NASA Contractor Report 191469

/75553
p. 55

Advanced Transport Operating System (ATOPS) Utility Library Software Description

Winston C. Clinedinst
Christopher J. Slominski
Richard W. Dickson
David A. Wolverton

Computer Sciences Corporation
Hampton, Virginia

Prepared For
Langley Research Center
under Contract NAS1-19038
April 1993

(NASA-CR-191469) ADVANCED TRANSPORT OPERATING SYSTEM (ATOPS) UTILITY LIBRARY SOFTWARE DESCRIPTION Final Report, Jun. 1988 - Nov. 1991 (Computer Sciences Corp.) 55 p

N93-32218

Unclas

G3/06 0175553

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-0001

TABLE OF CONTENTS

INTRODUCTION 1

ANGL 2

ANGL360 3

ASSIGN 4

BCDTIM 5

C_HDL 6
    REPORT_CHECK 7
    REPORT 8
    SHOW_TT 9
    EXCEPTIONS 10

CLIP 11

CLRBUF 13

DEGVAL 14

FMTDEG 15

FMTTIM 16

FRMFRQ 17

GET 18

GET_CHAR 19

GRID 20

LOCK 21

LUNAVA 22

LURTE 23

LURWY 24

LUSID 25

MAG_VAR 26

MAPCOM 27

MXM 28

MXT 29

MXV 30

OTS$FLOAT 31

P_LIST 32

POLAR 33

POSBTS 34

RET 35

SCOS 36

TIMVAL 37

UNPK 38

UVC 39

VCP 40

VDP 41

VMAX 42

VMG 43

VMIN 44

VSUM 45

VXM 46

XYZ 47

APPENDIX A 48

INTRODUCTION

This document describes the routines and functions of the Utility Library which are used by the flight software processes in the Digital Equipment Corporation VAX computers on the Transport Systems Research Vehicle. The software described herein is part of the baseline system released in November 1991.

Modules described in this document are organized alphabetically by name. Refer to Appendix A for a cross reference.

BIBLIOGRAPHY:
Digital Equipment Corp., VAX/VMS Manuals, Digital Equipment Corp. 1984.

MODULE NAME: ANGL (Function)

FILE NAME: ANGL.MAR

PURPOSE: To convert an angle from the 360° scale to the ±180° scale.

CALLED BY: See subroutine/function cross reference (appendix A).

CALLING SEQUENCE: angle = ANGL(angle)

Where: angle = 32-bit floating point angle in degrees

CALLS TO: None

DESCRIPTION:

This routine accepts any 32-bit floating-point number as input and returns an angle in the ±180° range. If the absolute value of the number is greater than 360°, the correct quadrant is determined by using a modulo technique and the resultant ±180° value is returned. No error checking is performed by this function. An assumption is made that the argument is in 32-bit floating-point format.

MODULE NAME:	ANGL360 (Function)
FILE NAME:	ANGL360.MAR
PURPOSE:	To convert input angle to ±360 degree range.
CALLED BY:	None
CALLING SEQUENCE:	Angle = ANGL360(Angle)
Where:	Angle = 32-bit floating point value in degrees
CALLS TO:	None

DESCRIPTION:

This routine accepts any single precision floating-point number as input and returns an angle in the ±360° range. The sign of the output is determined by the sign of the input. No error checking is performed by this function, and an assumption is made that the argument is in 32-bit floating-point format.

MODULE NAME: ASSIGN

FILE NAME: ASSIGN.FOR

PURPOSE: To assign an equivalent name in the default logical name table for a process.

CALLED BY: See subroutine/function cross reference (appendix A).

CALLING SEQUENCE: CALL ASSIGN(logical_name, equiv_name)

Where: logical_name = logical name to be defined (or presently existing)
equiv_name = equivalence name to be defined for the logical name.

CALLS TO: SYS$CRELNM, LIB$SIGNAL

DESCRIPTION:
This routine uses the equiv_name parameter to create an item list and invokes the $CRELNM system service (see VAX/VMS System Service Reference Manual for more details) to create the equivalence name. If an error is encountered, the LIB$SIGNAL library routine (see VAX/VMS Run Time Library Routines Reference Manual for more details) is called to log an error message.

MODULE NAME:	BCDTIM
FILE NAME:	BCDTIM.FOR
PURPOSE:	To convert the Data Acquisition System (DAS) time from BCD to ASCII and place it in a byte string.
CALLED BY:	IDENT, REPORT
CALLING SEQUENCE:	CALL BCDTIM(h, ms, label)

Where:

h	= word containing hours digits and minutes digits.
ms	= word containing seconds digits
label	= 7 byte output string in following format: HHMM:SS

CALLS TO: MVBITS (see Programming in VAX FORTRAN for details)

DESCRIPTION:

Each nibble of the H and MS word is a BCD digit in the range 0-9. Subroutine MVBITS is called to extract each nibble. An octal 60 is then added to this BCD value to create an ASCII byte that represents the BCD digit. BCDTIM outputs a 7 byte character string in the following format: HHMM:SS. No error checking is performed.

MODULE NAME: C_HDL (Function)

FILE NAME: C_HDL.FOR

PURPOSE: To interrupt VAX/VMS exception processing, provide a brief display on the error logging device, and record expanded data in a log file for later analysis.

CALLED BY: See subroutine/function cross reference (appendix A).

CALLING SEQUENCE: A. FORTRAN:

```
        CALL LIB$ESTABLISH(C_HDL)
```

B. MACRO:

```
        MOVAB   C_HDL,(FP)
```

CALLS TO: LIBSIM_TRAP, LIBFIXUP_FLT, SYS$GETTIM, SYS$PUTMSG, LIB$MATCH_COND

DESCRIPTION:

C_HDL is a user supplied VAX/VMS condition handler designed to process the "MTH$" utility type errors; floating-point overflow, and divide by zero (both trap and fault) and others. For the ATOPS implementation each process uses a call to "LIB$ESTABLISH" to establish C_HDL as the default exception handler. (See the VAX/VMS Run-Time Library Manual for more details).

