If set to 0, **GETPARAM** does not allow any parameters on the control statement.

param An array sufficiently large to receive all the parameter values

GETPARAM processes control statement parameter values from an already cracked control statement. If the statement has been continued across card images, **GETPARAM** automatically requests the next control statement and calls $CCS to crack it. Processing is determined by the rules set up by the PCT.

The PCT indicates default values for unspecified parameters. Through the PCT, the caller also indicates the following:

- If a parameter must be specified on the statement
- If a parameter is positional or keyword
- If a keyword parameter can have an equated value
- If a keyword parameter must have an equated value
- If any parameters are allowed

EXAMPLE

Example of control table definition in Fortran:

```
      INTEGER PERMFILE(2) PARAMS(15), TABLE(5,4), INPUT, LIBRARY(10), LIST
      EQUIVALENCE(PARAMS(1),INPUT),
     *            (PARAMS(2),PERMFILE),
     *            (PARAMS(4),LIBRARY(1)),
     *            (PARAMS(14),LIST)
      DATA PARAMS/15*0/
      DATA (TABLE(I,1),I=1,5)/'I'L,'$IN'L,'$IN'L,1,1/,
     -            (TABLE(I,2),I=1,5)/'P'L,0,-1,2,3/,
     -            (TABLE(I,3),I=1,5)/'LIB'L,-1,'$FTLIB'L,4,13/,
     -            (TABLE(I,4),I=1,5)/'LIST'L,0,1,14,14/
      CALL GETPARAM (TABLE,4,PARAMS)
```

This table (for a hypothetical program) tells **GETPARAM** that the only keywords to be accepted are **I, P, LIB**, and **LIST**. The -1 value means that **P** cannot appear alone (without an equal sign) and that **LIB** (with or without an equal sign) must appear in the control statement.

In this table, only one word is provided for the **I** parameter; therefore, if **I**=*xxx* appears in the control statement, the option *xxx* must not exceed 8 characters. The 2 words provided for the **P** parameter allow for the maximum of 16 characters or for two subparameters (up to 8 characters each) separated by a colon in the control statement. Ten words are provided for the **LIB** parameter so that up to ten subparameters (or five 2-word parameters) are allowed in the control statement. **GETPARAM** requires the keyword **LIST** to appear alone or not at all. If **LIST** is specified, the value returned in the Parameter Value Table is 1. **LIST** cannot be followed by an equal sign.

NOTES

The following two subparameters cannot be distinguished from one another in the **PARAMS** table:

A=A1234567:B1234567(Two 8-character parameters)
A=A1234567B1234567(One 16-character parameter)

Thus, the caller is responsible for restricting such cases.

The output array **PARAMS** must be as large as the largest subscript. If **PARAMS** is initialized to 0s, the programmer can determine how many words are returned by **GETPARAM** for multiword parameters such as **P** and **LIB**.

Because Fortran array numbering starts with 1, the array's base address is reduced by 1 in **GETPARAM**. Therefore, the CAL user must supply the table address + 1 (This is not true for **$GP**) in order to use labels directly in lieu of the Fortran subscripts.

The following characters should not be used in keywords: the colon, parentheses, period, comma, apostrophe, caret, and equal sign.

GETPARAM aborts if the control statement violates either the standard control statement syntax rules or the additional rules imposed by the PCT. If there are no errors, the array is filled with values from the control statement and/or with default values. The PCT is not altered by **GETPARAM**.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

IARGC – Returns number of command line arguments

SYNOPSIS

iargs = **IARGC( )**

DESCRIPTION

iargs Number of command line arguments passed to the program

If a program is invoked with the following command line, **IARGC** returns 3:

foo arg1 arg2 arg3

SEE ALSO

GETOPT(3C)

IMPLEMENTATION

This routine is available only to users of the UNICOS operating system.

NAME

ICEIL – Returns integer ceiling of a rational number

SYNOPSIS

i=**ICEIL**(*j*,*k*)

DESCRIPTION

j	The numerator of a rational number
k	The denominator of a rational number

ICEIL returns the integer ceiling of a rational number formed by two integer parameters. **ICEIL** is an integer function.

The value of the function i is the smallest integer larger than or equal to $\frac{j}{k}$.

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

IJCOM – Allows a job to communicate with another job

SYNOPSIS

CALL IJCOM(*status,array,lentry,nentry*)

DESCRIPTION

status — *status* is a literal value of the error (or, in the case of multiple errors, the literal value of the last error to occur). If *status* is not equal to **IJMS$OK, STAT** contains the literal error code. If multiple parameter blocks are used, all **STAT** fields must be examined if *status* is nonzero.

array — First element of the integer parameter block array. An installation-defined maximum number of parameter blocks (**I@MPBS**) can be specified in *array*. The array is *larray* words long, and each of the *nentry* parameter blocks in it is *lentry* words long. See the Interjob Communications Parameter Block (IJPB) table definition in the COS Reference Manual, publication SR-0011, for a description. You may ignore **LINK**; the system links the entries together for the user. In all cases, **FUNC, RID,** and **PLEN** are required in each parameter block, and the system sets **STAT** in each parameter block. The array length must equal *lentry * nentry.*

FUNC and **STAT** are literal strings (for example, set **FUNC** to **'IJM$OPEN'** to open a path.

'IJM$NOP'	Subfunction is a no op.
'IJM$REC'	Subfunction marks the job as receptive. **RCB** is required; all other words are ignored.
'IJM$OPEN'	Subfunction initiates an attempt to open a communication path with another job. **HLEN, TID,** and **NCB** are required; all other words are ignored.
'IJM$ACCE'	Subfunction accepts a request from another job to open communication. **TID, HLEN,** and **NCB** are required; all other words are ignored.
'IJM$REJE'	Subfunction rejects a request from another job to open communication. **TID** is required; all other words are ignored.
'IJM$SNDM'	Subfunction sends a message to another job. **NCB, TID, BADD,** and **BLEN** are required; all other words are ignored.
'IJM$SNDL'	Subfunction sends a message to an attached job's logfile. This is a privileged function. **TID, OVR, FCS, FCU, CLS,** and **BADD** are required; all other words are ignored.
'IJM$CLOS'	Closes a communication path. Either **NCB** and **TID** or neither are required; all other words are ignored. If **NCB** and **TID** are specified, only the path determined by **RID** and **TID** is closed; otherwise all communication paths with **RID** are closed.
'IJM$END'	Subfunction marks the job as not receptive. All other words are ignored. Existing communication paths are not affected.

lentry — Length of each parameter block entry in array; user-specified integer variable. lentry must equal **LE@IJPB** (**LE@IJPB** is defined in **$SYSTXT** as the length of the Interjob Communications Parameter Block).

nentry — Number of parameter blocks in the array; user-specified integer variable. Default is 1.

status Status; an integer variable set to 0 if no errors occurred. If *status* is nonzero, **STAT** contains the error code. If multiple parameter blocks are used, all **STAT** fields must be examined if status is not equal to **IJMS$OK** (if no errors occurred, *status*=**IJMS$OK**).

