

---

# OpenVMS RTL General Purpose (OTSS) Manual

Order Number: AA-PV6HD-TK

**April 2001**

This manual documents the general-purpose routines contained in the OTSS facility of the OpenVMS Run-Time Library.

**Revision/Update Information:** This manual supersedes the *OpenVMS RTL General Purpose (OTSS) Manual*, OpenVMS Alpha Version 7.1 and OpenVMS VAX Version 7.1.

**Software Version:** OpenVMS Alpha Version 7.3  
OpenVMS VAX Version 7.3

**Compaq Computer Corporation  
Houston, Texas**

---

© 2001 Compaq Computer Corporation

Compaq, VAX, VMS, and the Compaq logo Registered in U.S. Patent and Trademark Office.

OpenVMS is a trademark of Compaq Information Technologies Group, L.P. in the United States and other countries.

All other product names mentioned herein may be trademarks of their respective companies.

Confidential computer software. Valid license from Compaq required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is provided "as is" without warranty of any kind and is subject to change without notice. The warranties for Compaq products are set forth in the express limited warranty statements accompanying such products. Nothing herein should be construed as constituting an additional warranty.

ZK5933

The Compaq *OpenVMS* documentation set is available on CD-ROM.

This document was prepared using DECdocument, Version 3.3-1b.

---

# Contents

<b>Preface</b> .....	v
----------------------	---

## **Part I OTS\$ Overview**

### **1 Run-Time Library General Purpose (OTS\$) Facility**

1.1	Overview .....	1-1
1.2	Linking OTS\$ Routines on an Alpha System .....	1-4
1.2.1	64-Bit Addressing Support (Alpha Only) .....	1-4

## **Part II OTS\$ Reference Section**

OTS\$CNVOUT .....	OTS-3
OTS\$CVT_L_TB .....	OTS-5
OTS\$CVT_L_TI .....	OTS-7
OTS\$CVT_L_TL .....	OTS-9
OTS\$CVT_L_TO .....	OTS-11
OTS\$CVT_L_TU .....	OTS-13
OTS\$CVT_L_TZ .....	OTS-15
OTS\$CVT_T_x .....	OTS-17
OTS\$CVT_TB_L .....	OTS-22
OTS\$CVT_TI_L .....	OTS-25
OTS\$CVT_TL_L .....	OTS-27
OTS\$CVT_TO_L .....	OTS-29
OTS\$CVT_TU_L .....	OTS-32
OTS\$CVT_TZ_L .....	OTS-34
OTS\$DIVC_x .....	OTS-37
OTS\$DIV_PK_LONG .....	OTS-40
OTS\$DIV_PK_SHORT .....	OTS-44
OTS\$MOVE3 .....	OTS-46
OTS\$MOVE5 .....	OTS-48
OTS\$MULC_x .....	OTS-50
OTS\$POWC_xC_x .....	OTS-52
OTS\$POWC_xJ .....	OTS-55
OTS\$POWDD .....	OTS-57
OTS\$POWDJ .....	OTS-59
OTS\$POWDR .....	OTS-61
OTS\$POWGG .....	OTS-63
OTS\$POWGJ .....	OTS-66

OTSS\$POWHH_R3 (VAX Only) .....	OTS-68
OTSS\$POWHJ_R3 (VAX Only) .....	OTS-70
OTSS\$POWII .....	OTS-72
OTSS\$POWJJ .....	OTS-74
OTSS\$POWLULU .....	OTS-76
OTSS\$POWRD .....	OTS-78
OTSS\$POWRJ .....	OTS-80
OTSS\$POWRR .....	OTS-82
OTSS\$POWxLU .....	OTS-84
OTSS\$SCOPY_DXDX .....	OTS-86
OTSS\$SCOPY_R_DX .....	OTS-88
OTSS\$FREE1_DD .....	OTS-91
OTSS\$FREEM_DD .....	OTS-92
OTSS\$GET1_DD .....	OTS-93

## Index

### Tables

1-1	OTSS Conversion Routines .....	1-2
1-2	OTSS Division Routines .....	1-2
1-3	OTSS Move Data Routines .....	1-2
1-4	OTSS Multiplication Routine .....	1-2
1-5	OTSS Exponentiation Routines .....	1-3
1-6	OTSS Copy Source String Routines .....	1-3
1-7	OTSS Return String Area Routines .....	1-3
1-8	OTSS and Equivalent Math\$ Entry Points .....	1-4

---

# Preface

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

## Intended Audience

This manual is intended for system and application programmers who write programs that call OTSS Run-Time Library routines.

## Document Structure

This manual is organized into two parts as follows:

- Part I contains a brief overview of the OTSS routines in Chapter 1.
- Part II, the OTSS Reference Section, provides detailed reference information on each routine contained in the OTSS facility of the Run-Time Library. This information is presented using the documentation format described in *OpenVMS Programming Concepts Manual*. Routine descriptions appear in alphabetical order by routine name.

## Related Documents

The Run-Time Library routines are documented in a series of reference manuals. A description of how the Run-Time Library routines are accessed and of OpenVMS features and functionality available through calls to the OTSS Run-Time Library appears in the *OpenVMS Programming Concepts Manual*. Descriptions of other RTL facilities and their corresponding routines and usages are discussed in the following books:

- *Compaq Portable Mathematics Library*
- *OpenVMS VAX RTL Mathematics (MTH\$) Manual*
- *OpenVMS RTL DECTalk (DTK\$) Manual*<sup>1</sup>
- *OpenVMS RTL Library (LIB\$) Manual*
- *OpenVMS RTL Parallel Processing (PPL\$) Manual*<sup>1</sup>
- *OpenVMS RTL Screen Management (SMG\$) Manual*
- *OpenVMS RTL String Manipulation (STR\$) Manual*

The *Guide to the POSIX Threads Library* contains guidelines and reference information for Compaq POSIX Threads<sup>2</sup>, the Compaq Multithreading Run-Time Library.

---

<sup>1</sup> This manual has been archived but is available on the OpenVMS Documentation CD-ROM.

<sup>2</sup> Compaq POSIX Threads was formerly called DECthreads.

Application programmers using any programming language can refer to the *Guide to Creating OpenVMS Modular Procedures* for writing modular and reentrant code.

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 OpenVMS language software.

For a complete list and description of the manuals in the OpenVMS documentation set, see the *OpenVMS Version 7.3 New Features and Documentation Overview*.

For additional information about Compaq *OpenVMS* products and services, access the Compaq website at the following location:

<http://www.openvms.compaq.com/>

## Reader's Comments

Compaq welcomes your comments on this manual. Please send comments to either of the following addresses:

Internet	<b>openvmsdoc@compaq.com</b>
Mail	Compaq Computer Corporation OSSG Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

## How To Order Additional Documentation

Use the following World Wide Web address to order additional documentation:

<http://www.openvms.compaq.com/>

If you need help deciding which documentation best meets your needs, call 800-282-6672.

## Conventions

The following conventions are used in this manual:

Ctrl/ <i>x</i>	A sequence such as Ctrl/ <i>x</i> indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 <i>x</i>	A sequence such as PF1 <i>x</i> indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.
<span style="border: 1px solid black; padding: 2px;">Return</span>	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)  In the HTML version of this document, this convention appears as brackets, rather than a box.

...	<p>A horizontal ellipsis in examples indicates one of the following possibilities:</p> <ul style="list-style-type: none"> <li>• Additional optional arguments in a statement have been omitted.</li> <li>• The preceding item or items can be repeated one or more times.</li> <li>• Additional parameters, values, or other information can be entered.</li> </ul>
. . . . . .	<p>A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.</p>
( )	<p>In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.</p>
[ ]	<p>In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.</p>
{ }	<p>In command format descriptions, braces indicate a required choice of options; you must choose one of the options listed. Do not type the braces on the command line.</p>
<b>bold text</b>	<p>This typeface represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.</p>
<i>italic text</i>	<p>Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error <i>number</i>), in command lines (<i>PRODUCER=name</i>), and in command parameters in text (where <i>dd</i> represents the predefined code for the device type).</p>
UPPERCASE TEXT	<p>Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.</p>
Monospace text	<p>Monospace type indicates code examples and interactive screen displays.</p> <p>In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.</p>
-	<p>A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.</p>
numbers	<p>All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radices—binary, octal, or hexadecimal—are explicitly indicated.</p>



# Part I

---

## OTS\$ Overview

This part of the *OpenVMS RTL General Purpose (OTS\$) Manual* contains a general overview of the routines provided by the OpenVMS RTL General Purpose (OTS\$) Facility, and lists them by function.



---

# Run-Time Library General Purpose (OTSS) Facility

This chapter describes the OpenVMS Run-Time Library General Purpose (OTSS) facility. See the OTSS Reference Section for a detailed description of each routine within the OTSS facility.

Most of the OTSS routines were originally designed to support language compilers. Because they perform general-purpose functions, the routines were moved into the language-independent facility, OTSS.

## 1.1 Overview

The Run-Time Library General Purpose (OTSS) facility provides routines to perform general-purpose functions. These functions include data type conversions as part of a compiler's generated code, and some mathematical functions.

The OTSS facility contains routines to perform the following main tasks:

- Convert data types (see Table 1-1)
- Divide complex and packed decimal values (see Table 1-2)
- Move data to a specified destination address (see Table 1-3)
- Multiply complex values (see Table 1-4)
- Raise a base to an exponent (see Table 1-5)
- Copy a source string to a destination string (see Table 1-6)
- Return a string area to free storage (see Table 1-7)

Some restrictions apply if you link certain OTSS routines on an Alpha system. See Section 1.2 for more information about these restrictions.

# Run-Time Library General Purpose (OTS\$) Facility

## 1.1 Overview

**Table 1–1 OTS\$ Conversion Routines**

Routine Name	Function
OTSSCNVOUT	Convert a D-floating, G-floating, or H-floating value to a character string.
OTSSCVT_L_TB	Convert an unsigned integer to binary text.
OTSSCVT_L_TI	Convert a signed integer to signed integer text.
OTSSCVT_L_TL	Convert an integer to logical text.
OTSSCVT_L_TO	Convert an unsigned integer to octal text.
OTSSCVT_L_TU	Convert an unsigned integer to decimal text.
OTSSCVT_L_TZ	Convert an integer to hexadecimal text.
OTSSCVT_TB_L	Convert binary text to an unsigned integer value.
OTSSCVT_TI_L	Convert signed integer text to an integer value.
OTSSCVT_TL_L	Convert logical text to an integer value.
OTSSCVT_TO_L	Convert octal text to an unsigned integer value.
OTSSCVT_TU_L	Convert unsigned decimal text to an integer value.
OTSSCVT_T_x	Convert numeric text to a D-, F-, G-, or H-floating value.
OTSSCVT_TZ_L	Convert hexadecimal text to an unsigned integer value.

