VMS

digital

VMS RTL General Purpose (OTS$) Manual

Order Number AA–LA73A–TE

VMS RTL General Purpose (OTS$) Manual

Order Number: AA–LA73A–TE

April 1988

This manual documents the general purpose routines contained in the OTS$ facility of the VMS Run-Time Library.

Revision/Update Information:	This document supersedes the OTS$ section of the *VAX/VMS Run-Time Library Routines Reference Manual,* Version 4.4.
Software Version:	VMS Version 5.0

digital equipment corporation
maynard, massachusetts

April 1988

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

ZK4611

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire
03061

CANADA

Digital Equipment
of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

INTERNATIONAL

Digital Equipment Corporation
PSG Business Manager
c/o Digital§ local subsidiary
or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript™ printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

Contents

OTS$ REFERENCE SECTION

INDEX

TABLES

Preface

This manual provides users of the VMS operating system with detailed usage and reference information on general purpose routines supplied in the OTS$ facility of the Run-Time Library.

Run-Time Library routines can only be used in programs written in languages that produce native code for the VAX hardware. At present, these languages include VAX MACRO and the following compiled high-level languages:

VAX Ada
VAX BASIC
VAX BLISS-32
VAX C
VAX COBOL
VAX COBOL-74
VAX CORAL
VAX DIBOL
VAX FORTRAN
VAX Pascal
VAX PL/I
VAX RPG
VAX SCAN

Interpreted languages that can also access Run-Time Library routines include VAX DSM and DATATRIEVE.

Intended Audience

This manual is intended for system and application programmers who want to call Run-Time Library routines.

Document Structure

This manual is organized into two parts as follows:

- Part I provides a brief overview of the OTS$ routines.
- Part II provides detailed reference information on each routine contained in the OTS$ facility of the Run-Time Library. This information is presented using the documentation format described in the *Introduction to the VMS Run-Time Library*. Routine descriptions appear in alphabetical order by routine name.

Associated Documents

The Run-Time Library routines are documented in a series of reference manuals. A general overview of the Run-Time Library and a description of how the Run-Time Library routines are accessed are presented in the *Introduction to the VMS Run-Time Library*. Descriptions of the other RTL facilities and their corresponding routines and usages are discussed in the following books:

- The *VMS RTL DECtalk (DTK$) Manual*
- The *VMS RTL Library (LIB$) Manual*
- The *VMS RTL Mathematics (MTH$) Manual*
- The *VMS RTL Parallel Processing (PPL$) Manual*
- The *VMS RTL Screen Management (SMG$) Manual*
- The *VMS RTL String Manipulation (STR$) Manual*

The VAX Procedure Calling and Condition Handling Standard, which is documented in the *Introduction to System Routines*, contains useful information for anyone who wants to call Run-Time Library routines.

Application programmers of any language may refer to the *Guide to Creating VMS Modular Procedures* for the Modular Programming Standard and other guidelines.

High-level language programmers will find additional information on calling Run-Time Library routines in their language reference manual. Additional information may also be found in the language user's guide provided with your VAX language.

The *Guide to Using VMS Command Procedures* may also be useful.

For a complete list and description of the manuals in the VMS documentation set, see the *Overview of VMS Documentation*.

Conventions

Convention	Meaning
RET	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
$ SHOW TIME 05-JUN-1988 11:55:22	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

Other conventions used in the documentation of Run-Time Library routines are described in the *Introduction to the VMS Run-Time Library*.

1 Overview of the OTS$ Facility

This manual discusses the Run-Time Library OTS$ routines that perform general purpose functions. These functions include data type conversions as part of a compiler's generated code, and some mathematical functions.

Most of the OTS$ routines were originally designed to support language compilers. Because they provide useful functions, they were moved into the language-independent facility, OTS$.

The OTS$ facility provides you with routines that perform seven main tasks:

- Convert data types
- Divide complex and packed decimal values
- Move data to a specified destination address
- Multiply complex values
- Raise a base to an exponent
- Copy a source string to a destination string
- Return a string area to free storage

The following lists contain all of the OTS$ routines grouped according to their functions.

Table 1–1 OTS$ Routines

Conversion Routine	Function
OTS$CNVOUT	Convert a D-floating, G-floating, or H-floating value to a character string
OTS$CVT_L_TB	Convert an unsigned integer to binary text
OTS$CVT_L_TI	Convert a signed integer to signed integer text
OTS$CVT_L_TL	Convert an integer to logical text
OTS$CVT_L_TO	Convert an unsigned integer to octal text
OTS$CVT_L_TU	Convert an unsigned integer to decimal text
OTS$CVT_L_TZ	Convert an integer to hexadecimal text
OTS$CVT_TB_L	Convert binary text to an unsigned integer value
OTS$CVT_TI_L	Convert signed integer text to an integer value
OTS$CVT_TL_L	Convert logical text to an integer value
OTS$CVT_TO_L	Convert octal text to an integer value
OTS$CVT_TU_L	Convert unsigned decimal text to an integer value

Table 1–1 (Cont.) OTS$ Routines

Conversion Routine	Function
OTS$CVT_T_z	Convert numeric text to a D- or F-floating value
OTS$CVT_T_x	Convert numeric text to a G- or H-floating value
OTS$CVT_TZ_L	Convert hexadecimal text to an unsigned longword integer value

Division Routine	Function
OTS$DIVCx	Perform complex division
OTS$DIV_PK_LONG	Perform packed decimal division with a long divisor
OTS$DIV_PK_SHORT	Perform packed decimal division with a short divisor

Move Data Routine	Function
OTS$MOVE3	Move data without fill
OTS$MOVE5	Move data with fill

Multiplication Routine	Function
OTS$MULCx	Perform complex multiplication

Exponentiation Routine	Function
OTS$POWCxCx	Raise a complex base to a complex floating-point exponent
OTS$POWCxJ	Raise a complex base to a signed longword exponent
OTS$POWDD	Raise a D-floating base to a D-floating exponent
OTS$POWDR	Raise a D-floating base to an F-floating exponent
OTS$POWDJ	Raise a D-floating base to a longword integer exponent
OTS$POWGx	Raise a G-floating base to a G-floating or longword integer exponent
OTS$POWGJ	Raise a G-floating base to a longword integer exponent

Table 1–1 (Cont.) OTS$ Routines

Exponentiation Routine	Function
OTS$POWHx	Raise an H-floating base to a floating-point exponent
OTS$POWHJ	Raise an H-floating base to a longword integer exponent
OTS$POWII	Raise a word integer base to a word integer exponent
OTS$POWHJJ	Raise a longword integer base to a longword integer exponent
OTS$POWLULU	Raise an unsigned longword integer base to an unsigned longword integer exponent
OTS$POWxLU	Raise a floating-point base to an unsigned longword integer exponent
OTS$POWRD	Raise an F-floating base to a D-floating exponent
OTS$POWRJ	Raise an F-floating base to a longword integer exponent
OTS$POWRR	Raise an F-floating base to an F-floating exponent

Copy Source String Routine	Function
OTS$SCOPY_DXDX	Copy a source string passed by descriptor to a destination string
OTS$SCOPY_R_DX	Copy a source string passed by reference to a destination string

Return String Area Routine	Function
OTS$SFREE1_DD	Free one dynamic string
OTS$SFREEN_DD	Free *n* dynamic strings
OTS$SGET1_DD	Get one dynamic string

OTS$ Reference Section

This section provides detailed descriptions of the routines provided by the VMS RTL General Purpose (OTS$) Facility.

OTS$CNVOUT Convert D-floating, G-floating or H-floating Number to Character String

The Convert Floating to Character String routines convert a D-floating, G-floating or H-floating number to a character string in the FORTRAN E format.

FORMAT

OTS$CNVOUT *D-G-or-H-float-pt-input-val ,fixed-length-resultant-string ,digits-in-fraction*

OTS$CNVOUT_G *D-G-or-H-float-pt-input-val ,fixed-length-resultant-string ,digits-in-fraction*

OTS$CNVOUT_H *D-G-or-H-float-pt-input-val ,fixed-length-resultant-string ,digits-in-fraction*

RETURNS

VMS usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

D-G-or-H-float-pt-input-val

VMS usage:	**floating_point**
type:	**D_floating, G_floating, H_floating**
access:	**read only**
mechanism:	**by reference**

Value that OTS$CNVOUT converts to a character string. For OTS$CNVOUT, the **D-G-or-H-float-pt-input-val** argument is the address of a D-floating number containing the value. For OTS$CNVOUT_G, the **D-G-or-H-float-pt-input-val** argument is the address of a G-floating number containing the value. For OTS$CNVOUT_H, the **D-G-or-H-float-pt-input-val** argument is the address of an H-floating number containing the value.

fixed-length-resultant-string

VMS usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor, fixed length**

Output string into which OTS$CNVOUT writes the character string result of the conversion. The **fixed-length-resultant-string** argument is the address of a descriptor pointing to the output string.

digits-in-fraction

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of digits in the fractional portion of the result. The **digits-in-fraction** argument is an unsigned longword containing the number of digits to be written to the fractional portion of the result.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
SS$_ROPRAND	Floating reserved operand detected.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

OTS$CVT_L_TB Convert an Unsigned Integer to Binary Text

The Convert an Unsigned Integer to Binary Text routine converts an unsigned integer value of arbitrary length to binary representation in an ASCII text string. By default, a longword is converted.

FORMAT

OTS$CVT_L_TB *varying-input-value ,fixed-length-resultant-string [,number-of-digits] [,input-value-size]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

varying-input-value

VMS usage: **varying_arg**
type: **unspecified**
access: **read only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_L_TB converts to an unsigned decimal representation in an ASCII text string. (The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, or longword.) The **varying-input-value** argument is the address of the unsigned integer.

fixed-length-resultant-string

VMS usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

ASCII text string that OTS$CVT_L_TB creates when it converts the integer value. The **fixed-length-resultant-string** argument is the address of a descriptor pointing to this ASCII text string. The string is assumed to be of fixed length (DSC$K_CLASS_S).

number-of-digits

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Minimum number of digits in the binary representation to be generated. The **number-of-digits** argument is a signed longword containing this minimum number. This is an optional argument. If the minimum number of digits is omitted, the default is 1. If the actual number of significant digits is less than the minimum number of digits, leading zeros are produced. If the minimum number of digits is zero and the value of the integer to be converted is also zero, OTS$CVT_L_TB creates a blank string.

input-value-size

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Size of the integer to be converted, in bytes. The **input-value-size** argument is a signed longword containing the byte size. This is an optional argument. If the size is omitted, the default is 4 (longword).

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

EXAMPLE

```
     0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890123456789 0

     FTTY     D   F        4            TTY
     C* Initialize numeric value to be converted.
     C                     Z-ADD13       VALUE   90
     C          CVTLTB     EXTRN'OTS$CVT_L_TB'
     C* Convert the number to binary in a string.
     C                     CALL CVTLTB
     C                     PARM          VALUE      RL
     C                     PARMD         OUTSTR  4
     C* Display on the terminal the converted string.
     C          OUTSTR     DSPLYTTY
     C                     SETON                    LR
```

This RPG II program displays the string '1101' on the terminal.

OTS$CVT_L_TI Convert Signed Integer to Decimal Text

The Convert Signed Integer to Decimal Text routine converts a signed integer to a decimal ASCII text string. This routine supports FORTRAN Iw and Iw.m output and BASIC output conversion.

FORMAT

OTS$CVT_L_TI *varying-input-value ,fixed-length-resultant-string [,number-of-digits] [,input-value-size] [,flags-value]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

varying-input-value

VMS usage: **varying_arg**
type: **unspecified**
access: **read only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_L_TI converts to an unsigned decimal representation in an ASCII text string. (The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, or longword.) The **varying-input-value** argument is the address of the unsigned integer.

fixed-length-resultant-string

VMS usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed length**

Decimal ASCII text string that OTS$CVT_L_TI creates when it converts the signed integer. The **fixed-length-resultant-string** argument is the address of a descriptor pointing to this text string. The string is assumed to be of fixed length (DSC$K_CLASS_S).

number-of-digits

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Minimum number of digits to be generated when OTS$CVT_L_TI converts the signed integer to a decimal ASCII text string. The **number-of-digits** argument is a signed longword containing this number. This is an optional argument. If the minimum number of digits is omitted, the default value is 1. If the actual number of significant digits is smaller, OTS$CVT_L_TI inserts leading zeros into the output string. If **number-of-digits** is zero and **varying-input-value** is zero, OTS$CVT_L_TI writes a blank string to the output string.

input-value-size

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Size of the integer to be converted, in bytes. The **input-value-size** argument is a signed longword containing this value size. The value size must be either 1, 2, or 4. If value size is 1 or 2, the value is sign-extended to a longword before conversion. This is an optional argument. If the size is omitted, the default is 4 (longword).

flags-value

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Caller-supplied flags that you can use if you want OTS$CVT_L_TI to insert a plus sign before the converted number. The **flags-value** argument is an unsigned longword containing the flags.

The caller flags are defined as follows:

Bit 0	If set, a plus sign (+) is inserted before the first nonblank character in the output string; otherwise, the plus sign is omitted.

This is an optional argument. If **flags-value** is omitted, all bits are clear and the plus sign is not inserted.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

OTS$CVT_L_TL Convert Integer to Logical Text

The Convert Integer to Logical Text routine converts an integer to an ASCII text string representation using FORTRAN L (logical) format.

FORMAT

OTS$CVT_L_TL *longword-integer-value ,fixed-length-resultant-string*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

longword-integer-value

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by reference**

Value that OTS$CVT_L_TL converts to an ASCII text string. The **longword-integer-value** argument is the address of a signed longword containing this integer value.

fixed-length-resultant-string

VMS usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed length**

Output string that OTS$CVT_L_TL creates when it converts the integer value to an ASCII text string. The **fixed-length-resultant-string** argument is the address of a descriptor pointing to this ASCII text string.

The output string is assumed to be of fixed length (DSC$K_CLASS_S).

If *length* equals the fixed length of the output string, then the output string consists of (*length* – 1) blanks followed by the letter T if bit 0 is set, or the letter F if bit 0 is clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is of zero length (DSC$W_LENGTH= 0).