NOTE

IJCOM is available to all single-tasking job steps. At this time, interjob communication is prohibited to multitasking job steps.

SEE ALSO

The COS Reference Manual, publication SR-0011

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

ISHELL – Executes a UNICOS shell command

SYNOPSIS

ISTAT = ISHELL(*command***)**

DESCRIPTION

ISHELL has the following argument:

command Command to be given to the shell

ISHELL passes *command* to the shell **sh**(1) as input, as if *command* was entered at a terminal. The current process waits until the shell has completed, then returns the exit status.

EXAMPLE

```
ISTAT = ISHELL('rm -f *.o')
```

IMPLEMENTATION

This routine is available only to users of the UNICOS operating system.

NAME

JNAME – Returns the job name

SYNOPSIS

name=JNAME(*result*)

DESCRIPTION

name Job name; left-justified with trailing blanks.

result Returned job name

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

JSYMSET, JSYMGET – Changes a value for a JCL symbol or retrieve a JCL symbol

SYNOPSIS

CALL JSYMSET('*sym*'L,*val*[,*len*])
CALL JSYMGET('*sym*'L,*val*[,*len*])

DESCRIPTION

sym Valid JCL symbol name

val For **JSYMSET**, the actual value assigned to the symbol. For **JSYMGET**, *val* receives the actual value of the symbol if the value buffer is large enough and the symbol currently has a value.

len For **JSYMSET**, the length of *val* in words (elements). For **JSYMGET**, the length of the value buffer in words (elements). *len* is changed to the actual length of the symbol's value (less than or equal to the value buffer).

JSYMSET allows you to change a value for a JCL symbol. The value specified is the actual value given to the symbol; no evaluation is performed.

JSYMGET allows user programs to retrieve JCL symbols. **JSYMGET** also allows for the creation of JCL symbols if they do not exist. See the COS Reference Manual, publication SR-0011, for more information on JCL symbol definitions.

IMPLEMENTATION

These routines are available only to users of the COS operating system.

NAME

LGO – Loads an absolute program from a dataset containing a binary image as the first record

SYNOPSIS

CALL LGO('*dn*'L)

DESCRIPTION

The dataset name containing the absolute load module is represented by *dn*. LGO loads an absolute program from a local dataset containing the binary image as the first record. The loaded program is then executed. Control does not return to LGO.

Security privileges may be required sometimes when using LGO might seem appropriate (specifically, if you attempt to open a dataset using SDACCESS). Use CALLCSP as a more general replacement for this routine.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

CALLCSP

NAME

LOC – Returns memory address of variable or array

SYNOPSIS

address=**LOC**(*arg*)

DESCRIPTION

address Argument address (type integer)

arg Argument whose address is to be returned

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

MEMORY – Manipulates a job's memory allocation and/or its mode of field length reduction

SYNOPSIS

CALL MEMORY(*code,value*)

DESCRIPTION

code Determines what information or action is requested (blank-filled)

'UC' *value* specifies the number of words to be added to (if value is positive) or subtracted from (if *value* is negative) the end of the user code/data area.

'FL' *value* specifies the number of words of field length to be allocated to the job. If **FL** is specified and *value* is not, the new field length is set to the maximum allowed the job, and the job is placed in user mode for the duration of the job step.

'USER' The job is put in user-managed field length reduction mode. *value* is ignored.

'AUTO' The job is put in automatic field length reduction mode. *value* is ignored.

'MAXFL' The maximum field length allowed the job is returned in *value*.

'CURFL' The current field length is returned in *value*.

'TOTAL' The total amount of unused space in the job is returned in *value*.

value An integer value or variable when code is **'UC'** or **'FL'**. An integer variable that is to contain a returned value if code is **'CURFL'**, **'MAXFL'**, or **'TOTAL'**.

Memory can be added to or deleted from the end of the user code/data area by using the **'UC'** code. If the user code/data area is expanded, the new memory is initialized to an installation-defined value.

The job's field length can be changed by use of the **'FL'** code. The field length is set to the larger of the requested amount rounded up to the nearest multiple of 512-decimal words or the smallest multiple of 512-decimal words large enough to contain the user code/data, Logical File Table (LFT), Dataset Parameter Table (DSP), and buffer areas. The job is placed in user-managed field length reduction mode for the duration of the job step.

The job's mode of field length reduction can be changed by use of either the **'USER'** or **'AUTO'** code. When **'USER'** is specified, the job is placed in user mode until a subsequent request is made to return it to automatic mode. When **'AUTO'** is specified, the job is placed in automatic mode, and the field length is reduced to the smallest multiple of 512-decimal words that can contain the user code/data, LFT, DSP, and buffer areas.

The job's maximum or current field length can be determined by the **'MAXFL'** or **amount of unused space in the job can be determined by the 'TOTAL' code.**

The job is aborted if filling the request would result in a field length greater than the maximum allowed the job. The maximum is the smaller of the total number of words available to user jobs minus the job's Job Table Area (JTA) or the amount determined by the **MFL** parameter on the **JOB** statement.

EXAMPLE

Example 1:

```
CALL MEMORY('FL')
```

The job's field length is set to the maximum allowed the job, and the job is placed in user mode for the duration of the job step.

Example 2:

```
CALL MEMORY('AUTO')
```

The job's field length is reduced to a minimum, and the job is placed in automatic mode.