For more information on Run-Time Library conversion routines, see the CVT\$ reference section in the *OpenVMS RTL Library (LIB\$) Manual*.

**Table 1–2 OTS\$ Division Routines**

Routine Name	Function
OTSSDIVC <sub>x</sub>	Perform complex division.
OTSSDIV_PK_LONG	Perform packed decimal division with a long divisor.
OTSSDIV_PK_SHORT	Perform packed decimal division with a short divisor.

**Table 1–3 OTS\$ Move Data Routines**

Routine Name	Function
OTSSMOVE3	Move data without fill.
OTSSMOVE5	Move data with fill.

**Table 1–4 OTS\$ Multiplication Routine**

Routine Name	Function
OTSSMULC <sub>x</sub>	Perform complex multiplication.

## Run-Time Library General Purpose (OTSS\$) Facility 1.1 Overview

**Table 1–5 OTS\$ Exponentiation Routines**

Routine Name	Function
OTSSPOWCxCx	Raise a complex base to a complex floating-point exponent.
OTSSPOWCxJ	Raise a complex base to a signed longword exponent.
OTSSPOWDD	Raise a D-floating base to a D-floating exponent.
OTSSPOWDR	Raise a D-floating base to an F-floating exponent.
OTSSPOWDJ	Raise a D-floating base to a longword integer exponent.
OTSSPOWGG	Raise a G-floating base to a G-floating or longword integer exponent.
OTSSPOWGJ	Raise a G-floating base to a longword integer exponent.
†OTSSPOWHH_R3	Raise an H-floating base to an H-floating exponent.
†OTSSPOWHJ_R3	Raise an H-floating base to a longword integer exponent.
OTSSPOWII	Raise a word integer base to a word integer exponent.
OTSSPOWJJ	Raise a longword integer base to a longword integer exponent.
OTSSPOWLULU	Raise an unsigned longword integer base to an unsigned longword integer exponent.
OTSSPOWxLU	Raise a floating-point base to an unsigned longword integer exponent.
OTSSPOWRD	Raise an F-floating base to a D-floating exponent.
OTSSPOWRJ	Raise an F-floating base to a longword integer exponent.
OTSSPOWRR	Raise an F-floating base to an F-floating exponent.
†VAX specific.	

**Table 1–6 OTS\$ Copy Source String Routines**

Routine Name	Function
OTSSCOPY_DXDX	Copy a source string passed by descriptor to a destination string.
OTSSCOPY_R_DX	Copy a source string passed by reference to a destination string.

**Table 1–7 OTS\$ Return String Area Routines**

Routine Name	Function
OTSSFREE1_DD	Free one dynamic string.
OTSSFREEM_DD	Free <i>n</i> dynamic strings.
OTSSGET1_DD	Get one dynamic string.

# Run-Time Library General Purpose (OTSS) Facility

## 1.2 Linking OTSS Routines on an Alpha System

### 1.2 Linking OTSS Routines on an Alpha System

On Alpha systems, if you use the OTSS entry points for certain mathematics routines, you must link against the DPML\$SHR.EXE library. Alternately, you can use the equivalent math\$ entry point for the routine and link against either STARLET.OBJ or the DPML\$SHR.EXE library. Math\$ entry points are available only on Alpha systems.

Table 1–8 lists the affected OTSS entry points with their equivalent math\$ entry points. Refer to the *Compaq Portable Mathematics Library* for information about the math\$ entry points.

**Table 1–8 OTSS and Equivalent Math\$ Entry Points**

OTSS Entry Point	Math\$ Entry Point
OTSSDIVC	math\$cdiv_f
OTSSDIVCG_R3	math\$cdiv_g
OTSSMULCG_R3	math\$cmul_g
OTSSPOWCC	math\$cpow_f
OTSSPOWCGCG_R3	math\$cpow_g
OTSSPOWCJ	math\$cpow_fq
OTSSPOWGG	math\$pow_gg
OTSSPOWGJ	math\$pow_gq
OTSSPOWGLU	math\$pow_gq
OTSSPOWII	math\$pow_qq
OTSSPOWJJ	math\$pow_qq
OTSSPOWLULU	math\$pow_qq
OTSSPOWRJ	math\$pow_fq
OTSSPOWRLU	math\$pow_fq
OTSSPOWRR	math\$pow_ff

#### 1.2.1 64-Bit Addressing Support (Alpha Only)

On Alpha systems, the General Purpose (OTSS) routines provide 64-bit virtual addressing capabilities as follows:

- All OTSS RTL routines accept 64-bit addresses for arguments passed by reference.
- All OTSS RTL routines also accept either 32-bit or 64-bit descriptors for arguments passed by descriptor.

**Note**

The OTSS routines declared in `ots$routines.h` do not include prototypes for 64-bit data. You must provide your own generic prototypes for any OTSS functions you use.

See the *OpenVMS Programming Concepts Manual* for more information about 64-bit virtual addressing capabilities.

# Part II

---

## OTSS\$ Reference Section

This section provides detailed descriptions of the routines provided by the OpenVMS RTL General Purpose (OTSS) Facility.



---

## OTSS\$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

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

OTSS\$CNVOUT\_G D-G-or-H-float-pt-input-val ,fixed-length-resultant-string  
,digits-in-fraction

OTSS\$CNVOUT\_H D-G-or-H-float-pt-input-val ,fixed-length-resultant-string  
,digits-in-fraction (VAX only)

#### Returns

OpenVMS usage: cond\_value  
type: longword (unsigned)  
access: write only  
mechanism: by value

#### Arguments

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

OpenVMS usage: floating\_point  
type: D\_floating, G\_floating, H\_floating  
access: read only  
mechanism: by reference

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

OpenVMS usage: char\_string  
type: character string  
access: write only  
mechanism: by descriptor, fixed length

Output string into which OTSS\$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.

## OTSS\$ Routines

### OTSS\$CNVOUT

#### **digits-in-fraction**

OpenVMS 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.
OTSS\$_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks (*).

---

## OTSS\$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

OTSS\$CVT\_L\_TB varying-input-value, fixed-length-resultant-string [, number-of-digits]  
 [, input-value-size]

#### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

#### Arguments

##### varying-input-value

OpenVMS usage: varying\_arg  
 type: unspecified  
 access: read only  
 mechanism: by reference

Unsigned byte, word, or longword that OTSS\$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

OpenVMS usage: char\_string  
 type: character string  
 access: write only  
 mechanism: by descriptor, fixed length

ASCII text string that OTSS\$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 (CLASS\_S descriptor).

##### number-of-digits

OpenVMS 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. 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

## OTSS\$ Routines

### OTSS\$CVT\_L\_TB

value of the integer to be converted is also zero, OTSS\$CVT\_L\_TB creates a blank string.

#### **input-value-size**

OpenVMS 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.
OTSS_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks (*).

---

## OTSS\$CVT\_L\_TI

### Convert Signed Integer to Decimal Text

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

#### Format

OTSS\$CVT\_L\_TI varying-input-value ,fixed-length-resultant-string [,number-of-digits]  
 [,input-value-size] [,flags-value]

#### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

#### Arguments

##### varying-input-value

OpenVMS usage: varying\_arg  
 type: unspecified  
 access: read only  
 mechanism: by reference, fixed length

A signed integer that OTSS\$CVT\_L\_TI converts to a signed decimal representation in an ASCII text string. The **varying-input-value** argument is the address of the signed integer.

On VAX systems, the integer can be a signed byte, word, or longword. The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, or longword.

On Alpha systems, the integer can be a signed byte, word, longword, or quadword. The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, longword, or quadword.

##### fixed-length-resultant-string

OpenVMS usage: char\_string  
 type: character string  
 access: write only  
 mechanism: by descriptor

Decimal ASCII text string that OTSS\$CVT\_L\_TI creates when it converts the signed integer. The **fixed-length-resultant-string** argument is the address of a CLASS\_S descriptor pointing to this text string. The string is assumed to be of fixed length.

##### number-of-digits

OpenVMS usage: longword\_signed  
 type: longword (signed)  
 access: read only  
 mechanism: by value

## OTSS\$ Routines

### OTSS\$CVT\_L\_TI

Minimum number of digits to be generated when OTSS\$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. If the minimum number of digits is omitted, the default value is 1. If the actual number of significant digits is smaller, OTSS\$CVT\_L\_TI inserts leading zeros into the output string. If **number-of-digits** is zero and **varying-input-value** is zero, OTSS\$CVT\_L\_TI writes a blank string to the output string.

#### input-value-size

OpenVMS 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. If the size is omitted, the default is 4 (longword).

On VAX systems, the value size must be 1, 2, or 4. If value size is 1 or 2, the value is sign-extended to a longword before conversion.

On Alpha systems, the value size must be 1, 2, 4, or 8. If the value is 1, 2, or 4, the value is sign-extended to a quadword before conversion.

#### flags-value

OpenVMS usage: mask\_longword  
type: longword (unsigned)  
access: read only  
mechanism: by value

Caller-supplied flags that you can use if you want OTSS\$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 described in the following table:

Bit	Action if Set	Action if Clear
0	Insert a plus sign (+) before the first nonblank character in the output string.	Omit the plus sign.

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

## Condition Values Returned

SS\$NORMAL	Normal successful completion.
OTSS\$OUTCONERR	Output conversion error. Either the result would have exceeded the fixed-length string or the <b>input-value-size</b> is not a valid value. The output string is filled with asterisks (*).

---

## OTSS\$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

OTSS\$CVT\_L\_TL longword-integer-value ,fixed-length-resultant-string

#### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

#### Arguments

##### longword-integer-value

OpenVMS usage: longword\_signed  
 type: longword (signed)  
 access: read only  
 mechanism: by reference

Value that OTSS\$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

OpenVMS usage: char\_string  
 type: character string  
 access: write only  
 mechanism: by descriptor, fixed length

Output string that OTSS\$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 (CLASS\_S descriptor).

If bit 0 of **longword-integer-value** is set, OTSS\$CVT\_L\_TL stores the character T in the rightmost character of **fixed-length-resultant-string**. If bit 0 is clear, it stores the character F. In either case, it fills the remaining characters of **fixed-length-resultant-string** with blanks.