Each time C_HDL is invoked a counter is incremented. Therefore each process using this exception handler has the option of saving this counter in a global section for real time monitoring purposes. C_HDL contains three other user supplied routines: REPORT_CHECK, REPORT, and SHOW_TT. When C_HDL receives control, a subsidiary routine is invoked depending on the type of condition causing the exception. If the error occurred in mathematics procedures (with the exception of square root of a negative number) R0 and R1 are set to zero and control is passed to REPORT. If the exception was caused by an attempt to compute the square root of a negative number, a zero value and the SS$_CONTINUE status is returned, essentially ignoring the exception. If the exception is not one that C_HDL was designed to process, a status of SS$RESIGNAL is returned which causes the VAX/VMS default condition handler to service the exception.

For floating overflow fault (SS$_FLTOVF) and floating divide_fault (SS$_FLTDIV_F) the VAX/VMS Library Routine LIB$SIM_TRAP is called to convert the floating faults to floating traps which means these instructions will be executed again resulting in C_HDL being called again and this type of exception being counted twice. Subsequently control is passed to REPORT_CHECK. For the following traps - floating overflow, floating/decimal divided by zero, integer divide by zero, integer overflow - control is passed directly to REPORT_CHECK.

For a reserved operand (SS$_ROPPAND) the VAX/VMS LIB$FIXUP_FLT routine is called to change -0 to +0 and processing continues with C_HDL returning the SS$_CONTINUE status.

MODULE NAME:	REPORT_CHECK (Function)
FILE NAME:	C_HDL.FOR
PURPOSE:	To determine if an identical exception has occured in previous 15 seconds.
CALLED BY:	C_HDL
CALLING SEQUENCE:	c_hdl = REPORT_CHECK(sig)
Where:	c_hdl = return status of SS$_CONTINUE or SS$_RESIGNAL
CALLS TO:	None

DESCRIPTION:

REPORT_CHECK is a function that determines whether or not the identical exception has occurred previously and ensures that 15 seconds have elapsed prior to re-logging the error message. If it is the same exception and 15 seconds have not elapsed, then REPORT_CHECK does not log the message and returns the SS$_CONTINUE status.

If it is a new exception or 15 seconds have elapsed, the condition is changed to a warning only and REPORT is called to log the message. Then REPORT_CHECK returns the SS$_RESIGNAL status to cause the condition to be resignaled to the next handler.

MODULE NAME:	REPORT
FILE NAME:	C_HDL.FOR
PURPOSE:	To output an exception message.
CALLED BY:	REPORT_CHECK
CALLING SEQUENCE:	CALL REPORT(sig)
Where:	sig = the exception being signaled
CALLS TO:	BCDTIM, SYS$PUTMSG

DESCRIPTION:

REPORT is a subroutine that calls the SYS$PUTMSG system service to log an error in the log file for a process and to display an abbreviated error message on the system console display unit. The actual display is accomplished by the SHOW_TT function.

BCDTIM is then called to format the system time acquired from the Data Acquisition System for a time referencing entry placed in the log file for a process by calling SYS$PUTMSG a second time.

MODULE NAME:	SHOW_TT (Function)
FILE NAME:	C_HDL.FOR
PURPOSE:	To ouput exception message.
CALLED BY:	SYS$PUTMSG
CALLING SEQUENCE:	CALL SYS$PUTMSG(sig, SHOW_TT,,)

Where: sig = the exception being signaled
SHOW_TT = name of action routine for SYS$PUTMSG

CALLS TO: None

DESCRIPTION:

This is a function which outputs an abbreviated error message on the system display device. This function is activated when SYS$PUTMSG is called with SHOW_TT specified as an action routine. The abbreviated message has the format

'system error' in 'process'

where: **'system error'** is a VAX/VMS supplied error,
'process' is the name of the process in which the error occurred.

MODULE NAME: EXCEPTIONS

FILE NAME: EXCP.MSG

PURPOSE: Used by C_HDL to create system informational messages.

CALLED BY: N/A

CALLING SEQUENCE: N/A

CALLS TO: None

DESCRIPTION:

This module contains instructions used by the VAX/VMS Message facility to create an object file used for signaling user defined exceptions. The only user defined exception in the system is the informational message prefixed to exceptions reported in the process error logs. The prefixed message shows the VAX time and date, and the DAS time as follows:

```
%LOG-I-TIME, Error logged at 20-JUL-1991 16:38:39  (2240:25)
```

This code is accessed by the condition handler on the file C_HDL.FOR by the global symbol LOG_TIME which is created by the VAX/VMS message facility.

MODULE NAME: CLIP (Function)

FILE NAME: CLIP.FOR

PURPOSE: To process an x,y input pair to ensure that the vector lies within a defined box.

CALLED BY: See subroutine/function cross reference (appendix A).

CALLING SEQUENCE: Status = CLIP (x, y, left, right, top, bottom)

Where: Status = return code from CLIP as follows:

-1	- none of vector lies within box
0	- vector within box
1	- first point clipped to edge of box
2	- second point clipped to edge of box
3	- both points clipped to edge of box

x	= array of 2 x coordinates
y	= array of 2 y coordinates
left	= left side of box
right	= right side of box
top	= top of box
bottom	= bottom of box

CALLS TO: POSBTS

DESCRIPTION:

This function receives, as input, an x,y coordinate pair defining a line segment. It also receives coordinates defining all sides of a box (left, right, top, bottom) into which the line segment must fit. The POSBTS function is called to determine in which quadrant the vector endpoints lie. A return code of 0 indicates the endpoints are entirely inside the box. All possible return codes from POSBTS are indicated in the following diagram:

left right

top	1001	1000	1010
	0001	0000	0010
bottom	0101	0100	0110

CLIP then computes the slope of the line using the formula: $m=\frac{y2-y1}{x2-x1}$ and uses this to eliminate any points that lie outside the screen boundaries. This is done iteratively until the line lies inside the boundaries and the appropriate code is returned to the user.