Example 3:

```
CALL MEMORY('UC',-5)
CALL MEMORY('UC',IVAL)
```

where IVAL is -5

The job's user code/data area is reduced by 5 words.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

NACSED – Returns the edition of a previously-accessed permanent dataset

SYNOPSIS

ed=NACSED( )

DESCRIPTION

NACSED returns edition number *ed* in binary form for the permanent dataset most recently accessed by a call to ACCESS.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

OVERLAY – Loads an overlay and transfers control to the overlay entry point

SYNOPSIS

CALL OVERLAY(*n*L*dn*,lev_1,lev_2[,*recall*])

DESCRIPTION

n	Number of characters in *dn*
L	Left-justified; zero-filled.
dn	Dataset in which the overlay resides. Must be a character constant, integer variable, or an array element containing Hollerith data of not more than 7 characters.
lev_1	Overlay level 1 (LEV1)
lev_2	Overlay level 2 (LEV2)
recall	Optional recall parameter. To reexecute an overlay without reloading it, enter **6LRECALL**. If the overlay is not currently loaded, it will be loaded.

NOTES

This routine is used to implement **LDR**-style overlays. Cray Research recommends conversion to **SEGLDR**-style segments whenever possible. See the Segment Loader (SEGLDR) Reference Manual, publication SR-0066.

IMPLEMENTATION

This routine is available to users of both the COS and the UNICOS operating systems.

SEE ALSO

ldovl(1)
See the COS Reference Manual, publication SR-0011, for details of the **OVERLAY** routine.

NAME

PPL – Processes keywords of a directive

SYNOPSIS

CALL PPL(*cbuf,ctable,ltable,outarray,stattbl*)

DESCRIPTION

PPL processes the keywords for a given directive. Processing is governed by the Parameter Description Table, which has the same format as the table **GETPARAM** uses, except that the length of the table used by PPL is seven words with the two extra words unused.

cbuf Array containing the cracked image (usually prepared by **CRACK**, which is described in section 17)

ctable PPL control table

ltable Number of 7-word entries in PPL control table

outarray Array to receive parameter values

stattbl Three-word completion status code. On the first-time call, you must initialize the Return Status Table to zero. If PPL returns a status that is not normal, and PPL is called again with the invalid values left in, it attempts to recover.

Array element	Meaning
1	Return status code: **0** Normal termination **1** Required keyword not found **2** Output keyword overflow **3** Syntax error **4** Unknown or duplicate keyword **5** Unexpected separator encountered **6** Keyword cannot be equated **7** Keyword must have value **8** Maximum of 64 keywords exceeded **9** Invalid return status; cannot recover
2	Keyword in error
3	Ordinal keyword value

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

GETPARAM, CRACK

NAME

REMARK2, REMARK – Enters a message in the user and system log files

SYNOPSIS

CALL REMARK2(*message*)
CALL REMARK(*message*)

DESCRIPTION

message For **REMARK2**, message terminated by a 0 byte or a maximum of 79 characters. For **REMARK**, message terminated by a 0 byte or a 71-character message.

REMARK2 enters a message in the user and system log files. **REMARK** enters a message preceded by the prefix **'UT008 - '** in the user and system logfiles.

Under UNICOS, these routines write to **stderr** instead of the system logfile.

IMPLEMENTATION

These routines are available to users of both the COS and UNICOS operating systems.

NAME

REMARKF – Enters a formatted message in the user and system logfiles

SYNOPSIS

CALL REMARKF(*var,fvar,[fvar*$_2$*,...fvar*$_{12}$])

DESCRIPTION

var Variable containing the address of a format statement for **ENCODE**

fvar Address of variable

Up to 12 variables can be passed in arguments 2 through 13. The variables must be of type integer, real, or logical so that they each occupy only 1 word. The message is prefixed by **'UT009 – '** unless you supply a prefix. To supply the prefix, the characters '*b-b*' (*b*=blank) must appear in columns 6 through 8 of the formatted message.

EXAMPLE

Sample Fortran calling sequences with user-supplied prefixes:

```
10030   FORMAT ('CA001 - ', I4, ' errors')
        ASSIGN 10030 TO LABEL
        CALL REMARKF (LABEL, IERRCNT)
10770   FORMAT ('PD001 - ACCESS ', A8,A7,' ED=', I4, ';')
        ASSIGN 10770 TO LABEL
        CALL REMARKF (LABEL, DN(1), DN(2), ED)
```

Sample Fortran calling sequence without prefix:

```
10550   FORMAT ('LOOP EXECUTED ', I4, ' TIMES')
        ASSIGN 10550 TO LABEL
        CALL REMARKF (LABEL, LOOPCNT)
```

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

RERUN, NORERUN – Declares a job rerunnable/not rerunnable and instruct the system to begin or cease monitoring jobs for functions affecting rerunnability

SYNOPSIS

CALL RERUN(*param*)
CALL NORERUN(*param*)

DESCRIPTION

param One argument is required. For **RERUN**, if the argument is 0, the job can be rerun. If the argument is nonzero, the job cannot be rerun. For **NORERUN**, if the argument is 0, the system monitors for conditions causing the job to be flagged as not rerunnable. If nonzero, such conditions are not monitored.

RERUN declares a job rerunnable or not rerunnable.

NORERUN instructs the system to begin or cease monitoring jobs for functions affecting rerunnability.

IMPLEMENTATION

These routines are available only to users of the COS operating system.

NAME

SENSEBT – Determines whether bidirectional memory transfer is enabled or disabled

SYNOPSIS

CALL SENSEBT(*mode*)

DESCRIPTION

mode Transfer mode; *mode* has one of the following values:

= 1	Bidirectional memory transfer is enabled
= 0	Bidirectional memory transfer is disabled

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

SENSEFI – Determines if floating-point interrupts are permitted or prohibited

SYNOPSIS

CALL SENSEFI(*mode*)

DESCRIPTION

mode Interrupt mode:

mode=1	Permit interrupts
mode=0	Prohibit interrupts

IMPLEMENTATION

This routine is available to users of both the COS and UNICOS operating systems.

NAME

SETRPV – Conditionally transfers control to a specified routine

SYNOPSIS

CALL SETRPV(*rpvcode,rpvtab,mask***)**

DESCRIPTION

rpvcode Routine to which control is transferred

rpvtab A 40-word array reserved for system use

mask User mask specifying reprievable conditions

SETRPV transfers control to the specified routine when a user-selected reprievable condition occurs. **SETRPV** is called by address.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

SEE ALSO

See the Macros and Opdefs Reference Manual, publication SR-0012, for details of the **SETRPV** parameter formats.