EXAMPLE

```
5 !+
  ! This is an example program
  ! showing the use of OTS$CVT_L_TL.
  !-

  VALUE% = 10
  OUTSTR$ = ' '
  CALL OTS$CVT_L_TL(VALUE%, OUTSTR$)
  PRINT OUTSTR$
9 END
```

This BASIC example illustrates the use of OTS$CVT_L_TL. The output generated by this program is "F".

OTS$CVT_L_TO Convert Unsigned Integer to Octal Text

The Convert Unsigned Integer to Octal Text routine converts an unsigned integer to an octal ASCII text string. OTS$CVT_L_TO supports FORTRAN Ow and Ow.m output conversion formats.

FORMAT

OTS$CVT_L_TO *varying-input-value ,fixed-length-resultant-string [,number-of-digits] [,input-value-size]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

varying-input-value

VMS usage: **varying_arg**
type: **unspecified**
access: **read only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_L_TO converts to an unsigned decimal representation in an ASCII text string. (The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, or longword.) The **varying-input-value** argument is the address of the unsigned integer.

fixed-length-resultant-string

VMS usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed length**

Output string that OTS$CVT_L_TO creates when it converts the integer value to an octal ASCII text string. The **fixed-length-resultant-string** argument is the address of a descriptor pointing to the octal ASCII text string. The string is assumed to be of fixed length (DSC$K_CLASS_S).

number-of-digits

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Minimum number of digits that OTS$CVT_L_TO generates when it converts the integer value to an octal ASCII text string. The **number-of-digits** argument is a signed longword containing the minimum number of digits. This is an optional argument. If it is omitted, the default is 1. If the actual number of significant digits in the octal ASCII text string is less than the minimum number of digits, OTS$CVT_L_TO inserts leading zeros into the output string. If **number-of-digits** is zero and **varying-input-value** is zero, OTS$CVT_L_TO writes a blank string to the output string.

input-value-size

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Size of the integer to be converted, in bytes. The **input-value-size** argument is a signed longword containing the number of bytes in the integer to be converted by OTS$CVT_L_TO. This is an optional argument. If it is omitted, the default is 4 (longword).

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

OTS$CVT_L_TU Convert Unsigned Integer to Decimal Text

The Convert Unsigned Integer to Decimal Text routine converts a byte, word, or longword value to unsigned decimal representation in an ASCII text string. By default, a longword is converted.

FORMAT

OTS$CVT_L_TU *varying-input-value ,fixed-length-resultant-string [,number-of-digits] [,input-value-size]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

varying-input-value

VMS usage: **varying_arg**
type: **unspecified**
access: **read only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_L_TU converts to an unsigned decimal representation in an ASCII text string. (The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, or longword.) The **varying-input-value** argument is the address of the unsigned integer.

fixed-length-resultant-string

VMS usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed-length**

Output string (fixed-length) that OTS$CVT_L_TU creates when it converts the integer value to unsigned decimal representation in an ASCII text string. The **fixed-length-resultant-string** argument is the address of a descriptor pointing to this ASCII text string.

number-of-digits

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Minimum number of digits in the ASCII text string that OTS$CVT_L_TU creates. The **number-of-digits** argument is a signed longword containing the minimum number. This is an optional argument. If the minimum number of digits is omitted, the default is 1.

If the actual number of significant digits in the output string created is less than the minimum number, OTS$CVT_L_TU inserts leading zeros into the output string. If the minimum number of digits is zero and the integer value to be converted is also zero, OTS$CVT_L_TU writes a blank string to the output string.

input-value-size

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Size of the integer value to be converted, in bytes. The **input-value-size** argument is a signed longword containing the size of the integer value. This is an optional argument. If the size is omitted, the default is 4. The only values that OTS$CVT_L_TU allows are 1, 2 and 4. If any other value is specified, OTS$CVT_L_TU uses the default value, 4 (longword).

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

EXAMPLE

```
    0    |    1    |    2    |    3    |    4    |    5    |    6    |    7    |
1234567890123456789012345678901234567890123456789012345678901234567890
     FTTY     D  F        7             TTY
     C* Initialize numeric value to be converted.
     C                    Z-ADD32857     VALUE  90
     C                    Z-ADD7         DIGITS 90
     C          CVTLTU    EXTRN'OTS$CVT_L_TU'
     C* Convert the number to decimal in a string with 7 decimal digits.
     C                    CALL CVTLTU
     C                    PARM           VALUE     RL
     C                    PARMD          OUTSTR  7
     C                    PARMV          DIGITS
     C* Display on the terminal the converted string.
     C          OUTSTR    DSPLYTTY
     C                    SETON                    LR
```

This RPG II program displays the string '0032857' on the terminal screen.

OTS$CVT_L_TZ Convert Integer to Hexadecimal Text

The Convert Integer to Hexadecimal Text routine converts an unsigned integer to a hexadecimal ASCII text string. OTS$CVT_L_TZ supports FORTRAN Zw and Zw.m output conversion formats.

FORMAT

OTS$CVT_L_TZ *varying-input-value ,fixed-length-resultant-string [,number-of-digits] [,input-value-size]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

varying-input-value

VMS usage: **varying_arg**
type: **unspecified**
access: **read only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_L_TZ converts to an unsigned decimal representation in an ASCII text string. (The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, or longword.) The **varying-input-value** argument is the address of the unsigned integer.

fixed-length-resultant-string

VMS usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor, fixed length**

Output string that OTS$CVT_L_TZ creates when it converts the integer value to a hexadecimal ASCII text string. The **fixed-length-resultant-string** argument is the address of a descriptor pointing to this ASCII text string. The string is assumed to be of fixed length (DSC$K_CLASS_S).

number-of-digits

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Minimum number of digits in the ASCII text string that OTS$CVT_L_TZ creates when it converts the integer. The **number-of-digits** argument is a signed longword containing this minimum number. This is an optional argument. If it is omitted, the default is 1. If the actual number of significant digits in the text string that OTS$CVT_L_TZ creates is less than this minimum number, OTS$CVT_L_TZ inserts leading zeros in the output string. If the minimum number of digits is zero and the integer value to be converted is also zero, OTS$CVT_L_TZ writes a blank string to the output string.

input-value-size

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Size of the integer that OTS$CVT_L_TZ converts, in bytes. The **input-value-size** argument is a signed longword containing the value size. This is an optional argument. If the size is omitted, the default is 4 (longword).

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks.

EXAMPLE

```
with TEXT_IO; use TEXT_IO;
procedure SHOW_CONVERT is

   type INPUT_INT is new INTEGER range 0..INTEGER'LAST;

   INTVALUE : INPUT_INT := 256;
   HEXSTRING : STRING(1..11);

   procedure CONVERT_TO_HEX (I : in INPUT_INT; HS : out STRING);
   pragma INTERFACE (RTL, CONVERT_TO_HEX);
   pragma IMPORT_routine (INTERNAL => CONVERT_TO_HEX,
                          EXTERNAL => "OTS$CVT_L_TZ",
                          MECHANISM =>(REFERENCE,
                                       DESCRIPTOR (CLASS => S)));

begin
   CONVERT_TO_HEX (INTVALUE, HEXSTRING);
   PUT_LINE("This is the value of HEXSTRING");
   PUT_LINE(HEXSTRING);
end;
```

This Ada example uses OTS$CVT_L_TZ to convert a longword integer to hexadecimal text.

OTS$CVT_TB_L Convert Binary Text to Unsigned Integer

The Convert Binary Text to Unsigned Integer routine converts an ASCII text string representation of an unsigned binary value to an unsigned integer value of arbitrary length. By default, the result is a longword. Valid input characters are the blank and the digits 0 and 1. No sign is permitted.

FORMAT

OTS$CVT_TB_L *input-string ,varying-output-value [,output-value-size] [,flags-value]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

input-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Input string containing the string representation of an unsigned binary value that OTS$CVT_TB_L converts to an integer value. The **input-string** argument is the address of a descriptor pointing to the string.

varying-output-value

VMS usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_TB_L creates when it converts the ASCII text string. (The value of the **output-value-size** argument determines whether **varying-output-value** is a byte, word, or longword.) The **varying-output-value** argument is the address of the unsigned integer.

output-value-size

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Number of bytes to be occupied by the value created when OTS$CVT_TB_L converts the ASCII text string to an integer value. The **output-value-size** argument contains the value size. If **output-value-size** contains a zero or a negative number, OTS$CVT_TB_L returns an error code as the condition value. This is an optional argument. Valid values for the **output-value-size** argument are 1, 2, and 4; the contents determine whether the integer value that OTS$CVT_TB_L creates is a byte, word, or longword. If the number of bytes is omitted, the default is 4 (longword).

flags-value

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TB_L uses to determine how to interpret blanks and tabs. The **flags-value** argument contains the value of the user-supplied flags.

The flags are defined as follows:

Bit	Description
0	If set, OTS$CVT_TB_L ignores blanks. If clear, OTS$CVT_TB_L interprets blanks as zeros.
4	If set, OTS$CVT_TB_L ignores tabs. If clear, OTS$CVT_TB_L interprets tabs as invalid characters.

This is an optional argument. The default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow, or invalid **input-value-size** occurred.

EXAMPLE

```
1

    OPTION                              &
        TYPE = EXPLICIT

    !+
    !   This program demonstrates the use of OTS$CVT_TB_L from BASIC.
    !   Several binary numbers are read and then converted to their
    !   integer equivalents.
    !-

    !+
    !   DECLARATIONS
    !-

    DECLARE STRING BIN_STR
    DECLARE LONG BIN_VAL, I, RET_STATUS
    DECLARE LONG CONSTANT FLAGS = 17        ! 2^0 + 2^4
    EXTERNAL LONG FUNCTION OTS$CVT_TB_L (STRING, LONG, &
        LONG BY VALUE, LONG BY VALUE)

    !+
    !   MAIN PROGRAM
    !-
```

```
!+
!   Read the data, convert it to binary, and print the result.
!-

FOR I = 1 TO 5
    READ BIN_STR
    RET_STATUS = OTS$CVT_TB_L( BIN_STR, BIN_VAL, '4'L, FLAGS)
    PRINT BIN_STR;" treated as a binary number equals";BIN_VAL
NEXT I

!+
!   Done, end the program.
!-

GOTO 32767

999 Data    "1111", "1 111", "1011011", "11111111", "00000000"

32767 END
```

This BASIC example program demonstrates how to call OTS$CVT_TB_L to convert binary text to a longword integer.

The output generated by this BASIC program is as follows:

```
1111 treated as a binary number equals 15
1 111 treated as a binary number equals 15
1011011 treated as a binary number equals 91
11111111 treated as a binary number equals 255
00000000 treated as a binary number equals 0
```

OTS$CVT_TI_L Convert Signed Integer Text to Integer

The Convert Signed Integer Text to Integer routine converts an ASCII text string representation of a decimal number to a signed byte, word, or longword integer value. The result is a longword by default, but the calling program can specify a byte or a word value instead.

FORMAT

OTS$CVT_TI_L *fixed-or-dynamic-input-string ,varying-output-value [,output-value-size] [,flags-value]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

fixed-or-dynamic-input-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input ASCII text string that OTS$CVT_TI_L converts to a signed byte, word, or longword. The **fixed-or-dynamic-input-string** argument is the address of a descriptor pointing to the input string.

The syntax of a valid ASCII text input string is as follows:

[+ or –] <integer-digits>

OTS$CVT_TI_L always ignores leading blanks. A decimal point is assumed at the right of the input string.

varying-output-value

VMS usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_TI_L creates when it converts the ASCII text string. (The value of the **output-value-size** argument determines whether **varying-output-value** is a byte, word, or longword.) The **varying-output-value** argument is the address of the unsigned integer.

output-value-size

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Number of bytes to be occupied by the value created when OTS$CVT_TI_L converts the ASCII text string to an integer value. The **output-value-size** argument contains the number of bytes. If **output-value-size** contains a zero or a negative number, OTS$CVT_TI_L returns an error code as the condition value. This is an optional argument. Valid values for the **output-value-size** argument are 1, 2, and 4; the contents determine whether the integer value that OTS$CVT_TI_L creates is a byte, word, or longword. If the number of bytes is omitted, the default is 4 (longword).

flags-value

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TI_L uses to determine how blanks and tabs are interpreted. The **flags-value** argument is an unsigned longword containing the value of the flags.

Bit	Description
0	If set, OTS$CVT_TI_L ignores all blanks. If clear, OTS$CVT_TI_L ignores leading blanks but interprets blanks after the first legal character as zeros.
4	If set, OTS$CVT_TI_L ignores tabs. If clear, OTS$CVT_TI_L interprets tabs as invalid characters.

This is an optional argument. If **flags-value** is omitted, the default is that all bits are cleared.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_INPCONERR	Input conversion error: an invalid character in the input string; or the value overflows byte, word, or longword; or **input-value-size** is invalid. **Varying-input-value** is set to zero.

OTS$CVT_TL_L Convert Logical Text to Integer

The Convert Logical Text to Integer routine converts an ASCII text string representation of a FORTRAN-77 L format to a byte, word, or longword integer value. The result is a longword by default, but the calling program can specify a byte or a word value instead.

FORMAT

OTS$CVT_TL_L *fixed-or-dynamic-input-string ,varying-output-value [,output-value-size]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

fixed-or-dynamic-input-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text representation of a FORTRAN-77 L format that OTS$CVT_TL_L converts to a byte, word, or longword integer value. The **fixed-or-dynamic-input-string** argument is the address of a descriptor pointing to the input string.

The syntax of a valid ASCII text input string is either of the following:

Blank (end of string)

Blank Period Letter Character (end of string)

The elements in the preceding input string are defined as follows:

Term	Description
Blank	Zero or more blanks
Period	. or nothing
Letter	T, t, F, or f
Character	Zero or more of any character

varying-output-value

VMS usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_TL_L creates when it converts the ASCII text string. (The value of the **output-value-size** argument determines whether **varying-output-value** is a byte, word, or longword.) The **varying-output-value** argument is the address of the unsigned integer.

OTS$CVT_TL_L returns −1 as the contents of the **varying-output-value** argument if the character denoted by "Letter" is "T" or "t". Otherwise, OTS$CVT_TL_L sets **varying-output-value** to zero.

output-value-size

VMS usage:	**longword_signed**
type:	**longword (signed)**
access:	**read only**
mechanism:	**by value**

Number of bytes to be occupied by the value created when OTS$CVT_TL_L converts the ASCII text string to an integer value. The **output-value-size** argument contains the number of bytes. If **output-value-size** contains a zero or a negative number, OTS$CVT_TL_L returns an error code as the condition value. This is an optional argument. Valid values for the **output-value-size** argument are 1, 2, and 4; the contents determine whether the integer value that OTS$CVT_TI_L creates is a byte, word, or longword. If it is omitted, the default is 4 (longword).