MODULE NAME:	CLRBUF
FILE NAME:	CLRBUF.MAR
PURPOSE:	To fill a contiguous block of memory with zeroes.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL CLRBUF (block, length)
Where:	block = memory area to be cleared length = 16-bit integer word containing the number of bytes to clear
CALLS TO:	None

DESCRIPTION:

This routine uses the MACRO instruction 'MOVC' to move zeroes to the starting address specified in the first argument for the number of consecutive byte locations specified in the second parameter. No error checking is performed.

MODULE NAME:	DEGVAL (Function)
FILE NAME:	DEGVAL.MAR
PURPOSE:	To convert an ASCII string of LAT or LON to floating point.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	value = DEGVAL (string, length, error)

Where:

string	= byte array containing LAT or LON
length	= 16-bit integer string length
error	= 1-byte return code

CALLS TO: DIGITS, OTS$CVT_TU_L

DESCRIPTION:

This routine accepts a character string latitude or longitude value and returns the floating point representation in degrees. The length field is checked to determine whether the input is a latitude or longitude (seven characters denotes a latitude, eight characters denotes a longitude). Further error checking is performed to ensure that the seven character latitude begins with either N or S and that the eight character longitude begins with either E or W. The latitude is further checked to ensure that it does not exceed 89° 59' 59" and that longitude is checked to ensure it does not exceed 179° 59' 59".

An embedded subroutine, DIGITS, is called to convert the ASCII characters one section at a time - degrees, minutes, seconds. These values are then positionally adjusted and summed to form the resultant latitude or longitude. If any of the error checks fail, a one byte logical error flag is set true and returned to the caller.

DIGITS:

This is an embedded subroutine that calls the system routine OTS$CVT_TU_L. If any digit other than 0 through 9 is detected, the one byte error logical is set and the stack pointer altered such that the control returns to the error exit of DEGVAL. The error exit is also taken if the value returned exceeds the comparison value it receives from DEGVAL.

MODULE NAME:	FMTDEG
FILE NAME:	FMTDEG.MAR
PURPOSE:	To format a latitude or longitude into an ASCII string representing degrees, minutes and seconds.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL FMTDEG (input, string, type)

Where:

```
 input = 32-bit floating point value in degrees
string = output byte array
 type  = 0 denotes latitude
       = 1 denotes longitude
```

CALLS TO: OTS$CVT_L_TU

DESCRIPTION:

The 32-bit floating point input value is converted to an ASCII character string representing degrees, minutes and seconds. For a latitude input value, positive denotes north (N) and negative denotes south (S). For longitude inputs, positive denotes east (E) and negative denotes west (W). No error checking is done and the resultant character string is in the format XDD°MM'SS" for latitude and XDDD°MM'SS" for longitude, where:

```
  X = N, S, E, or W
DDD = degrees
 MM = minutes
 SS = seconds
```

The VAX/VMS routine library routine OTS$CVT_L_TU is called to convert the decimal value to an ASCII text string.

MODULE NAME:	FMTTIM
FILE NAME:	FMTTIM.MAR
PURPOSE:	To convert 4-byte integer time in seconds to an ASCII string representing hours, minutes, seconds.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL FMTTIM (input, string)

Where: input = long word integer containing seconds past midnight
string = output byte array

CALLS TO: OTS$CVT_L_TU

DESCRIPTION:

The longword of time (seconds past midnight) is fetched and processed to adjust for any values greater then one day (86,400 seconds). For purposes of flight path planning, a negative time could be passed to this routine indicating that the time falls into a previous day. The negative time is adjusted to create the correct time for the preceding day.

Following the previous tests, the hours, minutes and seconds are computed separately. Following each computation a call is made to the VAX/VMS library routine OTS$CVT_L_TU to convert the number to an ASCII value which is stored in the output byte array. The format of the output string is HHMM:SS where:

HH = hours since midnight
MM = minutes
SS = seconds

No error checking is done except to limit the input time to within ± 24 hours.

MODULE NAME:	FRMFRQ
FILE NAME:	FRMFRQ.MAR
PURPOSE:	To convert a 2/5 code frequency (navigation data base format) to an ASCII representation of its decimal value on the CDU.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL FRMFRQ (input, string)
Where:	input = longword containing 2/5 code string = output byte array to contain frequency code
CALLS TO:	None

DESCRIPTION:

This procedure converts a navaid tuning frequency from 2/5 code to ASCII for display on the CDU. The frequency is input in a form where only the one's digit and the tenth's digit are in true 2/5 form. The hundred's digit is assumed to always be 1, the hundredth's digit is limited to zero or 5 and the ten's digit is limited to values between zero and three. Each of these digits is processed separately as they require special limits on their values. The 2/5 code (in the one's and tenth's digits) is located in tables and the corresponding ASCII values are moved to the output buffer for display in the form XXX.XX. If a value is input in which any digit cannot be located in the 2/5 tables, a row of question marks is output to indicate an error.

MODULE NAME:	GET (Function)
FILE NAME:	GET.MAR
PURPOSE:	To fetch a data item from the navigation data base.
CALLED BY:	See subroutine/function cross reference (appendix A).
ENTRY POINTS:	GET_BYTE, GET_LONG, GET_REAL, GET_WORD
CALLING SEQUENCE:	value = GET_XXXX(address)
Where:	address = address of desired data in navigation data base.
CALLS TO:	None

DESCRIPTION:

This function fetches the longword of data beginning at the address passed and places it in register 0. The function type in the FORTRAN calling routine determines whether a byte, word, longword, or real is returned to the caller. No error messages or status is returned.

MODULE NAME:	GET_CHAR (Function)
FILE NAME:	GET_CHAR.MAR
PURPOSE:	To fetch a character string from the navigation data base.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	clnf = GET_CHAR(address)

Where:	address =	beginning address of desired character string in navigation data base.
	clnf =	CHARACTER*N variable

CALLS TO: None

DESCRIPTION:

This function fetches data from the navigation data base beginning with the input address. GET_CHAR is defined in each module that uses it as a CHARACTER*N function. "N" can be different for each module since information about the return data is passed by descriptor. No error message or status is returned.