#### Condition Values Returned

SS\$NORMAL	Normal successful completion.
OTSS_OUTCONERR	Output conversion error. The result would have exceeded the fixed-length string; the output string is of zero length (descriptor LENGTH field contains 0).

## OT\$ Routines

### OT\$CVT\_L\_TL

#### Example

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

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

This BASIC example illustrates the use of OT\$CVT\_L\_TL. The output generated by this program is 'F'.

---

## OTSS\$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. OTSS\$CVT\_L\_TO supports Fortran Ow and Ow.m output conversion formats.

#### Format

OTSS\$CVT\_L\_TO varying-input-value ,fixed-length-resultant-string [,number-of-digits]  
 [,input-value-size]

#### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

#### Arguments

##### varying-input-value

OpenVMS usage: varying\_arg  
 type: unspecified  
 access: read only  
 mechanism: by reference

Unsigned byte, word, or longword that OTSS\$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

OpenVMS usage: char\_string  
 type: character string  
 access: write only  
 mechanism: by descriptor, fixed length

Output string that OTSS\$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 (CLASS\_S descriptor).

##### number-of-digits

OpenVMS usage: longword\_signed  
 type: longword (signed)  
 access: read only  
 mechanism: by value

Minimum number of digits that OTSS\$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. 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, OTSS\$CVT\_L\_TO inserts

## OTSS\$ Routines

### OTSS\$CVT\_L\_TO

leading zeros into the output string. If **number-of-digits** is 0 and **varying-input-value** is 0, OTSS\$CVT\_L\_TO writes a blank string to the output string.

#### input-value-size

OpenVMS 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 OTSS\$CVT\_L\_TO. If it is omitted, the default is 4 (longword).

### Condition Values Returned

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

---

## OTSS\$CVT\_L\_TU

### Convert Unsigned Integer to Decimal Text

The Convert Unsigned Integer to Decimal Text routine converts an unsigned integer value to its unsigned decimal representation in an ASCII text string.

#### Format

OTSS\$CVT\_L\_TU varying-input-value ,fixed-length-resultant-string [,number-of-digits]  
 [,input-value-size]

#### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

#### Arguments

##### varying-input-value

OpenVMS usage: varying\_arg  
 type: unspecified  
 access: read only  
 mechanism: by reference

An unsigned integer that OTSS\$CVT\_L\_TU converts to an unsigned decimal representation in an ASCII text string. The **varying-input-value** argument is the address of the unsigned integer.

On VAX systems, the integer can be an unsigned byte, word, or longword. (The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, or longword.)

On Alpha systems, the integer can be an unsigned byte, word, longword, or quadword. (The value of the **input-value-size** argument determines whether **varying-input-value** is a byte, word, longword, or quadword.)

##### fixed-length-resultant-string

OpenVMS usage: char\_string  
 type: character string  
 access: write only  
 mechanism: by descriptor, fixed length

Output string that OTSS\$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

OpenVMS usage: longword\_signed  
 type: longword (signed)  
 access: read only  
 mechanism: by value

## OTSS\$ Routines

### OTSS\$CVT\_L\_TU

Minimum number of digits in the ASCII text string that OTSS\$CVT\_L\_TU creates. The **number-of-digits** argument is a signed longword containing the minimum number. 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, OTSS\$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, OTSS\$CVT\_L\_TU writes a blank string to the output string.

#### input-value-size

OpenVMS 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. If the size is omitted, the default is 4 (longword).

On VAX systems, the value size must be 1, 2, or 4.

On Alpha systems, the value size must be 1, 2, 4, or 8.

## Condition Values Returned

SS\$NORMAL	Normal successful completion.
OTSS_OUTCONERR	Output conversion error. Either the result would have exceeded the fixed-length string or the <b>input-value-size</b> is not a valid value. The output string is filled with asterisks (*).

---

## OTSS\$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. OTSS\$CVT\_L\_TZ supports Fortran Zw and Zw.m output conversion formats.

#### Format

OTSS\$CVT\_L\_TZ varying-input-value ,fixed-length-resultant-string [,number-of-digits]  
 [,input-value-size]

#### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

#### Arguments

##### varying-input-value

OpenVMS usage: varying\_arg  
 type: unspecified  
 access: read only  
 mechanism: by reference

Unsigned byte, word, or longword that OTSS\$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

OpenVMS usage: char\_string  
 type: character string  
 access: write only  
 mechanism: by descriptor, fixed length

Output string that OTSS\$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 (CLASS\_S descriptor).

##### number-of-digits

OpenVMS usage: longword\_signed  
 type: longword (signed)  
 access: read only  
 mechanism: by value

Minimum number of digits in the ASCII text string that OTSS\$CVT\_L\_TZ creates when it converts the integer. The **number-of-digits** argument is a signed longword containing this minimum number. If it is omitted, the default is 1. If the actual number of significant digits in the text string that OTSS\$CVT\_L\_TZ creates is less than this minimum number, OTSS\$CVT\_L\_TZ inserts leading zeros in the output string. If the minimum number of digits is zero and the integer

## OTSS\$ Routines

### OTSS\$CVT\_L\_TZ

value to be converted is also zero, OTSS\$CVT\_L\_TZ writes a blank string to the output string.

#### input-value-size

OpenVMS usage: longword\_signed  
type: longword (signed)  
access: read only  
mechanism: by value

Size of the integer that OTSS\$CVT\_L\_TZ converts, in bytes. The **input-value-size** argument is a signed longword containing the value size. If the size is omitted, the default is 4 (longword).

### Condition Values Returned

SS\$NORMAL	Normal successful completion.
OTSS\$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 => "OTSS$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 OTSS\$CVT\_L\_TZ to convert a longword integer to hexadecimal text.

## OTSS\$CVT\_T\_x Convert Numeric Text to D-, F-, G-, or H-Floating Value

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

### Format

OTSS\$CVT\_T\_D fixed-or-dynamic-input-string ,floating-point-value [,digits-in-fraction] [,scale-factor] [,flags-value] [,extension-bits]  
 OTSS\$CVT\_T\_F fixed-or-dynamic-input-string ,floating-point-value [,digits-in-fraction] [,scale-factor] [,flags-value] [,extension-bits]  
 OTSS\$CVT\_T\_G fixed-or-dynamic-input-string ,floating-point-value [,digits-in-fraction] [,scale-factor] [,flags-value] [,extension-bits]  
 OTSS\$CVT\_T\_H fixed-or-dynamic-input-string ,floating-point-value [,digits-in-fraction] [,scale-factor] [,flags-value] [,extension-bits]

### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

### Arguments

**fixed-or-dynamic-input-string**  
 OpenVMS 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 OTSS\$CVT\_T\_x converts to a D-floating, F-floating, 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:

$$\left[ \left[ \begin{matrix} + \\ - \end{matrix} \right] \left[ \text{<digits>} \right] \left[ \cdot \right] \left[ \text{<digits>} \right] \left\{ \begin{matrix} E \\ e \\ D \\ d \\ Q \\ q \end{matrix} \right\} \left[ \text{<blanks>} \right] \left[ \begin{matrix} + \\ - \end{matrix} \right] \right] \left[ \text{<digits>} \right]$$

VM-0710A-A1

## OTSS\$ Routines

### OTSS\$CVT\_T\_x

E, e, D, d, Q, and q are the possible exponent letters. They are semantically equivalent. Other elements in the preceding syntax are defined as follows:

Term	Description
blanks	One or more blanks
digits	One or more decimal digits

#### floating-point-value

OpenVMS usage: floating\_point  
type: D\_floating, F\_floating, G\_floating, H\_floating  
access: write only  
mechanism: by reference

Floating-point value that OTSS\$CVT\_T\_x creates when it converts the input string. The **floating-point-value** argument is the address of the floating-point value. The data type of **floating-point-value** depends on the called routine as shown in the following table:

Routine	floating-point-value Data Type
OTSS\$CVT_T_D	D-floating
OTSS\$CVT_T_F	F-floating
OTSS\$CVT_T_G	G-floating
OTSS\$CVT_T_H	H-floating

#### digits-in-fraction

OpenVMS 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. If the number of digits is omitted, the default is zero.

#### scale-factor

OpenVMS 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. If the scale factor is omitted, the default is zero.

**flags-value**

OpenVMS usage: mask\_longword  
 type: longword (unsigned)  
 access: read only  
 mechanism: by value

User-supplied flags. The **flags-value** argument contains the user-supplied flags described in the following table:

Bit	Action if Set	Action if Clear
0	Ignore blanks.	Interpret blanks as zeros.
1	Allow only E or e exponents. (This is consistent with Fortran semantics.)	Allow E, e, D, d, Q and q exponents. (This is consistent with BASIC semantics.)
2	Interpret an underflow as an error.	Do not interpret an underflow as an error.
3	Truncate the value.	Round the value.
4	Ignore tabs.	Interpret tabs as invalid characters.
5	An exponent must begin with a valid exponent letter.	The exponent letter can be omitted.
6	Always apply the scale factor.	Apply the scale factor only if there is no exponent present in the string.

If you omit the **flags-value** argument, OTSS\$CVT\_T\_x defaults all flags to clear.

**extension-bits (D-, F-floating)**

OpenVMS usage: byte\_unsigned  
 type: byte (unsigned)  
 access: write only  
 mechanism: by reference

**extension-bits (G-, H-floating)**

OpenVMS usage: word\_unsigned  
 type: word (unsigned)  
 access: write only  
 mechanism: by reference

Extra precision bits. The **extension-bits** argument is the address of a word 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 left-justified in this argument, as follows:

Routine	Number of Bits Returned	Data Type
OTSS\$CVT_T_D	8	Byte (unsigned)
OTSS\$CVT_T_F	8	Byte (unsigned)
OTSS\$CVT_T_G	11	Word (unsigned)
OTSS\$CVT_T_H	15	Word (unsigned)

A value represented by extension bits is suitable for use as the extension operand in an EMOD instruction.

## OTS\$ Routines

### OTS\$CVT\_T\_x

The extra precision bits returned for H-floating may not be precise because OTS\$CVT\_T\_H carries its calculations to only 128 bits. However the error should be small.

### Description

The OTS\$CVT\_T\_D, OTS\$CVT\_T\_F, OTS\$CVT\_T\_G, and OTS\$CVT\_T\_H routines support Fortran D, E, F, and G input type conversion as well as similar types for other languages.

These routines provide run-time support for BASIC and Fortran input statements.

Although Alpha systems do not generally support H-floating operations, you can use OTS\$CVT\_T\_H to convert a text string to an H-floating value on an Alpha system.