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_INPCONERR	Invalid character in the input string or invalid **input-value-size**; **varying-input-value** is set to zero.

OTS$CVT_TO_L Convert Octal Text to Signed Integer

The Convert Octal Text to Signed Integer routine converts an ASCII text string representation of an octal value to a signed integer of an arbitrary length. The result is a longword by default, but the calling program can specify a byte, word, or longword.

FORMAT

OTS$CVT_TO_L *fixed-or-dynamic-input-string ,varying-output-value [,output-value-size] [,flags-value]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

fixed-or-dynamic-input-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text string representation of an octal value that OTS$CVT_TO_L converts to a signed integer. The **fixed-or-dynamic-input-string** argument is the address of a descriptor pointing to the input string. The valid input characters are blanks and the digits 0 through 7. No sign is permitted.

varying-output-value

VMS usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Signed byte, word, or longword that OTS$CVT_TO_L creates when it converts the ASCII text string. (The value of the **output-value-size** argument determines whether **varying-output-value** is a byte, word, or longword.) The **varying-output-value** argument is the address of the signed integer.

output-value-size

VMS usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the signed integer value. The **output-value-size** argument contains the number of bytes. If the content of the **output-value-size** argument is zero or a negative number, OTS$CVT_TO_L returns an error. This is an optional argument. If the number of bytes is omitted, the default is 4 (longword).

flags-value

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TO_L uses to determine how blanks within the input string are interpreted. The **flags-value** argument contains the user-supplied flags.

Bit 0	If set, OTS$CVT_TO_L ignores all blanks. If clear, OTS$CVT_TO_L interprets blanks as zeros.

This is an optional argument. If **flags-value** is omitted, the default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow, or invalid **input-value-size** occurred.

EXAMPLE

```
OCTAL_CONV: PROCEDURE OPTIONS (MAIN) RETURNS (FIXED BINARY (31));

%INCLUDE $STSDEF;       /* Include definition of return status values      */
DECLARE OTS$CVT_TO_L ENTRY
        (CHARACTER (*),         /* Input string passed by descriptor       */
        FIXED BINARY (31),      /* Returned value passed by reference      */
        FIXED BINARY VALUE,     /* Size for returned value passed by value */
        FIXED BINARY VALUE)     /* Flags passed by value                   */
        RETURNS (FIXED BINARY (31))     /* Return status                   */
        OPTIONS (VARIABLE);     /* Arguments may be omitted                */

DECLARE INPUT CHARACTER (10);
DECLARE VALUE FIXED BINARY (31);
DECLARE SIZE  FIXED BINARY(31) INITIAL(4) READONLY STATIC; /* Longword     */
DECLARE FLAGS FIXED BINARY(31) INITIAL(1) READONLY STATIC; /* Ignore blanks */

ON ENDFILE (SYSIN) STOP;

DO WHILE ('1'B);                /* Loop continuously, until end of file    */
        PUT SKIP (2);
        GET LIST (INPUT) OPTIONS (PROMPT ('Octal value: '));
        STS$VALUE = OTS$CVT_TO_L (INPUT, VALUE, SIZE, FLAGS);
        IF ^STS$SUCCESS THEN RETURN (STS$VALUE);
        PUT SKIP EDIT (INPUT, 'Octal equals', VALUE, 'Decimal')
                (A,X,A,X,F(10),X,A);
        END;

END OCTAL_CONV;
```

This PL/I program translates an octal value in ASCII into a fixed binary value. The program is run interactively; simply press CTRL/Z to quit.

```
$ RUN OCTOL
Octal value: 1
1  Octal equals 1 Decimal
Octal value: 11
11   Octal equals   9 Decimal
Octal value: 1017346
1017346 Octal equals 274150 Decimal
Octal value: CTRL/Z
```

OTS$CVT_TU_L Convert Unsigned Decimal Text to Integer

The Convert Unsigned Decimal Text to Integer routine converts an ASCII text string representation of an unsigned decimal value to an unsigned byte, word, or longword value. By default, the result is a longword. Valid input characters are the space and the digits 0 through 9. No sign is permitted.

FORMAT

OTS$CVT_TU_L *fixed-length-input-string ,varying-output-value [,output-value-size] [,flags-value]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

fixed-length-input-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length**

Input string (fixed-length) containing an ASCII text string representation of an unsigned decimal value that OTS$CVT_TU_L converts to a byte, word, or longword value. The **fixed-length-input-string** argument is the address of a descriptor pointing to the input string.

varying-output-value

VMS usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_TU_L creates when it converts the ASCII text string. (The value of the **output-value-size** argument determines whether **varying-output-value** is a byte, word, or longword.) The **varying-output-value** argument is the address of the unsigned integer.

output-value-size

VMS usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the value created when OTS$CVT_TU_L converts the input string. The **output-value-size** argument contains the number of bytes. OTS$CVT_TU_L allows value sizes of 1, 2 and 4. If any other value is specified, or if **output-value-size** is omitted, OTS$CVT_TU_L uses the default, 4.

flags-value

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TU_L uses to determine how blanks and tabs are interpreted. The **flags-value** argument contains the user-supplied flags.

Bit	Description
0	If set, OTS$CVT_TU_L ignores blanks. If clear, OTS$CVT_TU_L interprets blanks as zeros.
4	If set, OTS$CVT_TU_L ignores tabs. If clear, OTS$CVT_TU_L interprets tabs as invalid characters.

Flags-value is an optional argument. If it is omitted, the default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow or invalid **input-value-size** occurred.

OTS$CVT_T_z Convert Numeric Text to D- or F-Floating Value

The Convert Numeric Text to D- or F-Floating routines convert an ASCII text string representation of a numeric value to a D-floating or F-floating value.

FORMAT

OTS$CVT_T_D *fixed-or-dynamic-input-string ,floating-point-value [,digits-in-fraction] [,scale-factor] [,flags-value] [,extension-bits]*

OTS$CVT_T_F *fixed-or-dynamic-input-string ,floating-point-value [,digits-in-fraction] [,scale-factor] [,flags-value] [,extension-bits]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

fixed-or-dynamic-input-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text string representation of a numeric value that OTS$CVT_T_z converts to a D-floating or F-floating value. The **fixed-or-dynamic-input-string** argument is the address of a descriptor pointing to the input string.

The syntax of a valid input string is as follows:

Blank Sign Digit Period Digit$^{\textit{letter blank sign OR sign digit}}$

The elements in the preceding input string are defined as follows:

Term	Description
Blank	Zero or more blanks
Sign	+, −, or nothing
Digit	Zero or more decimal digits
Period	. or nothing
Letter	E, e, D, d, Q, or q

There is no difference in semantics among any of the six valid exponent letters (E, e, D, d, Q, q).

floating-point-value

VMS usage: **floating_point**
type: **D_floating, F_floating**
access: **write only**
mechanism: **by reference**

Floating-point value that OTS$CVT_T_z creates when it converts the input string. The **floating-point-value** argument is the address of the floating-point value. For OTS$CVT_T_D, **floating-point-value** is a D-floating number. For OTS$CVT_T_F, **floating-point-value** is an F-floating number.

digits-in-fraction

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of digits in the fraction if no decimal point is included in the input string. The **digits-in-fraction** argument contains the number of digits. This is an optional argument. If the number of digits is omitted, the default is zero.

scale-factor

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Scale factor. The **scale-factor** argument contains the value of the scale factor. If bit 6 of the **flags-value** argument is clear, the resultant value is divided by $10^{\text{scale-factor}}$ unless the exponent is present. If bit 6 of **flags-value** is set, the scale factor is always applied. This is an optional argument. If the scale factor is omitted, the default is zero.

flags-value

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags. The **flags-value** argument contains the user-supplied flags.

Bit 0	If set, OTS$CVT_T_z ignores blanks. If clear, OTS$CVT_T_z interprets blanks as zeros.
Bit 1	If set, OTS$CVT_T_z allows only E or e exponents. If clear, OTS$CVT_T_z allows E, e, D, d, Q and q exponents. (Bit 1 is clear for BASIC and set for FORTRAN.)
Bit 2	If set, OTS$CVT_T_z interprets an underflow as an error. If clear, OTS$CVT_T_z does not interpret an underflow as an error.
Bit 3	If set, OTS$CVT_T_z truncates the value. If clear, OTS$CVT_T-z rounds the value.

Bit 4	If set, OTS$CVT_T_z ignores tabs. If clear, OTS$CVT_T_z interprets tabs as invalid characters.
Bit 5	If set, an exponent must begin with a valid exponent letter. If clear, the exponent letter can be omitted.
Bit 6	If set, OTS$CVT_T_z always applies the scale factor. If clear, OTS$CVT_T_z applies the scale factor only if there is no exponent present in the string.

If **flags-value** is omitted, all bits are clear.

extension-bits

VMS usage: **byte_signed**
type: **byte (signed)**
access: **write only**
mechanism: **by reference**

Extra precision bits. The **extension-bits** argument is the address of a byte containing the extra precision bits. If **extension-bits** is present, **floating-point-value** is not rounded, and the first *n* bits after truncation are returned in this argument. For D-floating and F-floating values, *n* equals 8 and the bits are returned as a byte.

These values are suitable for use as the extension operand in an EMOD instruction.

DESCRIPTION

These routines support FORTRAN D, E, F, and G input type conversion as well as similar types for other languages.

OTS$CVT_T_D and OTS$CVT_T_F provide run-time support for BASIC and FORTRAN input statements.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_INPCONERR	Input conversion error; an invalid character in the input string, or the value is outside the range that can be represented. **Floating-point-value** is set to +0.0 (not reserved operand –0.0).

EXAMPLE

```
C+
C This is a FORTRAN program demonstrating the use of
C OTS$CVT_T_F.
C-

       REAL*4 A
       CHARACTER*10 T(5)
       DATA T/'1234567+23','8.786534+3','-983476E-3','-23.734532','45'/
       DO 2 I = 1, 5
       TYPE 1,I,T(I)
 1     FORMAT(' Input string ',I1,' is ',A10)
```

```
C+
C B is the return status.
C T(I) is the string to be converted to an
C F-floating point value.  A is the F-floating
C point conversion of T(I).  %VAL(5) means 5 digits
C are in the fraction if no decimal point is in
C the input string T(I).
C-
        B = OTS$CVT_T_F(T(I),A,%VAL(5),,)
        TYPE *,' Output of OTSCVT_T_F is     ',A
        TYPE *,'  '
 2      CONTINUE
        END
```

This FORTRAN example demonstrates the use of OTS$CVT_T_F. The output generated by this program is as follows:

```
Input string 1 is 1234567+23
 Output of OTSCVT_T_F is      1.2345669E+24
Input string 2 is 8.786534+3
 Output of OTSCVT_T_F is       8786.534
Input string 3 is -983476E-3
 Output of OTSCVT_T_F is     -9.8347599E-03
Input string 4 is -23.734532
 Output of OTSCVT_T_F is      -23.73453
Input string 5 is 45
 Output of OTSCVT_T_F is       45000.00
```

OTS$CVT_T_z Convert Numeric Text to G- or H-Floating Value

The Convert Numeric Text to G- or H-Floating routines convert an ASCII text string representation of a numeric value to a G-floating or H-floating value.

FORMAT

OTS$CVT_T_G *fixed-or-dynamic-input-string ,floating-point-value [,digits-in-fraction] [,scale-factor] [,flags-value] [,extension-bits]*

OTS$CVT_T_H *fixed-or-dynamic-input-string ,floating-point-value [,digits-in-fraction] [,scale-factor] [,flags-value] [,extension-bits]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

fixed-or-dynamic-input-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text string representation of a numeric value that OTS$CVT_T_z converts to a G-floating or H-floating value. The **fixed-or-dynamic-input-string** argument is the address of a descriptor pointing to the input string.

The syntax of a valid input string is as follows:

Blank Sign Digit Period Digit$^{letter\ blank\ sign\ OR\ sign\ digit}$

The elements in the preceding input string are defined as follows:

Term	Description
Blank	Zero or more blanks
Sign	+, −, or nothing
Digit	Zero or more decimal digits
Period	. or nothing
Letter	E, e, D, d, Q, or q

There is no difference in semantics among any of the six valid exponent letters (E, e, D, d, Q, q).

floating-point-value

VMS usage: **floating_point**
type: **G_floating, H_floating**
access: **write only**
mechanism: **by reference**

Floating-point value that OTS$CVT_T_z creates when it converts the input string. The **floating-point-value** argument is the address of the floating-point value. For OTS$CVT_T_G, **floating-point-value** is a G-floating number. For OTS$CVT_T_H, **floating-point-value** is an H-floating number.

digits-in-fraction

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of digits in the fraction if no decimal point is included in the input string. The **digits-in-fraction** argument contains the number of digits. This is an optional argument. If the number of digits is omitted, the default is zero.

scale-factor

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Scale factor. The **scale-factor** argument contains the value of the scale factor. If bit 6 of the **flags-value** argument is clear, the resultant value is divided by $10^{\text{scale-factor}}$ unless the exponent is present. If bit 6 of **flags-value** is set, the scale factor is always applied. This is an optional argument. If the scale factor is omitted, the default is zero.

flags-value

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags. The **flags-value** argument contains the user-supplied flags.

Bit 0 — If set, OTS$CVT_T_z ignores blanks. If clear, OTS$CVT_T_z interprets blanks as zeros.

Bit 1 — If set, OTS$CVT_T_z allows only E or e exponents. If clear, OTS$CVT_T_z allows E, e, D, d, Q and q exponents. (Bit 1 is clear for BASIC and set for FORTRAN.)

Bit 2 — If set, OTS$CVT_T_z interprets an underflow as an error. If clear, OTS$CVT_T_z does not interpret an underflow as an error.

Bit 3 — If set, OTS$CVT_T_z truncates the value. If clear, OTS$CVT_T-z rounds the value.

Bit 4	If set, OTS$CVT_T_z ignores tabs. If clear, OTS$CVT_T_z interprets tabs as invalid characters.
Bit 5	If set, an exponent must begin with a valid exponent letter. If clear, the exponent letter may be omitted.
Bit 6	If set, OTS$CVT_T_z always applies the scale factor. If clear, OTS$CVT_T_z applies the scale factor only if there is no exponent present in the string.