MODULE NAME:	GRID
FILE NAME:	GRID.FOR
PURPOSE:	To compute an x,y grid displacement between two positions designated by their latitude and longitude values.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL GRID(lat1, lon1, lat2, lon2, x, y)

Where:

lat1	=	32-bit floating point reference latitude (input)
lon1	=	32-bit floating point reference longitude (input)
lat2	=	32-bit floating point destination latitude (input)
lon2	=	32-bit floating point destination longitude (input)
x	=	x grid displacement from reference lat/lon to destination lat/lon in feet(output)
y	=	y grid displacement from reference lat/lon to destination lat/lon in feet(output)

CALLS TO: None

DESCRIPTION:

This subroutine computes an x,y grid displacement between two positions designated by their respective latitude and longitude values. The local radius values for the north/south and east/west directions are computed using the reference latitude/longitude values. These reference values are saved in LATSV and LONSV so that the south radius computations are not repeated when GRID is called with identical reference values.

The x,y displacement in feet is then computed and returned to the caller. No error detection is performed, i.e; the values input are treated as real numbers in degrees and no error status is returned.

MODULE NAME:	LOCK
FILE NAME:	LOCK.MAR
PURPOSE:	To lock all pages of a process' working set into memory to prevent page swapping by VMS.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	status = SYS$CMEXEC(LOCK)
Where:	LOCK = name of routine to be executed in executive mode
CALLS TO:	$GETJPIW, $LKWSET

DESCRIPTION:

This routine must be in Executive Mode to perform its function. It calls the $GETJPI system service to retrieve the address of the first free page at the end of the program region (P0) of the process. The beginning address defaults to hexadecimal 200. The beginning address and ending address are used in a call to the $LKWSET system service to lock the specified range of pages in the working set (see the VAX/VMS System Services Reference Manual for a description of system services). Register 0 is set to indicate a successful status at the beginning of this routine, therefore no error status is returned and success is assumed.

MODULE NAME:	LUNAVA
FILE NAME:	LUNAVA.MAR
PURPOSE:	To locate in the navigation data base, return the address, and optionally return latitude and longitude of any one of the following: airfield, navaid or geographic reference point.
CALLED BY:	See subroutine/function cross reference (appendix A).

```
CALLING SEQUENCE: CALL LUNAVA (% REF(name), addr, lat, lon)
                    "   LUARP  (           "            )
                    "   LUGRP  (           "            )
```

Where:

name	=	byte array containing name of item to search for
addr	=	longword address of item in navigation database (output)
lat	=	32-bit floating point latitude in degrees (output)
lon	=	32-bit floating point longitude in degrees (output)

CALLS TO: None

DESCRIPTION:

Entry occurs at one of the following points, depending on whether an airfield, navaid, or geographic reference point in the navigation data base is to be accessed.

LUARP:	airfield
LUNAVA:	navaid
LUGRP:	Geographic Reference Point (GRP).

For the LUARP entry point, a four character name in the data base will be compared to the supplied argument. For the LUNAVA entry point, three characters are compared and for the LUGRP entry point, five characters are compared.

This routine searches the navigation data base for the name of the requested airfield, navaid or GRP. It searches each longitudinal strip by using the data base index block (INDBLK) to find a pointer to the first GRP, navaid or airfield in a longitudinal strip. If a match is found, the address of the item is returned. If it is not found, a zero address is returned. If four parameters are passed to this routine, the latitude and longitude of the requested item are also returned.

MODULE NAME:	LURTE
FILE NAME:	LURTE.MAR
PURPOSE:	To locate in the navigation data base and return the address of a Jet airway, Victor airway or Route name.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL LURTE (%REF(name), addr) CALL LUVIC (") CALL LUJET (")
Where:	name = 6 char name of route addr = longword (32-bit) address of route in navigation data base (AADCOM)
CALLS TO:	None

DESCRIPTION:

Control passes to the following entry points depending on whether a Jet Airway, Victor Airway or Route name in the navigation data base is to be accessed:

LUVIC:	Victor airway
LURTE:	Route name
LUJET:	Jet airway

The Victor airway, Jet airway or Route name is entered via the Control Display Unit (CDU). This routine searches the navigation data base to find a route/victor airway/jet airway of the same name. If a match is found, the address is returned, otherwise a zero is returned.

MODULE NAME:	LURWY
FILE NAME:	LURWY.MAR
PURPOSE:	To look up a runway address in the navigation data base at a given airfield.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL LURWY(%REF(name), afad, ryad[, lat, lon])

Where:

```
name = 3 char runway name
afad  = (input) airfield longword address
ryad  = (output) runway longword address
 lat  = (output) 32-bit floating point latitude in degrees
 lon  = (output) 32-bit floating point longitude in degrees
```

CALLS TO: None

DESCRIPTION:

This routine searches the navigation data base for a runway beginning at the given airfield address. If a match is found, the runway address is returned. If not, a zero is returned as the address. If five parameters were passed to the routine, the latitude and longitude of the runway threshold is also returned.

MODULE NAME:	LUSID
FILE NAME:	LUSID.MAR
PURPOSE:	To locate SID, STAR, or APPROACH names in the navigation database and return their address and type.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL LUSID (% REF(name), afad, ad, typ)

Where:
name = (INPUT) 6 character item name
afad = (INPUT) longword address of airfield
ad = (OUTPUT) longword address of item
typ = (OUTPUT) type of item

1 = STAR
0 = SID
-1 = APPROACH

CALLS TO: None

DESCRIPTION:

This routine searches for the name of the item amongst the Standard Instrument Departures (SIDs) at the given airfield. If a match is found, the address of the SID is returned as well as a type value of zero to indicate the item is a SID. If not found, the routine then searches the Standard Terminal Arrival Route (STAR) names at the given airfield. If a match is found, the address and a type value of 0 are returned. Otherwise the routine searches the approach names at the given airfield, and returns the address and type value of -1 if a match is found. If the item is not found as any of the three types, a zero address is returned.