### 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. The <b>floating-point-value</b> and <b>extension-bits</b> arguments, if present, are 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$CVT_T_F(T(I),A,%VAL(5),)
      TYPE *, ' Output of OTS$CVT_T_F is ',A
      TYPE *, ' '
2      CONTINUE
      END
```

**This Fortran example demonstrates the use of OTSS\$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\_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. The integer value can be of arbitrary length but is typically a byte, word, longword, or quadword. The default size of the result is a longword.

#### Format

OTS\$CVT\_TB\_L fixed-or-dynamic-input-string ,varying-output-value  
[,output-value-size] [,flags-value]

#### Returns

OpenVMS usage: cond\_value  
type: longword (unsigned)  
access: write only  
mechanism: by value

#### Arguments

##### fixed-or-dynamic-input-string

OpenVMS 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 unsigned integer value. 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 and 1. No sign is permitted.

##### varying-output-value

OpenVMS usage: varying\_arg  
type: unspecified  
access: write only  
mechanism: by reference

Unsigned integer of specified size that OTS\$CVT\_TB\_L creates when it converts the ASCII text string. The **varying-output-value** argument is the address of the integer. The value of the **output-value-size** argument determines the size in bytes of the output value.

##### output-value-size

OpenVMS usage: longword\_signed  
type: longword (signed)  
access: read only  
mechanism: by value

Arbitrary number of bytes to be occupied by the unsigned integer output value. The **output-value-size** argument contains a value that equals the size in bytes of the output value. If the value of **output-value-size** is zero or a negative number, OTS\$CVT\_TB\_L returns an input conversion error. If you omit the **output-value-size** argument, the default is 4 (longword).

**flags-value**

OpenVMS usage: mask\_longword  
 type: longword (unsigned)  
 access: read only  
 mechanism: by value

User-supplied flag that OTSS\$CVT\_TB\_L uses to determine how to interpret blanks within the input string. The **flags-value** argument contains this user-supplied flag.

OTSS\$CVT\_TB\_L defines the flag as follows:

Bit	Action if Set	Action if Clear
0	Ignore blanks.	Interpret blanks as zeros.

If you omit the **flags-value** argument, OTSS\$CVT\_TB\_L defaults all flags to clear.

**Condition Values Returned**

SS\$NORMAL	Normal successful completion.
OTSS\$INPCONERR	Input conversion error. OTSS\$CVT_TB_L encountered an invalid character in the <b>fixed-or-dynamic-input-string</b> , an overflow of <b>varying-output-value</b> , or an invalid <b>output-value-size</b> . In the case of an invalid character or of an overflow, <b>varying-output-value</b> is set to zero.

**Example**

```

OPTION                                     &
      TYPE = EXPLICIT

!+
!  This program demonstrates the use of OTSS$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 OTSS$CVT_TB_L (STRING, LONG, &
      LONG BY VALUE, LONG BY VALUE)

```

## OTSS\$ Routines OTSS\$CVT\_TB\_L

```
!+
!  MAIN PROGRAM
!-

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

FOR I = 1 TO 5
  READ BIN_STR
  RET_STATUS = OTSS$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 OTSS\$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
```

---

## OTSS\$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 signed decimal number to a signed integer value. The default size of the result is a longword.

#### Format

OTSS\$CVT\_TI\_L fixed-or-dynamic-input-string ,varying-output-value  
 [,output-value-size] [,flags-value]

#### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

#### Arguments

##### fixed-or-dynamic-input-string

OpenVMS usage: char\_string  
 type: character string  
 access: read only  
 mechanism: by descriptor, fixed-length or dynamic string

Input ASCII text string that OTSS\$CVT\_TI\_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 syntax of a valid ASCII text input string is as follows:

[ + <integer-digits>  
 - ]

OTSS\$CVT\_TI\_L always ignores leading blanks.

##### varying-output-value

OpenVMS usage: varying\_arg  
 type: unspecified  
 access: write only  
 mechanism: by reference

Signed integer that OTSS\$CVT\_TI\_L creates when it converts the ASCII text string. The **varying-output-value** argument is the address of the signed integer. The value of the **output-value-size** argument determines the size of **varying-output-value**.

##### output-value-size

OpenVMS usage: longword\_signed  
 type: longword (signed)  
 access: read only  
 mechanism: by value

## OTSS\$ Routines

### OTSS\$CVT\_TI\_L

Number of bytes to be occupied by the value created when OTSS\$CVT\_TI\_L converts the ASCII text string to an integer value. The **output-value-size** argument contains the number of bytes in **varying-output-value**.

On VAX systems, valid values for the **output-value-size** argument are 1, 2, and 4. The value determines whether the integer value that OTSS\$CVT\_TI\_L creates is a byte, word, or longword.

On Alpha systems, valid values for the **output-value-size** argument are 1, 2, 4, and 8. The value determines whether the integer value that OTSS\$CVT\_TI\_L creates is a byte, word, longword, or quadword.

For VAX and Alpha systems, if you specify a 0 (zero) or omit the **output-value-size** argument, the size of the output value defaults to 4 (longword). If you specify any other value, OTSS\$CVT\_TI\_L returns an input conversion error.

#### flags-value

OpenVMS usage: mask\_longword  
type: longword (unsigned)  
access: read only  
mechanism: by value

User-supplied flags that OTSS\$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	Action if Set	Action if Clear
0	Ignore all blanks.	Ignore leading blanks but interpret blanks after the first legal character as zeros.
4	Ignore tabs.	Interpret tabs as invalid characters.

If you omit the **flags-value** argument, OTSS\$CVT\_TI\_L defaults all flags to clear.

### Condition Values Returned

SS\$NORMAL	Normal successful completion.
OTSS\$INPCONERR	Input conversion error. OTSS\$CVT_TI_L encountered an invalid character in the <b>fixed-or-dynamic-input-string</b> , an overflow of <b>varying-output-value</b> , or an invalid <b>output-value-size</b> . In the case of an invalid character or of an overflow, <b>varying-output-value</b> is set to zero.

## OTSS\$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 signed integer.

### Format

OTSS\$CVT\_TL\_L fixed-or-dynamic-input-string ,varying-output-value  
 [,output-value-size]

### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

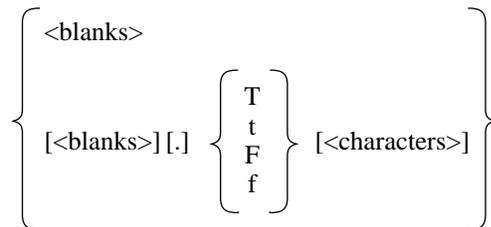
### Arguments

#### fixed-or-dynamic-input-string

OpenVMS 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 OTSS\$CVT\_TL\_L converts to a signed integer value. The **fixed-or-dynamic-input-string** argument is the address of a descriptor pointing to the input string.

Common ASCII text representations of a FORTRAN-77 logical are .TRUE., .FALSE., T, t, F, and f. In practice, an OTSS\$CVT\_TL\_L input string is valid if it adheres to the following syntax:



VM-0711A-AI

One of the letters T, t, F, or f is required. Other elements in the preceding syntax are defined as follows:

Term	Description
blanks	One or more blanks
characters	One or more of any character

## OTSS\$ Routines

### OTSS\$CVT\_TL\_L

#### **varying-output-value**

OpenVMS usage: varying\_arg  
type: unspecified  
access: write only  
mechanism: by reference

Signed integer that OTSS\$CVT\_TL\_L creates when it converts the ASCII text string. The **varying-output-value** argument is the address of the signed integer. The value of the **output-value-size** argument determines the size in bytes of the signed integer.

OTSS\$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, OTSS\$CVT\_TL\_L sets **varying-output-value** to zero.

#### **output-value-size**

OpenVMS usage: longword\_signed  
type: longword (signed)  
access: read only  
mechanism: by value

Number of bytes to be occupied by the signed integer created when OTSS\$CVT\_TL\_L converts the ASCII text string to an integer value. The **output-value-size** argument contains a value that equals the size in bytes of the output value. If **output-value-size** contains a zero or a negative number, OTSS\$CVT\_TL\_L returns an input conversion error.

On VAX systems, valid values for the **output-value-size** argument are 1, 2, and 4. The value determines whether the integer value that OTSS\$CVT\_TL\_L creates is a byte, word, or longword.

On Alpha systems, valid values for the **output-value-size** argument are 1, 2, 4, and 8. This value determines whether the integer value that OTSS\$CVT\_TL\_L creates is a byte, word, longword, or quadword.

For VAX and Alpha systems, if you omit the **output-value-size** argument, the default is 4 (longword).

## Condition Values Returned

SS\$NORMAL	Normal successful completion.
OTSS\$INPCONERR	Input conversion error. OTSS\$CVT_TL_L encountered an invalid character in the <b>fixed-or-dynamic-input-string</b> or an invalid <b>output-value-size</b> . In the case of an invalid character <b>varying-output-value</b> is set to zero.

---

## OTSS\$CVT\_TO\_L

### Convert Octal Text to Unsigned Integer

The Convert Octal Text to Unsigned Integer routine converts an ASCII text string representation of an unsigned octal value to an unsigned integer. The integer value can be of arbitrary length but is typically a byte, word, longword, or quadword. The default size of the result is a longword.

#### Format

OTSS\$CVT\_TO\_L fixed-or-dynamic-input-string ,varying-output-value  
 [,output-value-size] [,flags-value]

#### Returns

OpenVMS usage: cond\_value  
 type: longword (unsigned)  
 access: write only  
 mechanism: by value

#### Arguments

##### fixed-or-dynamic-input-string

OpenVMS usage: char\_string  
 type: character string  
 access: read only  
 mechanism: by descriptor, fixed-length or dynamic string

Input string containing the string representation of an unsigned octal value that OTSS\$CVT\_TO\_L converts to an unsigned 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

OpenVMS usage: varying\_arg  
 type: unspecified  
 access: write only  
 mechanism: by reference

Unsigned integer of specified size that OTSS\$CVT\_TO\_L creates when it converts the ASCII text string. The **varying-output-value** argument is the address of the unsigned integer. The value of the **output-value-size** argument determines the size in bytes of the output value.

##### output-value-size

OpenVMS usage: longword\_signed  
 type: longword integer (signed)  
 access: read only  
 mechanism: by value

Arbitrary number of bytes to be occupied by the unsigned integer output value. The **output-value-size** argument contains a value that equals the size in bytes of the output value. If the value of **output-value-size** is zero or a negative number, OTSS\$CVT\_TO\_L returns an input conversion error. If you omit the **output-value-size** argument, the default is 4 (longword).

## OTS\$ Routines

### OTS\$CVT\_TO\_L

#### flags-value

OpenVMS usage: mask\_longword  
type: longword (unsigned)  
access: read only  
mechanism: by value

User-supplied flag that OTS\$CVT\_TO\_L uses to determine how to interpret blanks within the input string. The **flags-value** argument contains the user-supplied flag described in the following table:

Bit	Action if Set	Action if Clear
0	Ignore all blanks.	Interpret blanks as zeros.

If you omit the **flags-value** argument, OTS\$CVT\_TO\_L defaults the flag to clear.

### Condition Values Returned

SS\$NORMAL	Normal successful completion.
OTS\$INPCONERR	Input conversion error. OTS\$CVT_TO_L encountered an invalid character in the <b>fixed-or-dynamic-input-string</b> , an overflow of <b>varying-output-value</b> , or an invalid <b>output-value-size</b> . In the case of an invalid character or of an overflow, <b>varying-output-value</b> is set to zero.

### 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; press Ctrl/Z to quit.

