OpenVMS RTL General Purpose (OTS$) Manual

Order Number: AA–PV6HD–TK

April 2001

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

Revision/Update Information: This manual supersedes the OpenVMS RTL General Purpose (OTS$) 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
Contents

Preface ........................................................................................................ 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$DIVCx ......................................................................................... OTS–37
OTS$DIV_PK_LONG ............................................................................... OTS–40
OTS$DIV_PK_SHORT ............................................................................. OTS–44
OTS$MOVE3 ......................................................................................... OTS–46
OTS$MOVE5 ......................................................................................... OTS–48
OTS$MULCx ......................................................................................... OTS–50
OTS$POWCxCx ................................................................................... OTS–52
OTS$POWCxj ......................................................................................... OTS–55
OTS$POWDD ......................................................................................... OTS–57
OTS$POWDJ ......................................................................................... OTS–59
OTS$POWDR ......................................................................................... OTS–61
OTS$POWGG ......................................................................................... OTS–63
OTS$POWGJ ......................................................................................... OTS–66
Index

Tables

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1–1</td>
<td>OTS$ Conversion Routines</td>
<td></td>
<td>1–2</td>
</tr>
<tr>
<td>1–2</td>
<td>OTS$ Division Routines</td>
<td></td>
<td>1–2</td>
</tr>
<tr>
<td>1–3</td>
<td>OTS$ Move Data Routines</td>
<td></td>
<td>1–2</td>
</tr>
<tr>
<td>1–4</td>
<td>OTS$ Multiplication Routine</td>
<td></td>
<td>1–2</td>
</tr>
<tr>
<td>1–5</td>
<td>OTS$ Exponentiation Routines</td>
<td></td>
<td>1–3</td>
</tr>
<tr>
<td>1–6</td>
<td>OTS$ Copy Source String Routines</td>
<td></td>
<td>1–3</td>
</tr>
<tr>
<td>1–7</td>
<td>OTS$ Return String Area Routines</td>
<td></td>
<td>1–3</td>
</tr>
<tr>
<td>1–8</td>
<td>OTS$ and Equivalent Math$ Entry Points</td>
<td></td>
<td>1–4</td>
</tr>
</tbody>
</table>
Preface

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

Intended Audience

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

Document Structure

This manual is organized into two parts as follows:

• Part I contains a brief overview of the OTS$ routines in Chapter 1.
• Part II, the OTS$ Reference Section, provides detailed reference information on each routine contained in the OTS$ facility of the Run-Time Library. This information is presented using the documentation format described in 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 OTS$ 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
• OpenVMS RTL Library (LIB$) Manual
• OpenVMS RTL Parallel Processing (PPL$) Manual
• 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, the Compaq Multithreading Run-Time Library.

1 This manual has been archived but is available on the OpenVMS Documentation CD-ROM.
2 Compaq POSIX Threads was formerly called DECTreads.
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: openvmsdoc@compaq.com
Mail:
Compaq Computer Corporation
OSSG Documentation Group, ZK03-4/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/x: A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.

- PF1 x: A sequence such as PF1 x indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.

- Return: 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.
A horizontal ellipsis in examples indicates one of the following possibilities:

- Additional optional arguments in a statement have been omitted.
- The preceding item or items can be repeated one or more times.
- Additional parameters, values, or other information can be entered.

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.

In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.

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.

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.

This typeface represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.

Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error number), in command lines (/PRODUCER=name), and in command parameters in text (where dd represents the predefined code for the device type).

Monospace type indicates code examples and interactive screen displays.

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.

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.

All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radices—binary, octal, or hexadecimal—are explicitly indicated.
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.
This chapter describes the OpenVMS Run-Time Library General Purpose (OTS$) facility. See the OTS$ Reference Section for a detailed description of each routine within the OTS$ facility.

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

1.1 Overview

The Run-Time Library General Purpose (OTS$) 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 OTS$ 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 OTS$ routines on an Alpha system. See Section 1.2 for more information about these restrictions.
### Table 1–1 OTS$ Conversion Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$CNVOUT</td>
<td>Convert a D-floating, G-floating, or H-floating value to a character string.</td>
</tr>
<tr>
<td>OTS$CVT_L_TB</td>
<td>Convert an unsigned integer to binary text.</td>
</tr>
<tr>
<td>OTS$CVT_L_TI</td>
<td>Convert a signed integer to signed integer text.</td>
</tr>
<tr>
<td>OTS$CVT_L_TL</td>
<td>Convert an integer to logical text.</td>
</tr>
<tr>
<td>OTS$CVT_L_TO</td>
<td>Convert an unsigned integer to octal text.</td>
</tr>
<tr>
<td>OTS$CVT_L_TU</td>
<td>Convert an unsigned integer to decimal text.</td>
</tr>
<tr>
<td>OTS$CVT_L_TZ</td>
<td>Convert an integer to hexadecimal text.</td>
</tr>
<tr>
<td>OTS$CVT_TB_L</td>
<td>Convert binary text to an unsigned integer value.</td>
</tr>
<tr>
<td>OTS$CVT_TI_L</td>
<td>Convert signed integer text to an integer value.</td>
</tr>
<tr>
<td>OTS$CVT_TL_L</td>
<td>Convert logical text to an integer value.</td>
</tr>
<tr>
<td>OTS$CVT_TO_L</td>
<td>Convert octal text to an unsigned integer value.</td>
</tr>
<tr>
<td>OTS$CVT_TU_L</td>
<td>Convert unsigned decimal text to an integer value.</td>
</tr>
<tr>
<td>OTS$CVT_T_x</td>
<td>Convert numeric text to a D-, F-, G-, or H-floating value.</td>
</tr>
<tr>
<td>OTS$CVT_TZ_L</td>
<td>Convert hexadecimal text to an unsigned integer value.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$DIVCx</td>
<td>Perform complex division.</td>
</tr>
<tr>
<td>OTS$DIV_PK_LONG</td>
<td>Perform packed decimal division with a long divisor.</td>
</tr>
<tr>
<td>OTS$DIV_PK_SHORT</td>
<td>Perform packed decimal division with a short divisor.</td>
</tr>
</tbody>
</table>

### Table 1–3 OTS$ Move Data Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$MOVE3</td>
<td>Move data without fill.</td>
</tr>
<tr>
<td>OTS$MOVE5</td>
<td>Move data with fill.</td>
</tr>
</tbody>
</table>