If **flags-value** is omitted, all bits are clear.

extension-bits

VMS usage: **word_signed**
type: **word (signed)**
access: **write only**
mechanism: **by reference**

Extra precision bits. The **extension-bits** argument is the address of a signed word integer containing the extra precision bits. If present, **floating-point-value** is not rounded, and the first *n* bits after truncation are returned in this argument. For G-floating and H-floating, *n* equals 11 and 15, respectively, and the bits are returned as a word, left-justified.

These values are suitable for use as the extension operand in an EMOD instruction.

The extra precision bits returned for H-floating may not be precise because calculations are only carried to 128 bits. However, the error should be small.

DESCRIPTION

These routines support FORTRAN D, E, F, and G input type conversion as well as similar types for other languages.

OTS$CVT_T_G and OTS$CVT_T_H provide run-time support for BASIC and FORTRAN input statements.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_INPCONERR	Input conversion error; an invalid character in the input string, or the value is outside the range that can be represented. **Floating-point-value** is set to +0.0 (not reserved operand –0.0).

OTS$CVT_TZ_L Convert Hexadecimal Text to Unsigned Integer

The Convert Hexadecimal Text to Unsigned Integer routine converts an ASCII text string representation of an unsigned hexadecimal value to an unsigned integer of an arbitrary length. The result is a longword by default, but the calling program can specify a byte, word, or longword value.

FORMAT

OTS$CVT_TZ_L *fixed-or-dynamic-input-string ,varying-output-value [,output-value-size] [,flags-value]*

RETURNS

VMS usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

fixed-or-dynamic-input-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor, fixed-length or dynamic string**

Input string containing an ASCII text string representation of an unsigned hexadecimal value that OTS$CVT_TZ_L converts to an unsigned integer. The **fixed-or-dynamic-input-string** argument is the address of a descriptor pointing to the input string. Valid input characters are the space, the digits 0 through 9, and the letters A through F (lowercase letters a through f are acceptable). No sign is permitted.

varying-output-value

VMS usage: **varying_arg**
type: **unspecified**
access: **write only**
mechanism: **by reference**

Unsigned byte, word, or longword that OTS$CVT_TZ_L creates when it converts the ASCII text string. (The value of the **output-value-size** argument determines whether **varying-output-value** is a byte, word, or longword.) The **varying-output-value** argument is the address of the unsigned integer.

output-value-size

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Number of bytes occupied by the integer value. The **output-value-size** argument contains the number of bytes. If the value size is zero or a negative number, OTS$CVT_TZ_L returns an input conversion error. This is an optional argument. If the number of bytes is omitted, the default is 4 (longword).

flags-value

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

User-supplied flags that OTS$CVT_TZ_L uses to determine how blanks are interpreted. The **flags-value** argument is an unsigned longword containing these user-supplied flags.

Bit 0 — If set, OTS$CVT_TZ_L ignores blanks. If set, OTS$CVT_TZ_L interprets blanks as zeros.

This is an optional argument. If **flags-value** is omitted, the default is that all bits are clear.

CONDITION VALUES RETURNED

SS$_NORMAL	Normal successful completion.
OTS$_INPCONERR	Input conversion error. An invalid character, overflow, or invalid **output-value-size** occurred.

EXAMPLES

1

```
10      !+
        ! This BASIC program converts a character string representing
        ! a hexadecimal value to a longword.
        !-

100     !+
        ! Illustrate (and test) OTS convert hex-string to longword
        !-

        EXTERNAL LONG FUNCTION OTS$CVT_TZ_L
        EXTERNAL LONG CONSTANT OTS$_INPCONERR
        INPUT "Enter hex numeric";HEXVAL$
        RET_STAT% = OTS$CVT_TZ_L(HEXVAL$, HEX% )
        PRINT "Conversion error " IF RET_STAT% = OTS$_INPCONERR
        PRINT "Decimal value of ";HEXVAL$;" is";HEX%              &
             IF RET_STAT% <> OTS$_INPCONERR
```

This BASIC example accepts a hexadecimal numeric string, converts it to a decimal integer, and prints the result. One sample of the output generated by this program is as follows:

```
$ RUN HEX
Enter hex numeric?  A
Decimal value of A is 10
```

2

```
HEX_CONV: PROCEDURE OPTIONS (MAIN) RETURNS (FIXED BINARY (31));

%INCLUDE $STSDEF;       /* Include definition of return status values       */
DECLARE OTS$CVT_TZ_L ENTRY
        (CHARACTER (*),         /* Input string passed by descriptor        */
        FIXED BINARY (31),      /* Returned value passed by reference       */
        FIXED BINARY VALUE,     /* Size for returned value passed by value  */
        FIXED BINARY VALUE)     /* Flags passed by value                    */
        RETURNS (FIXED BINARY (31))     /* Return status                    */
        OPTIONS (VARIABLE);     /* Arguments may be omitted                 */

DECLARE INPUT CHARACTER (10);
DECLARE VALUE FIXED BINARY (31);
DECLARE FLAGS FIXED BINARY(31) INITIAL(1) READONLY STATIC; /* Ignore blanks */

ON ENDFILE (SYSIN) STOP;

DO WHILE ('1'B);                /* Loop continuously, until end of file     */
        PUT SKIP (2);
        GET LIST (INPUT) OPTIONS (PROMPT ('Hex value: '));
        STS$VALUE = OTS$CVT_TZ_L (INPUT, VALUE, , FLAGS);
        IF ^STS$SUCCESS THEN RETURN (STS$VALUE);
        PUT SKIP EDIT (INPUT, 'Hex equals', VALUE, 'Decimal')
                        (A,X,A,X,F(10),X,A);
        END;

END HEX_CONV;
```

This PL/I example translates a hexadecimal value in ASCII into a fixed binary value. This program continues to prompt for input values until the user types CTRL/Z.

One sample of the output generated by this program is as follows:

```
$ RUN HEX
Hex value:  1A
1A       Hex equals       26 Decimal

Hex value:  C
C        Hex equals       12 Decimal

Hex value:  CTRL/Z
```

OTS$DIVCx Complex Division

The Complex Division routines return a complex result of a division on complex numbers.

FORMAT

OTS$DIVC *complex-dividend ,complex-divisor*
OTS$DIVCD_R3 *complex-dividend ,complex-divisor*
OTS$DIVCG_R3 *complex-dividend ,complex-divisor*

Each of these three formats corresponds to one of the three floating-point complex types.

RETURNS

VMS usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**write only**
mechanism:	**by value**

Complex result of complex division. OTS$DIVC returns an F-floating complex number. OTS$DIVCD_R3 returns a D-floating complex number. OTS$DIVCG_R3 returns a G-floating complex number.

ARGUMENTS

complex-dividend

VMS usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**read only**
mechanism:	**by value**

Complex dividend. The **complex-dividend** argument contains a floating-point complex value. For OTS$DIVC, **complex-dividend** is an F-floating complex number. For OTS$DIVCD_R3, **complex-dividend** is a D-floating complex number. For OTS$DIVCG_R3, **complex-dividend** is a G-floating complex number.

complex-divisor

VMS usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**read only**
mechanism:	**by value**

Complex divisor. The **complex-divisor** argument contains the value of the divisor. For OTS$DIVC, **complex-divisor** is an F-floating complex number. For OTS$DIVCD_R3, **complex-divisor** is a D-floating complex number. For OTS$DIVCG_R3, **complex-divisor** is a G-floating complex number.

DESCRIPTION

These routines return a complex result of a division on complex numbers.

The complex result is computed as follows:

1. Let (a,b) represent the complex dividend.
2. Let (c,d) represent the complex divisor.
3. Let (r,i) represent the complex quotient.

The results of this computation are as follows:

$$r = (ac + bd)/(c^2 + d^2)$$

$$i = (bc - ad)/(c^2 + d^2)$$

CONDITION VALUES SIGNALED

SS$_FLTDIV_F	Arithmetic fault. Floating-point division by zero.
SS$_FLTOV_F	Arithmetic fault. Floating-point overflow.

EXAMPLES

1

```
C+
C    This FORTRAN example forms the complex
C    quotient of two complex numbers using
C    OTS$DIVC and the FORTRAN random number
C    generator RAN.
C
C    Declare Z1, Z2, Z_Q, and OTS$DIVC as complex values.
C    OTS$DIVC will return the complex quotient of Z1 divided
C    by Z2:  Z_Q = OTS$DIVC( %VAL(REAL(Z1)), %VAL(AIMAG(Z1),
C    %VAL(REAL(Z2)), %VAL(AIMAG(Z2))
C-

       COMPLEX Z1,Z2,Z_Q,OTS$DIVC
C+
C    Generate a complex number.
C-
       Z1 = (8.0,4.0)
C+
C    Generate another complex number.
C-
       Z2 = (1.0,1.0)
C+
C    Compute the complex quotient of Z1/Z2.
C-
       Z_Q = OTS$DIVC( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)), %VAL(REAL(Z2)),
    +                  %VAL(AIMAG(Z2)))
       TYPE *, ' The complex quotient of',Z1,' divided by ',Z2,' is'
       TYPE *, '     ',Z_Q
       END
```

This FORTRAN program demonstrates how to call OTS$DIVC. The output generated by this program is as follows:

```
The complex quotient of (8.000000,4.000000) divided by  (1.000000,1.000000) is
     (6.000000,-2.000000)
```

2

```
C+
C    This FORTRAN example forms the complex
C    quotient of two complex numbers by using
C    OTS$DIVCG_R3 and the FORTRAN random number
C    generator RAN.
C
C     Declare Z1, Z2, and Z_Q as complex values. OTS$DIVCG_R3
C     will return the complex quotient of Z1 divided by Z2:
C     Z_Q = Z1/Z2
C-

        COMPLEX*16 Z1,Z2,Z_Q
C+
C    Generate a complex number.
C-
        Z1 = (8.0,4.0)
C+
C    Generate another complex number.
C-
        Z2 = (1.0,1.0)
C+
C    Compute the complex quotient of Z1/Z2.
C-
        Z_Q = Z1/Z2
        TYPE *, ' The complex quotient of',Z1,' divided by ',Z2,' is'
        TYPE *, '     ',Z_Q
        END
```

This FORTRAN example uses the OTS$DIVCG_R3 entry point instead. Notice the difference in the precision of the output generated:

```
 The complex quotient of (8.000000000000000,4.000000000000000) divided by
(1.000000000000000,1.000000000000000) is
     (6.000000000000000,-2.000000000000000)
```

OTS$DIV_PK_LONG Packed Decimal Division with Long Divisor

The Packed Decimal Division with Long Divisor routine divides fixed-point decimal data, which is stored in packed decimal form, when precision and scale requirements for the quotient call for multiple precision division. The divisor must have a precision of thirty or thirty-one digits.

FORMAT

OTS$DIV_PK_LONG *packed-decimal-dividend ,packed-decimal-divisor ,divisor-precision ,packed-decimal-quotient ,quotient-precision ,precision-data ,scale-data*

RETURNS

None.

ARGUMENTS

packed-decimal-dividend

VMS usage: **varying_arg**
type: **packed decimal string**
access: **read only**
mechanism: **by reference**

Dividend. The **packed-decimal-dividend** argument is the address of a packed decimal string that contains the shifted dividend.

Before being passed as input, the **packed-decimal-dividend** argument is always multiplied by 10^c where *c* is defined as follows:

```
c = 31 - prec(packed-decimal-dividend)
```

Mutiplying **packed-decimal-dividend** by 10^c makes **packed-decimal-dividend** a 31-digit number.

packed-decimal-divisor

VMS usage: **varying_arg**
type: **packed decimal string**
access: **read only**
mechanism: **by reference**

Divisor. The **packed-decimal-divisor** argument is the address of a packed decimal string that contains the divisor.

divisor-precision

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Precision of the divisor. The **divisor-precision** argument is a signed word that contains the precision of the divisor. The high-order bits are filled with zeros.

packed-decimal-quotient

VMS usage: **varying_arg**
type: **packed decimal string**
access: **write only**
mechanism: **by reference**

Quotient. The **packed-decimal-quotient** argument is the address of the packed decimal string into which OTS$DIV_PK_LONG writes the quotient.

quotient-precision

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Precision of the quotient. The **quotient-precision** argument is a signed word that contains the precision of the quotient. The high-order bits are filled with zeros.

precision-data

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Additional digits of precision required. The **precision-data** argument is a signed word that contains the value of the additional digits of precision required.

OTS$DIV_PK_LONG computes the **precision-data** argument as follows:

```
precision-data = scale(packed-decimal-quotient)
+ scale(packed-decimal-divisor)
- scale(packed-decimal-dividend)
- 31 + prec(packed-decimal-dividend)
```

scale-data

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Scale factor of the decimal point. The **scale-data** argument is a signed word that contains the scale data.

OTS$DIV_PK_LONG defines the **scale-data** argument as follows:

```
scale-data = 31 - prec(packed-decimal-divisor)
```

DESCRIPTION

Before using this routine, you should determine whether it is best to use OTSDIV_PK_LONG, OTSDIV_PK_SHORT, or the VAX instruction DIVP. To determine this, you must first calculate *b*, where *b* is defined as follows:

```
b = scale(packed-decimal-quotient)
+ scale(packed-decimal-divisor)
- scale(packed-decimal-dividend)
+ prec(packed-decimal-dividend)
```

If *b* is greater than 31, then OTS$DIV_PK_LONG can be used to perform the division. If *b* is less than 31, you could use the instruction DIVP instead.

Once you have determined that you cannot use DIVP, you need to determine whether you should use OTS$DIV_PK_LONG or OTS$DIV_PK_SHORT. To determine this, you must examine the value of **scale-data**. If **scale-data** is less than or equal to 1, then you should use OTS$DIV_PK_LONG. If **scale-data** is greater than 1, you should use OTS$DIV_PK_SHORT instead.

CONDITION VALUE SIGNALED

SS$_FLTDIV	Fatal error. Division by zero.