MODULE NAME:	MAG_VAR (Function)
FILE NAME:	MAG_VAR.MAR
PURPOSE:	To compute a magnetic variation estimate at a specified latitude/longitude.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	magvar = MAG_VAR(latref, lonref)
Where:	latref = 32-bit floating point reference latitude in degrees lonref = 32-bit floating point reference longitude in degrees
CALLS TO:	None

DESCRIPTION:

This function is called from FM/FC software to compute a magnetic variation estimate at some desired locality. The method used for computing the magnetic variation is a two dimensional interpolation on values from a table. The table contains magnetic variation values for the entire globe with the exception of latitudes above North 73.125 or below South 73.125. Values are included for each 11.25 degree step in latitude and longitude.

This function first determines the quadrant containing the input position. Then the magnetic variation values for the four corners are found in the table. Finally, the following equation is used to compute the magnetic variation estimate:

$$magvar = m11 + D\_LAT * (m12 - m11) + D\_LON * (m21 - m11) + D\_LAT * D\_LON * (m22 + m11 - m12 - m21)$$

where:

m11	= magnetic variation at lower left of quadrant
m12	= magnetic variation at upper left of quadrant
m21	= magnetic variation at lower right of quadrant
m22	= magnetic variation at upper right of quadrant
D_LAT	= (latref - LAT11)/11.25
D_LON	= (lonref - LON11)/11.25
LAT11	= latitude at lower left of quadrant
LON11	= longitude at lower left of quadrant

No error checking is performed on the input position.

MODULE NAME:	MAPCOM
FILE NAME:	MAPCOM.MAR
PURPOSE:	To map to one or more global sections.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL MAPCOM(% VAL(selection), %VAL(access))

Where: selection = a longword containing a bit string with each 'on' bit selecting a global section.

access = for each bit set in 'selection' parameter, a corresponding 'on' bit in this parameter denotes write privilege.

-OR-

CALL MAPCOM(name)

Where: name = address of character string containing a process name.

CALLS TO: $MGBLSC

DESCRIPTION:

This procedure is called to map one or more global sections in physical memory to a process' virtual memory. The names of the global sections to map and the associated access privileges for each process are contained in a MACRO language file named MAPTBL.MAR. Specifically, three global symbols in the MAPTBL.MAR module are referenced by MAPCOM, these include BITS, ENTRIES, and TABLE. This table is used to find process names when the second calling sequence (noted previously) is used.

The first calling sequence is used by a process whose name is not included in the default table, or if selection or access privileges different from the default values for a process are desired.

The process containing the caller is terminated if the attempt to map the global section fails (the $MGBLSC VAX/VMS System Service returns an error code).

MODULE NAME:	MXM
FILE NAME:	MXM.MAR
PURPOSE:	To multiply matrices.
CALLED BY:	None
CALLING SEQUENCE:	CALL MXM (m1, r1, c1, m2, c2, m3)

Where:

m1	= 32-bit floating point matrix 1
r1	= 16-bit integer number of rows in m1
c1	= 16-bit integer number of columns in m1
m2	= 32-bit floating point matrix 2
c2	= 16-bit integer number of columns in m2
m3	= 32-bit floating point output matrix

CALLS TO: None

DESCRIPTION:

This routine performs multiplication of matrices containing single precision floating point data (32 bits). The matrices must meet the criteria that the number of columns in matrix 1 is equal to the number of rows in matrix 2.

MODULE NAME:	MXT
FILE NAME:	MXT.MAR
PURPOSE:	To calculate matrix times matrix transpose.
CALLED BY:	None
CALLING SEQUENCE:	CALL MXT (m1, r1, c1, m2, r2, m3);

Where:

m1	= 32-bit floating point matrix 1
r1	= 16-bit integer number of rows in m1
c1	= 16-bit integer number of columns in m1
m2	= 32-bit floating point matrix 2
c2	= 16-bit integer number of columns in m2
m3	= 32-bit floating point output matrix

CALLS TO: None

DESCRIPTION:

This routine performs multiplication of matrices containing single precision floating point data (32 bits). M2 is transposed prior to multiplication. The matrices must meet the criteria that the number of columns in matrix 1 is equal to the number of columns in matrix 2.

MODULE NAME:	MXV
FILE NAME:	MXV.MAR
PURPOSE:	Perform matrix times vector multiplication.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL MXV (m, v, ml, vl, dv)

Where:

m	= 32-bit floating point matrix
v	= 32-bit floating point vector
ml	= 16-bit integer count of rows in matrix
vl	= 16-bit integer vector length
dv	= 32-bit floating point destination vector

CALLS TO: None

DESCRIPTION:

Subroutine MXV multiplies a matrix times a vector, and stores the result in the destination vector. The matrix and vector are assumed to contain single precision floating-point data (32 bits). The number of columns in the matrix must equal the number of elements in the vector.

MODULE NAME:	OTS$FLOAT
FILE NAME:	OTS$FLOAT.MAR
PURPOSE:	To convert a floating point value to ASCII text.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL OTS$FLOAT(value, output, %VAL(f_digits))

Where: value = floating point value
output = output string
f_digits = digits of fraction desired

CALLS TO:	OTS$CVI_L_TI (VAX/VMS RunTime Library Routine)

DESCRIPTION:

This routine converts a single precision floating-point value (32 bits) to ASCII text. Both the integer portion and fractional portion of the number must be individually less than the maximum 32-bit integer (4, 294, 967, 296). The smallest fractional portion allowed is 10^{-9}. If an overflow in the integer portion or fractional portion of the number is detected, the output string is asterisk filled and control is returned to the caller. If the fraction width is greater than the total field width, if there is an integer portion and the fraction width exceeds the total string length, or if the integer portion is not large enough to contain the integer and a minus sign if the value is negative, the output string is asterisk filled.

If the input parameters pass the previously described validity checks, the integer portion and the fractional portion of the number are separated and integerized. A call is then made to OTS$CVT_L_TI to convert each portion to ASCII and place them in the output string.

MODULE NAME:	P_LIST
FILE NAME:	P_LIST.MAR
PURPOSE:	To provide information about the parameter list input to the caller.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL P_LIST(cnt, list)

Where:

cnt	=	returned byte value denoting argument count
list	=	string of discretes identifying each parameter as null or a value. A 32-bit integer value.