### Table 1–4 OTS$ Multiplication Routine

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$MULCx</td>
<td>Perform complex multiplication.</td>
</tr>
</tbody>
</table>
### Table 1–5 OTS$ Exponentiation Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$POWCxCx</td>
<td>Raise a complex base to a complex floating-point exponent.</td>
</tr>
<tr>
<td>OTS$POWCxJ</td>
<td>Raise a complex base to a signed longword exponent.</td>
</tr>
<tr>
<td>OTS$POWDD</td>
<td>Raise a D-floating base to a D-floating exponent.</td>
</tr>
<tr>
<td>OTS$POWDR</td>
<td>Raise a D-floating base to an F-floating exponent.</td>
</tr>
<tr>
<td>OTS$POWDJ</td>
<td>Raise a D-floating base to a longword integer exponent.</td>
</tr>
<tr>
<td>OTS$POWGG</td>
<td>Raise a G-floating base to a G-floating or longword integer exponent.</td>
</tr>
<tr>
<td>OTS$POWGJ</td>
<td>Raise a G-floating base to a longword integer exponent.</td>
</tr>
<tr>
<td>†OTS$POWHH R3</td>
<td>Raise an H-floating base to an H-floating exponent.</td>
</tr>
<tr>
<td>†OTS$POWHJ R3</td>
<td>Raise an H-floating base to a longword integer exponent.</td>
</tr>
<tr>
<td>OTS$POWI</td>
<td>Raise a word integer base to a word integer exponent.</td>
</tr>
<tr>
<td>OTS$POWJ J</td>
<td>Raise a longword integer base to a longword integer exponent.</td>
</tr>
<tr>
<td>OTS$POWLULU</td>
<td>Raise an unsigned longword integer base to an unsigned longword integer exponent.</td>
</tr>
<tr>
<td>OTS$POWXLU</td>
<td>Raise a floating-point base to an unsigned longword integer exponent.</td>
</tr>
<tr>
<td>OTS$POWRD</td>
<td>Raise an F-floating base to a D-floating exponent.</td>
</tr>
<tr>
<td>OTS$POWRJ</td>
<td>Raise an F-floating base to a longword integer exponent.</td>
</tr>
<tr>
<td>OTS$POWRR</td>
<td>Raise an F-floating base to an F-floating exponent.</td>
</tr>
</tbody>
</table>

†VAX specific.

### Table 1–6 OTS$ Copy Source String Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$SCOPY_DXDX</td>
<td>Copy a source string passed by descriptor to a destination string.</td>
</tr>
<tr>
<td>OTS$SCOPY_R_DX</td>
<td>Copy a source string passed by reference to a destination string.</td>
</tr>
</tbody>
</table>

### Table 1–7 OTS$ Return String Area Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$SFREE1_DD</td>
<td>Free one dynamic string.</td>
</tr>
<tr>
<td>OTS$SFREEN_DD</td>
<td>Free n dynamic strings.</td>
</tr>
<tr>
<td>OTS$SGET1_DD</td>
<td>Get one dynamic string.</td>
</tr>
</tbody>
</table>
1.2 Linking OTS$ Routines on an Alpha System

On Alpha systems, if you use the OTS$ 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 OTS$ entry points with their equivalent math$ entry points. Refer to the Compaq Portable Mathematics Library for information about the math$ entry points.

<table>
<thead>
<tr>
<th>OTS$ Entry Point</th>
<th>Math$ Entry Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$DIVC</td>
<td>math$cdiv_f</td>
</tr>
<tr>
<td>OTS$DIVCG_R3</td>
<td>math$cdiv_g</td>
</tr>
<tr>
<td>OTS$MULCG_R3</td>
<td>math$cmul_g</td>
</tr>
<tr>
<td>OTS$POWCC</td>
<td>math$cpow_f</td>
</tr>
<tr>
<td>OTS$POWCGCG_R3</td>
<td>math$cpow_g</td>
</tr>
<tr>
<td>OTS$POWCJ</td>
<td>math$cpow_fq</td>
</tr>
<tr>
<td>OTS$POWG</td>
<td>math$pow_qg</td>
</tr>
<tr>
<td>OTS$POWj</td>
<td>math$pow_qq</td>
</tr>
<tr>
<td>OTS$POWGLU</td>
<td>math$pow_qq</td>
</tr>
<tr>
<td>OTS$POWII</td>
<td>math$pow_qq</td>
</tr>
<tr>
<td>OTS$POWJ</td>
<td>math$pow_qq</td>
</tr>
<tr>
<td>OTS$POWLULU</td>
<td>math$pow_qq</td>
</tr>
<tr>
<td>OTS$POWR</td>
<td>math$pow_fq</td>
</tr>
<tr>
<td>OTS$POWRLU</td>
<td>math$pow_fq</td>
</tr>
<tr>
<td>OTS$POWRR</td>
<td>math$pow_ff</td>
</tr>
</tbody>
</table>

1.2.1 64-Bit Addressing Support (Alpha Only)

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

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

Note

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

See the OpenVMS Programming Concepts Manual for more information about 64-bit virtual addressing capabilities.
This section provides detailed descriptions of the routines provided by the OpenVMS RTL General Purpose (OTS$) Facility.
OTS$CNVOUT
Convert D-Floating, G-Floating or H-Floating Number to Character String

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

Format

OTS$CNVOUT    D-G-or-H-float-pt-input-val ,fixed-length-resultant-string ,digits-in-fraction
OTS$CNVOUT_G  D-G-or-H-float-pt-input-val ,fixed-length-resultant-string ,digits-in-fraction
OTS$CNVOUT_H  D-G-or-H-float-pt-input-val ,fixed-length-resultant-string ,digits-in-fraction (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 OTS$CNVOUT converts to a character string. For OTS$CNVOUT, the D-G-or-H-float-pt-input-val argument is the address of a D-floating number containing the value. For OTS$CNVOUT_G, the D-G-or-H-float-pt-input-val argument is the address of a G-floating number containing the value. For OTS$CNVOUT_H, the D-G-or-H-float-pt-input-val argument is the address of an H-floating number containing the value.

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

Output string into which OTS$CNVOUT writes the character string result of the conversion. The fixed-length-resultant-string argument is the address of a descriptor pointing to the output string.
**OTS$ Routines**
**OTS$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.
- **OTS$$_OUTCONERR**  Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks (*).
OTS$ Routines
OTS$CVT_L_TB

OTS$CVT_L_TB
Convert an Unsigned Integer to Binary Text

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

Format

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

Returns

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

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

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

OTS–5
value of the integer to be converted is also zero, OTS$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

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>OTS$_OUTCONERR</td>
<td>Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks (*).</td>
</tr>
</tbody>
</table>
OTS$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

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

input-value-size
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 OTS$CVT_L_TI to insert a plus sign before the converted number. The flags-value argument is an unsigned longword containing the flags.

The caller flags are described in the following table:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Action if Set</th>
<th>Action if Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Insert a plus sign (+) before the first nonblank character in the output string.</td>
<td>Omit the plus sign.</td>
</tr>
</tbody>
</table>

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

Condition Values Returned

| SS$_NORMAL | Normal successful completion. |
| OTS$_OUTCONERR | Output conversion error. Either the result would have exceeded the fixed-length string or the input-value-size is not a valid value. The output string is filled with asterisks (*). |
OTS$CVT_L_TL
Convert Integer to Logical Text

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

Format

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

Returns

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

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

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

If bit 0 of longword-integer-value is set, OTS$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$_NORMAL        Normal successful completion.
OTS$_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).
Example

```
5 !+
! This is an example program
! showing the use of OTS$CVT_L_TL.
!-
VALUE% = 10
OUTSTR$ = ' 'CALL OTS$CVT_L_TL(VALUE%, OUTSTR$)
PRINT OUTSTR$
9 END
```

This BASIC example illustrates the use of OTS$CVT_L_TL. The output generated by this program is ‘F’.
OTS$ CVT_L_TO
Convert Unsigned Integer to Octal Text

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

Format

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

Returns

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

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

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

number-of-digits
OpenVMS usage: longword_signed
type: longword (signed)
access: read only
mechanism: by value

Minimum number of digits that OTS$CVT_L_TO generates when it converts the integer value to an octal ASCII text string. The number-of-digits argument is a signed longword containing the minimum number of digits. If it is omitted, the default is 1. If the actual number of significant digits in the octal ASCII text string is less than the minimum number of digits, OTS$CVT_L_TO inserts
leading zeros into the output string. If `number-of-digits` is 0 and `varying-input-value` is 0, OTS$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 OTS$CVT_L_TO. If it is omitted, the default is 4 (longword).