EXAMPLE

1

```
OPTION                                  &
    TYPE = EXPLICIT

!+
!   This program uses OTS$DIV_PK_LONG to perform packed decimal
!   division.
!-

!+
!   DECLARATIONS
!-

DECLARE DECIMAL (31, 2)     NATIONAL_DEBT
DECLARE DECIMAL (30, 3)     POPULATION
DECLARE DECIMAL (10, 5)     PER_CAPITA_DEBT

EXTERNAL SUB OTS$DIV_PK_LONG (DECIMAL(31,2), DECIMAL (30, 3), &
    WORD BY VALUE, DECIMAL(10, 5), WORD BY VALUE, WORD BY VALUE, &
    WORD BY VALUE)

!+
!   Prompt the user for the required input.
!-

INPUT   "Enter national debt: ";NATIONAL_DEBT
INPUT   "Enter current population: ";POPULATION
```

```
!+
!   Perform the division and print the result.
!
!   scale(divd) = 2
!   scale(divr) = 3
!   scale(quot) = 5
!
!   prec(divd) = 31
!   prec(divr) = 30
!   prec(quot) = 10
!
!   prec-data  = scale(quot) + scale(divr) - scale(divd) - 31 +
!                prec(divd)
!   prec-data  =   5       +   3       -     2    - 31 +   31
!   prec-data  = 6
!
!   b = scale(quot) + scale(divr) - scale(divd) + prec(divd)
!   b =   5      +   3      -     2    +    31
!   b = 37
!
!   c = 31 - prec(divd)
!   c = 31 -   31
!   c = 0
!
!   scale-data = 31 - prec(divr)
!   scale-data = 31 -   30
!   scale-data = 1
!
!   b is greater than 31, so either OTS$DIV_PK_LONG or
!      OTS$DIV_PK_SHORT may be used to perform the division.
!      If b is less than or equal to 31, then the DIVP
!      instruction may be used.
!
!   scale-data is less than or equal to 1, so OTS$DIV_PK_LONG
!      should be used instead of OTS$DIV_PK_SHORT.
!
!-
CALL OTS$DIV_PK_LONG( NATIONAL_DEBT, POPULATION, '30'W, PER_CAPITA_DEBT, &
        '10'W, '6'W, '1'W)

PRINT   "The per capita debt is ";PER_CAPITA_DEBT
END
```

This BASIC example program uses OTS$DIV_PK_LONG to perform packed decimal division. One example of the output generated by this program is as follows:

```
$ RUN DEBT
Enter national debt: ?  12345678
Enter current population: ?  1212
The per capita debt is 10186.20297
```

OTS$DIV_PK_SHORT Packed Decimal Division with Short Divisor

The Packed Decimal Division with Short Divisor routine divides fixed-point decimal data when precision and scale requirements for the quotient call for multiple-precision division.

FORMAT

OTS$DIV_PK_SHORT *packed-decimal-dividend*
,packed-decimal-divisor
,divisor-precision
,packed-decimal-quotient
,quotient-precision
,precision-data

RETURNS

None.

ARGUMENTS

packed-decimal-dividend

VMS usage: **varying_arg**
type: **packed decimal string**
access: **read only**
mechanism: **by reference**

Dividend. The **packed-decimal-dividend** argument is the address of a packed decimal string that contains the shifted dividend.

Before being passed as input, the **packed-decimal-dividend** argument is always multiplied by 10^c where *c* is defined as follows:

```
c = 31 - prec(packed-decimal-dividend)
```

Multiplying **packed-decimal-dividend** by 10^c makes **packed-decimal-dividend** a 31-digit number.

packed-decimal-divisor

VMS usage: **varying_arg**
type: **packed decimal string**
access: **read only**
mechanism: **by reference**

Divisor. The **packed-decimal-divisor** argument is the address of a packed decimal string that contains the divisor.

divisor-precision

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Precision of the divisor. The **divisor-precision** argument is a signed word integer that contains the precision of the divisor. The high-order bits are filled with zeros.

packed-decimal-quotient

VMS usage: **varying_arg**
type: **packed decimal string**
access: **write only**
mechanism: **by reference**

Quotient. The **packed-decimal-quotient** argument is the address of a packed decimal string into which OTS$DIV_PK_SHORT writes the quotient.

quotient-precision

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Precision of the quotient. The **quotient-precision** argument is a signed word that contains the precision of the quotient. The high-order bits are filled with zeros.

precision-data

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Additional digits of precision required. The **precision-data** argument is a signed word that contains the value of the additional digits of precision required.

OTS$DIV_PK_SHORT computes the **precision-data** argument as follows:

```
precision-data = scale(packed-decimal-quotient)
+ scale(packed-decimal-divisor)
- scale(packed-decimal-dividend)
- 31 + prec(packed-decimal-dividend)
```

DESCRIPTION

Before using this routine, you should determine whether it is best to use OTSDIV_PK_LONG, OTSDIV_PK_SHORT, or the VAX instruction DIVP. To determine this, you must first calculate *b*, where *b* is defined as follows:

```
b = scale(packed-decimal-quotient) + scale(packed-decimal-divisor) -
  scale(packed-decimal-dividend) + prec(packed-decimal-dividend)
```

If *b* is greater than 31, then OTS$DIV_PK_SHORT can be used to perform the division. If *b* is less than 31, you could use the VAX instruction DIVP instead.

Once you have determined that you cannot use DIVP, you need to determine whether you should use OTS$DIV_PK_LONG or OTS$DIV_PK_SHORT. To determine this, you must examine the value of **scale-data**. If **scale-data** is less than or equal to 1, then you should use OTS$DIV_PK_LONG. If **scale-data** is greater than 1, you should use OTS$DIV_PK_SHORT instead.

CONDITION VALUE SIGNALED

SS$_FLTDIV	Fatal error. Division by zero.

OTS$MOVE3 Move Data Without Fill

The Move Data Without Fill routine moves up to $2^{31}-1$ bytes (2,147,483,647 bytes) from a specified source address to a specified destination address.

FORMAT

OTS$MOVE3 *length-value ,source-array ,destination-array*

corresponding jsb entry point

OTS$MOVE3_R5

RETURNS

None.

ARGUMENTS

length-value

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Number of bytes of data to move. The **length-value** argument is a signed longword that contains the number of bytes to move. The value of **length-value** may range from 0 to 2,147,483,647 bytes.

source-array

VMS usage: **vector_byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference, array reference**

Data to be moved by OTS$MOVE3. The **source-array** argument contains the address of an unsigned byte array that contains this data.

destination-array

VMS usage: **vector_byte_unsigned**
type: **byte (unsigned)**
access: **write only**
mechanism: **by reference, array reference**

Address into which **source-array** will be moved. The **destination-array** argument is the address of an unsigned byte array into which OTS$MOVE3 writes the source data.

OTS$MOVE3

DESCRIPTION

OTS$MOVE3 performs the same function as the VAX MOVC3 instruction except that the **length-value** is a longword integer rather than a word integer. When called from the JSB entry point, the register outputs of OTS$MOVE3_R5 follow the same pattern as those of the MOVC3 instruction:

R0	0
R1	Address of one byte beyond the source string
R2	0
R3	Address of one byte beyond the destination string
R4	0
R5	0

For more information, see the description of the MOVC3 instruction in the *VAX Architecture Manual*. See also the routine LIB$MOVC3, which is a callable version of the MOVC3 instruction.

CONDITION VALUES RETURNED

None.

OTS$MOVE5 Move Data with Fill

The Move Data with Fill routine moves up to $2^{31}-1$ bytes (2,147,483,647 bytes) from a specified source address to a specified destination address, with separate source and destination lengths, and with fill. Overlap of the source and destination arrays does not affect the result.

FORMAT

OTS$MOVE5 *longword-int-source-length ,source-array ,fill-value ,longword-int-dest-length ,destination-array*

corresponding jsb entry point

OTS$MOVE5_R5

RETURNS

None.

ARGUMENTS

longword-int-source-length

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Number of bytes of data to move. The **longword-int-source-length** argument is a signed longword that contains this number. The value of **longword-int-source-length** may range from 0 to 2,147,483,647.

source-array

VMS usage: **vector_byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by reference, array reference**

Data to be moved by OTS$MOVE5. The **source-array** argument contains the address of an unsigned byte array that contains this data.

fill-value

VMS usage: **byte_unsigned**
type: **byte (unsigned)**
access: **read only**
mechanism: **by value**

Character used to pad the source data if **longword-int-source-length** is less than **longword-int-dest-length**. The **fill-value** argument contains the address of an unsigned byte that is this character.

OTS$MOVE5

longword-int-dest-length

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Size of the destination area in bytes. The **longword-int-dest-length** argument is a signed longword containing this size. The value of **longword-int-dest-length** may range from 0 through 2,147,483,647.

destination-array

VMS usage: **vector_byte_unsigned**
type: **byte (unsigned)**
access: **write only**
mechanism: **by reference, array reference**

Address into which **source-array** is moved. The **destination-array** argument is the address of an unsigned byte array into which OTS$MOVE5 writes the source data.

DESCRIPTION

OTS$MOVE5 performs the same function as the VAX MOVC5 instruction except that the **longword-int-source-length** and **longword-int-dest-length** arguments are longword integers rather than word integers. When called from the JSB entry point, the register outputs of OTS$MOVE5_R5 follow the same pattern as those of the MOVC5 instruction:

R0	Number of unmoved bytes remaining in source string
R1	Address of one byte beyond the source string
R2	0
R3	Address of one byte beyond the destination string
R4	0
R5	0

For more information, see the description of the MOVC5 instruction in the *VAX Architecture Manual*. See also the routine LIB$MOVC5, which is a callable version of the MOVC5 instruction.

CONDITION VALUES RETURNED

None.

OTS$MULCx Complex Multiplication

The Complex Multiplication routines calculate the complex product of two complex values.

FORMAT

OTS$MULCD_R3 *complex-multiplier ,complex-multiplicand*

OTS$MULCG_R3 *complex-multiplier ,complex-multiplicand*

These formats correspond to the D-floating and G-floating complex types.

RETURNS

VMS usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **write only**
mechanism: **by value**

Complex result of multiplying two complex numbers. OTS$MULCD_R3 returns a D-floating complex number. OTS$MULCG_R3 returns a G-floating complex number.

ARGUMENTS

complex-multiplier

VMS usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**

Complex multiplier. The **complex-multiplier** argument contains the complex multiplier. For OTS$MULCD_R3, **complex-multiplier** is a D-floating complex number. For OTS$MULCG_R3, **complex-multiplier** is a G-floating complex number.

complex-multiplicand

VMS usage: **complex_number**
type: **D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**

Complex multiplicand. The **complex-multiplicand** argument contains the complex multiplicand. For OTS$MULCD_R3, **complex-multiplicand** is a D-floating complex number. For OTS$MULCG_R3, **complex-multiplicand** is an F-floating complex number.

DESCRIPTION

OTS$MULCD_R3 and OTS$MULCG_R3 calculate the complex product of two complex values.

The complex product is computed as follows:

1 Let (a,b) represent the complex multiplier.

2 Let (c,d) represent the complex multiplicand.

3 Let (r,i) represent the complex product.

The results of this computation are as follows:

$$(a,b) * (c,d) = (ac - bd) + \sqrt{-1}(ad + bc)$$
$$Therefore : r = ac - bd$$
$$Therefore : i = ad + bc$$

CONDITION VALUES SIGNALED

MTH$_FLOOVEMAT	Floating-point overflow in math library.
SS$_ROPRAND	Reserved operand. OTS$MULCx encountered a floating-point reserved operand because of incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by DIGITAL.

EXAMPLE

```
C+
C    This FORTRAN example forms the product of
C    two complex numbers using OTS$MULCD_R3
C    and the FORTRAN random number generator RAN.
C
C    Declare Z1, Z2, and Z_Q as complex values. OTS$MULCD_R3
C    returns the complex product of Z1 times Z2:
C    Z_Q = Z1 * Z2
C-

        COMPLEX*16 Z1,Z2,Z_Q
C+
C    Generate a complex number.
C-
        Z1 = (8.0,4.0)
C+
C    Generate another complex number.
C-
        Z2 = (2.0,3.0)
C+
C    Compute the complex product of Z1*Z2.
C-
        Z_Q = Z1 * Z2
        TYPE *, ' The complex product of',Z1,' times ',Z2,' is'
        TYPE *, '      ',Z_Q
        END
```

This FORTRAN example uses OTS$MULCD_R3 to multiply two complex numbers. The output generated by this program is as follows:

```
 The complex product of (8.000000000000000,4.000000000000000) times
(2.000000000000000,3.000000000000000) is
      (4.000000000000000,32.00000000000000)
```

OTS$POWCxCx Raise a Complex Base to a Complex Floating-Point Exponent

The Raise a Complex Base to a Complex Floating-Point Exponent routines raise a complex base to a complex exponent.

FORMAT

OTS$POWCC *complex-base ,complex-exponent-value*
OTS$POWCDCD_R3 *complex-base ,complex-exponent-value*
OTS$POWCGCG_R3 *complex-base ,complex-exponent-value*

Each of these three formats corresponds to one of the three floating-point complex types.

RETURNS

VMS usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **write only**
mechanism: **by value**

Result of raising a complex base to a complex exponent. OTS$POWCC returns an F-floating complex number. OTS$POWCDCD_R3 returns a D-floating complex number. OTS$POWCGCG_R3 returns a G-floating complex number.

ARGUMENTS

complex-base

VMS usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**

Complex base. The **complex-base** argument contains the value of the base. For OTS$POWCC, **complex-base** is an F-floating complex number. For OTS$POWCDCD_R3, **complex-base** is a D-floating complex number. For OTS$POWCGCG_R3, **complex-base** is a G-floating complex number.

complex-exponent-value

VMS usage: **complex_number**
type: **F_floating complex, D_floating complex, G_floating complex**
access: **read only**
mechanism: **by value**

Complex exponent. The **complex-exponent-value** argument contains the value of the exponent. For OTS$POWCC, **complex-exponent-value** is an

F-floating complex number. For OTS$POWCDCD_R3, **complex-exponent-value** is a D-floating complex number. For OTS$POWCGCG_R3, **complex-exponent-value** is a G-floating complex number.

DESCRIPTION

OTS$POWCC, OTS$POWCDCD_R3 and OTS$POWCGCG_R3 raise a complex base to a complex exponent. The American National Standard FORTRAN-77 (ANSI X3.9–1978) defines complex exponentiation as follows:

$$x^y = exp(y * log(x))$$

In this example, x and y are type COMPLEX.