CALLS TO: None

DESCRIPTION:

This routine fetches the last argument pointer from the frame buffer. It then obtains the parameter count, returns it to the caller in CNT and uses it as a loop counter for testing the parameter list for null or actual parameters. A one-bit in the output word denotes an actual parameter and a zero-bit indicates a null parameter. A sample usage of P_LIST follows:

```
CALL SUB1(arc,, 3)
SUBROUTINE SUB1(p1, p2, p3, p4)
CALL P_LIST(cnt, list)
```

For the example CNT=3 and list = 00000005 hexadecimal.

MODULE NAME:	POLAR
FILE NAME:	POLAR.MAR
PURPOSE:	To convert a unit vector in rectangular coordinates to polar coordinates in degrees.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL POLAR(vector, lat, lon)

Where:	vector =	32-bit floating point unit vector in rectangular coordinate
	lat =	location to receive 32-bit floating point latitude in degrees
	lon =	location to receive 32-bit floating point longitude in degrees

CALLS TO: ASIND, ATAN2D

DESCRIPTION:

This routine accepts a unit vector in rectangular coordinates of the form [X, Y, Z] and returns polar coordinates in degrees, assuming a unit sphere. The following operations are used for this operation:

```
lat = ASIND (vector(1))
lon = ATAN2D (-vector(2), vector (3)).
```

No error checking is performed.

MODULE NAME: POSBTS (Function)

FILE NAME: POSBTS.FOR

PURPOSE: To determine which quadrant a point is in relative to a clip box.

CALLED BY: See subroutine/function cross reference (appendix A).

CALLING SEQUENCE: status = POSBTS (x, y, left, right, top, bottom)

Where:

x	=	x coordinate of endpoint
y	=	y coordinate of endpoint
left	=	left side of box
right	=	right side of box
top	=	top of box
bottom	=	bottom of box

CALLS TO: None

DESCRIPTION:

This routine determines in which quadrant a vector endpoint lies. The x, y coordinate and the box definition (left, right, top, bottom) are input to POSBTS and it supplies a return code indicating where the endpoint lies relative to the defined box as follows:

	left		right
top	1001	1000	1010
	0001	0000	0010
bottom	0101	0100	0110

A return code of zero indicates that the vector endpoint is inside the box.

MODULE NAME:	RET
FILE NAME:	RET.MAR
PURPOSE:	To provide FORTRAN procedures the capability to return to modules other than the caller.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL RET(n)
Where:	n = desired return level (longword integer)
CALLS TO:	None

DESCRIPTION:

This routine uses the input return level as a loop counter as it unwinds the stack frames. An input value of 2, for example, causes control to pass to the caller's caller. Extreme caution should be exercised in the use of this routine since no error checking is performed. Passing an incorrect level number has unpredictable results.

MODULE NAME:	SCOS
FILE NAME:	SCOS.MAR
PURPOSE:	To compute the SINE and COSINE of an angle measured in degrees.
CALLED BY:	ACCPRC, APPREF, BLOW, CRBSC, EARTH_VEC, ERAD, EXECUTE, FIX_ERAD, HNAVFS, HNAVML, HNAVSL, LATCMD, NEW_POS, PATH, POINTS, PROJPOINT, RSCON, TRALCBA, UNITVEC, XFORM, XYZIN
CALLING SEQUENCE:	CALL SCOSD (angle, sine, cosine)

Where: angle = 32-bit floating point angle in degrees (input)
sine = 32-bit floating point sine of angle (output)
cosine = 32-bit floating point cosine of angle (output)

CALLS TO: None

DESCRIPTION:

The SCOS algorithm is based on the property of sines and cosines that all possible absolute values for each are contained in any 45° sector, although they may be swapped. Adjacent sectors are, in a sense, mirror images of each other. For example:

	45°	90°		
Input	Sector	Quadrant	SINE	COSINE
10°	0	1	.1736	.9848
80°	1	1	.9848	.1736
100°	2	2	.9848	-.1736
170°	3	2	.1736	-.9848

The code first limits the input to the 0-360 range and normalizes it to a 0-1.0 number representing its fraction of 360°. This fraction is multiplied by 8.0. The integer part of the product (0-7) is the number of the 45° sector of the input. The fractional part of the product is the normalized fraction of the input within the 45° sector. Since the adjacent sectors are mirror images of each other, but for the swapping, the odd sectors (1,3,5,7) are folded back over the prior sector by subtracting the fraction from 1.0. The resulting fraction is converted to radians, the required input for the polynomial expansion (POLYF) which is used to compute the sine and cosine based on locally defined tables of coefficients (SIN_TBL & COS_TBL).

Lookup tables are used with the sector number as a pointer to determine whether the values must be swapped, and to determine the signs of the outputs. The sine and cosine are swapped if the sector of the input was 1, 2, 5, or 6. The sine becomes negative if the input was in quadrant 3 or 4, the cosine becomes negative if it was in quadrant 2 or 3.

MODULE NAME:	TIMVAL (Function)
FILE NAME:	TIMVAL.MAR
PURPOSE:	To convert an ASCII string representing time of day to a longword integer value representing seconds past midnight.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	itime = TIMVAL(string, length, error)

Where:

string	= byte array containg time
length	= 16-bit integer string length
error	= byte to contain error code,
0	= successful completion
FF_{16}	= error

CALLS TO: OTS$CVT_TU_L

DESCRIPTION:

This subroutine converts an ASCII string representing the time of day to a longword integer value representing seconds past midnight. Two input formats are acceptable as follows: HHMM or HHMM:SS

where: HH = hour of day
MM = minutes
SS = seconds

Format is determined by the length parameter with '4' denoting HHMM and '7' denoting HHMM:SS. Any other input length causes an error status to be returned. The input string is tested to ensure that hours are less then 24, minutes less than 60, and seconds less than 60. The VAX/VMS Runtime Library routine OTS$CVT_TU_L is called to convert each digit portion of the time to a longword integer. These values, in seconds, are accumulated and returned to the caller.