**Condition Values Returned**

- **SS$_NORMAL**
  Normal successful completion.
- **OTS$_OUTCONERR**
  Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks (*).
OTS$CVT_L_TU
Convert Unsigned Integer to Decimal Text

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

Format

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

number-of-digits
OpenVMS usage: longword_signed
type: longword (signed)
access: read only
mechanism: by value
Minimum number of digits in the ASCII text string that OTS$CVT_L_TU creates. The `number-of-digits` argument is a signed longword containing the minimum number. If the minimum number of digits is omitted, the default is 1.

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

`input-value-size`
- 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$_NORMAL**: Normal successful completion.
- **OTS$_OUTCONERR**: Output conversion error. Either the result would have exceeded the fixed-length string or the `input-value-size` is not a valid value. The output string is filled with asterisks (*).
OTS$CVT_L_TZ
Convert Integer to Hexadecimal Text

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

Format

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

Returns

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

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

Output string that OTS$CVT_L_TZ creates when it converts the integer value to a hexadecimal ASCII text string. The fixed-length-resultant-string argument is the address of a descriptor pointing to this ASCII text string. The string is assumed to be of fixed length (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 OTS$CVT_L_TZ creates when it converts the integer. The number-of-digits argument is a signed longword containing this minimum number. If it is omitted, the default is 1. If the actual number of significant digits in the text string that OTS$CVT_L_TZ creates is less than this minimum number, OTS$CVT_L_TZ inserts leading zeros in the output string. If the minimum number of digits is zero and the integer
value to be converted is also zero, OTS$CVT_L_TZ writes a blank string to the output string.

**input-value-size**
- **OpenVMS usage:** longword_signed
- **type:** longword (signed)
- **access:** read only
- **mechanism:** by value

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

**Condition Values Returned**

<table>
<thead>
<tr>
<th>Condition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>OTS$OUTCONERR</td>
<td>Output conversion error. The result would have exceeded the fixed-length string; the output string is filled with asterisks (*).</td>
</tr>
</tbody>
</table>

**Example**

```ada
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,
  DESCRIPTOR (CLASS => S));
begin
  CONVERT_TO_HEX (INTVALUE, HEXSTRING);
  PUT_LINE("This is the value of HEXSTRING");
  PUT_LINE(HEXSTRING);
end;
```

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

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

Returns

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 OTS$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:

```
[<blanks>] [+][<digits>][.][<digits>]
{[<digits>] [Ee][Dd][Qq][<digits>] [+] [-] [blanks>][<digits>]
```

VM-0710A-A1
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:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blanks</td>
<td>One or more blanks</td>
</tr>
<tr>
<td>digits</td>
<td>One or more decimal digits</td>
</tr>
</tbody>
</table>

**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 OTS$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:

<table>
<thead>
<tr>
<th>Routine</th>
<th>floating-point-value Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$CVT_T_D</td>
<td>D-floating</td>
</tr>
<tr>
<td>OTS$CVT_T_F</td>
<td>F-floating</td>
</tr>
<tr>
<td>OTS$CVT_T_G</td>
<td>G-floating</td>
</tr>
<tr>
<td>OTS$CVT_T_H</td>
<td>H-floating</td>
</tr>
</tbody>
</table>

**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:

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

If you omit the **flags-value** argument, OTS$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:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Number of Bits Returned</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$CVT_T_D</td>
<td>8</td>
<td>Byte (unsigned)</td>
</tr>
<tr>
<td>OTS$CVT_T_F</td>
<td>8</td>
<td>Byte (unsigned)</td>
</tr>
<tr>
<td>OTS$CVT_T_G</td>
<td>11</td>
<td>Word (unsigned)</td>
</tr>
<tr>
<td>OTS$CVT_T_H</td>
<td>15</td>
<td>Word (unsigned)</td>
</tr>
</tbody>
</table>

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$NORMAL Normal successful completion.
OTSS$INPCONERR Input conversion error; an invalid character in
the input string, or the value is outside the range
that can be represented. The floating-point-
value and extension-bits 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 OTSSCVT_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 'I,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 = OTSSCVT_T_F(T(I),A,%VAL(5),,)
   TYPE 'A Output of OTSCVT_T_F is ',A
   TYPE 'A
2 CONTINUE
END
This Fortran example demonstrates the use of OTS$CVT_T_F. The output generated by this program is as follows:

Input string 1 is 1234567+23  
Output of OTSCVT_T_F is 1.2345669E+24

Input string 2 is 8.786534+3  
Output of OTSCVT_T_F is 8786.534

Input string 3 is -983476E-3  
Output of OTSCVT_T_F is -9.8347599E-03

Input string 4 is -23.734532  
Output of OTSCVT_T_F is -23.73453

Input string 5 is 45  
Output of OTSCVT_T_F is 45000.00
OTS$ Routines
OTS$CVT_TB_L

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 OTS$CVT_TB_L uses to determine how to interpret blanks within the input string. The flags-value argument contains this user-supplied flag.

OTS$CVT_TB_L defines the flag as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Action if Set</th>
<th>Action if Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ignore blanks.</td>
<td>Interpret blanks as zeros.</td>
</tr>
</tbody>
</table>

If you omit the flags-value argument, OTS$CVT_TB_L defaults all flags to clear.

Condition Values Returned

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

Example

```
OPTION & TYPE = EXPLICIT
! This program demonstrates the use of OTS$CVT_TB_L from BASIC.
! Several binary numbers are read and then converted to their
! integer equivalents.
!-
!+ DECLARATIONS
!-
DECLARE STRING BIN_STR
DECLARE LONG BIN_VAL, I, RET_STATUS
DECLARE LONG CONSTANT FLAGS = 17 ! 2^0 + 2^4
EXTERNAL LONG FUNCTION OTS$CVT_TB_L (STRING, LONG, &
    LONG BY VALUE, LONG BY VALUE)
```
This BASIC example program demonstrates how to call OTS$CVT_TB_L to convert binary text to a longword integer.

The output generated by this BASIC program is as follows:

- 1111 treated as a binary number equals 15
- 1 111 treated as a binary number equals 15
- 1011011 treated as a binary number equals 91
- 11111111 treated as a binary number equals 255
- 00000000 treated as a binary number equals 0
OTS$CVT_TI_L
Convert Signed Integer Text to Integer

The Convert Signed Integer Text to Integer routine converts an ASCII text string representation of a signed decimal number to a signed integer value. The default size of the result is a longword.

