# HP OpenVMS/Hangul RTL Korean Processing (HSY$) Manual

Order Number: BA322-90027

**May 2005**

This manual documents the library routines contained in the HSY$ facility of the OpenVMS/Hangul Run-Time Library.

| | |
|---|---|
| **Revision/Update Information:** | This document supersedes the Introduction to the Multi-byte Processing Run Time Library HSYSHR manual, Version 6.0 |
| **Software Version:** | OpenVMS/Hangul I64 Version 8.2<br>OpenVMS/Hangul Alpha Version 7.3-2 |

**Hewlett-Packard Company**
**Palo Alto, California**

Printed in Singapore

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

# Contents

## Tables

# Preface

This manual provides users of the HP OpenVMS/Hangul operating system with detailed usage and reference information on library routines supplied in the HSY$ facility of the OpenVMS/Hangul Run-Time Library for Korean processing.

## Intended Audience

This manual is intended for application programmers who want to write applications for Korean processing.

## Document Structure

This manual is organized into two parts as follows:

- The introductory chapters provide reference material on specific types of HSY$ library routines and Korean processing concepts.

  Chapter 1 provides a brief overview of the HSY$ facility and lists the HSY$ routines and their functions.

  Chapter 2 provides an overview of the concept of Korean characters and their representation in the HP OpenVMS/Hangul operating system.

- The HSY$ Reference Section describes each library routine contained in the HSY$ Run-Time Library facility in OpenVMS/Hangul. This information is presented using the documentation format described in *OpenVMS Programming Interfaces: Calling a System Routine.*

## Associated Document

A description of how the Run-Time Library routines are accessed is presented in *OpenVMS Programming Interface: Calling a System Routine.* The HSY$ Run-Time Library routines can be used with other RTL facilities provided in OpenVMS and OpenVMS/Hangul. Descriptions of the other RTL facilities and their corresponding routines are presented in the following books:

- *OpenVMS/Hangul RTL Korean Screen Management (SMG$) Manual*
- *OpenVMS RTL Library (RTL$) Manual*
- *OpenVMS VAX RTL Mathematics (MTH$) Manual*
- *OpenVMS RTL General Purpose (OTS$) Manual*
- *OpenVMS RTL String Manipulation (STR$) Manual*

Application programmers using any programming language can refer to *Guide to Creating OpenVMS Modular Procedures* for writing modular and reentrant code, and *OpenVMS/Hangul User Guide* for understanding the DEC Hangul character set.

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

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

For additional information about HP OpenVMS products and services, visit the following World Wide Web address:

`http://www.hp.com/go/openvms`

## Conventions

The following conventions may be 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.)<br>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:<br>• Additional optional arguments in a statement have been omitted.<br>• The preceding item or items can be repeated one or more times.<br>• Additional parameters, values, or other information can be entered. |
| .<br>.<br>. | 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, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line. |
| { } | In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line. |
| **bold type** | Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason. |
| *italic type* | Italic type 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). |
| `Example` | This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX commands and pathnames, PC-based commands and folders, and certain elements of the C programming language. |
| UPPERCASE TYPE | Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege. |
| - | 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. |
| numbers | All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated. |

# 1
# INTRODUCTION

The OpenVMS/Hangul Korean Processing Run Time Library (or simply HSYSHR) is a library of prewritten, commonly-used routines that perform a wide variety of multi-byte Korean language processing operations. It represents the HSY$ facility of the OpenVMS/Hangul Run-Time Library. All HSY$ routines follow the OpenVMS Procedure Calling Standard. They are callable from any programming languages supported in OpenVMS/Hangul, thus increasing program flexibility.

## 1.1 Organization of the HSYSHR

Routines in HSYSHR are grouped according to the types of tasks they perform. Altogether, there are nine groups of routines. All routine names are prefixed by the facility code HSY$. Those routines prefixed by HSY$DX_ pass strings by descriptor, otherwise strings are passed by the address of the starting position of the string. Table 1–1 shows the nine groups of HSY$ routines.

**Table 1–1 HSYSHR routine groups**

| Group | Types of Tasks Performed |
|---|---|
| String Routines | Perform manipulation of strings containing multi-byte or mixed ASCII and multi-byte characters. |
| Read Write Routines | Perform read and write of ASCII and multi-byte characters in user buffers. |
| Pointer Routines | Perform character pointer manipulation. |
| Comparison Routines | Perform comparison of strings containing multi-byte or mixed ASCII multi-byte characters. |
| Searching Routines | Perform searching of substrings in buffer containing multi-byte or mixed ASCII and multi-byte characters. |
| Counting Routines | Perform counting of bytes and characters in buffer containing multi-byte or mixed ASCII and multi-byte characters. |
| Character Type Routines | Perform checking of different classes of local language symbols and characters. |
| Date Time Routines | Provide local language date time format. |
| Conversion Routines | Perform various multi-byte character specific conversion. |

Table 1–2 to Table 1–10 list all routines available for each of the aforementioned groups, followed by brief statements of the routines' functions.