```
$ RUN OCTAL  
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 integer value. By default, the size of the result is a longword.

#### Format

OTS\$CVT\_TU\_L fixed-or-dynamic-input-string ,varying-output-value  
[,output-value-size] [,flags-value]

#### Returns

OpenVMS usage: cond\_value  
type: longword (unsigned)  
access: write only  
mechanism: by value

#### Arguments

##### **fixed-or-dynamic-input-string**

OpenVMS usage: char\_string  
type: character string  
access: read only  
mechanism: by descriptor

Input string containing an ASCII text string representation of an unsigned decimal value that OTS\$CVT\_TU\_L converts to an unsigned integer value. The **fixed-or-dynamic-input-string** argument is the address of a descriptor pointing to the input string. Valid input characters are the space and the digits 0 through 9. No sign is permitted.

##### **varying-output-value**

OpenVMS usage: varying\_arg  
type: unspecified  
access: write only  
mechanism: by reference

Unsigned integer that OTS\$CVT\_TU\_L creates when it converts the ASCII text string. The **varying-output-value** argument is the address of the unsigned integer. The value of the **output-value-size** argument determines the size of **varying-output-value**.

##### **output-value-size**

OpenVMS 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 in **varying-output-value**.

On VAX systems, valid values for the **output-value-size** argument are 1, 2, and 4. The value determines whether the integer value that OTSS\$CVT\_TU\_L creates is a byte, word, or longword.

On Alpha systems, valid values for the **output-value-size** argument are 1, 2, 4, and 8. The value determines whether the integer value that OTSS\$CVT\_TU\_L creates is a byte, word, longword, or quadword.

For VAX and Alpha systems, if you specify a 0 (zero) or omit the **output-value-size** argument, the size of the output value defaults to 4 (longword). If you specify any other value, OTSS\$CVT\_TU\_L returns an input conversion error.

**flags-value**

OpenVMS usage: mask\_longword  
 type: longword (unsigned)  
 access: read only  
 mechanism: by value

User-supplied flags that OTSS\$CVT\_TU\_L uses to determine how blanks and tabs are interpreted. The **flags-value** argument contains the user-supplied flags as described in the following table:

Bit	Action if Set	Action if Clear
0	Ignore all blanks.	Ignore leading blanks but interpret blanks after the first legal character as zeros.
4	Ignore tabs.	Interpret tabs as invalid characters.

If you omit the **flags-value** argument, OTSS\$CVT\_TU\_L defaults all flags to clear.

**Condition Values Returned**

SS\$NORMAL	Normal successful completion.
OTSS\$INPCONERR	Input conversion error. OTSS\$CVT_TU_L encountered an invalid character in the <b>fixed-or-dynamic-input-string</b> , overflow of <b>varying-output-value</b> , or an invalid <b>output-value-size</b> . In the case of an invalid character or of an overflow, <b>varying-output-value</b> is set to zero.

---

## 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. The integer value can be of arbitrary length but is typically a byte, word, longword, or quadword. The default size of the result is a longword.

#### Format

OTS\$CVT\_TZ\_L fixed-or-dynamic-input-string ,varying-output-value  
[,output-value-size] [,flags-value]

#### Returns

OpenVMS usage: cond\_value  
type: longword (unsigned)  
access: write only  
mechanism: by value

#### Arguments

##### fixed-or-dynamic-input-string

OpenVMS usage: char\_string  
type: character string  
access: read only  
mechanism: by descriptor, fixed-length or dynamic string

Input string containing the 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. The valid input characters are blanks, the digits 0 through 7, and the letters A through F. Letters can be uppercase or lowercase. No sign is permitted.

##### varying-output-value

OpenVMS usage: varying\_arg  
type: unspecified  
access: write only  
mechanism: by reference

Unsigned integer of specified size that OTS\$CVT\_TZ\_L creates when it converts the ASCII text string. The **varying-output-value** argument is the address of the unsigned integer. The value of the **output-value-size** argument determines the size in bytes of the output value.

##### output-value-size

OpenVMS usage: longword\_signed  
type: longword (signed)  
access: read only  
mechanism: by value

Arbitrary number of bytes to be occupied by the unsigned integer output value. The **output-value-size** argument contains a value that equals the size in bytes of the output value. If the value of **output-value-size** is zero or a negative

number, OTSS\$CVT\_TZ\_L returns an input conversion error. If you omit the **output-value-size** argument, the default is 4 (longword).

**flags-value**

OpenVMS usage: mask\_longword  
 type: longword (unsigned)  
 access: read only  
 mechanism: by value

User-supplied flags that OTSS\$CVT\_TZ\_L uses to determine how to interpret blanks within the input string. The **flags-value** argument contains these user-supplied flags as described in the following table:

Bit	Action if Set	Action if Clear
0	Ignore all blanks.	Interpret blanks as zeros.

If you omit the **flags-value** argument, OTSS\$CVT\_TZ\_L defaults the flag to clear.

**Condition Values Returned**

SS\$ _NORMAL	Normal successful completion.
OTSS\$ _INPCONERR	Input conversion error. OTSS\$CVT_TZ_L encountered an invalid character in the <b>fixed-or-dynamic-input-string</b> , overflow of <b>varying-output-value</b> , or an invalid <b>output-value-size</b> . In the case of an invalid character or of an overflow, <b>varying-output-value</b> is set to zero.

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

## OTSS\$ Routines OTSS\$CVT\_TZ\_L

```
2. HEX_CONV: PROCEDURE OPTIONS (MAIN) RETURNS (FIXED BINARY (31));
%INCLUDE $STSDEF;          /* Include definition of return status values */
DECLARE OTSS$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$VALUE = OTSS$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 presses 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
```

---

## OTSS\$DIVCx Complex Division

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

### Format

OTSS\$DIVC complex-dividend ,complex-divisor

OTSS\$DIVCD\_R3 complex-dividend ,complex-divisor (VAX only)

OTSS\$DIVCG\_R3 complex-dividend ,complex-divisor

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

### Returns

OpenVMS usage: complex\_number  
 type: F\_floating complex, D\_floating complex, G\_floating complex  
 access: write only  
 mechanism: by value

Complex result of complex division. OTSS\$DIVC returns an F-floating complex number. OTSS\$DIVCD\_R3 returns a D-floating complex number. OTSS\$DIVCG\_R3 returns a G-floating complex number.

### Arguments

#### complex-dividend

OpenVMS 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 OTSS\$DIVC, **complex-dividend** is an F-floating complex number. For OTSS\$DIVCD\_R3, **complex-dividend** is a D-floating complex number. For OTSS\$DIVCG\_R3, **complex-dividend** is a G-floating complex number.

#### complex-divisor

OpenVMS 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 OTSS\$DIVC, **complex-divisor** is an F-floating complex number. For OTSS\$DIVCD\_R3, **complex-divisor** is a D-floating complex number. For OTSS\$DIVCG\_R3, **complex-divisor** is a G-floating complex number.

## OTS\$ Routines OTS\$DIVCx

### 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:

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

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

On Alpha systems, some restrictions apply when linking OTS\$DIVC or OTS\$DIVCG\_R3. See Chapter 1 for more information about these restrictions.

### Condition Values Signaled

SS\$FLTDIV_F	Arithmetic fault. Floating-point division by zero.
SS\$FLTOVF_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 OTSS$DIVCG_R3 and the Fortran random number
   C generator RAN.
   C
   C Declare Z1, Z2, and Z_Q as complex values. OTSS$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 OTSS\$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 30 or 31 digits.

#### Format

OTS\$DIV\_PK\_LONG packed-decimal-dividend ,packed-decimal-divisor  
,divisor-precision ,packed-decimal-quotient ,quotient-precision  
,precision-data ,scale-data

#### Returns

OpenVMS usage: cond\_value  
type: longword (unsigned)  
access: write only  
mechanism: by value

#### Arguments

##### packed-decimal-dividend

OpenVMS 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:

$c = 31 - \text{prec}(\text{packed-decimal-dividend})$

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

##### packed-decimal-divisor

OpenVMS 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

OpenVMS 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**

OpenVMS 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 OTSS\$DIV\_PK\_LONG writes the quotient.

**quotient-precision**

OpenVMS 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**

OpenVMS 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.

OTSS\$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**

OpenVMS 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.

OTSS\$DIV\_PK\_LONG defines the **scale-data** argument as follows:

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

## OTSS\$ Routines

### OTSS\$DIV\_PK\_LONG

#### Description

On VAX systems, before using this routine, you should determine whether it is best to use OTSS\$DIV\_PK\_LONG, OTSS\$DIV\_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 OTSS\$DIV\_PK\_LONG can be used to perform the division. If *b* is less than 31, you could use the instruction DIVP instead.

When using this routine on an OpenVMS Alpha system or on an OpenVMS VAX system and you have determined that you cannot use DIVP, you need to determine whether you should use OTSS\$DIV\_PK\_LONG or OTSS\$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 OTSS\$DIV\_PK\_LONG. If **scale-data** is greater than 1, you should use OTSS\$DIV\_PK\_SHORT instead.