NAME

SMACH, CMACH – Returns machine epsilon, small/large normalized numbers

SYNOPSIS

result=SMACH(*int*)

result=CMACH(*int*)

DESCRIPTION

result Machine constant returned

int An integer from 1 to 3. Any other value returns an error message to the logfile. For **SMACH**, *int* indicates that one of the following machine constants is to be returned:

Int	Constant	Description
1	.7105E-14	The machine epsilon (the smallest number ε such that $1.\pm\ \varepsilon \neq 1$).
2	.1290E-2449	A number close to the smallest normalized, representable number
3	.7750E+2450	A number close to the largest normalized, representable number

For **CMACH**, *int* indicates that one of the following machine constants is to be returned:

Int	Constant	Description
1	.7105E-14	The machine epsilon (the smallest number ε such that $1.\pm\ \varepsilon \neq 1$).
2	.1348E+1216	A number close to the square root of the smallest normalized, representable number
3	.7421E+1217	A number close to the square root of the largest normalized, representable number

The use of CMACH(2) and CMACH(3) prevents overflow during complex division.

These functions are calculated by Fortran versions of **SMACH** and **CMACH** (see the Basic Linear Algebra Subprograms for Fortran Usage by Chuck L. Lawson, Richard J. Hanson, Davis R. Kincaid, and Fred T. Crow, published by Sandia Laboratories, Albuquerque, 1977, publication number SAND77-0898).

IMPLEMENTATION

These routines are availale to users of both the COS and UNICOS operating systems.

NAME

SSWITCH – Tests the sense switch

SYNOPSIS

CALL SSWITCH(*swnum,result*)

DESCRIPTION

swnum Switch number (integer)

result *result* is 1 if the switch value ranges from 1 to 6 and the switch is on. *result* is 2 if the switch value is less than 1 or greater than 6, or if the switch is off (type integer).

IMPLEMENTATION

This routine is available only to users of the COS operating system.

NAME

SYSTEM – Makes requests of the operating system

SYNOPSIS

status=**SYSTEM**(*function*,arg_1,arg_2)

DESCRIPTION

status	Status returned in S1 register (function dependent)
function	System action request number. This is the octal code of the desired system action request. The requests (which all begin with the characters **f$**) and their codes are described in the COS Internal Reference Manual Volume II: STP, publication SM-0141. The code is the jump table address (relative offset) of the function.
arg_1	Optional argument (required by some requests)
arg_2	Optional argument (required by some requests)

NOTE

Use of the SYSTEM command by other than CRI systems programmers is discouraged, as the details of systems request formats are subject to change. In most cases, there is a library routine which performs the desired functions and makes changes in request formats transparent to your program.

IMPLEMENTATION

This routine is available only to users of the COS operating system.

18. INTERFACES TO C LIBRARY ROUTINES

A number of Fortran callable interfaces to C library routines are available under UNICOS. These routines give a Fortran programmer access to an extensive number of routines and system calls found in the C library. The interfaces are simple routines which resolve calling sequence differences and provide uppercase entry point names. Argument lists and return values should match those of the corresponding C routine, except where noted otherwise. Data types need to be handled as follows:

- C character data should be defined as Fortran integer and terminated by a null (zero) byte; 'L' Hollerith data handles this for 1-7 characters in length.
- C pointers should be handled by Fortran integers
- Other C data types are compatible with their Fortran counterparts

Interface routines should be coded as Fortran functions.

Example:

```
INTEGER FOPEN, FWRITE
ISTREAM = FOPEN( 'filenm'L, 'w+'L )
IF ( ISTREAM .EQ. 0 ) THEN
        PRINT *,' FOPEN failed '
        CALL ABORT
ENDIF
J = FWRITE( IDA(1), N, 8, ISTREAM )
```

If an argument to one of these routines is a file name, as in the above example, the name must be word-aligned and terminated by a null byte.

The following set of interface routines are provided in the standard CRAY X-MP UNICOS libraries. Refer to the appropriate Cray manuals for specific usage information.

C Library Reference Manual (SR-0136)		
Purpose	Name	Heading
Terminate a program and specify status	exit	**exit**
Close or flush a stream	**fclose**	**fclose**
Get integer file descriptor associated with stream	**fileno**	**ferror**
Open a stream	**fopen** **fdopen** **freopen**	**fopen**
Get a string from a stream	**fgets**	**gets**
Put a string on a stream	**fputs**	**puts**
Binary I/O	**fread** **fwrite**	**fread**
Reposition a file pointer in a stream	**fseek** **ftell**	**fseek**
Return value for environment name	**getenv**	**getenv**
Get option letter from argument vector	**getopt**	**getopt**
Make a unique file name	**mktemp**	**mktemp**
Change or add value to the environment	**putenv**	**putenv**
Create a name for a temporary file	**tempnam**	**tempnam**

The argument list of the **getenv** routine differs from that of the corresponding C routine. See the man page in this section for the correct syntax when calling **getenv** from Fortran.

UNICOS System Calls Manual (SR-2012)		
Purpose	Name	Heading
Determine accessibility of a file	**access**	**access**
Close a file descriptor	**close**	**close**
Allocate storage for a file	**ialloc**	**ialloc**
Move read/write file pointer	**lseek**	**lseek**
Change data segment space allocation	**sbreak** **sbrk**	**brk**
Provide signal control Fortran interface to **sigctl** Pascal interface to **sigctl**	**sigctl** **fsigctl** **psigctl**	**sigctl**
Specify what to do upon receipt of a signal Fortran interface to **signal** Pascal interface to **signal**	**signal** **fsignal** **psignal**	**signal**
Change size of secondary data segment	**ssbreak**	**ssbreak**
Read, write to secondary data segment	**ssread** **sswrite**	**ssread**
Get file status	**stat**	**stat**
Get time	**time**	**time**
Set and get file creation mark	**umask**	**umask**
Get name of current operating system	**uname**	**uname**
Remove directory entry	**unlink**	**unlink**

The argument lists of the **uname** and **time** routines differ from those of the corresponding C routines. No arguments can be used with the Fortran call to **time**. See the man page in this section for the correct syntax when calling **uname** from Fortran.