MODULE NAME:	UNPK
FILE NAME:	UNPK.MAR
PURPOSE:	To unpack a packed discrete and store the results.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL UNPK(input, count, b_array)

Where:

input =	16-bit packed discrete word
count =	16-bit integer number of bits to unpack
+ :	unpacking proceeds left to right (15-0)
– :	unpacking proceeds right to left (0-15)
b_array =	boolean array to contain discrete bytes. Note: address is incremented for each store if 'count' is positive, decremented if 'count' is negative.

CALLS TO: None

DESCRIPTION:

Procedure UNPK permits a specified number of bits (count) to be unpacked from the packed discrete word (input) and stored as discrete bytes at a starting address specified by b_array. If the COUNT is positive, unpacking proceeds from left to right and the storage address is incremented for subsequent moves. Otherwise, unpacking goes from right to left and the storage address is decremented. Thus, "UNPK(MCONF, 5, MLSC)" unpacks and stores 5 discrete words at MLSC, MLSC + 1, etc.

MODULE NAME:	UVC
FILE NAME:	UVC.MAR
PURPOSE:	To compute a unit vector.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL UVC (V, L, UV)

Where:	V	= 32-bit floating point vector
	L	= 16-bit integer vector length
	UV	= destination 32-bit floating point unit vector

CALLS TO: MTH$SQRT_R3 (See VAX/VMS Runtime Library Mathematic Manual)

DESCRIPTION:

Subroutine UVC computes the unit vector with direction V and length L, defined as: UV = V / ABS(V). No error checking is performed.

MODULE NAME:	VCP
FILE NAME:	VCP.MAR
PURPOSE:	To compute a vector cross product.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL VCP (U, V, W)

Where: U = 3 dimensional 32-bit floating point input vector
V = 3 dimensional 32-bit floating point input vector
W = 3 dimensional 32-bit floating point output vector
W = U X V

CALLS TO: None

DESCRIPTION:

Subroutine VCP computes the cross product of two 3 dimensional vectors, defined as: U X V = (u2 v3 - u3 v2), (u3 v1 - u1 v3), (u1 v2 - u2 v1). This computation is performed in double precision mode. The resulting vector (W) is converted to single-precision floating point (32 bit). No error checking is performed.

MODULE NAME: VDP (Function)

FILE NAME: VDP.MAR

PURPOSE: To perform a vector-dot-product on two 3-dimensional vectors.

CALLED BY: See subroutine/function cross reference (appendix A).

CALLING SEQUENCE: Dot_Product = VDP(U, V)

Where: U = 3 dimensional 32-bit floating point vector
V = 3 dimensional 32-bit floating point vector

CALLS TO: None

DESCRIPTION:
This routine performs a dot-product function on two 3-dimensional vectors. E.g, DP = (u1 v1) + (u2 v2) + (u3 v3). No error checking is performed.

MODULE NAME:	VMAX (Function)
FILE NAME:	VMAX.MAR
PURPOSE:	To find the largest element of a vector.
CALLED BY:	Not Used
CALLING SEQUENCE:	A = VMAX(vector, length)
Where:	vector = 32-bit floating point vector length = 16-bit integer vector length
CALLS TO:	None

DESCRIPTION:

VMAX finds and returns the largest element of a vector. No error checking is performed.

MODULE NAME:	VMG (Function)
FILE NAME:	VMG.MAR
PURPOSE:	To compute vector magnitude.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	MAG = VMG (vec, vec_lgth)

Where: vec = 32-bit floating point vector
vec_lgth = 16-bit integer vector length

CALLS TO: MTH$SQRT_R3 (See VAX/VMS Runtime Library mathematics manual)

DESCRIPTION:
Subroutine VMG computes the magnitude of a vector, defined as:

$$MAG = SQRT(v1\ v1 + v2\ v2 + v3\ v3 + \ldots v_n\ v_n) = SQRT(V \bullet V)\ .$$

No error checking is performed.

MODULE NAME: VMIN (Function)

FILE NAME: VMIN.MAR

PURPOSE: To find the smallest element of a vector.

CALLED BY: Not Used

CALLING SEQUENCE: A = VMIN(vector, length)

Where: vector = 32-bit floating point vector
length = 16-bit integer vector length

CALLS TO: None

DESCRIPTION:
VMIN finds and returns the smallest element of a vector. No error checking is performed.

MODULE NAME:	VSUM (Function)
FILE NAME:	VSUM.MAR
PURPOSE:	To compute the sum of the elements of a vector.
CALLED BY:	Not Used
CALLING SEQUENCE:	A = VSUM (vector, length)

Where: vector = 32-bit floating point vector
length = 16-bit integer vector length

CALLS TO:	None

DESCRIPTION:
VSUM produces the sum of the elements of a vector. No error checking is performed.

MODULE NAME: VXM

FILE NAME: VXM.MAR

PURPOSE: To perform vector times matrix multiply.

CALLED BY: CFILT

CALLING SEQUENCE: CALL VXM(V, M, VL, MW, U)

```
Where:  V = 32-bit floating point vector
        M = 32-bit floating point matrix
       VL = 16-bit integer vector length
       MW = 16-bit integer matrix width (# columns)
        U = 32-bit floating point destination vector
```

CALLS TO: None

DESCRIPTION:

Subroutine VXM multiplies a vector times a matrix, and stores the result in the destination vector. The input vector length must equal the number of rows in the matrix or the results are unpredictable. No error checking is performed.

MODULE NAME:	XYZ
FILE NAME:	XYZ.FOR
PURPOSE:	To create a unit vector pointing from the earth's center to a latitude/longitude on the surface.
CALLED BY:	See subroutine/function cross reference (appendix A).
CALLING SEQUENCE:	CALL XYZ(lat, lon, vector)

Where:	lat =	32-bit floating point latitude on the earth's surface (input)
	lon =	32-bit floating point longitude on the earth's surface (input)
	vector =	3 dimensional 32-bit floating point unit vector computed using lat/lon (output)

CALLS TO:	SCOSD

DESCRIPTION:

This module converts from polar coordinates in degrees to the corresponding unit vector in rectangular coordinates. The output unit vector has the following form:

```
vector(1) =  sin(lat)
vector(2) = -sin(lon) cos(lat)
vector(3) =  cos(lon) cos(lat)
```

No error checking is performed.