#### Condition Value Signaled

SS\$FLTDIV	Fatal error. Division by zero.
------------	--------------------------------

#### Example

```
1
  OPTION                                &
    TYPE = EXPLICIT

  !+
  !   This program uses OTSS$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 OTSS$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
```

## OTSS\$ Routines OTSS\$DIV\_PK\_LONG

```

!+
! 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 OTSS$DIV_PK_LONG or
! OTSS$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 OTSS$DIV_PK_LONG
! should be used instead of OTSS$DIV_PK_SHORT.
!
!-

CALL OTSS$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 OTSS\$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

OpenVMS usage: cond\_value  
type: longword (unsigned)  
access: write only  
mechanism: by value

#### Arguments

##### packed-decimal-dividend

OpenVMS 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:

$c = 31 - \text{prec}(\text{packed-decimal-dividend})$

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

##### packed-decimal-divisor

OpenVMS 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

OpenVMS 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; high-order bits are filled with zeros.

**packed-decimal-quotient**

OpenVMS 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 OTSS\$DIV\_PK\_SHORT writes the quotient.

**quotient-precision**

OpenVMS 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; high-order bits are filled with zeros.

**precision-data**

OpenVMS 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.

OTSS\$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**

On VAX systems, before using this routine, you should determine whether it is best to use OTSS\$DIV\_PK\_LONG, OTSS\$DIV\_PK\_SHORT, or the VAX instruction DIVP. To determine this, you must first calculate *b*, where *b* is defined as follows:

$$b = \text{scale}(\text{packed-decimal-quotient}) + \text{scale}(\text{packed-decimal-divisor}) - \text{scale}(\text{packed-decimal-dividend}) + \text{prec}(\text{packed-decimal-dividend})$$

If *b* is greater than 31, then OTSS\$DIV\_PK\_SHORT can be used to perform the division. If *b* is less than 31, you could use the VAX instruction DIVP instead.

When using this routine on an OpenVMS Alpha system or on an OpenVMS VAX system and you have determined that you cannot use DIVP, you need to determine whether you should use OTSS\$DIV\_PK\_LONG or OTSS\$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 OTSS\$DIV\_PK\_LONG. If **scale-data** is greater than 1, you should use OTSS\$DIV\_PK\_SHORT instead.

**Condition Value Signaled**

SS\$FLT DIV	Fatal error. Division by zero.
-------------	--------------------------------

## OTS\$MOVE3

### Move Data Without Fill

The Move Data Without Fill routine moves up to  $2^{32} - 1$  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

OpenVMS 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

OpenVMS 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

OpenVMS 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.

## Description

OTSS\$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 OTSS\$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 Reference 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  $2^{32} - 1$  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**

OpenVMS 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**

OpenVMS 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**

OpenVMS 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.

**longword-int-dest-length**

OpenVMS 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**

OpenVMS 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 OTSS\$MOVE5 writes the source data.

**Description**

OTSS\$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 OTSS\$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 Reference Manual*. See also the routine LIB\$MOVC5, which is a callable version of the MOVC5 instruction.

**Condition Values Returned**

None.

## OTSS\$MULCx Complex Multiplication

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

### Format

OTSS\$MULCD\_R3 complex-multiplier ,complex-multiplicand (VAX only)

OTSS\$MULCG\_R3 complex-multiplier ,complex-multiplicand

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

### Returns

OpenVMS usage: complex\_number  
type: D\_floating complex, G\_floating complex  
access: write only  
mechanism: by value

Complex result of multiplying two complex numbers. OTSS\$MULCD\_R3 returns a D-floating complex number. OTSS\$MULCG\_R3 returns a G-floating complex number.

### Arguments

#### complex-multiplier

OpenVMS 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 OTSS\$MULCD\_R3, **complex-multiplier** is a D-floating complex number. For OTSS\$MULCG\_R3, **complex-multiplier** is a G-floating complex number.

#### complex-multiplicand

OpenVMS 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 OTSS\$MULCD\_R3, **complex-multiplicand** is a D-floating complex number. For OTSS\$MULCG\_R3, **complex-multiplicand** is a G-floating complex number.

### Description

OTSS\$MULCD\_R3 and OTSS\$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*

On Alpha systems, some restrictions apply when linking OTSS\$MULCG\_R3. See Chapter 1 for more information about these restrictions.

### Condition Values Signaled

SS\$_FLTOVF_F	Floating value overflow can occur.
SS\$_ROPRAND	Reserved operand. OTSS\$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 Compaq.

### Example

```

C+
C   This Fortran example forms the product of
C   two complex numbers using OTSS$MULCD_R3
C   and the Fortran random number generator RAN.
C
C   Declare Z1, Z2, and Z_Q as complex values. OTSS$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 OTSS\$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
      (4.000000000000000,32.000000000000000)

```

## 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 (VAX only)

OTS\$POWCGCG\_R3 complex-base ,complex-exponent-value

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

### Returns

OpenVMS 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

OpenVMS 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

OpenVMS 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

OTSS\$POWCC, OTSS\$POWCDCD\_R3 and OTSS\$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:

$$x^y = \text{exp}(y * \log(x))$$

In this example,  $x$  and  $y$  are type COMPLEX.

On Alpha systems, some restrictions apply when linking OTSS\$POWCC or OTSS\$POWCGCG\_R3. See Chapter 1 for more information about these restrictions.

## 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 OTSS$POWCC.
   C
   C   Declare Z1, Z2, Z3, and OTSS$POWCC as complex values. Then OTSS$POWCC
   C   returns the complex result of Z1**Z2:  Z3 = OTSS$POWCC(Z1,Z2),
   C   where Z1 and Z2 are passed by value.
   C-

           COMPLEX Z1,Z2,Z3,OTSS$POWCC
C+
C   Generate a complex base.
C-
           Z1 = (2.0,3.0)
C+
C   Generate a complex power.
C-
           Z2 = (1.0,2.0)
C+
C   Compute the complex value of Z1**Z2.
C-
           Z3 = OTSS$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 OTSS\$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)

```

## OTSS\$ Routines OTSS\$POWCxCx

```
2. C+
C   This Fortran example raises a complex base to a complex
C   power using OTSS$POWCGCG_R3.
C
C   Declare Z1, Z2, and Z3 as complex values. OTSS$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 OTSS\$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).

---

## OTSS\$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

OTSS\$POWCJ complex-base ,longword-integer-exponent

OTSS\$POWCDJ\_R3 complex-base ,longword-integer-exponent (VAX only)

OTSS\$POWCGJ\_R3 complex-base ,longword-integer-exponent (VAX only)

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

#### Returns

OpenVMS 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. OTSS\$POWCJ returns an F-floating complex number. OTSS\$POWCDJ\_R3 returns a D-floating complex number. OTSS\$POWCGJ\_R3 returns a G-floating complex number. In each format, the result and base are of the same data type.

#### Arguments

##### complex-base

OpenVMS 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 OTSS\$POWCJ, **complex-base** is an F-floating complex number. For OTSS\$POWCDJ\_R3, **complex-base** is a D-floating complex number. For OTSS\$POWCGJ\_R3, **complex-base** is a G-floating complex number.

##### longword-integer-exponent

OpenVMS 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.

## OTS\$ Routines OTS\$POWCxJ

### 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 $(\text{base}^{**2^i})$ , where $i$ is each nonzero bit in <b>longword-integer-exponent</b> .
(0.,0.)	$\leq 0$	Undefined exponentiation.
Not (0.,0.)	$< 0$	The product of $(\text{base}^{**2^i})$ , where $i$ is each nonzero bit in <b>longword-integer-exponent</b> .
Not (0.,0.)	0	(1.0,0.0)

On Alpha systems, some restrictions apply when linking OTS\$POWCJ. See Chapter 1 for more information about these restrictions.

### 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).
```

---

## OTSS\$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

OTSS\$POWDD D-floating-point-base ,D-floating-point-exponent

#### Returns

OpenVMS usage: floating\_point  
 type: D\_floating  
 access: write only  
 mechanism: by value

Result of raising a D-floating base to a D-floating exponent.

#### Arguments

##### D-floating-point-base

OpenVMS 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

OpenVMS 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

OTSS\$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 OTSS\$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

## OTS\$ Routines

### OTS\$POWDD

Base	Exponent	Result
> 0	> 0	$2^{[exponent*\log_2(base)]}$
> 0	= 0	1.0
> 0	< 0	$2^{[exponent*\log_2(base)]}$

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 <b>D-floating-point-base</b> is zero and <b>D-floating-point-exponent</b> is zero or negative, or if the <b>D-floating-point-base</b> is negative.

---

## OTSS\$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

OTSS\$POWDJ D-floating-point-base ,longword-integer-exponent

#### Returns

OpenVMS usage: floating\_point  
 type: D\_floating  
 access: write only  
 mechanism: by value

Result of raising a D-floating base to a longword exponent.

#### Arguments

##### D-floating-point-base

OpenVMS 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

OpenVMS 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

OTSS\$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 <b>longword-integer-exponent</b> .
> 0	= 0	1.0
= 0	= 0	Undefined exponentiation.

## OTS\$ Routines

### OTS\$POWDJ

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

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 <b>D-floating-point-base</b> is zero and <b>longword-integer-exponent</b> is zero or negative, or if the <b>D-floating-point-base</b> is negative.

---

## OTSS\$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

OTSS\$POWDR D-floating-point-base ,F-floating-point-exponent

#### Returns

OpenVMS usage: floating\_point  
 type: D\_floating  
 access: write only  
 mechanism: by value

Result of raising a D-floating base to an F-floating exponent.

#### Arguments

##### D-floating-point-base

OpenVMS 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

OpenVMS 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

OTSS\$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.

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

Base	Exponent	Result
= 0	> 0	0.0
= 0	= 0	Undefined exponentiation
= 0	< 0	Undefined exponentiation

## OTS\$ Routines

### OTS\$POWDR

Base	Exponent	Result
< 0	Any	Undefined exponentiation
> 0	> 0	$2^{[exponent*\log_2(base)]}$
> 0	= 0	1.0
> 0	< 0	$2^{[exponent*\log_2(base)]}$

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 <b>D-floating-point-base</b> is zero and <b>F-floating-point-exponent</b> is zero or negative, or if the <b>D-floating-point-base</b> is negative.

---

## OTSS\$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

OTSS\$POWGG G-floating-point-base ,G-floating-point-exponent

#### Returns

OpenVMS usage: floating\_point  
 type: G\_floating  
 access: write only  
 mechanism: by value

Result of raising a G-floating base to a G-floating exponent.

#### Arguments

##### G-floating-point-base

OpenVMS usage: floating\_point  
 type: G\_floating  
 access: read only  
 mechanism: by value

Base that OTSS\$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

OpenVMS usage: floating\_point  
 type: G\_floating  
 access: read only  
 mechanism: by value

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

#### Description

OTSS\$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 OTSS\$POWGG is as follows:

Base	Exponent	Result
= 0	> 0	0.0
= 0	= 0	Undefined exponentiation
= 0	< 0	Undefined exponentiation
< 0	Any	Undefined exponentiation

## OTSS\$ Routines

### OTSS\$POWGG

Base	Exponent	Result
> 0	> 0	$2^{[exponent*\log_2(base)]}$
> 0	= 0	1.0
> 0	< 0	$2^{[exponent*\log_2(base)]}$

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.