**Format**

```
OTS$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 OTS$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>  
  -    ]
```

OTS$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 OTS$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
Number of bytes to be occupied by the value created when OTS$CVT_TI_L converts the ASCII text string to an integer value. The output-value-size argument contains the number of bytes 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 OTS$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 OTS$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, OTS$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 OTS$CVT_TI_L uses to determine how blanks and tabs are interpreted. The flags-value argument is an unsigned longword containing the value of the flags.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Action if Set</th>
<th>Action if Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ignore all blanks.</td>
<td>Ignore leading blanks but interpret blanks after the first legal character as zeros.</td>
</tr>
<tr>
<td>4</td>
<td>Ignore tabs.</td>
<td>Interpret tabs as invalid characters.</td>
</tr>
</tbody>
</table>

If you omit the flags-value argument, OTS$CVT_TI_L defaults all flags to clear.

Condition Values Returned

<table>
<thead>
<tr>
<th>SS$_NORMAL</th>
<th>Normal successful completion.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTS$_INPCONERR</td>
<td>Input conversion error. OTS$CVT_TI_L encountered an invalid character in the fixed-or-dynamic-input-string, an overflow of varying-output-value, or an invalid output-value-size. In the case of an invalid character or of an overflow, varying-output-value is set to zero.</td>
</tr>
</tbody>
</table>
OTS$CVT_TL_L
Convert Logical Text to Integer

The Convert Logical Text to Integer routine converts an ASCII text string representation of a FORTRAN-77 L format to a signed integer.

Format

OTS$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 OTS$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 OTS$CVT_TL_L input string is valid if it adheres to the following syntax:

\[
\begin{align*}
\text{<blanks>} \\
[\text{<blanks>}][] \begin{cases}
T \\
F \\
f 
\end{cases}
\text{[<characters>]}
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blanks</td>
<td>One or more blanks</td>
</tr>
<tr>
<td>characters</td>
<td>One or more of any character</td>
</tr>
</tbody>
</table>
OTS$ Routines
OTS$CVT_TL_L

varying-output-value
OpenVMS usage: varying_arg
type: unspecified
access: write only
mechanism: by reference

Signed integer that OTS$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.

OTS$CVT_TL_L returns –1 as the contents of the varying-output-value argument if the character denoted by “letter” is T or t. Otherwise, OTS$CVT_TL_L sets varying-output-value to zero.

output-value-size
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 OTS$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, OTS$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 OTS$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 OTS$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 ValuesReturned

SS$_NORMAL Normal successful completion.
OTS$_INPCONERR Input conversion error. OTS$CVT_TL_L encountered an invalid character in the fixed-or-dynamic-input-string or an invalid output-value-size. In the case of an invalid character varying-output-value is set to zero.
OTS$ Routines
OTS$CVT_TO_L

OTS$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

OTS$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 onlymechanism: by value

Arguments

fixed-or-dynamic-input-string
OpenVMS usage: char_string
type: character stringaccess: read onlymechanism: by descriptor, fixed-length or dynamic string

Input string containing the string representation of an unsigned octal value that OTS$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: unspecifiedaccess: write onlymechanism: by reference

Unsigned integer of specified size that OTS$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 onlymechanism: 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_TO_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 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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Action if Set</th>
<th>Action if Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ignore all blanks.</td>
<td>Interpret blanks as zeros.</td>
</tr>
</tbody>
</table>

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 fixed-or-dynamic-input-string, an overflow of varying-output-value, or an invalid output-value-size. In the case of an invalid character or of an overflow, varying-output-value is set to zero.

Example

```pli
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$CVT_TU_L converts the input string. The **output-value-size** argument contains the number of bytes in **varying-output-value**.
On VAX systems, valid values for the `output-value-size` argument are 1, 2, and 4. The value determines whether the integer value that OTS$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 OTS$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, OTS$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 OTS$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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Action if Set</th>
<th>Action if Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ignore all blanks.</td>
<td>Ignore leading blanks but interpret blanks after the first legal character as zeros.</td>
</tr>
<tr>
<td>4</td>
<td>Ignore tabs.</td>
<td>Interpret tabs as invalid characters.</td>
</tr>
</tbody>
</table>

If you omit the `flags-value` argument, OTS$CVT_TU_L defaults all flags to clear.

### Condition Values Returned

- **SS$$_N$$ORMAL**: Normal successful completion.  
- **OTS$$_INPCONERR**: Input conversion error. OTS$CVT_TU_L encountered an invalid character in the `fixed-or-dynamic-input-string`, overflow of `varying-output-value`, or an invalid `output-value-size`. In the case of an invalid character or of an overflow, `varying-output-value` is set to zero.
OTS$ Routines
OTS$CVT_TZ_L

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

<table>
<thead>
<tr>
<th>OTS$CVT_TZ_L</th>
<th>fixed-or-dynamic-input-string</th>
<th>varying-output-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[.output-value-size]</td>
<td>[.flags-value]</td>
</tr>
</tbody>
</table>

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, OTS$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 OTS$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:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Action if Set</th>
<th>Action if Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ignore all blanks.</td>
<td>Interpret blanks as zeros.</td>
</tr>
</tbody>
</table>

If you omit the `flags-value` argument, OTS$CVT_TZ_L defaults the flag to clear.

**Condition Values Returned**

- **SS$_NORMAL**: Normal successful completion.  
- **OTS$_INPCONERR**: Input conversion error. OTS$CVT_TZ_L encountered an invalid character in the fixed-or-dynamic-input-string, overflow of varying-output-value, or an invalid output-value-size. In the case of an invalid character or of an overflow, varying-output-value 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$CVT_TZ_L  
   EXTERNAL LONG CONSTANT OTS$_INPCONERR  
   INPUT "Enter hex numeric";HEXVAL$  
   RET_STAT% = OTS$CVT_TZ_L(HEXVAL$ , HEX% )  
   PRINT "Conversion error " IF RET_STAT% = OTS$_INPCONERR  
   PRINT "Decimal value of ";HEXVAL$;" is";HEX% &  
   IF RET_STAT% <> OTS$_INPCONERR

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

$ RUN HEX  
Enter hex numeric? A  
Decimal value of A is 10
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
```
OTS$DIVCx
Complex Division

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

Format

OTS$DIVC complex-dividend ,complex-divisor
OTS$DIVCD_R3 complex-dividend ,complex-divisor (VAX only)
OTS$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. OTS$DIVC returns an F-floating complex number. OTS$DIVCD_R3 returns a D-floating complex number. OTS$DIVCG_R3 returns a G-floating complex number.

Arguments

complex-dividend
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 OTS$DIVC, complex-dividend is an F-floating complex number. For OTS$DIVCD_R3, complex-dividend is a D-floating complex number. For OTS$DIVCG_R3, complex-dividend is a G-floating complex number.

complex-divisor
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 OTS$DIVC, complex-divisor is an F-floating complex number. For OTS$DIVCD_R3, complex-divisor is a D-floating complex number. For OTS$DIVCG_R3, complex-divisor is a G-floating complex number.
OTS$ 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:
\[
\begin{align*}
    r &= \frac{ac + bd}{c^2 + d^2} \\
    i &= \frac{bc - ad}{c^2 + d^2}
\end{align*}
\]
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+
   This Fortran example forms the complex quotient of two complex numbers using OTSSDIVC and the Fortran random number generator RAN.
   Declare \(Z1, Z2, Z_Q,\) and OTSSDIVC as complex values.
   OTSSDIVC will return the complex quotient of \(Z1\) divided by \(Z2: Z_Q = OTSSDIVC( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)), %VAL(REAL(Z2)), %VAL(AIMAG(Z2)))\)
   ```fortran
   COMPLEX Z1,Z2,Z_Q,OTS$DIVC
   Z1 = (8.0,4.0)  
   Z2 = (1.0,1.0)  
   Z_Q = OTS$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$DIVC. The output generated by this program is as follows:
   The complex quotient of (8.000000,4.000000) divided by (1.000000,1.000000) is (6.000000,-2.000000)
```
This Fortran example forms the complex quotient of two complex numbers by using OTS$DIVCG_R3 and the Fortran random number generator RAN.