The third argument of the Fortran routines **ssread** and **sswrite** specifies the number of words to be read or written. This is different from the corresponding system call. The Fortran programmer should not call **ssbreak, ssread,** or **sswrite** in a program that accesses the SDS using the **assign**(1) command.

NAME

getenv – Returns value for environment name

SYNOPSIS

INTEGER GETENV
INTEGER *value(valuesz)*
int = GETENV(*name,value,valuesz*)

DESCRIPTION

int GETENV returns 1 if *name* was found in the environment and 0 if not.

name The name of the environmental variable for which GETENV searches in the environment list. The name must be left-justified and terminated with a zero byte.

value The value to which *name* is set, if found, in the current environment. This is a character string, and the *value* variable must be big enough to handle it.

valuesz Maximum number of words to hold string returned in *value*.

IMPLEMENTATION

This routine is available only to users of the UNICOS operating system.

SEE ALSO

getenv(3C) in the C Library Reference Manual, publication SR-0136
sh(1) in the UNICOS User Commands Reference Manual, publication SR-2011

NAME

GETOPT – Gets an option letter from an argument vector

SYNOPSIS

INTEGER FUNCTION GETOPT(*options,arg*)
CHARACTER(*) *options*
CHARACTER(*) *arg*

INTEGER FUNCTION GETOPT(*options,arg,argsz*)
CHARACTER(*) *options*
INTEGER *arg*(*)
INTEGER *argsz*

INTEGER GETVARG
morearg = **GETVARG**(*varg,vargsz*)

INTEGER GETOARG
morearg = **GETOARG**(*oarg,oargsz*)

DESCRIPTION

GETOPT returns the next option letter as the integer value of that ASCII code. For example, if the next option letter is **a**, the **GETOPT** returns with the value 97. If there is no next option letter, **GETOPT** returns zero. The **CHAR** routine can then be called to convert the integer back into a character.

The *options* argument is a string of recognized option letters. If the option letter encountered does not match one of the letters in the *options* string, an error is generated. If a letter in *options* is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space.

The *arg* argument returns the value of the argument following the option letter encountered. If *arg* is declared as a character variable, *argsz* need not be specified. If *arg* is declared as an integer array, *argsz* must be specified as the size of the array. The argument string is returned as characters packed in the integer array, terminated by a null byte.

If a letter in *options* is followed by a semicolon (;), zero or more arguments are expected for the option. You must then call **GETVARG** to get the variable arguments until **GETVARG** returns 0 before the next call to **GETOPT**.

The next variable argument is copied into the array *varg* (of size *vargsz*). **GETVARG** returns 0 when no more variable arguments exist.

After **GETOPT** returns 0, you can call **GETOARG** to get the remaining arguments from the command line.

GETOARG returns 0 if there are no more arguments. The next remaining argument is copied into the array *oarg* (of size *oargsz*).

If **GETOPT** is not used, **GETOARG** can be called to get the command line arguments in order, starting with the first argument.

EXAMPLE

The following example shows how the options of a command might be processed using **GETOPT**. This example assumes the options **a** and **b**, which have arguments, and **x** and **y**, which do not.

```
CHARACTER*8 OPTIONS
CHARACTER*80 ARGMNTS
CHARACTER OPTLET
```

```
      INTEGER OPTVAL
      DATA OPTIONS/'a:b:xy'/

 100  CONTINUE
      OPTVAL = GETOPT(OPTIONS, ARGMNTS)
      IF(OPTVAL .EQ. 0) GOTO 200
      OPTLET = CHAR(OPTVAL)
      IF (OPTLET .EQ. 'a') THEN
      * Analyze arguments from ARGMNTS
       ELSEIF (OPTLET .EQ. 'b') THEN
      * Analyze arguments from ARGMNTS
       ELSEIF (OPTLET .EQ. 'x') THEN
      * Process x option
       ELSEIF (OPTLET .EQ. 'y') THEN
      * Process y option
      ENDIF
 200  CONTINUE
```

The following example illustrates the use of **GETOPT** and **GETOARG** together.

```
      program test
      external getopt,getoarg
      integer getopt, getoarg
      integer arglen
      parameter (arglen=10)
      integer opt,done,argbuf(arglen)

10    CONTINUE
      OPT = GETOPT ('abo:',ARGBUF,ARGLEN)
      IF (OPT .GT. 0) THEN
        IF (OPT .EQ. 'a'R) THEN
          print '(a)' , ' option -a- present '

        ELSEIF (OPT .EQ. 'b'R) THEN
          print '(a)' , ' option -b- present '

        ELSEIF (OPT .EQ. 'o'R) THEN
          print '(a,a8)' , ' option -o- present-',argbuf(1)

        ELSE
C           unknown option
          print '(a,a8)' , ' bad option present-',opt
        ENDIF
        GO TO 10
      ENDIF
C  all options processed.
C
C  Get arguments
 20   CONTINUE
      DONE = GETOARG(ARGBUF,ARGLEN)
      IF(DONE .NE. 0) THEN
       print '(a,a8)' , ' argument present-',argbuf(1)
       GO TO 20
      ENDIF
```

```
C  done processing arguments
      end
```

RETURN VALUE

The value of **GETOPT** is 0 when no option characters can be found. **GETOPT** prints an error message on **stderr** and returns a question mark when it encounters an option letter not included in *options*.

NAME

uname – Gets name of current operating system

SYNOPSIS

CALL UNAME(*sysname, nodename, release, version, machine*)

DESCRIPTION

The **uname** routine returns information identifying the current operating system. The arguments, which are all of type **CHARACTER**, are as follows:

sysname Current operating system name

nodename Name by which the system is known on a communications network

release Release of the operating system

version Release version of the operating system

machine Standard name identifying the hardware on which the operating system is running

IMPLEMENTATION

This routine is available only to users of the UNICOS operating system.

SEE ALSO

uname(1) in the UNICOS User Commands Reference Manual, publication SR-2011
uname(2) in the UNICOS System Calls Reference Manual, publication SR-2012

19. MISCELLANEOUS UNICOS ROUTINES

This section contains descriptions of various specialized UNICOS libraries or miscellaneous routines that are not included elsewhere in this manual.