CONDITION VALUES SIGNALED

MTH$_INVARGMAT	Invalid argument in math library. Base is (0.,0.).
MTH$_FLOOVEMAT	Floating-point overflow in math library.
SS$_ROPRAND	Reserved operand.

EXAMPLES

1

```
C+
C    This FORTRAN example raises a complex base to a complex
C    power using OTS$POWCC.
C
C    Declare Z1, Z2, Z3, and OTS$POWCC as complex values. Then OTS$POWCC
C    returns the complex result of Z1**Z2:    Z3 = OTS$POWCC(Z1,Z2),
C    where Z1 and Z2 are passed by value.
C-
        COMPLEX Z1,Z2,Z3,OTS$POWCC
C+
C    Generate a complex base.
C-
        Z1 = (2.0,3.0)
C+
C    Generate a complex power.
C-
        Z2 = (1.0,2.0)
C+
C    Compute the complex value of Z1**Z2.
C-
        Z3 = OTS$POWCC( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)),
     +                 %VAL(REAL(Z2)), %VAL(AIMAG(Z2)))
        TYPE *, ' The value of',Z1,'**',Z2,' is',Z3
        END
```

This FORTRAN example uses OTS$POWCC to raise an F-floating complex base to an F-floating complex exponent.

The output generated by this program is as follows:

```
 The value of (2.000000,3.000000)** (1.000000,2.000000) is
(-0.4639565,-0.1995301)
```

2

```
C+
C    This FORTRAN example raises a complex base to a complex
C    power using OTS$POWCGCG_R3.
C
C    Declare Z1, Z2, and Z3 as complex values. OTS$POWCGCG_R3
C    returns the complex result of Z1**Z2: Z3 = Z1**Z2.
C-
        COMPLEX*16 Z1,Z2,Z3
C+
C    Generate a complex base.
C-
        Z1 = (2.0,3.0)
C+
C    Generate a complex power.
C-
        Z2 = (1.0,2.0)
C+
C    Compute the complex value of Z1**Z2.
C-
        Z3 = Z1**Z2
        TYPE 1,Z1,Z2,Z3
   1    FORMAT(' The value of (',F11.8,',',F11.8,')**(',F11.8,
     +  ',',F11.8,') is (',F11.8,',',F11.8,').')
        END
```

This FORTRAN example program shows how to use OTS$POWCGCG_R3. Notice the high precision in the output generated by this program:

```
The value of ( 2.00000000, 3.00000000)**( 1.00000000, 2.00000000) is
(-0.46395650,-0.46395650).
```

OTS$POWCxJ Raise a Complex Base to a Signed Longword Integer Exponent

The Raise a Complex Base to a Signed Longword Integer Exponent routines return the complex result of raising a complex base to an integer exponent.

FORMAT

OTS$POWCJ *complex-base ,longword-integer-exponent*
OTS$POWCDJ_R3 *complex-base ,longword-integer-exponent*
OTS$POWCGJ_R3 *complex-base ,longword-integer-exponent*

Each of these three formats corresponds to one of the three floating-point complex types.

RETURNS

VMS usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**write only**
mechanism:	**by value**

Complex result of raising a complex base to an integer exponent. OTS$POWCJ returns an F-floating complex number. OTS$POWCDJ_R3 returns a D-floating complex number. OTS$POWCGJ_R3 returns a G-floating complex number. In each format, the result and base are of the same data type.

ARGUMENTS

complex-base

VMS usage:	**complex_number**
type:	**F_floating complex, D_floating complex, G_floating complex**
access:	**read only**
mechanism:	**by value**

Complex base. The **complex-base** argument contains the complex base. For OTS$POWCJ, **complex-base** is an F-floating complex number. For OTS$POWCDJ_R3, **complex-base** is a D-floating complex number. For OTS$POWCGJ_R3, **complex-base** is a G-floating complex number.

longword-integer-exponent

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Exponent. The **longword-integer-exponent** argument is a signed longword containing the exponent.

DESCRIPTION

OTS$POWCJ, OTS$POWCDJ_R3, and OTS$POWCGJ_R3 return the complex result of raising a complex base to an integer exponent. The complex result is as follows:

Base	Exponent	Result
Any	>0	The product of (base*2^i), where *i* is each nonzero bit in **longword-integer-exponent**
(0.,0.)	≤ 0	Undefined exponentiation
Not (0.,0.)	<0	The product of (base*2^i), where *i* is each nonzero bit in **longword-integer-exponent**
Not (0.,0.)	0	(1.0,0.0)

CONDITION VALUES SIGNALED

SS$_FLTDIV	Floating-point division by zero.
SS$_FLTOVF	Floating-point overflow.
MTH$_UNDEXP	Undefined exponentiation.

EXAMPLE

```
C+
C    This FORTRAN example raises a complex base to
C    a NONNEGATIVE integer power using OTS$POWCJ.
C
C    Declare Z1, Z2, Z3, and OTS$POWCJ as complex values.
C    Then OTS$POWCJ returns the complex result of
C    Z1**Z2:   Z3 = OTS$POWCJ(Z1,Z2),
C    where Z1 and Z2 are passed by value.
C-
        COMPLEX Z1,Z3,OTS$POWCJ
        INTEGER Z2
C+
C    Generate a complex base.
C-
        Z1 = (2.0,3.0)
C+
C    Generate an integer power.
C-
        Z2 = 2
```

```
C+
C    Compute the complex value of Z1**Z2.
C-
       Z3 = OTS$POWCJ( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)), %VAL(Z2))
       TYPE 1,Z1,Z2,Z3
  1    FORMAT(' The value of (',F10.8,',',F11.8,')**',I1,' is
     + (',F11.8,',',F12.8,').')
       END
```

The output generated by this FORTRAN program is as follows:

```
The value of (2.00000000, 3.00000000)**2 is
(-5.00000000, 12.00000000).
```

OTS$POWDD Raise a D-floating Base to a D-floating Exponent

The Raise a D-floating Base to a D-floating Exponent routine raises a D-floating base to a D-floating exponent.

FORMAT

OTS$POWDD *D-floating-point-base ,D-floating-point-exponent*

RETURNS

VMS usage: **floating_point**
type: **D_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

D-floating-point-base

VMS usage: **floating_point**
type: **D_floating**
access: **read only**
mechanism: **by value**

Base. The **D-floating-point-base** argument is a D-floating number containing the base.

D-floating-point-exponent

VMS usage: **floating_point**
type: **D_floating**
access: **read only**
mechanism: **by value**

Exponent. The **D-floating-point-exponent** argument is a D-floating number that contains the exponent.

DESCRIPTION

OTS$POWDD raises a D-floating base to a D-floating exponent.

The internal calculations and the floating-point result have the same precision as the base value.

The D-floating result for OTS$POWDD is given by the following:

Base	Exponent	Result
$=0$	>0	0.0
$=0$	$=0$	Undefined exponentiation
$=0$	<0	Undefined exponentiation
<0	Any	Undefined exponentiation
>0	>0	$2^{[exponent*\log2(base)]}$
>0	$=0$	1.0
>0	<0	$2^{[exponent*\log2(base)]}$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative, or if the base is negative.

CONDITION VALUES SIGNALED

MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **D-floating-point-base** is zero and **D-floating-point-exponent** is zero or negative, or if the **D-floating-point-base** is negative.

OTS$POWDR Raise a D-floating Base to an F-floating Exponent

The Raise a D-floating Base to an F-floating Exponent routine raises a D-floating base to an F-floating exponent.

FORMAT

OTS$POWDR *D-floating-point-base ,F-floating-point-exponent*

RETURNS

VMS usage: **floating_point**
type: **D_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

D-floating-point-base

VMS usage: **floating_point**
type: **D_floating**
access: **read only**
mechanism: **by value**

Base. The **D-floating-point-base** argument is a D-floating number containing the base.

F-floating-point-exponent

VMS usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by value**

Exponent. The **F-floating-point-exponent** argument is an F-floating number that contains the exponent.

DESCRIPTION

OTS$POWDR raises a D-floating base to an F-floating exponent.

The internal calculations and the floating-point result have the same precision as the base value.

OTS$POWDR converts the F-floating exponent to a D-floating number. The D-floating result for OTS$POWDR is given by the following:

OTS$POWDR

Base	Exponent	Result
$= 0$	> 0	0.0
$= 0$	$= 0$	Undefined exponentiation
$= 0$	< 0	Undefined exponentiation
< 0	Any	Undefined exponentiation
> 0	> 0	$2^{[exponent*\log2(base)]}$
> 0	$= 0$	1.0
> 0	< 0	$2^{[exponent*\log2(base)]}$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative, or if the base is negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **D-floating-point-base** is zero and **F-floating-point-exponent** is zero or negative, or if the **D-floating-point-base** is negative.

OTS$POWDJ Raise a D-floating Base to a Longword Exponent

The Raise a D-floating Base to a Longword Exponent routine raises a D-floating base to a longword exponent.

FORMAT

OTS$POWDJ *D-floating-point-base ,longword-integer-exponent*

RETURNS

VMS usage: **floating_point**
type: **D_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

D-floating-point-base

VMS usage: **floating_point**
type: **D_floating**
access: **read only**
mechanism: **by value**

Base. The **D-floating-point-base** argument is a D-floating number containing the base.

longword-integer-exponent

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Exponent. The **longword-integer-exponent** argument is a signed longword that contains the signed longword integer exponent.

DESCRIPTION

OTS$POWDJ raises a D-floating base to a longword exponent.

The internal calculations and the floating-point result have the same precision as the base value.

The floating-point result is as follows:

Base	Exponent	Result
Any	> 0	Product of ($base*2^i$) where *i* is each nonzero bit position in **longword-integer-exponent**
> 0	$= 0$	1.0
$= 0$	$= 0$	Undefined exponentiation
< 0	$= 0$	1.0
> 0	< 0	1.0/ ($base*2^i$), where *i* is each nonzero bit position in **longword-integer-exponent**
$= 0$	< 0	Undefined exponentiation
< 0	< 0	1.0/ ($base*2^i$) where *i* is each nonzero bit position in **longword-integer-exponent**

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **D-floating-point-base** is zero and **longword-integer-exponent** is zero or negative, or if the **D-floating-point-base** is negative.

OTS$POWGG Raise a G-floating Base to a G-floating Exponent

The Raise a G-floating Base to a G-floating Exponent routine raises a G-floating base to a G-floating exponent.

FORMAT

OTS$POWGG *G-floating-point-base ,G-floating-point-exponent*

RETURNS

VMS usage: **floating_point**
type: **G_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

G-floating-point-base
VMS usage: **floating_point**
type: **G_floating**
access: **read only**
mechanism: **by value**

Base that OTS$POWGG raises to a G-floating exponent. The **G-floating-point-base** argument is a G-floating number containing the base.

G-floating-point-exponent
VMS usage: **floating_point**
type: **G_floating**
access: **read only**
mechanism: **by value**

Exponent to which OTS$POWGG raises the base. The **G-floating-point-exponent** argument is a G-floating number containing the exponent.

DESCRIPTION

OTS$POWGG raises a G-floating base to a G-floating exponent.

The internal calculations and the floating-point result have the same precision as the base value.

The G-floating result for OTS$POWGG is as follows:

Base	Exponent	Result
$= 0$	> 0	0.0
$= 0$	$= 0$	Undefined exponentiation
$= 0$	< 0	Undefined exponentiation
< 0	Any	Undefined exponentiation
> 0	> 0	$2^{[exponent*\log2(base)]}$
> 0	$= 0$	1.0
> 0	< 0	$2^{[exponent*\log2(base)]}$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative, or if the base is negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponent. This error is signaled if **G-floating-point-base** is zero and **G-floating-point-exponent** is zero or negative, or if **G-floating-point-base** is negative.

EXAMPLE

```
C+
C   This example demonstrates the use of OTS$POWGG,
C   which raises a G-floating point base
C   to a G-floating point power.
C-
        REAL*8 X,Y,RESULT,OTS$POWGG
C+
C   The arguments of OTS$POWGG are passed by value. FORTRAN can
C   only pass INTEGER and REAL*4 expressions as VALUE. Since
C   INTEGER and REAL*4 values are one longword long, while REAL*8
C   values are two longwords long, equate the base (and power) to
C   two-dimensional INTEGER vectors. These vectors will be passed
C   by VALUE.
C-
        INTEGER N(2),M(2)
        EQUIVALENCE (N(1),X), (M(1),Y)
        X = 8.0
        Y = 2.0
C+
C  To pass X by value, pass N(1) and N(2) by value. Similarly for Y.
C-
        RESULT = OTS$POWGG(%VAL(N(1)),%VAL(N(2)),%VAL(M(1)),%VAL(M(2)))
        TYPE *,' 8.0**2.0 IS ',RESULT
        X = 9.0
        Y = -0.5
C+
```

```
C   In FORTRAN, OTS$POWWGG is indirectly called by simply using the
C   exponentiation operator.
C-
        RESULT = X**Y
        TYPE *,' 9.0**-0.5 IS ',RESULT
        END
```

This FORTRAN example uses OTS$POWGG to raise a G-floating base to a G-floating exponent.

The output generated by this example is as follows:

```
8.0**2.0 IS    64.0000000000000
9.0**-0.5 IS  0.333333333333333
```

OTS$POWGJ Raise a G-floating Base to a Longword Exponent

The Raise a G-floating Base to a Longword Exponent routine raises a G-floating base to a longword exponent.

FORMAT

OTS$POWGJ *G-floating-point-base ,longword-integer-exponent*

RETURNS

VMS usage: **floating_point**
type: **G_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

G-floating-point-base

VMS usage: **floating_point**
type: **G_floating**
access: **read only**
mechanism: **by value**

Base that OTS$POWGJ raises to a longword exponent. The **G-floating-point-base** argument is a G-floating number containing the base.

longword-integer-exponent

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Exponent to which OTS$POWGJ raises the base. The **longword-integer-exponent** argument is a signed longword containing the exponent.

DESCRIPTION

OTS$POWGJ raises a G-floating base to a longword exponent.

The internal calculations and the floating-point result have the same precision as the base value.