Declare Z1, Z2, and Z_Q as complex values. OTS$DIVCG_R3 will return the complex quotient of Z1 divided by Z2:

\[ Z_Q = \frac{Z1}{Z2} \]

```
COMPLEX*16 Z1,Z2,Z_Q

C+ Generate a complex number.
C-  Z1 = (8.0,4.0)
C+ Generate another complex number.
C-  Z2 = (1.0,1.0)
C+ Compute the complex quotient of Z1/Z2.
C-  Z_Q = Z1/Z2
```

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

The complex quotient of (8.000000000000000,4.000000000000000) divided by (1.000000000000000,1.000000000000000) is (6.000000000000000,-2.000000000000000)
OTS$ Routines
OTS$DIV_PK_LONG

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 OTS$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.

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

\[
\text{precision-data} = \text{scale(packed-decimal-quotient)} + \text{scale(packed-decimal-divisor)} - \text{scale(packed-decimal-dividend)} - 31 + \text{prec(packed-decimal-dividend)}
\]

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.

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

\[
\text{scale-data} = 31 - \text{prec(packed-decimal-divisor)}
\]
OTS$ Routines
OTS$DIV_PK_LONG

Description

On VAX systems, before using this routine, you should determine whether it is best to use OTS$DIV_PK_LONG, OTS$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 OTS$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 OTS$DIV_PK_LONG or OTS$DIV_PK_SHORT. To determine this, you must examine the value of \textit{scale-data}. If \textit{scale-data} is less than or equal to 1, then you should use OTS$DIV_PK_LONG. If \textit{scale-data} is greater than 1, you should use OTS$DIV_PK_SHORT instead.

Condition Value Signaled

SS$_FLTDIV  
Fatal error. Division by zero.

Example

1

OPTION &
   TYPE = EXPLICIT

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

!+ 
! DECLARATIONS
!-

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

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

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

INPUT "Enter national debt: ";NATIONAL_DEBT
INPUT "Enter current population: ";POPULATION
OTSS Routines

OTS$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 OTS$DIV_PK_LONG or
! OTS$DIV_PK_SHORT may be used to perform the division.
! If b is less than or equal to 31, then the DIVP
! instruction may be used.
!
! scale-data is less than or equal to 1, so OTS$DIV_PK_LONG
! should be used instead of OTS$DIV_PK_SHORT.
!
!-

CALL OTS$DIV_PK_LONG( NATIONAL_DEBT, POPULATION, '30'W, PER_CAPITA_DEBT, &
'10'W, '6'W, '1'W)

PRINT "The per capita debt is ";PER_CAPITA_DEBT
END

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

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

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


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 OTS$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.

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

\[
\text{precision-data} = \text{scale}(\text{packed-decimal-quotient}) + \text{scale}(\text{packed-decimal-divisor}) - \text{scale}(\text{packed-decimal-dividend}) - 31 + \text{prec}(\text{packed-decimal-dividend})
\]

Description

On VAX systems, before using this routine, you should determine whether it is best to use OTS$DIV_PK_LONG, OTS$DIV_PK_SHORT, or the VAX instruction DIVP. To determine this, you must first calculate \( b \), where \( b \) is defined as follows:

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

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

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 OTS$DIV_PK_LONG or OTS$DIV_PK_SHORT. To determine this, you must examine the value of scale-data. If scale-data is less than or equal to 1, then you should use OTS$DIV_PK_LONG. If scale-data is greater than 1, you should use OTS$DIV_PK_SHORT instead.

Condition Value Signaled

SS$_FLTDIV Fatal error. Division by zero.
OTS$ Routines
OTS$MOVE3

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

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

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

For more information, see the description of the MOVC3 instruction in the VAX Architecture 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.
The OTS$MOVE5 routine performs the same function as the VAX MOV C5 instruction except that the `longword-int-source-length` and `longword-int-dest-length` arguments are longword integers rather than word integers. When called from the JSB entry point, the register outputs of OTS$MOVE5_R5 follow the same pattern as those of the MOV C5 instruction:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>Number of unmoved bytes remaining in source string</td>
</tr>
<tr>
<td>R1</td>
<td>Address of one byte beyond the source string</td>
</tr>
<tr>
<td>R2</td>
<td>0</td>
</tr>
<tr>
<td>R3</td>
<td>Address of one byte beyond the destination string</td>
</tr>
<tr>
<td>R4</td>
<td>0</td>
</tr>
<tr>
<td>R5</td>
<td>0</td>
</tr>
</tbody>
</table>

For more information, see the description of the MOV C5 instruction in the VAX Architecture Reference Manual. See also the routine LIB$MOV C5, which is a callable version of the MOV C5 instruction.

### Condition Values Returned

None.
OTS$ Routines
OTS$MULCx

OTS$MULCx
Complex Multiplication

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

Format

OTS$MULCD_R3 complex-multiplier ,complex-multiplicand (VAX only)
OTS$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_float complex, G_float complex
access: write only
mechanism: by value

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

Arguments

complex-multiplier
OpenVMS usage: complex_number
type: D_float complex, G_float complex
access: read only
mechanism: by value

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

complex-multiplicand
OpenVMS usage: complex_number
type: D_float complex, G_float complex
access: read only
mechanism: by value

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

Description

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

The complex product is computed as follows:

1. Let (a,b) represent the complex multiplier.
2. Let \((c,d)\) represent the complex multiplicand.
3. Let \((r,i)\) represent the complex product.

The results of this computation are as follows:

\[
\begin{align*}
(a, b) \cdot (c, d) &= (ac - bd) + \sqrt{-1}(ad + bc) \\
\text{Therefore: } r &= ac - bd \\
\text{Therefore: } i &= ad + bc
\end{align*}
\]

On Alpha systems, some restrictions apply when linking OTS$MULCG_R3. See Chapter 1 for more information about these restrictions.

**Condition Values Signaled**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$$_F$$LTOVF_F</td>
<td>Floating value overflow can occur.</td>
</tr>
<tr>
<td>SS$$_R$$ROPRAND</td>
<td>Reserved operand. OTS$MULCx encountered a floating-point reserved operand because of incorrect user input. A floating-point reserved operand is a floating-point datum with a sign bit of 1 and a biased exponent of zero. Floating-point reserved operands are reserved for future use by Compaq.</td>
</tr>
</tbody>
</table>

**Example**

This Fortran example forms the product of two complex numbers using OTS$MULCD_R3 and the Fortran random number generator RAN.

Declare Z1, Z2, and Z_Q as complex values. OTS$MULCD_R3 returns the complex product of Z1 times Z2:

\[
Z_Q = Z1 \times Z2
\]

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

The complex product of \((8.000000000000000,4.000000000000000)\) times \\
\((2.000000000000000,3.000000000000000)\) is \\
\((4.000000000000000,32.000000000000000)\)
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

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

\[ x^y = \exp(y \cdot \log(x)) \]

In this example, x and y are type COMPLEX.