Miscellaneous Routines and Libraries		
Purpose	Name	Entry
Update CRT screens	**CURSES**	**CURSES**
System call interface to Fortran	**SYSCALL**	**SYSCALL**
Text interface to X Window System	**XIO**	**XIO**
C language X Window System Interface Library	**XLIB**	**XLIB**

NAME

curses – Updates CRT screens

SYNOPSIS

#include <curses.h>
cc [flags] files **–lcurses** [libraries]

DESCRIPTION

The **curses** routines give you a method of updating screens with reasonable optimization. In order to initialize the routines, the routine **initscr()** must be called before any of the other routines that deal with windows and screens are used. The routine **endwin()** should be called before exiting. To get character-at-a-time input without echoing, (most interactive, screen oriented-programs want this) after calling **initscr()** you should call **"nonl(); cbreak(); noecho();"**

The full **curses** interface permits manipulation of data structures called **windows** that can be thought of as two dimensional arrays of characters representing all or part of a CRT screen. A default window called **stdscr** is supplied, and others can be created with **newwin**. Windows are referred to by variables declared **WINDOW***, the type **WINDOW*** is defined in **curses.h** to be a C structure. These data structures are manipulated with functions described below, among which the most basic are **move**, and **addch**. (More general versions of these functions are included with names beginning with 'w', allowing you to specify a window. The routines not beginning with 'w' affect **stdscr**.) Then **refresh()** is called, telling the routines to make the user's CRT screen look like **stdscr**.

Mini-Curses is a subset of **curses** that does not allow manipulation of more than one window. To invoke this subset, use **-DMINICURSES** as a **cc** option. This level is smaller and faster than full **curses**.

If the environment variable **TERMINFO** is defined, any program using **curses** checks for a local terminal definition before checking in the standard place. For example, if the standard place is **/usr/lib/terminfo**, and **TERM** is set to **vt100**, then normally the compiled file is found in **/usr/lib/terminfo/v/vt100**. (The **v** is copied from the first letter of **vt100** to avoid creation of huge directories.) However, if **TERMINFO** is set to **/usr/mark/myterms**, **curses** first checks **/opusr/mark/myterms/v/vt100**, and if that fails, checks **/usr/lib/terminfo/v/vt100**. This is useful for developing experimental definitions or when write permission in **/usr/lib/terminfo** is not available.

FUNCTIONS

Routines listed here may be called when using the full **curses**. Those marked with an asterisk may be called when using Mini-Curses.

Routine	Description
addch(*ch*)*	Adds a character to stdscr (like **putchar**) (wraps to next line at end of line)
addstr(*str*)*	Calls **addch** with each character in *str*
attroff(*attrs*)*	Turns off attributes named
attron(*attrs*)*	Turns on attributes named
attrset(*attrs*)*	Sets current attributes to *attrs*
baudrate()*	Current terminal speed
beep()*	Sounds beep on terminal
box(*win, vert, hor*)	Draws a box around edges of *win* *vert* and *hor* are characters to use for vertical and horizontal edges of box

Routine	Description
clear( )	Clears stdscr
clearok(*win*, *bf*)	Clears screen before next redraw of *win*
clrtobot( )	Clears to bottom of stdscr
clrtoeol( )	Clears to end of line on stdscr
cbreak( )*	Sets **cbreak** mode
delay_output(*ms*)*	Inserts *ms* millisecond pause in output
delch( )	Deletes a character
deleteln( )	Deletes a line
delwin(*win*)	Deletes *win*
doupdate( )	Updates screen from all **wnooutrefresh**
echo( )*	Sets echo mode
endwin( )*	Ends window modes
erase( )	Erases stdscr
erasechar( )	Returns user's erase character
fixterm( )	Restores tty to "in curses" state
flash( )	Flashs screen or beep
flushinp( )*	Throws away any typeahead
getch( )*	Gets a character from tty
getstr(*str*)	Gets a string through stdscr
gettmode( )	Establishes current tty modes
getyx(*win*, *y*, *x*)	Gets (*y*, *x*) co-ordinates
has_ic( )	True if terminal can do insert character
has_il( )	True if terminal can do insert line
idlok(*win*, *bf*)*	Uses terminal's insert/delete line if bf != 0
inch( )	Gets char at current (*y*, *x*) co-ordinates
initscr( )*	Initializes screens
insch(*c*)	Inserts a character
insertln( )	Inserts a line
intrflush(*win*, *bf*)	Interrupts flush output if *bf* is **TRUE**
keypad(*win*, *bf*)	Enables keypad input
killchar( )	Returns current user's kill character
leaveok(*win*, *flag*)	OK to leave cursor anywhere after refresh if flag!=0 for *win*, otherwise cursor must be left at current position.
longname( )	Returns verbose name of terminal
meta(*win*, *flag*)*	Allows meta characters on input if flag != 0
move(*y*, *x*)*	Moves to (*y*, *x*) on stdscr
mvaddch(*y*, *x*, *ch*)	Move(*y*, *x*) then **addch**(*ch*)
mvaddstr(*y*, *x*, *str*)	similar...
mvcur(*oldrow*, *oldcol*, *newrow*, *newcol*)	Low level cursor motion
mvdelch(*y*, *x*)	like **delch**, but **move**(*y*, *x*) first
mvgetch(*y*, *x*)	etc.
mvgetstr(*y*, *x*)	
mvinch(*y*, *x*)	
mvinsch(*y*, *x*, *c*)	
mvprintw(*y*, *x*, *fmt*, *args*)	
mvscanw(*y*, *x*, *fmt*, *args*)	
mvwaddch(*win*, *y*, *x*, *ch*)	