APPENDIX A

SUBROUTINE/FUNCTION CROSS REFERENCE

ANGL is called by routines :
DATSEL DMA ENGAGE_CAS ERAD FILL HNAVFS HNAVSL
HOLD_INIT HOLD_INPUT INBOUND INBRG INT_LEG LATCMD
LEGSW MLOG MSPLGC NAVEXC PFD_NASA POINTS PTHPOS
RADIAL RCOM REFRESH_HOLD RWYMGR SELTRK STAR
TOPEXC TRALCBA WINDOW

ASSIGN is called by routines :
DSPFST DSPSLW FCFAST FMFAST SLOW

BCDTIM is called by routines :
IDENT

CLIP is called by routines :
AREAS DMA LEG MAP_AIRWAY RUNWAY TURN

CLRBUF is called by routines :
CLEAN_CON INIT_PLAN XLAT_RTE

C_HDL is called by routines :
DSPFST DSPSLW FCFAST FMFAST SLOW

DEGVAL is called by routines :
INITUP WPT_ID

FMTDEG is called by routines :
ACTION AIR_PAGE FUNC_INP_FIX INITPOS INITUP PROCESS_ARP
PROCESS_GRP PROCESS_NAV PROGRESS

FMTTIM is called by routines :
DASDUMP DSP_TIME ECHO_TIME FLT_TYPE HOLD_INPUT IDENT
LEG_END PROGRESS REFRESH_HOLD RTA_LN10 RTA_LN9
SNAPDUMP TEXT

FRMFRQ is called by routines :
APPREF PROCESS_NAV PROGRESS

GET_CHAR is called by routines :
ACTION APPREF CLEAN_PPT DSPOT ECHO ORG_RWY PROGRESS
SET_SIDLINE STRIPS WAYPOINT WPT

GRID is called by routines :
AREAS ARPSMB COMP_ANG COMP_IP_DTG FIND_LEG_RAD

GET_XY
LEG_BRNG NAVAID NAVEXC NAVMLS OPTION PASSBY PLAN POS_INFO PTHPOS RADIAL RUNWAY STRIPS TURN XYPOS

LOCK is called by routines :
DSPFST DSPSLW FCFAST FMFAST SLOW

LUNAVA is called by routines :
ACTION DATA_INP_FIX PROCESS_NAV WPT_ID

LURTE is called by routines :
COMPANY RTE_ID

LURWY is called by routines :
DATA_IN MOD_ROUTE PROCESS_RWY WPT_ID

LUSID is called by routines :
FIND_RTE MOD_ROUTE

MAG_VAR is called by routines :
ERAD INTERCEPT MAGV RTE_INTC WPT WPT_ID

MAPCOM is called by routines :
DDSTAR DSPFST DSPHDL DSPSLW DSTAR FCFAST FMFAST HDL SLOW VIEW

MXV is called by routines :
ACCPRC CFILT CRBSC GPSPRC SCREEN XFORM

OTS$FLOAT is called by routines :
APPREF EPRLIM FIX_INFO FLT_TYPE PFINIT PROGRESS REFRESH_HOLD SHOW_GPS TKOFF

P_LIST is called by routines :
MAKE_WPT NEW_CON

POLAR is called by routines :
PROJECT

POSBTS is called by routines :
ARPSMB NAVAID NAVEXC OPTION PLAN RADIAL RUNWAY STRIPS

RET is called by routines :
FIND_EMPTY WPT_ADDR

TIMVAL is called by routines :
IDENT TIME_IN

UNPK is called by routines :
MLSEX MSPLGC

UVC is called by routines :
AAA HVGUID PATH PROJECT

VCP is called by routines :
AAA HVG2 PATH STAR TRALCBA

VDP is called by routines :
AAA AB_IP_LL FIND_LEG_AB HVG2 HVGUID PATH STAR
TRALCBA

VMG is called by routines :
CRBSC CTLBLK PATH STAR

VXM is called by routines :
CFILT

XYZ is called by routines :
PATH STAR

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response including the time for reviewing instructions searching existing data sources gathering and maintaining the data needed and completing and reviewing the collection of information Send comments regarding this burden estimate or any other aspect of this collection of information including suggestions for reducing this burden to Washington Headquarters Services Directorate for Information Operations and Reports 1215 Jefferson Davis Highway Suite 1204 Arlington VA 22202-4302 and to the Office of Management and Budget Paperwork Reduction Project (0704 0188) Washington DC 20503			
1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE April 1993	3. REPORT TYPE AND DATES COVERED Contractor Report (June 1988 - Nov. 1991)	
4. TITLE AND SUBTITLE Advanced Transport Operating System (ATOPS) Utility Library Software Description			5. FUNDING NUMBERS C NAS1-19038 WU 505-64-13
6. AUTHOR(S) Winston C. Clinedinst, Christopher J. Slominski, Richard W. Dickson, and David A. Wolverton			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Computer Sciences Corporation 3217 North Armistead Avenue Hampton, VA 23666			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, VA 23681-0001			10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA CR-191469
11. SUPPLEMENTARY NOTES Langley Technical Monitor: Robert A. Kudlinski Final Report			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified–Unlimited Subject Category 06			12b. DISTRIBUTION CODE
13. ABSTRACT *(Maximum 200 words)* The individual software processes used in the flight computers on-board the ATOPS aircraft have many common functional elements. A library of commonly used software modules was created for general uses among the processes. The library includes modules for mathematical computations, data formatting, system database interfacing, and condition handling. This document describes the modules available in the library and their associated calling requirements.			
14. SUBJECT TERMS ATOPS, Flight software, Utility software, VAX, VMS			15. NUMBER OF PAGES 54
			16. PRICE CODE A04
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298(Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102