The floating-point result is as follows:

Base	Exponent	Result
Any	> 0	Product of (base*2^i) where *i* is each nonzero bit position in **longword-integer-exponent**
> 0	$= 0$	1.0
$= 0$	$= 0$	Undefined exponentiation
< 0	$= 0$	1.0
> 0	< 0	1.0/ (base*2^i), where *i* is each nonzero bit position in **longword-integer-exponent**
$= 0$	< 0	Undefined exponentiation
< 0	< 0	1.0/ (base*2^i) where *i* is each nonzero bit position in **longword-integer-exponent**

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponent. This error is signaled if **G-floating-point-base** is zero and **longword-integer-exponent** is zero or negative, or if **G-floating-point-base** is negative.

OTS$POWHH_R3 Raise an H-floating Base to an H-floating Exponent

The Raise an H-floating Base to an H-floating Exponent routine raises an H-floating base to an H-floating exponent.

FORMAT

OTS$POWHH_R3 *H-floating-point-base ,H-floating-point-exponent*

RETURNS

VMS usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

H-floating-point-base

VMS usage: **floating_point**
type: **H_floating**
access: **read only**
mechanism: **by value**

Base. The **H-floating-point-base** argument is an H-floating number containing the base.

H-floating-point-exponent

VMS usage: **floating_point**
type: **H_floating**
access: **read only**
mechanism: **by value**

Exponent. The **H-floating-point-exponent** argument is an H-floating number that contains the H-floating exponent.

DESCRIPTION

OTS$POWHH_R3 raises an H-floating base to an H-floating exponent.

The internal calculations and the floating-point result have the same precision as the base value.

The H-floating result for OTS$POWHH_R3 is as follows:

Base	Exponent	Result
$= 0$	> 0	0.0
$= 0$	$= 0$	Undefined exponentiation
$= 0$	< 0	Undefined exponentiation
< 0	Any	Undefined exponentiation
> 0	> 0	$2^{[exponent*\log2(base)]}$
> 0	$= 0$	1.0
> 0	< 0	$2^{[exponent*\log2(base)]}$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative, or if the base is negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **H-floating-point-base** is zero and **H-floating-point-exponent** is zero or negative, or if the **H-floating-point-base** is negative.

EXAMPLE

```
C+
C Example of OTS$POWHH, which raises an H_floating
C point base to an H_floating point power. In FORTRAN,
C it is not directly called.
C-
        REAL*16 X,Y,RESULT
        X = 9877356535.0
        Y = -0.5837653
C+
C In FORTRAN, OTS$POWWHH is indirectly called by simply using the
C exponentiation operator.
C-
        RESULT = X**Y
        TYPE *,' 9877356535.0**-0.5837653 IS ',RESULT
        END
```

This FORTRAN example demonstrates how to call OTS$POWHH_R3 to raise an H-floating base to an H-floating power.

The output generated by this program is as follows:

```
9877356535.0**-0.5837653 IS   1.4637791459946283574823435982054Z7E-0006
```

OTS$POWHJ_R3 Raise an H-floating Base to a Longword Exponent

The Raise an H-floating Base to a Longword Exponent routine raises an H-floating base to a longword exponent.

FORMAT

OTS$POWHJ_R3 *H-floating-point-base ,longword-integer-exponent*

RETURNS

VMS usage: **floating_point**
type: **H_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

H-floating-point-base

VMS usage: **floating_point**
type: **H_floating**
access: **read only**
mechanism: **by value**

Base. The **H-floating-point-base** argument is an H-floating number containing the base.

longword-integer-exponent

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Exponent. The **longword-integer-exponent** argument is a signed longword that contains the signed longword exponent.

DESCRIPTION

OTS$POWHJ_R3 raises an H-floating base to a longword exponent.

The internal calculations and the floating-point result have the same precision as the base value.

The floating-point result is as follows:

Base	Exponent	Result
Any	> 0	Product of (base*2^i) where *i* is each nonzero bit position in **longword-integer-exponent**
> 0	$= 0$	1.0
$= 0$	$= 0$	Undefined exponentiation
< 0	$= 0$	1.0
> 0	< 0	1.0/ (base*2^i), where *i* is each nonzero bit position in **longword-integer-exponent**
$= 0$	< 0	Undefined exponentiation
< 0	< 0	1.0/ (base*2^i) where *i* is each nonzero bit position in **longword-integer-exponent**

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **H-floating-point-base** is zero and **longword-integer-exponent** is zero or negative, or if the **H-floating-point-base** is negative.

OTS$POWII Raise a Word Base to a Word Exponent

The Raise a Word Base to a Word Exponent routine raises a word base to a word exponent.

FORMAT

OTS$POWII *word-integer-base ,word-integer-exponent*

RETURNS

VMS usage: **word_signed**
type: **word (signed)**
access: **write only**
mechanism: **by value**

ARGUMENTS

word-integer-base

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Base. The **word-integer-base** argument is a signed word containing the base.

word-integer-exponent

VMS usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by value**

Exponent. The **word-integer-exponent** argument is a signed word containing the exponent.

CONDITION VALUES SIGNALED

SS$_FLTDIV	Arithmetic trap. This error is signaled by the hardware if a floating-point division by zero occurs.
SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **word-integer-base** is zero and **word-integer-exponent** is zero or negative, or if **word-integer-base** is negative.

OTS$POWJJ Raise a Longword Base to a Longword Exponent

The Raise a Longword Base to a Longword Exponent routine raises a signed longword base to a signed longword exponent.

FORMAT

OTS$POWJJ *longword-integer-base ,longword-integer-exponent*

RETURNS

VMS usage:	**longword_signed**
type:	**longword (signed)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

longword-integer-base

VMS usage:	**longword_signed**
type:	**longword (signed)**
access:	**read only**
mechanism:	**by value**

Base. The **longword-integer-base** argument is a signed longword containing the base.

longword-integer-exponent

VMS usage:	**longword_signed**
type:	**longword (signed)**
access:	**read only**
mechanism:	**by value**

Exponent. The **longword-integer-exponent** argument is a signed longword containing the exponent.

CONDITION VALUES SIGNALED

SS$_FLTDIV	Arithmetic trap. This error is signaled by the hardware if a floating-point division by zero occurs.
SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **longword-integer-base** is zero and **longword-integer-exponent** is zero or negative, or if **longword-integer-base** is negative.

OTS$POWLULU Raise an Unsigned Longword Base to an Unsigned Longword Exponent

The Raise an Unsigned Longword Base to an Unsigned Longword Exponent routine raises an unsigned longword integer base to an unsigned longword integer exponent.

FORMAT

OTS$POWLULU *unsigned-lword-int-base, unsigned-lword-int-exponent*

RETURNS

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

ARGUMENTS

unsigned-lword-int-base

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Unsigned longword integer base. The **unsigned-lword-int-base** argument contains the value of the integer base.

unsigned-lword-int-exponent

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Unsigned longword integer exponent. The **unsigned-lword-int-exponent** argument contains the value of the integer exponent.

DESCRIPTION

OTS$POWLULU returns the unsigned longword integer result of raising an unsigned longword integer base to an unsigned longword integer exponent. Note that overflow cannot occur in this routine. If the result or intermediate result is greater than 32 bits, the low-order 32 bits are used.

CONDITION VALUES SIGNALED

MTH$_UNDEXP	Both the base and exponent values are zero.

OTS$POWxLU Raise a Floating-Point Base to an Unsigned Longword Integer Exponent

The Raise a Floating-Point Base to an Unsigned Longword Integer Exponent routines raises a floating-point base to an unsigned longword integer exponent.

FORMAT

OTS$POWRLU *floating-point-base ,unsigned-lword-int-exponent*
OTS$POWDLU *floating-point-base ,unsigned-lword-int-exponent*
OTS$POWGLU *floating-point-base ,unsigned-lword-int-exponent*
OTS$POWHLU_R3 *floating-point-base ,unsigned-lword-int-exponent*

RETURNS

VMS usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **write only**
mechanism: **by value**

Result of raising a floating-point base to an unsigned longword integer exponent. OTS$POWRLU returns an F-floating number. OTS$POWDLU returns a D-floating number. OTS$POWGLU returns a G-floating number. OTS$POWHLU_R3 returns an H-floating number.

ARGUMENTS

floating-point-base

VMS usage: **floating_point**
type: **F_floating, D_floating, G_floating, H_floating**
access: **read only**
mechanism: **by value**

Floating-point base. The **floating-point-base** argument contains the value of the base. For OTS$POWRLU, **floating-point-base** is an F-floating number. For OTS$POWDLU, **floating-point-base** is a D-floating number. For OTS$POWGLU, **floating-point-base** is a G-floating number. For OTS$POWHLU_R3, **floating-point-base** is an H-floating number.

unsigned-lword-int-exponent

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Integer exponent. The **unsigned-lword-int-exponent** argument contains the value of the unsigned longword integer exponent.

DESCRIPTION

OTS$POWRLU, OTS$POWDLU, OTS$POWGLU, and OTS$POWHLU_R3 return the result of raising a floating-point base to an unsigned longword integer exponent. The floating-point result is as follows:

Base	Exponent	Result
Any	>0	Product of (base*2^i) where *i* is each nonzero bit position in **longword-integer-exponent**
>0	$=0$	1.0
$=0$	$=0$	Undefined exponentiation
<0	$=0$	1.0

CONDITION VALUES SIGNALED

MTH$_FLOOVEMAT	Floating-point overflow in math library
MTH$_FLOUNDMAT	Floating-point underflow in math library. This can only occur if the caller has floating-point underflow enabled.
MTH$_UNDEXP	Undefined exponentiation. This occurs if both the **floating-point-base** and **unsigned-longword-integer-exponent** arguments are zero.

OTS$POWRD Raise an F-floating Base to a D-floating Exponent

The Raise an F-floating Base to a D-floating Exponent routine raises an F-floating base to a D-floating exponent.

FORMAT

OTS$POWRD *F-floating-point-base ,D-floating-point-exponent*

RETURNS

VMS usage: **floating_point**
type: **D_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

F-floating-point-base

VMS usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by value**

Base. The **F-floating-point-base** argument is an F-floating number containing the base.

D-floating-point-exponent

VMS usage: **floating_point**
type: **D_floating**
access: **read only**
mechanism: **by value**

Exponent. The **D-floating-point-exponent** argument is a D-floating number that contains the exponent.

DESCRIPTION

OTS$POWRD raises an F-floating base to a D-floating exponent.

The internal calculations and the floating-point result have the same precision as the base value.

OTS$POWRD first converts the F-floating base to D-floating. The D-floating result for OTS$POWRD is as follows:

Base	Exponent	Result
$= 0$	> 0	0.0
$= 0$	$= 0$	Undefined exponentiation
$\neq 0$	< 0	Undefined exponentiation
< 0	Any	Undefined exponentiation
> 0	> 0	$2^{[exponent*\mathrm{LOG2}(base)]}$
> 0	$= 0$	1.0
> 0	< 0	$2^{[exponent*\mathrm{LOG2}(base)]}$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative, or if the base is negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **F-floating-point-base** is zero and **D-floating-point-exponent** is zero or negative, or if **F-floating-point-base** is negative.

EXAMPLE

```
C+
C   This FORTRAN example demonstrates the use
C   of OTS$POWRD, which raises an F-floating point
C   base to a D-floating point exponent. The result is a
C   D-floating value.
C-

        REAL*4 X
        REAL*8 Y,RESULT,OTS$POWRD
        INTEGER M(2)
        EQUIVALENCE (M(1),Y)
        X = 9768.0
        Y = 9.0

C+
C  The arguments of OTS$POWRD are passed by value.
C-

        RESULT = OTS$POWRD(%VAL(X),%VAL(M(1)),%VAL(M(2)))
        TYPE *,' 9768.0**9.0 IS ',RESULT
        X = 7689.0
        Y = -0.587436654545

C+
C   In FORTRAN, OTS$POWRD is indirectly called by simply
C   using the exponentiation operator.
C-
```

```
RESULT = X**Y
TYPE *,' 7689.0**-0.587436654545 IS ',RESULT
END
```

This FORTRAN example uses OTS$POWRD to raise an F-floating base to a D-floating exponent. Notice the difference in the precision of the result produced by this routine in comparison to the result produced by OTS$POWRR.

The output generated by this program is as follows:

```
9768.0**9.0 IS   8.0956338648832908E+35
7689.0**-0.587436654545 IS   5.2155199252836588E-03
```

OTS$POWRJ Raise an F-floating Base to a Longword Exponent

The Raise an F-floating Base to a Longword Exponent routine raises an F-floating base to a longword exponent.

FORMAT

OTS$POWRJ *F-floating-point-base ,longword-integer-exponent*

RETURNS

VMS usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

F-floating-point-base

VMS usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by value**

Base. The **F-floating-point-base** argument is an F-floating number containing the base.

longword-integer-exponent

VMS usage: **longword_signed**
type: **longword (signed)**
access: **read only**
mechanism: **by value**

Exponent. The **longword-integer-exponent** argument is a signed longword that contains the longword exponent.

DESCRIPTION

OTS$POWRJ raises an F-floating base to a longword exponent.

The internal calculations and the floating-point result have the same precision as the base value.

The floating-point result is as follows:

Base	Exponent	Result
Any	> 0	Product of (base*2^i) where *i* is each nonzero bit position in **longword-integer-exponent**
> 0	$= 0$	1.0
$= 0$	$= 0$	Undefined exponentiation
< 0	$= 0$	1.0
> 0	< 0	1.0/ (base*2^i), where *i* is each nonzero bit position in **longword-integer-exponent**
$= 0$	< 0	Undefined exponentiation
< 0	< 0	1.0/ (base*2^i) where *i* is each nonzero bit position in **longword-integer-exponent**

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **F-floating-point-base** is zero and **longword-integer-exponent** is zero or negative, or if **F-floating-point-base** is negative.

OTS$POWRR Raise an F-floating Base to an F-floating Exponent

The Raise an F-floating Base to an F-floating Exponent routine raises an F-floating base to an F-floating exponent.

FORMAT

OTS$POWRR *F-floating-point-base ,F-floating-point-exponent*

RETURNS

VMS usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by value**

ARGUMENTS

F-floating-point-base

VMS usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by value**

Base. The **F-floating-point-base** argument is an F-floating number containing the base.

F-floating-point-exponent

VMS usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by value**

Exponent. The **F-floating-point-exponent** argument is an F-floating number that contains the exponent.

DESCRIPTION

OTS$POWRR raises an F-floating base to an F-floating exponent.

The internal calculations and the floating-point result have the same precision as the base value.