On Alpha systems, some restrictions apply when linking OTS$POWCC or OTS$POWCGCG_R3. See Chapter 1 for more information about these restrictions.

Condition Values Signaled

- MTH$_INVAR$GMA$T$: Invalid argument in math library. Base is (0.,0.).
- MTH$_FLOOVE$MAT: Floating-point overflow in math library.
- SS$_ROPR$AND: Reserved operand.

Examples

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

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

The output generated by this program is as follows:

The value of (2.000000,3.000000)** (1.000000,2.000000) is
(-0.4639565,-0.1995301)
This Fortran example raises a complex base to a complex power using OTS$POWCGCG_R3.

Declare Z1, Z2, and Z3 as complex values. OTS$POWCGCG_R3 returns the complex result of Z1**Z2: Z3 = Z1**Z2.

```
COMPLEX*16 Z1,Z2,Z3

Z1 = (2.0,3.0)
Z2 = (1.0,2.0)
Z3 = Z1**Z2

TYPE 1,Z1,Z2,Z3
1 FORMAT(' The value of (',F11.8,',',F11.8,')**(',F11.8,+ ',',F11.8,') is (',F11.8,',',F11.8,').')
END
```

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

The value of ( 2.00000000, 3.00000000)**( 1.00000000, 2.00000000) is (-0.46395650,-0.46395650).
OTS$POWCxJ
Raise a Complex Base to a Signed Longword Integer Exponent

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

Format

OTS$POWCJ  complex-base,longword-integer-exponent
OTS$POWCDJ_R3  complex-base,longword-integer-exponent (VAX only)
OTS$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. OTS$POWCJ returns an F-floating complex number. OTS$POWCDJ_R3 returns a D-floating complex number. OTS$POWCGJ_R3 returns a G-floating complex number. In each format, the result and base are of the same data type.

Arguments

complex-base
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 OTS$POWCJ, complex-base is an F-floating complex number. For OTS$POWCDJ_R3, complex-base is a D-floating complex number. For OTS$POWCGJ_R3, complex-base is a G-floating complex number.

longword-integer-exponent
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$POWxCi

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:

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>&gt; 0</td>
<td>The product of (base**2^i), where i is each nonzero bit in longword-integer-exponent.</td>
</tr>
<tr>
<td>(0.,0.)</td>
<td>≤ 0</td>
<td>Undefined exponentiation.</td>
</tr>
<tr>
<td>Not (0.,0.)</td>
<td>&lt; 0</td>
<td>The product of (base**2^i), where i is each nonzero bit in longword-integer-exponent.</td>
</tr>
<tr>
<td>Not (0.,0.)</td>
<td>0</td>
<td>(1.0,0.0)</td>
</tr>
</tbody>
</table>

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-
C+ COMPLEX Z1,Z3,OTS$POWCJ
C+ INTEGER Z2
C+
C Generate a complex base.
C-
C+ Z1 = (2.0,3.0)
C+
C Generate an integer power.
C-
C+ Z2 = 2
C+
C Compute the complex value of Z1**Z2.
C-
C+ Z3 = OTS$POWCJ( %VAL(REAL(Z1)), %VAL(AIMAG(Z1)), %VAL(Z2) )
C+ TYPE 1,Z1,Z2,Z3
C+ FORMAT(’ The value of (’ ,F10.8,’ ’,F11.8,’’)**’,I1,’ is
C+ ’,F11.8,’ ’,F12.8,’).’) END
C
The output generated by this Fortran program is as follows:
The value of (2.00000000, 3.00000000)**2 is
(-5.00000000, 12.00000000).
```

OTS–56
**OTS$POWDD**  
*Raise a D-Floating Base to a D-Floating Exponent*

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

**Format**

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

**Returns**

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**

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

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

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

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0</td>
<td>&gt; 0</td>
<td>0.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>= 0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Any</td>
<td>Undefined exponentiation</td>
</tr>
</tbody>
</table>
OTS$ Routines
OTS$POWDD

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>&gt; 0</td>
<td>(2^{[\text{exponent}\times\log_2(\text{base})]})</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>(2^{[\text{exponent}\times\log_2(\text{base})]})</td>
</tr>
</tbody>
</table>

Floating-point overflow can occur.

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

**Condition Values Signaled**

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

Raise a D-Floating Base to a Longword Exponent

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

Format

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

Returns

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

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

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

The floating-point result is as follows:

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>&gt; 0</td>
<td>Product of (base**2^i), where i is each nonzero bit position in longword-integer-exponent.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation.</td>
</tr>
</tbody>
</table>
OTS$ Routines
OTS$POWDJ

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>1.0/(base**2^i), where i is each nonzero bit position in longword-integer-exponent.</td>
</tr>
<tr>
<td>= 0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>&lt; 0</td>
<td>1.0/(base**2^i), where i is each nonzero bit position in longword-integer-exponent.</td>
</tr>
</tbody>
</table>

Floating-point overflow can occur.
Undefined exponentiation occurs if the base is zero and the exponent is zero or negative.

**Condition Values Signaled**

- **SS$_$FLTOVF**  
  Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
- **MTH$_$FLOOVEMAT**  
  Floating-point overflow in math library.
- **MTH$_$FLOUNDMAT**  
  Floating-point underflow in math library.
- **MTH$_$UNDEXP**  
  Undefined exponentiation. This error is signaled if D-floating-point-base is zero and longword-integer-exponent is zero or negative, or if the D-floating-point-base is negative.
OTS$POWDR
Raise a D-Floating Base to an F-Floating Exponent

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

Format

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

Returns

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

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

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

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

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&gt; 0</td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>= 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation</td>
</tr>
</tbody>
</table>
OTS$ Routines
OTS$POWDR

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>Any</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&gt; 0</td>
<td>(2^{[\text{exponent} \times \log_2(\text{base})]})</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>(2^{[\text{exponent} \times \log_2(\text{base})]})</td>
</tr>
</tbody>
</table>

Floating-point overflow can occur.

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

**Condition Values Signaled**

- **SS$_{FLTOVF}$**
  - Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.

- **MTH$_{FLOOVEMAT}$**
  - Floating-point overflow in math library.

- **MTH$_{FLOUNDMAT}$**
  - Floating-point underflow in math library.

- **MTH$_{UNDEXP}$**
  - Undefined exponentiation. This error is signaled if **D-float-point-base** is zero and **F-float-point-exponent** is zero or negative, or if the **D-float-point-base** is negative.
OTS$POWGG
Raise a G-Floating Base to a G-Floating Exponent

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

Format

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

Returns

OpenVMS usage: floating_point
type:       G_floating
access:     read 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 OTS$POWGG raises to a G-floating exponent. The G-floating-point-base argument is a G-floating number containing the base.

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

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

Description

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

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

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

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0</td>
<td>&gt; 0</td>
<td>0.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>= 0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Any</td>
<td>Undefined exponentiation</td>
</tr>
</tbody>
</table>
OTS$ Routines
OTS$POWGG

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>&gt; 0</td>
<td>$2^{(\text{exponent}\times\log_2(\text{base})]}$</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>$2^{(\text{exponent}\times\log_2(\text{base})]}$</td>
</tr>
</tbody>
</table>

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 OTS$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 $\text{G-floating-point-base}$ is zero and $\text{G-floating-point-exponent}$ is zero or negative, or if $\text{G-floating-point-base}$ is negative.

**Example**

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

OTS–64
This Fortran example uses OTS$POWGG to raise a G-floating base to a G-floating exponent.

The output generated by this example is as follows:

\[
\begin{align*}
8.0^{2.0} & \quad \text{IS} \quad 64.00000000000000 \\
9.0^{0.5} & \quad \text{IS} \quad 0.3333333333333333
\end{align*}
\]
OTS$ Routines
OTS$POWGJ

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:

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>&gt; 0</td>
<td>Product of (base**2^i), where i is each nonzero bit position in longword-integer-exponent.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation.</td>
</tr>
<tr>
<td>Base</td>
<td>Exponent</td>
<td>Result</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>(1.0/(\text{base}^{*2^i})), where (i) is each nonzero bit position in \text{longword-integer-exponent}.</td>
</tr>
<tr>
<td>= 0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>&lt; 0</td>
<td>(1.0/(\text{base}^{*2^i})), where (i) is each nonzero bit position in \text{longword-integer-exponent}.</td>
</tr>
</tbody>
</table>

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 OTS$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$\_FLOOVMAT**
  - Floating-point overflow in math library.

- **MTH$\_FLOUNDMAT**
  - Floating-point underflow in math library.

- **MTH$\_UNDEXP**
  - Undefined exponent. This error is signaled if \(\text{G-floating-point-base}\) is zero and \(\text{longword-integer-exponent}\) is zero or negative, or if \(\text{G-floating-point-base}\) is negative.
OTS$ Routines
OTS$POWHH_R3 (VAX Only)

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:

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&gt; 0</td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>= 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Any</td>
<td>Undefined exponentiation</td>
</tr>
</tbody>
</table>
## OTS$ Routines

### OTS$POWHH_R3 (VAX Only)

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>&gt; 0</td>
<td>$2^{\text{exponent}-\text{log}_2(\text{base})}$</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>$2^{\text{exponent}-\text{log}_2(\text{base})}$</td>
</tr>
</tbody>
</table>

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$_F$LTOVF**: Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
- **MTH$_F$LOOEMAT**: Floating-point overflow in math library.
- **MTH$_F$LOUNDMAT**: Floating-point underflow in math library.
- **MTH$_U$NDEXP**: Undefined exponentiation. This error is signaled if \( \text{H-floating-point-base} \) is zero and \( \text{H-floating-point-exponent} \) is zero or negative, or if the \( \text{H-floating-point-base} \) is negative.

### Example

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

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

The output generated by this program is as follows:

```
9877356535.0**-0.5837653 IS 1.463779145994628343598205427E-0006
```
OTS$ Routines
OTS$POWHJ_R3 (VAX Only)

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:       read 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:

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>&gt; 0</td>
<td>Product of ( (\text{base}^{*2^i}) ), where ( i ) is each nonzero bit position in longword-integer-exponent.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation.</td>
</tr>
</tbody>
</table>
## OTS$ Routines

OTS$POWHJ_R3 (VAX Only)

### Table: OTS$POWHJ_R3 Routines

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>( \frac{1.0}{(\text{base}^{2^i})} ), where ( i ) is each nonzero bit position in \text{longword-integer-exponent}.</td>
</tr>
<tr>
<td>= 0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>&lt; 0</td>
<td>( \frac{1.0}{(\text{base}^{2^i})} ), where ( i ) is each nonzero bit position in \text{longword-integer-exponent}.</td>
</tr>
</tbody>
</table>

Floating-point overflow can occur.

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

### Condition Values Signaled

- **SS$_FLTOVF**
  - Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.
- **MTH$_FLOOVMAT**
  - Floating-point overflow in math library.
- **MTH$_FLONDMAT**
  - Floating-point underflow in math library.
- **MTH$_UNDEXP**
  - Undefined exponentiation. This error is signaled if \( H\text{-floating-point-base} \) is zero and \( \text{longword-integer-exponent} \) is zero or negative, or if the \( H\text{-floating-point-base} \) is negative.
OTS$ Routines
OTS$POWII

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

- **SS$_FLTOVF**
  Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.

- **MTH$_UNDEXP**
  Undefined exponentiation. This error is signaled if `word-integer-base` is zero and `word-integer-exponent` is zero or negative, or if `word-integer-base` is negative.
OTS$ Routines
OTS$POWJJ

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SS$_FLTDIV</strong></td>
<td>Arithmetic trap. This error is signaled by the hardware if a floating-point division by zero occurs.</td>
</tr>
<tr>
<td><strong>SS$_FLTOVF</strong></td>
<td>Arithmetic trap. This error is signaled by the hardware if a floating-point overflow occurs.</td>
</tr>
<tr>
<td><strong>MTH$_UNDEXP</strong></td>
<td>Undefined exponentiation. This error is signaled if \texttt{longword-integer-base} is zero and \texttt{longword-integer-exponent} is zero or negative, or if \texttt{longword-integer-base} is negative.</td>
</tr>
</tbody>
</table>
OTS$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:  read 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$_{\text{UNEXP}}$  Both the base and exponent values are zero.
OTS$ Routines
OTS$POWRD

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-float to a D-float 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:

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0</td>
<td>&gt; 0</td>
<td>0.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>= 0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation</td>
</tr>
</tbody>
</table>
### OTS$ Routines

#### OTS$POWRD

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>Any</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&gt; 0</td>
<td>(2^{\text{exponent} \cdot \log_2(\text{base})})</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>(2^{\text{exponent} \cdot \log_2(\text{base})})</td>
</tr>
</tbody>
</table>

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

### Example

```fortran
REAL*4 X
REAL*8 Y,RESULT,OTS$POWRD
INTEGER M(2)
EQUIVALENCE (M(1),Y)
X = 9768.0
Y = -0.587436654545

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

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

This Fortran example uses OTS$POWRD to raise an **F-floating base** to a **D-floating exponent**. Notice the difference in the precision of the result produced by this routine in comparison to the result produced by OTS$POWRR. The output generated by this program is as follows:

- 9768.0**9.0 IS 8.0956338648832906E+35
- 7689.0**-0.587436654545 IS 5.2155199252836588E-03
OTS$ Routines
OTS$POWRJ

**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: read 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:

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>&gt; 0</td>
<td>Product of (base**2^i), where i is each nonzero bit position in <strong>longword-integer-exponent</strong>.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation.</td>
</tr>
<tr>
<td>Base</td>
<td>Exponent</td>
<td>Result</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>1.0/(base**2^i), where i is each nonzero bit position in longword-integer-exponent.</td>
</tr>
<tr>
<td>= 0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>&lt; 0</td>
<td>1.0/(base**2^i), where i is each nonzero bit position in longword-integer-exponent.</td>
</tr>
</tbody>
</table>

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 OTS$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 F-floating-point-base is zero and longword-integer-exponent is zero or negative, or if F-floating-point-base is negative.
OTS$ POWRR
Raise an F-Floating Base to an F-Floating Exponent

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

Format

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

Returns

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:

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>= 0</td>
<td>&gt; 0</td>
<td>0.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>= 0</td>
<td>&lt; 0</td>
<td>Undefined exponentiation</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Any</td>
<td>Undefined exponentiation</td>
</tr>
</tbody>
</table>
### OTS$ Routines

#### OTS$POWRR

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>&gt; 0</td>
<td>$2^{\text{exponent} \cdot \log_2(\text{base})}$</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>&lt; 0</td>
<td>$2^{\text{exponent} \cdot \log_2(\text{base})}$</td>
</tr>
</tbody>
</table>

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 OTS$POWRR. 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$_FLOOEMAT**
  - Floating-point overflow in math library.