Routine	Description
mvwaddstr(*win, y, x, str***)**	
mvwdelch(*win, y, x***)**	
mvwgetch(*win, y, x***)**	
mvwgetstr(*win, y, x***)**	
mvwin(*win, by, bx***)**	
mvwinch(*win, y, x***)**	
mvwinsch(*win, y, x, c***)**	
mvwprintw(*win, y, x, fmt, args***)**	
mvwscanw(*win, y, x, fmt, args***)**	
newpad(*nlines, ncols***)**	Creates a new pad with given dimensions
newterm(*type, fd***)**	Sets up new terminal of given type to output on *fd*
newwin(*lines, cols, begin_y, begin_x***)**	Creates a new window
nl()*	Sets newline mapping
nocbreak()*	Unsets **cbreak** mode
nodelay(*win, bf***)**	Enables **nodelay** input mode through **getch**
noecho()*	Unsets **echo** mode
nonl()*	Unsets newline mapping
noraw()*	Unsets **raw** mode
overlay(*win1, win2***)**	Overlays *win1* on *win2*
overwrite(*win1, win2***)**	Overwrites *win1* on top of *win2*
pnoutrefresh(*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol***)**	Like **prefresh** but with no output until **doupdate** called
prefresh(*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol***)**	Refreshes from pad starting with given upper left corner of pad with output to given portion of screen
printw(*fmt, arg1, arg2, ...***)**	Does **printf** on stdscr
raw()*	Sets raw mode
refresh()*	Makes current screen look like stdscr
resetterm()*	Sets tty modes to "out of curses" state
resetty()*	Resets tty flags to stored value
saveterm()*	Saves current modes as "in curses" state
savetty()*	Stores current tty flags
scanw(*fmt, arg1, arg2, ...***)**	Does scanf through stdscr
scroll(*win***)**	Scrolls *win* one line
scrollok(*win, flag***)**	Allows terminal to scroll if flag != 0
set_term(*new***)**	Now talk to terminal *new*
setscrreg(*t, b***)**	Sets user scrolling region to lines *t* through *b*
setterm(*type***)**	Establishes terminal with given type
setupterm(*term, filenum, errret***)**	
standend()*	Clears standout mode attribute
standout()*	Sets standout mode attribute
subwin(*win, lines, cols, begin_y, begin_x***)**	Creates a subwindow

Routine	Description
touchwin(*win*)	Changes all of *win*
traceoff()	Turns off debugging trace output
traceon()	Turns on debugging trace output
typeahead(*fd*)	Use file descriptor *fd* to check typeahead
unctrl(*ch*)*	Printable version of *ch*
waddch(*win, ch*)	Adds character to *win*
waddstr(*win, str*)	Adds string to *win*
wattroff(*win, attrs*)	Turns off *attrs* in *win*
wattron(*win, attrs*)	Turns on *attrs* in *win*
wattrset(*win, attrs*)	Sets *attrs* in *win* to *attrs*
wclear(*win*)	Clears *win*
wclrtobot(*win*)	Clears to bottom of *win*
wclrtoeol(*win*)	Clears to end of line on *win*
wdelch(*win, c*)	Deletes character from *win*
wdeleteln(*win*)	Deletes line from *win*
werase(*win*)	Erases *win*
wgetch(*win*)	Gets a character through *win*
wgetstr(*win, str*)	Gets a string through *win*
winch(*win*)	Gets character at current (*y, x*) in *win*
winsch(*win, c*)	Inserts character into *win*
winsertln(*win*)	Inserts line into *win*
wmove(*win, y, x*)	Sets current (*y, x*) co-ordinates on *win*
wnoutrefresh(*win*)	Refreshes but no screen output
wprintw(*win, fmt, arg1, arg2, ...*)	Does **printf** on *win*
wrefresh(*win*)	Makes screen look like *win*
wscanw(*win, fmt, arg1, arg2, ...*)	Do **scanf** through *win*
wsetscrreg(*win, t, b*)	Sets scrolling region of *win*
wstandend(*win*)	Clears standout attribute in *win*
wstandout(*win*)	Sets standout attribute in *win*

TERMINFO LEVEL ROUTINES

These routines should be called by programs wishing to deal directly with the terminfo database. Due to the low level of this interface, use of them is discouraged. Initially, **setupterm** should be called. This defines the set of terminal dependent variables defined in **terminfo(4F)**. The include files **<curses.h>** and **<term.h>** should be included to get the definitions for these strings, numbers, and flags. Parmeterized strings should be passed through **tparm** to instantiate them. All **terminfo** strings (including the output of **tparm**) should be printed with **tputs** or **putp**. Before exiting, **resetterm** should be called to restore the tty modes. (Programs desiring shell escapes or suspending with control Z can call **resetterm** before the shell is called and **fixterm** after returning from the shell.)

Routine	Description
fixterm()	Restores tty modes for **terminfo** use (called by **setupterm**)
resetterm()	Resets tty modes to state before program entry

Routine	Description
setupterm(*term, fd, rc*)	Reads in database. Terminal type is the character string *term*, all output is to UNCOS System file descriptor *fd*. A status value is returned in the integer pointed to by *rc*: 1 is normal. The simplest call would be **setupterm(0, 1, 0)** which uses all defaults.
tparm(*str, p1, p2, ..., p9*)	Instantiates string *str* with parameters p_i.
tputs(*str, affcnt, putc*)	Applies padding information to string *str*. *affcnt* is the number of lines affected, or 1 if not applicable. *Putc* is a **putchar**-like function to which the characters are passed, one at a time.
putp(*str*)	Calls **tputs** (*str*, 1, *putchar*)
vidputs(*attrs, putc*)	Outputs the string to put terminal in video attribute mode *attrs*, which is any combination of the attributes listed below. Characters are passed to **putchar**-like function *putc*.
vidattr(*attrs*)	Like **vidputs** but outputs through **putchar**

TERMCAP COMPATIBILITY ROUTINES

These routines were included as a conversion aid for programs that use termcap. Their parameters are the same as for **termcap**. They are emulated using the **terminfo** database. They may go away at a later date.

Routine	Description
tgetent(*bp, name*)	Looks up **termcap** entry for *name*
tgetflag(*id*)	Gets Boolean entry for *id*
tgetnum(*id*)	Gets numeric entry for *id*
tgetstr(*id, area*)	Gets string entry for *id*
tgoto(*cap, col, row*)	Applies parameters to given *cap*
tputs(*cap, affcnt, fn*)	Applies padding to *cap* calling *fn* as **putchar**

ATTRIBUTES

The following video attributes can be passed to the functions **attron,attroff,attrset.**

Attribute	Description
A_STANDOUT	Terminal's best highlighting mode
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_BLANK	Blanking (invisible)
A_PROTECT	Protected
A_ALTCHARSET	Alternate character set

FUNCTION KEYS

The following function keys might be returned by **getch** if **keypad** has been enabled. Note that not all of these are currently supported, due to lack of definitions in **terminfo** or the terminal not transmitting a unique code when the key is pressed.