On Alpha systems, some restrictions apply when linking OTSS\$POWGG. See Chapter 1 for more information about these restrictions.

### 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 <b>G-floating-point-base</b> is zero and <b>G-floating-point-exponent</b> is zero or negative, or if <b>G-floating-point-base</b> is negative.

### Example

```

C+
C This example demonstrates the use of OTSS$POWGG,
C which raises a G-floating point base
C to a G-floating point power.
C-
      REAL*8 X,Y,RESULT,OTSS$POWGG
C+
C The arguments of OTSS$POWGG are passed by value. Fortran can
C only pass INTEGER and REAL*4 expressions as VALUE. Since
C INTEGER and REAL*4 values are one longword long, while REAL*8
C values are two longwords long, equate the base (and power) to
C two-dimensional INTEGER vectors. These vectors will be passed
C by VALUE.
C-
      INTEGER N(2),M(2)
      EQUIVALENCE (N(1),X), (M(1),Y)
      X = 8.0
      Y = 2.0
C+
C To pass X by value, pass N(1) and N(2) by value. Similarly for Y.
C-
      RESULT = OTSS$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, OTSS$POWGG 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 OTSS\$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 64.00000000000000  
9.0**-0.5 IS 0.3333333333333333
```

---

## 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

OpenVMS usage: floating\_point  
type: G\_floating  
access: write only  
mechanism: by value

Result of raising a G-floating base to a longword exponent.

#### Arguments

##### G-floating-point-base

OpenVMS 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

OpenVMS 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 <b>longword-integer-exponent</b> .
> 0	= 0	1.0
= 0	= 0	Undefined exponentiation.

---

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

Floating-point overflow can occur.

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

On Alpha systems, some restrictions apply when linking OTSS\$POWGJ. See Chapter 1 for more information about these restrictions.

### 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 <b>G-floating-point-base</b> is zero and <b>longword-integer-exponent</b> is zero or negative, or if <b>G-floating-point-base</b> is negative.

---

## OTS\$POWHH\_R3 (VAX Only)

### Raise an H-Floating Base to an H-Floating Exponent

On VAX systems, 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

OpenVMS usage: floating\_point  
type: H\_floating  
access: write only  
mechanism: by value

Result of raising an H-floating base to an H-floating exponent.

#### Arguments

##### H-floating-point-base

OpenVMS 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

OpenVMS 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

---

Base	Exponent	Result
> 0	> 0	$2^{[exponent*\log_2(base)]}$
> 0	= 0	1.0
> 0	< 0	$2^{[exponent*\log_2(base)]}$

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 <b>H-floating-point-base</b> is zero and <b>H-floating-point-exponent</b> is zero or negative, or if the <b>H-floating-point-base</b> is negative.

### Example

```

C+
C Example of OTSS$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, OTSS$POWHH 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 OTSS\$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 1.463779145994628357482343598205427E-0006
```

---

## OTS\$POWHJ\_R3 (VAX Only)

### Raise an H-Floating Base to a Longword Exponent

On VAX systems, 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

OpenVMS usage: floating\_point  
type: H\_floating  
access: write only  
mechanism: by value

Result of raising an H-floating base to a longword exponent.

#### Arguments

##### H-floating-point-base

OpenVMS 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

OpenVMS 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 <b>longword-integer-exponent</b> .
> 0	= 0	1.0
= 0	= 0	Undefined exponentiation.

---

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

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 <b>H-floating-point-base</b> is zero and <b>longword-integer-exponent</b> is zero or negative, or if the <b>H-floating-point-base</b> 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

OpenVMS usage: word\_signed  
type: word (signed)  
access: write only  
mechanism: by value

Result of raising a word base to a word exponent.

#### Arguments

##### **word-integer-base**

OpenVMS 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**

OpenVMS 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.

#### Description

The OTS\$POWII routine raises a word base to a word exponent.

On Alpha systems, some restrictions apply when linking OTS\$POWII. See Chapter 1 for more information about these restrictions.

### Condition Values Signaled

SS\$FLT DIV

Arithmetic trap. This error is signaled by the hardware if a floating-point division by zero occurs.

SS\$FLT OVF

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

OpenVMS usage: longword\_signed  
type: longword (signed)  
access: write only  
mechanism: by value

Result of raising a signed longword base to a signed longword exponent.

#### Arguments

##### **longword-integer-base**

OpenVMS 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**

OpenVMS 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

The OTS\$POWJJ routine raises a signed longword base to a signed longword exponent.

On Alpha systems, some restrictions apply when linking OTS\$POWJJ. See Chapter 1 for more information about these restrictions.

### 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

OpenVMS usage: longword\_unsigned  
type: longword (unsigned)  
access: write only  
mechanism: by value

Result of raising an unsigned longword integer base to an unsigned longword integer exponent.

#### Arguments

##### **unsigned-lword-int-base**

OpenVMS 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**

OpenVMS 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.

On Alpha systems, some restrictions apply when linking OTS\$POWLULU. See Chapter 1 for more information about these restrictions.

**Condition Values Signaled**

MTH\$UNDEXP

Both the base and exponent values 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

OpenVMS usage: floating\_point  
type: D\_floating  
access: write only  
mechanism: by value

Result of raising an F-floating base to a D-floating exponent.

#### Arguments

##### F-floating-point-base

OpenVMS 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

OpenVMS 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
= 0	< 0	Undefined exponentiation

---

Base	Exponent	Result
< 0	Any	Undefined exponentiation
> 0	> 0	$2^{[exponent * \text{LOG}_2(\text{base})]}$
> 0	= 0	1.0
> 0	< 0	$2^{[exponent * \text{LOG}_2(\text{base})]}$

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 <b>F-floating-point-base</b> is zero and <b>D-floating-point-exponent</b> is zero or negative, or if <b>F-floating-point-base</b> is negative.

### Example

```

C+
C   This Fortran example uses OTSS$POWRD, to raise an F-floating point
C   base to a D-floating point exponent. The result is a D-floating value.
C-
      REAL*4 X
      REAL*8 Y,RESULT,OTSS$POWRD
      INTEGER M(2)
      EQUIVALENCE (M(1),Y)
      X = 9768.0
      Y = 9.0

C+
C   The arguments of OTSS$POWRD are passed by value.
C-
      RESULT = OTSS$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, OTSS$POWRD is indirectly called by the exponentiation operator.
C-
      RESULT = X**Y
      TYPE *, ' 7689.0**-0.587436654545 IS ',RESULT
      END

```

This Fortran example uses OTSS\$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 OTSS\$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

OpenVMS usage: floating\_point  
type: F\_floating  
access: write only  
mechanism: by value

Result of raising an F-floating base to a longword exponent.

#### Arguments

##### F-floating-point-base

OpenVMS 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

OpenVMS 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 <b>longword-integer-exponent</b> .
> 0	= 0	1.0
= 0	= 0	Undefined exponentiation.

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

Floating-point overflow can occur.

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

On Alpha systems, some restrictions apply when linking OTSS\$POWRJ. See Chapter 1 for more information about these restrictions.

### 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 <b>F-floating-point-base</b> is zero and <b>longword-integer-exponent</b> is zero or negative, or if <b>F-floating-point-base</b> 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

OpenVMS usage: floating\_point  
type: F\_floating  
access: write only  
mechanism: by value

Result of raising an F-floating base to an F-floating exponent.

#### Arguments

##### F-floating-point-base

OpenVMS 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

OpenVMS 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

---

Base	Exponent	Result
> 0	> 0	$2^{[exponent*\log_2(base)]}$
> 0	= 0	1.0
> 0	< 0	$2^{[exponent*\log_2(base)]}$

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.

On Alpha systems, some restrictions apply when linking OTSS\$POWRR. See Chapter 1 for more information about these restrictions.

### Condition Values Signaled

SS\$FLTUVF	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 <b>F-floating-point-base</b> is zero and <b>F-floating-point-exponent</b> is zero or negative, or if <b>F-floating-point-base</b> is negative.