- **MTH$_FLOUNDMAT**
  - Floating-point underflow in math library.

- **MTH$_UNDEXP**
  - Undefined exponentiation. This error is signaled if \textbf{F-floating-point-base} is zero and \textbf{F-floating-point-exponent} is zero or negative, or if \textbf{F-floating-point-base} is negative.

### Example

```fortran
C+  C This Fortran example demonstrates the use
C+  C of OTS$POWRR, which raises an F-floating
C+  C point base to an F-floating point power.
C-
REAL*4 X,Y,RESULT,OTSSPOWRR
X = 8.0
Y = 2.0

C+  C The arguments of OTS$POWRR are passed by value.
C-
RESULT = OTS$POWRR(%VAL(X),%VAL(Y))
TYPE *,’ 8.0**2.0 IS ’,RESULT
X = 9.0
Y = -0.5

C+  C In Fortran, OTS$POWRR is indirectly called by simply
C+  C using the exponentiation operator.
C-
RESULT = X**Y
TYPE *,’ 9.0**-0.5 IS ’,RESULT
END
```

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

- $8.0^{2.0}$ is $64.00000$
- $9.0^{-0.5}$ is $0.333333$
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

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

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>&gt; 0</td>
<td>Product of (base*2^i), where i is each nonzero bit position in longword-integer-exponent.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
<tr>
<td>= 0</td>
<td>= 0</td>
<td>Undefined exponentiation.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>= 0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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

Condition Values Signaled

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

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

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

Format

OTS$SCOPY_DXDX source-string,destination-string

Corresponding JSB Entry Point

OTS$SCOPY_DXDX6

Returns

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

OTS$SCOPY_DXDX copies a source string to a destination string. It passes the source string by descriptor. If the length of the source string is greater than the length of the destination string, OTS$SCOPY_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 OTS$SCOPY_DXDX depend on the descriptor class of the destination string. The following table describes these actions for each descriptor class:

<table>
<thead>
<tr>
<th>Descriptor Class</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>S, Z, SD, A, NCA</td>
<td>Copy the source string. If needed, space fill or truncate on the right.</td>
</tr>
<tr>
<td>D</td>
<td>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.</td>
</tr>
<tr>
<td>VS</td>
<td>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 (CURLEN) to the actual number of bytes copied.</td>
</tr>
</tbody>
</table>

**Condition Values Signaled**

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

OTS$SCOPY_R_DX

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 (CURLEN) is rewritten.

Description

OTS$SCOPY_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, OTS$SCOPY_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 OTS$SCOPY_R_DX depend on the descriptor class of the destination string. The following table describes these actions for each descriptor class:

<table>
<thead>
<tr>
<th>Descriptor Class</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>S, Z, SD, A, NCA</td>
<td>Copy the source string. If needed, space fill or truncate on the right.</td>
</tr>
<tr>
<td>D</td>
<td>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.</td>
</tr>
<tr>
<td>VS</td>
<td>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 (CURLEN) field to the actual number of bytes copied.</td>
</tr>
</tbody>
</table>
Condition Values Signaled

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

Example

A Fortran example that demonstrates the manipulation of dynamic strings appears at the end of OTS$SGET1_DD. This example uses OTS$SCOPY_R_DX, OTS$SGET1_DD, and OTS$SFREE1_DD.
OTSSFREE1_DD
Strings, Free One Dynamic

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

Format

    OTSSFREE1_DD  dynamic-descriptor

Corresponding JSB Entry Point

    OTSSFREE1_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

    OTSSFREE1_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

    OTS$_FATINTERR     Fatal internal error.

Example

    A Fortran example that demonstrates the manipulation of dynamic strings appears at the end of OTSSGET1_DD. This example uses OTSSFREE1_DD, OTSSGET1_DD, and OTSSCOPY_R_DX.
The Free n Dynamic Strings routine takes as input a vector of one or more dynamic string areas and returns them to free storage.

Format

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

Corresponding JSB Entry Point

```
OTS$SFREEN_DD6
```

Returns

None.

Arguments

**descriptor-count-value**
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 OTS$SFREEN_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

OTS$SFREEN_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

```
OTS$_FATINTERR  Fatal internal error.
```
OTS$SGET1_DD
Strings, Get One Dynamic

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

Format

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

Corresponding JSB Entry Point

OTS$SGET1_DD_R6

Returns

None.

Arguments

word-integer-length-value
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
word-integer-length-value 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

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

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

**OTS$SGET1_DD**

### Condition Values Signaled

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

### Example

```c
PROGRAM STRING_TEST

C+ This program demonstrates the use of some dynamic string manipulation routines.

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+ Initialize the output descriptor. It should be empty.

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

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

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

C+ Initialize a temporary descriptor.

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

C+ Prompt the user.

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

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

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

OTS--94
The user has typed Ctrl/Z. Output the data we read.

Free the storage allocated to the dynamic strings.

END

This Fortran example program demonstrates the manipulation of dynamic strings using OTS$SGET1_DD, OTS$SFREE1_DD, and OTS$SCOPY_R_DX.
Index

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_TLS 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
OTSSDIVCD_R3 routine, OTS–37
OTSSDIVC_x routine, OTS–37
OTSSDIVC_pk_long routine, OTS–40
OTSSDIVC_pk_short routine, OTS–44
OTSSMOVE3 routine, OTS–46
OTSSMOVE5 routine, OTS–48
OTSSMULCD_R3 routine, OTS–50
OTSSMULCG_R3 routine, OTS–50
OTSSpowcx_x routine, OTS–52
OTSSpowx_j routine, OTS–55
OTSSPOWDD routine, OTS–57
OTSSPOWDJ routine, OTS–59
OTSSPOWDLU routine, OTS–84
OTSSPOWDR routine, OTS–61
OTSSPOWWG routine, OTS–63
OTS$POWGj routine, OTS–66
OTS$POWGLU routine, OTS–84
OTS$POWHH_R3 routine, OTS–68
OTS$POWHJ_R3 routine, OTS–70
OTS$POWHLU_R3 routine, OTS–84
OTS$POWII routine, OTS–72
OTS$POWJJ routine, OTS–74
OTS$POWLULU routine, OTS–76
OTS$POWRD routine, OTS–78
OTS$POWRJ routine, OTS–80
OTS$POWRLU routine, OTS–84
OTS$POWRRR routine, OTS–82
OTS$SCOPY_DXDX routine, OTS–86
OTS$SCOPY_R_DX routine, OTS–88
OTS$SGET1_DD routine, OTS–93
OTS$SGET1_DD routine, OTS–93

R
Run-time library routines
genral 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