Name	Value	Key name
KEY_BREAK	0401	Break key (unreliable)
KEY_DOWN	0402	The four arrow keys ...
KEY_UP	0403	
KEY_LEFT	0404	
KEY_RIGHT	0405	...
KEY_HOME	0406	Home key (upward+left arrow)
KEY_BACKSPACE	0407	Backspace (unreliable)
KEY_F0	0410	Function keys. Space for 64 is reserved.
KEY_F(*n*)	(KEY_F0+(*n*))	Formula for *fn*.
KEY_DL	0510	Delete line
KEY_IL	0511	Insert line
KEY_DC	0512	Delete character
KEY_IC	0513	Insert character or enter insert mode
KEY_EIC	0514	Exit insert character mode
KEY_CLEAR	0515	Clear screen
KEY_EOS	0516	Clear to end of screen
KEY_EOL	0517	Clear to end of line
KEY_SF	0520	Scroll 1 line forward
KEY_SR	0521	Scroll 1 line backwards (reverse)
KEY_NPAGE	0522	Next page
KEY_PPAGE	0523	Previous page
KEY_STAB	0524	Set tab
KEY_CTAB	0525	Clear tab
KEY_CATAB	0526	Clear all tabs
KEY_ENTER	0527	Enter or send (unreliable)
KEY_SRESET	0530	Soft (partial) reset (unreliable)
KEY_RESET	0531	Reset or hard reset (unreliable)
KEY_PRINT	0532	Print or copy
KEY_LL	0533	Home down or bottom (lower left)

IMPLEMENTATION

These routines are available only to users of the UNICOS operating system.

SEE ALSO

terminfo(4F) in the UNICOS File Formats and Special Files Reference Manual, publication SR-2014

NAME

xio – Text interface to the X Window System

SYNOPSIS

Display *
xstart(program, disp, evfunc)
char *program;
char *disp;
int (*evfunc)();
TEXT *
xopen(prompt, geom)
char *prompt;
char *geom;
xclose(win)
TEXT *win;
TEXT *
xtitle(pwin)
TEXT *pwin;
xprintf(win, format [, arg] ...)
TEXT *win;
char *format;
xputc(c, win)
TEXT *win;
char c;
xputs(s, win)
TEXT *win;
char *s;
xflush(win)
TEXT *win;
xevents()
xselect(win, mask)
TEXT *win;
long mask;
xunselect(win, mask)
TEXT *win;
long mask;
xconfigure(win, nw, nh, xw, xh)
TEXT *win;
int nw, nh, xw, xh;
Window
xfindwindow(prompt)
int (*prompt)();

DESCRIPTION

These functions provide a standard I/O like interface to the X Window System to a single display. The **xstart** routine is used initialize the display. **program** is used to extract the following variables from **~/.Xdefaults:**

BodyFont	**BorderWidth**	**Foreground**	**Background**	**Border**
ReverseVideo				

If disp is nonzero, it refers to the display name. If it is zero then the environment varaiable **DISPLAY** is used as the display name. The **evfunc** is used by the **xevent** function (see below). **xstart** returns non zero if the contact is made with the display.

The **xopen** routine is used to open a new window on the display started by **xstart** The **geom** argument specifies a standard X geometry (i.e =width x height + xoff + yoff). **xopen** returns a non null TEXT pointer if it succeeds.

xclose closes and destroys the window refered to by **win**

xtitle returns a TEXT pointer to a one line title subwindow contained in the window **pwin**. It is a violation to open a title in a title or try to open more than one title in a window.

xprintf, xputc, xputs, and **xflush** work as their **stdio** counterparts **fprintf, fputc, fputs**, and **fflush.**

xevents handles X events and calls **evfunc** from above for any event it does not know how to deal with. It passes **evfunc** a pointer to the **XEvent** structure. This routine must be called whenever there is input waiting on the file descriptor associated with X (**dpyno()** in C will return the file descriptor).

xselect allows the selection of more events on the TEXT window.

xunseletc allows the deselections of events selected via **xselect.**

xconfigure sets a minimum and maximum size for the TEXT window. Setting any value to 0 will remove the limit for that value.

xfindwindow grabs the server, makes the mouse a target, calls the **prompt** routine (which should ask the user to select a window) and returns the window ID of the window selected.

IMPLEMENTATION

These routines are available only to users of the UNICOS operating system.

SEE ALSO

Complete documentation for the text interface to the X Window System, is in the Xlib – C Language X Interface Protocol Version 10 by Jim Gettys and Tony Della Fera of the Digital Equipment Corporation, and Ron Newman of the Massachusetts Institute of Technology.

NOTE

The **X Window System** is a trademark of MIT.

NAME

Xlib – C Language X Window System Interface Library

SYNOPSIS

#include <X/Xlib.h>

DESCRIPTION

This library is the low level interface for C to the X protocol, which supports the X Window System, X Version 10, January 1986, from M.I.T. At present, the X Window System comprises more than 150 subroutines.

This library gives complete access to all capability provided by the X Window System (protocol version 10), and is intended to be the basis for other higher level libraries for use with X.

FILES

/usr/include/X/Xlib.h, /usr/lib/libX.a

IMPLEMENTATION

This library is available only to users of the UNICOS operating system.

SEE ALSO

Complete documentation for the C language interface to the X Window System, is in the Xlib – C Language X Interface Protocol Version 10 by Jim Gettys and Tony Della Fera of the Digital Equipment Corporation, and Ron Newman of the Massachusetts Institute of Technology.

INDEX

Index

READER'S COMMENT FORM

Programmer's Library Reference Manual SR-0113 C

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
____ other (please specify): ____________________________
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
____for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name ________________________
Title ________________________
Company ________________________
Telephone ________________________
Today's Date ________________________

Address ________________________
City ________________________
State/ Country ________________________
Zip Code ________________________

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

STAPLE

READER'S COMMENT FORM

Programmer's Library Reference Manual SR-0113 C

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
____ other (please specify): ______________________________
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
____for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name ____________________________
Title _____________________________
Company __________________________
Telephone _________________________
Today's Date _______________________

Address __________________________
City _____________________________
State/ Country _____________________
Zip Code __________________________

102 641771

FOLD

CUT ALONG THIS LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY
RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

STAPLE