### Example

```

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

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

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

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

C+
C In Fortran, OTSS$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 OTSS\$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\$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 raise 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 (VAX only)

#### Returns

OpenVMS 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.

On VAX systems, OTS\$POWHLU\_R3 returns an H-floating number.

#### Arguments

##### floating-point-base

OpenVMS 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

OpenVMS 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

OTSS\$POWRLU, OTSS\$POWDLU, OTSS\$POWGLU, and OTSS\$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 <b>longword-integer-exponent</b> .
> 0	= 0	1.0
= 0	= 0	Undefined exponentiation.
< 0	= 0	1.0

On Alpha systems, some restrictions apply when linking OTSS\$POWRLU or OTSS\$POWGLU. See Chapter 1 for more information about these restrictions.

## 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 <b>floating-point-base</b> and <b>unsigned-longword-integer-exponent</b> arguments are zero.

## OTS\$COPY\_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\$COPY\_DXDX source-string ,destination-string

#### Corresponding JSB Entry Point

OTS\$COPY\_DXDX6

#### Returns

OpenVMS usage: word\_unsigned  
type: word (unsigned)  
access: write only  
mechanism: by value

Number of bytes not moved to the destination string if the length of **source-string** is greater than the length of **destination-string**. The value is 0 (zero) otherwise.

#### Arguments

##### **source-string**

OpenVMS 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**

OpenVMS 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.

## Description

OTSS\$COPY\_DXDX copies a source string to a destination string. It passes the source string by descriptor. If the length of the source string is greater than the length of the destination string, OTSS\$COPY\_DXDX returns the number of bytes not moved to the destination string. If the length of the source string is less than or equal to the length of the destination string, it returns 0 (zero). All error conditions except truncation are signaled; truncation is ignored.

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). On return, R0 through R5 and the PSL are as they would be after a VAX MOVC5 instruction. R0 through 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*.

The actions taken by OTSS\$COPY\_DXDX depend on the descriptor class of the destination string. The following table describes these actions for each descriptor class:

Descriptor Class	Action
S, Z, SD, A, NCA	Copy the source string. If needed, space fill or truncate on the right.
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.
VS	Copy source string to destination string up to the limit of the destination descriptor's MAXSTRLEN field with no padding. Adjust the string's current length field (CURLLEN) to the actual number of bytes copied.

## Condition Values Signaled

OTSS\$_FATINTERR	Fatal internal error.
OTSS\$_INVSTRDES	Invalid string descriptor.
OTSS\$_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

OpenVMS usage: word\_unsigned  
type: word (unsigned)  
access: write only  
mechanism: by value

Number of bytes not moved to the destination string if the length of the source string pointed to by **source-string-address** is greater than the length of **destination-string**. Otherwise, the value is 0 (zero).

#### Arguments

##### **word-int-source-length-val**

OpenVMS 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**

OpenVMS 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**

OpenVMS 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. OTS\$SCOPY\_R\_DX determines the appropriate action based on the descriptor's CLASS field. The descriptor's LENGTH field alone or both the POINTER and LENGTH fields can be modified if

the string is dynamic. For varying strings, the string's current length (CURLLEN) is rewritten.

## Description

OTSS\$COPY\_R\_DX copies a source string to a destination string. It passes the source string by reference preceded by a length argument. The length argument, **word-int-source-length-val**, is passed by value.

If the length of the source string is greater than the length of the destination string, OTSS\$COPY\_R\_DX returns the number of bytes not moved to the destination string. If the length of the source string is less than or equal to the length of the destination string, it returns 0 (zero). All conditions except truncation are signaled; truncation is ignored.

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. On return, R0 through R5 and the PSL are as they would be after a VAX MOVC5 instruction. R0 through 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*.

The actions taken by OTSS\$COPY\_R\_DX depend on the descriptor class of the destination string. The following table describes these actions for each descriptor class:

Descriptor Class	Action
S, Z, SD, A, NCA	Copy the source string. If needed, space fill or truncate on the right.
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.
VS	Copy source string to destination string up to the limit of the descriptor's MAXSTRLEN field with no padding. Adjust the string's current length (CURLLEN) field to the actual number of bytes copied.

## OTSS\$ Routines

### OTSS\$COPY\_R\_DX

#### Condition Values Signaled

OTSS_FATINTERR	Fatal internal error.
OTSS_INVSTRDES	Invalid string descriptor.
OTSS_INSVIRMEM	Insufficient virtual memory.

#### Example

A Fortran example that demonstrates the manipulation of dynamic strings appears at the end of OTSS\$GET1\_DD. This example uses OTSS\$COPY\_R\_DX, OTSS\$GET1\_DD, and OTSS\$FREE1\_DD.

## OTSS\$FREE1\_DD Strings, Free One Dynamic

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

### Format

OTSS\$FREE1\_DD dynamic-descriptor

### Corresponding JSB Entry Point

OTSS\$FREE1\_DD6

### Returns

None.

### Argument

#### **dynamic-descriptor**

OpenVMS 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

OTSS\$FREE1\_DD deallocates the described string space and flags the descriptor as describing no string at all. The descriptor's POINTER and LENGTH fields contain 0.

### Condition Value Signaled

OTSS\_FATINTERR Fatal internal error.

### Example

A Fortran example that demonstrates the manipulation of dynamic strings appears at the end of OTSS\$GET1\_DD. This example uses OTSS\$FREE1\_DD, OTSS\$GET1\_DD, and OTSS\$COPY\_R\_DX.

## OTSS\$FREEN\_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

OTSS\$FREEN\_DD descriptor-count-value ,first-descriptor

#### Corresponding JSB Entry Point

OTSS\$FREEN\_DD6

#### Returns

None.

#### Arguments

##### **descriptor-count-value**

OpenVMS usage: longword\_unsigned  
type: longword (unsigned)  
access: read only  
mechanism: by value

Number of adjacent descriptors to be flagged as having no allocated area (the descriptor's POINTER and LENGTH fields contain 0) and to have their allocated areas returned to free storage by OTSS\$FREEN\_DD. The **descriptor-count-value** argument is an unsigned longword containing this number.

##### **first-descriptor**

OpenVMS 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

OTSS\$FREEN\_DD6 deallocates the described string space and flags each descriptor as describing no string at all. The descriptor's POINTER and LENGTH fields contain 0.

#### Condition Values Signaled

OTSS\_FATINTERR Fatal internal error.

---

## OTSS\$GET1\_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

OTSS\$GET1\_DD word-integer-length-value ,dynamic-descriptor

#### Corresponding JSB Entry Point

OTSS\$GET1\_DD\_R6

#### Returns

None.

#### Arguments

##### **word-integer-length-value**

OpenVMS 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**

OpenVMS 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 (CLASS = 2). The LENGTH field is set to **word-integer-length-value** and the POINTER field is set to the string area allocated (first byte beyond the header).

#### Description

OTSS\$GET1\_DD allocates a specified number of bytes of dynamic virtual memory to a specified string descriptor. This routine is identical to OTSS\$COPY\_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 OTSS\$GET1\_DD allocates new space.

# OTSS\$ Routines

## OTSS\$GET1\_DD

### Condition Values Signaled

OTSS_FATINTERR	Fatal internal error.
OTSS_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 OTSS$GET1_DD(%VAL(0), DSC)

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

CALL OTSS$GET1_DD(%VAL(2), CRLF_DSC)
CRLF = CHAR(13)//CHAR(10)
CALL OTSS$COPY_R_DX( %VAL(2), %REF(CRLF(1:1)), CRLF_DSC)

C+
C   Initialize a temporary descriptor.
C-

CALL OTSS$GET1_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 OTSS$COPY_R_DX( %VAL(DATA_LEN),
1     %REF(DATA_LINE(1:1)),
2     TEMP_DSC)
  CALL STR$CONCAT( DSC, DSC, TEMP_DSC, CRLF_DSC )
END DO
```

## OTSS\$ Routines OTSS\$GET1\_DD

```
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 OTSS$FREE1_DD( DSC )
    CALL OTSS$FREE1_DD( CRLF_DSC )
    CALL OTSS$FREE1_DD( TEMP_DSC )

C+
C   End of program.
C-

    STOP
    END
```

**This Fortran example program demonstrates the manipulation of dynamic strings using OTSS\$GET1\_DD, OTSS\$FREE1\_DD, and OTSS\$COPY\_R\_DX.**



## C

---

### Complex numbers

- division of, OTS-38
- multiplication of, OTS-50

### Conversions

- binary text to unsigned integer, OTS-22
  - floating-point to character string, OTS-3
  - hexadecimal text to unsigned integer, OTS-34
  - integer to binary text, OTS-5, OTS-7
  - integer to decimal text, OTS-13
  - integer to Fortran L format, OTS-9
  - integer to hexadecimal text, OTS-15
  - integer to logical text, OTS-9
  - integer to octal text, OTS-11
  - logical text to integer, OTS-27
  - numeric text to floating-point, OTS-17
  - octal text to unsigned integer, OTS-29
  - signed integer text to integer, OTS-25
  - unsigned decimal to integer, OTS-32
- Copying strings, OTS-86

## D

---

### Division

- complex number, OTS-38
- packed decimal, OTS-42, OTS-45

### Dynamic length strings

- allocating, OTS-93
- deallocating, OTS-91, OTS-92

## E

---

### Exponentiation

- complex base to complex exponent, OTS-53
- complex base to signed integer exponent, OTS-56
- D-floating base, OTS-57, OTS-59, OTS-61
- F-floating base, OTS-78, OTS-80, OTS-82
- G-floating base, OTS-63, OTS-66
- H-floating base, OTS-68, OTS-70
- signed longword base, OTS-74
- word base to word exponent, OTS-72

## M

---

### Memory allocation

- for dynamic length strings, OTS-93
- freeing dynamic length strings, OTS-91, OTS-92

### Multiplication

- of complex numbers, OTS-50

## O

---

- OTSSCNVOUT routine, OTS-3
- OTSSCNVOUT\_D routine, OTS-3
- OTSSCNVOUT\_G routine, OTS-3
- OTSSCNVOUT\_H routine, OTS-3
- OTSSCVT\_L\_TB routine, OTS-5
- OTSSCVT\_L\_TI routine, OTS-7
- OTSSCVT\_L\_TL routine, OTS-9
- OTSSCVT\_L\_TO routine, OTS-11
- OTSSCVT\_L\_TU routine, OTS-13
- OTSSCVT\_L\_TZ routine, OTS-15
- OTSSCVT\_TB\_L routine, OTS-22
- OTSSCVT\_TI\_L routine, OTS-25
- OTSSCVT\_TL\_L routine, OTS-27
- OTSSCVT\_TO\_L routine, OTS-29
- OTSSCVT\_TU\_L routine, OTS-32
- OTSSCVT\_TZ\_L routine, OTS-34
- OTSSCVT\_T\_x routine, OTS-17
- OTSSDIVCD\_R3 routine, OTS-37
- OTSSDIVCG\_R3 routine, OTS-37
- OTSSDIVCx routine, OTS-37
- OTSSDIV\_PK\_LONG routine, OTS-40
- OTSSDIV\_PK\_SHORT routine, OTS-44
- OTSSMOVE3 routine, OTS-46
- OTSSMOVE5 routine, OTS-48
- OTSSMULCD\_R3 routine, OTS-50
- OTSSMULCG\_R3 routine, OTS-50
- OTSSPOWCxCx routine, OTS-52
- OTSSPOWCxJ routine, OTS-55
- OTSSPOWDD routine, OTS-57
- OTSSPOWDJ routine, OTS-59
- OTSSPOWDLU routine, OTS-84
- OTSSPOWDR routine, OTS-61
- OTSSPOWGG routine, OTS-63

OTSSPOWGJ routine, OTS-66  
OTSSPOWGLU routine, OTS-84  
OTSSPOWHH\_R3 routine, OTS-68  
OTSSPOWHJ\_R3 routine, OTS-70  
OTSSPOWHLU\_R3 routine, OTS-84  
OTSSPOWII routine, OTS-72  
OTSSPOWJJ routine, OTS-74  
OTSSPOWLULU routine, OTS-76  
OTSSPOWRD routine, OTS-78  
OTSSPOWRJ routine, OTS-80  
OTSSPOWRLU routine, OTS-84  
OTSSPOWRR routine, OTS-82  
OTSSSCOPY\_DXDX routine, OTS-86  
OTSSSCOPY\_R\_DX routine, OTS-88  
OTSSSFREE1\_DD routine, OTS-91  
OTSSSFREEN\_DD routine, OTS-92

OTSSSGET1\_DD routine, OTS-93

## **R**

---

Run-time library routines  
    general purpose, 1-1

## **S**

---

Strings  
    copying by descriptor, OTS-86  
    copying by reference, OTS-88  
    dynamic length  
        allocating, OTS-93  
        deallocating, OTS-91, OTS-92