The F-floating result for OTS$POWRR is as follows:

Base	Exponent	Result
$=0$	>0	0.0
$=0$	$=0$	Undefined exponentiation
$=0$	<0	Undefined exponentiation
<0	Any	Undefined exponentiation
>0	>0	$2^{[exponent*\log2(base)]}$
>0	$=0$	1.0
>0	<0	$2^{[exponent*\log2(base)]}$

Floating-point overflow can occur.

Undefined exponentiation occurs if the base is zero and the exponent is zero or negative, or if the base is negative.

CONDITION VALUES SIGNALED

SS$_FLTOVF	Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
MTH$_FLOOVEMAT	Floating-point overflow in math library.
MTH$_FLOUNDMAT	Floating-point underflow in math library.
MTH$_UNDEXP	Undefined exponentiation. This error is signaled if **F-floating-point-base** is zero and **F-floating-point-exponent** is zero or negative, or if **F-floating-point-base** is negative.

EXAMPLE

```
C+
C  This FORTRAN example demonstrates the use
C  of OTS$POWRR, which raises an F-floating
C  point base to an F-floating point power.
C-

       REAL*4 X,Y,RESULT,OTS$POWRR
       X = 8.0
       Y = 2.0

C+
C  The arguments of OTS$POWRR are passed by value.
C-

       RESULT = OTS$POWRR(%VAL(X),%VAL(Y))
       TYPE *,' 8.0**2.0 IS ',RESULT
       X = 9.0
       Y = -0.5

C+
C  In FORTRAN, OTS$POWRR is indirectly called by simply
C  using the exponentiation operator.
C-

       RESULT = X**Y
       TYPE *,' 9.0**-0.5 IS ',RESULT
       END
```

This FORTRAN example uses OTS$POWRR to raise an F-floating point base to an F-floating point exponent. The output generated by this program is as follows:

```
8.0**2.0 IS     64.00000
9.0**-0.5 IS    0.3333333
```

OTS$SCOPY_DXDX Copy a Source String Passed by Descriptor to a Destination String

The Copy a Source String Passed by Descriptor to a Destination String routine copies a source string to a destination string. Both strings are passed by descriptor.

FORMAT

OTS$SCOPY_DXDX *source-string ,destination-string*

corresponding jsb entry point

OTS$SCOPY_DXDX6

RETURNS

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by value**

If **source-string** contains more characters than **destination-string**, and the JSB entry point is used, R0 contains the number of characters that were not copied.

ARGUMENTS

source-string

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Source string. The **source-string** argument is the address of a descriptor pointing to the source string. The descriptor class can be unspecified, fixed length, dynamic, scalar decimal, array, noncontiguous array, or varying.

destination-string

VMS usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string. The **destination-string** argument is the address of a descriptor pointing to the destination string. The class field determines the appropriate action.

See the Description section for further information.

OTS$SCOPY_DXDX

DESCRIPTION

OTS$SCOPY_DXDX copies a source string to a destination string. All error conditions except truncation are signaled; truncation is ignored.

OTS$SCOPY_DXDX passes the source string by descriptor. In addition, an equivalent JSB entry point is provided, with R0 being the first argument (the descriptor of the source string), and R1 the second (the descriptor of the destination string).

For the CALL entry point, R0 (return status) is as it would be after a MOVC5 instruction. For the JSB entry point, R0:R5 and the PSL are as they would be after a MOVC5 instruction. R0:R5 contain the following:

R0	Number of bytes of source string not moved to destination string
R1	Address one byte beyond the last copied byte in the source string
R2	0
R3	Address one byte beyond the destination string
R4	0
R5	0

For further information, see the *VAX Architecture Reference Manual.*

Depending on the class of the destination string, the actions described below occur:

Class Field	Action
DSC$K_CLASS_S,Z,SD,A,NCA	Copy the source string. If needed, space fill or truncate on the right.
DSC$K_CLASS_D	If the area specified by the destination descriptor is large enough to contain the source string, copy the source string and set the new length in the destination descriptor. If the area specified is not large enough, return the previous space allocation (if any) and then dynamically allocate the amount of space needed. Copy the source string and set the new length and address in the destination descriptor.
DSC$K_CLASS_VS	Copy source string to destination string up to the limit of DSC$W_MAXSTRLEN with no padding. Adjust current length field to actual number of bytes copied.

CONDITION VALUES SIGNALED

OTS$_FATINTERR	Fatal internal error.
OTS$_INVSTRDES	Invalid string descriptor.
OTS$_INSVIRMEM	Insufficient virtual memory.

OTS$SCOPY_R_DX Copy a Source String Passed by Reference to a Destination String

The Copy a Source String Passed by Reference to a Destination String routine copies a source string passed by reference to a destination string.

FORMAT

OTS$SCOPY_R_DX *word-int-source-length-val ,source-string-address ,destination-string*

corresponding jsb entry point

OTS$SCOPY_R_DX6

RETURNS

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by value**

If **source-string-address** contains more characters than **destination-string**, and the JSB entry point is used, R0 contains the number of characters that were not copied.

ARGUMENTS

word-int-source-length-val

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Length of the source string. The **word-int-source-length-val** argument is an unsigned word integer containing the length of the source string.

source-string-address

VMS usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by reference**

Source string. The **source-string-address** argument is the address of the source string.

destination-string

VMS usage: **char_string**
type: **character string**
access: **write only**
mechanism: **by descriptor**

Destination string. The **destination-string** argument is the address of a descriptor pointing to the destination string. The class field determines the appropriate action. The length field (DSC$W_LENGTH) alone or both the address (DSC$A_POINTER) and length fields can be modified if the string is dynamic. For varying strings, the current length is rewritten.

DESCRIPTION

OTS$SCOPY_R_DX copies a source string to a destination string. All conditions except truncation are signaled; truncation is ignored. Input scalars are passed by value.

OTS$SCOPY_R_DX passes the source string by reference preceded by a length argument. In addition, an equivalent JSB entry point is provided, with R0 being the first argument, R1 the second, and R2 the third, if any. The length argument is passed in bits 15:0 of the appropriate register.

For the CALL entry point, R0 (return status) is as it would be after a MOVC5 instruction. For the JSB entry point, R0:R5 and the PSL are as they would be after a MOVC5 instruction. R0:R5 contain the following:

R0	Number of bytes of source string not moved to destination string
R1	Address one byte beyond the last copied byte in the source string
R2	0
R3	Address one byte beyond the destination string
R4	0
R5	0

For additional information, see the *VAX Architecture Reference Manual.*

Depending on the class of the destination string, the actions described below occur:

Class Field	Action
DSC$K_CLASS_S,Z,SD,A,NCA	Copy the source string. If needed, space fill or truncate on the right.
DSC$K_CLASS_D	If the area specified by the destination descriptor is large enough to contain the source string, copy the source string and set the new length in the destination descriptor. If the area specified is not large enough, return the previous space allocation (if any) and then dynamically allocate the amount of space needed. Copy the source string and set the new length and address in the destination descriptor.
DSC$K_CLASS_VS	Copy source string to destination string up to the limit of DSC$W_MAXSTRLEN with no padding. Adjust current length field to actual number of bytes copied.

CONDITION VALUES SIGNALED

OTS$_FATINTERR	Fatal internal error.
OTS$_INVSTRDES	Invalid string descriptor.
OTS$_INSVIRMEM	Insufficient virtual memory.

EXAMPLE

A FORTRAN example demonstrating dynamic string manipulation appears at the end of OTS$SGET1_DD. This example uses OTS$SCOPY_R_DX, OTS$SGET1_DD, and OTS$SFREE1_DD.

OTS$SFREE1_DD Strings, Free One Dynamic

The Free One Dynamic String routine returns one dynamic string area to free storage.

FORMAT

OTS$SFREE1_DD *dynamic-descriptor*

corresponding jsb entry point **OTS$SFREE1_DD6**

RETURNS

None.

ARGUMENTS

dynamic-descriptor
VMS usage: **quadword_unsigned**
type: **quadword (unsigned)**
access: **modify**
mechanism: **by reference**

Dynamic string descriptor. The **dynamic-descriptor** argument is the address of the dynamic string descriptor. The descriptor is assumed to be dynamic and its class field is not checked.

DESCRIPTION

OTS$SFREE1_DD deallocates the described string space and flags the descriptor as describing no string at all (DSC$A_POINTER = 0 and DSC$W_LENGTH = 0).

CONDITION VALUE SIGNALED

OTS$_FATINTERR	Fatal internal error.

EXAMPLE

A FORTRAN example demonstrating dynamic string manipulation appears at the end of OTS$SGET1_DD. This example uses OTS$SFREE1_DD, OTS$SGET1_DD, and OTS$SCOPY_R_DX.

OTS$SFREEn_DD Strings, Free n Dynamic

The Free *n* Dynamic Strings routine takes as input a vector of one or more dynamic string areas and returns them to free storage.

FORMAT

OTS$SFREEN_DD *descriptor-count-value ,first-descriptor*

corresponding jsb entry point

OTS$SFREEN_DD6

RETURNS

None.

ARGUMENTS

descriptor-count-value

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

Number of adjacent descriptors to be flagged as having no allocated area (DSC$A_POINTER = 0 and DSC$W_LENGTH = 0) and to have their allocated areas returned to free storage by OTS$SFREEN_DD. The **descriptor-count-value** argument is an unsigned longword containing this number.

first-descriptor

VMS usage: **quadword_unsigned**
type: **quadword (unsigned)**
access: **modify**
mechanism: **by reference**

First string descriptor of an array of string descriptors. The **first-descriptor** argument is the address of the first string descriptor. The descriptors are assumed to be dynamic, and their class fields are not checked.

DESCRIPTION

OTS$SFREEN_DD6 deallocates the described string space and flags each descriptor as describing no string at all (DSC$A_POINTER = 0 and DSC$W_LENGTH = 0).

CONDITION VALUES SIGNALED

OTS$_FATINTERR	Fatal internal error.

OTS$SGET1_DD Strings, Get One Dynamic

The Get One Dynamic String routine allocates a specified number of bytes of dynamic virtual memory to a specified string descriptor.

FORMAT

OTS$SGET1_DD *word-integer-length-value ,dynamic-descriptor*

corresponding jsb entry point

OTS$SGET1_DD_R6

RETURNS

None.

ARGUMENTS

word-integer-length-value

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by value**

Number of bytes to be allocated. The **word-integer-length-value** argument contains the number of bytes. The amount of storage allocated is automatically rounded up. If the number of bytes is zero, a small number of bytes is allocated.

dynamic-descriptor

VMS usage: **quadword_unsigned**
type: **quadword (unsigned)**
access: **modify**
mechanism: **by reference**

Dynamic string descriptor to which the area is to be allocated. The **dyn-str** argument is the address of the dynamic string descriptor. The class field is not checked but it is set to dynamic (DSC$B_CLASS = 2). The length field (DSC$W_LENGTH) is set to **word-integer-length-value** and the address field (DSC$A_POINTER) is set to the string area allocated (first byte beyond the header).

DESCRIPTION

OTS$SGET1_DD allocates a specified number of bytes of dynamic virtual memory to a specified string descriptor. This routine is identical to OTS$SCOPY_DXDX except that no source string is copied. You can write anything you want in the allocated area.

If the specified string descriptor already has dynamic memory allocated to it, but the amount allocated is either greater than or less than **word-integer-length-value**, that space is deallocated before OTS$SGET1_DD allocates new space.

CONDITION VALUES SIGNALED

OTS$_FATINTERR	Fatal internal error.
OTS$_INSVIRMEM	Insufficient virtual memory.

EXAMPLE

```
      PROGRAM STRING_TEST
C+
C     This program demonstrates the use of some dynamic string
C     manipulation routines.
C-

C+
C     DECLARATIONS
C-

      IMPLICIT NONE
      CHARACTER*80    DATA_LINE
      INTEGER*4       DATA_LEN, DSC(2), CRLF_DSC(2), TEMP_DSC(2)
      CHARACTER*2     CRLF

C+
C     Initialize the output descriptor.  It should be empty.
C-

      CALL OTS$SGET1_DD(%VAL(0), DSC)

C+
C     Initialize a descriptor to the string CRLF and copy the
C     character CRLF to it.
C-

      CALL OTS$SGET1_DD(%VAL(2), CRLF_DSC)
      CRLF = CHAR(13)//CHAR(10)
      CALL OTS$SCOPY_R_DX( %VAL(2), %REF(CRLF(1:1)), CRLF_DSC)

C+
C     Initialize a temporary descriptor.
C-

      CALL OTS$SGET1_DD(%VAL(0), TEMP_DSC)

C+
C     Prompt the user.
C-

      WRITE(6, 999)
999   FORMAT(1X, 'Enter your message, end with CTRL/Z.')

C+
C     Read lines of text from the terminal until end-of-file.
C     Concatenate each line to the previous input.  Include a
C     CRLF between each line.
C-

      DO WHILE (.TRUE.)
          READ(5, 998, ERR = 10) DATA_LEN, DATA_LINE
998       FORMAT(Q,A)
          CALL OTS$SCOPY_R_DX( %VAL(DATA_LEN),
     1        %REF(DATA_LINE(1:1)),
     2        TEMP_DSC)
          CALL STR$CONCAT( DSC, DSC, TEMP_DSC, CRLF_DSC )
      END DO
```

```
C+
C       The user has typed CTRL/Z.  Output the data we read.
C-

10      CALL LIB$PUT_OUTPUT( DSC )
C+
C       Free the storage allocated to the dynamic strings.
C-

        CALL OTS$SFREE1_DD( DSC )
        CALL OTS$SFREE1_DD( CRLF_DSC )
        CALL OTS$SFREE1_DD( TEMP_DSC )

C+
C       End of program.
C-

        STOP
        END
```

This FORTRAN example program demonstrates dynamic string manipulation using OTS$SGET1_DD, OTS$SFREE1_DD, and OTS$SCOPY_R_DX.

Index

C

D

E

M

O

R

S

Reader's Comments

VMS RTL General Purpose (OTS$) Manual
AA–LA73A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

____

What I like best about this manual is ______________________________

____

____

What I like least about this manual is ______________________________

____

____

I found the following errors in this manual:

Page Description

______ ______________________________

______ ______________________________

______ ______________________________

______ ______________________________

______ ______________________________

Additional comments or suggestions to improve this manual:

____

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. ______________

Company ______________________________ Date ______________

Mailing Address ______________________________

______________________________ Phone ______________

-- Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here