**Table 1–2 String Routines**

| Routine Name | Function |
|---|---|
| HSY$CH_MOVE | Moves a substring from a specified source buffer to a specified destination buffer. |
| HSY$TRIM | Trims trailing one-byte and multi-byte spaces and TAB characters. |
| HSY$TRUNC | Returns the position of the first character that follows the truncated string. |
| HSY$DX_TRIM | Trims trailing one-byte and multi-byte spaces and TAB characters. |
| HSY$DX_TRUNC | Truncates the input string to the specified length. |

**Table 1–3 Read Write Routines**

| Routine Name | Function |
|---|---|
| HSY$CH_GCHAR | Reads the current character. |
| HSY$CH_GNEXT | Reads the current character. |
| HSY$CH_NEXTG | Reads the next character, skipping the current character. |
| HSY$CH_RCHAR | Reads the current character. |
| HSY$CH_RNEXT | Reads the current character. |
| HSY$CH_RPREV | Reads the previous character. |
| HSY$DX_RCHAR | Reads the current character. |
| HSY$DX_RNEXT | Reads the current character. |
| HSY$CH_PCHAR | Writes a specified character to the current position of a buffer. |
| HSY$CH_PNEXT | Writes a specified character to the current position of a buffer. |
| HSY$CH_WCHAR | Writes a specified character to the current position of a buffer. |
| HSY$CH_WNEXT | Writes a specified character to the current position of a buffer. |
| HSY$DX_WCHAR | Writes a specified character. |
| HSY$DX_WNEXT | Writes a specified character. |

**Table 1–4 Pointer Routines**

| Routine Name | Function |
|---|---|
| HSY$SKPC | Skips a specified character. |
| HSY$CH_CURR | Points to the first byte of the current character. |
| HSY$CH_NEXT | Points to the first byte of the next character. |

(continued on next page)

**Table 1–4 (Cont.) Pointer Routines**

| Routine Name | Function |
|---|---|
| HSY$CH_PREV | Points to the first byte of the previous character. |
| HSY$POS_CURR | Points to the first byte of the current character. |
| HSY$POS_NEXT | Points to the first byte of the next character. |
| HSY$POS_PREV | Points to the first byte of the previous character. |
| HSY$DX_SKPC | Skips a specified character. |
| HSY$DX_POS_CURR | Points to the first byte of the current character. |
| HSY$DX_POS_NEXT | Points to the first byte of the next character. |
| HSY$DX_POS_PREV | Points to the first byte of the previous character. |

**Table 1–5 Comparison Routines**

| Routine Name | Function |
|---|---|
| HSY$COMPARE | Compares two specified strings. |
| HSY$STR_EQUAL | Checks if two specified character strings are equal. |
| HSY$DX_STR_EQUAL | Checks if two specified character strings are equal. |

**Table 1–6 Searching Routines**

| Routine Name | Function |
|---|---|
| HSY$LOCC | Locates the position of the first occurrence of the specified character. |
| HSY$POSITION | Searches the first occurrence of a specified substring in the input string. |
| HSY$STR_SEARCH | Searches the first occurrence of a specified substring in the input string with conversion performed prior to comparing the characters. |
| HSY$STR_START | Checks if the specified substring is found in another input string and starts from the first byte of the input string. |
| HSY$DX_LOCC | Locates the position of the first occurrence of the specified character. |
| HSY$DX_POSITION | Searches the first occurrence of a substring in a specified string. |
| HSY$DX_STR_SEARCH | Searches the first occurrence of a specified substring in the input string. |
| HSY$DX_STR_START | Checks if the specified substring is found in another input string and starts from the first byte of the input string. |

**Table 1–7 Counting Routines**

| Routine Name | Function |
|---|---|
| HSY$CH_SIZE | Tells the byte length of the specified character. |
| HSY$CH_NCHAR | Returns the number of characters in a specified string. |
| HSY$CH_NBYTE | Counts the number of bytes of a character string. |
| HSY$DX_NOF_CHAR | Returns the number of characters in a specified number of bytes. |
| HSY$DX_NOF_BYTE | Counts the number of bytes of a character string. |

**Table 1–8 Character Type Routines**

| Routine Name | Function |
|---|---|
| HSY$IS_VALID | Checks if the input character is a valid multi-byte character. |
| HSY$IS_IDEOGRAPH | Checks if the input multi-byte character is an ideographic multi-byte character. |
| HSY$IS_DESCRIPTION | Checks if the input character is a multi-byte local language punctuation. |
| HSY$IS_TECHNICAL | Checks if the input character is a scientific or mathematical multi-byte symbol character. |
| HSY$IS_UNIT | Checks if the input character is a multi-byte standard unit symbol character. |
| HSY$IS_GENERAL | Checks if the input character is a multi-byte general symbol character. |
| HSY$IS_LINE_DRAWING | Checks if the input character is a multi-byte line drawing symbol character. |
| HSY$IS_DIGIT | Checks if the input character is a one-byte or multi-byte numeric digit. |
| HSY$IS_ROMAN | Checks if the input character is a one-byte or multi-byte English letter. |
| HSY$IS_GREEK | Checks if the input character is a multi-byte Greek letter. |
| HSY$IS_RUSSIAN | Checks if the input character is a multi-byte Russian letter. |
| HSY$IS_ALPHA | Checks if the input character is a Greek, Russian or Roman letter. |
| HSY$IS_UPPER | Checks if the input character is an upper case Greek, Russian or Roman letter. |
| HSY$IS_LOWER | Checks if the input character is a lower case Greek, Russian or Roman letter. |
| HSY$IS_HIRAGANA | Checks if the input character is a multi-byte Japanese Hiragana character. |
| HSY$IS_KATAKANA | Checks if the input character is a multi-byte Japanese Katakana character. |

(continued on next page)

**Table 1–8 (Cont.) Character Type Routines**

| Routine Name | Function |
|---|---|
| HSY$IS_KANA | Checks if the input character is a multi-byte Japanese Kana character. |
| HSY$IS_PARENTHESIS | Checks if the input character is a multi-byte parenthesis symbol character. |
| HSY$IS_LEFT_PARENTHESIS | Checks if the input character is a multi-byte left parenthesis symbol character. |
| HSY$IS_RIGHT_ PARENTHESIS | Checks if the input character is a multi-byte right parenthesis symbol character. |
| HSY$IS_NO_FIRST | Checks if the input character is a multi-byte "NO FIRST" character. |
| HSY$IS_NO_LAST | Checks if the input character is a multi-byte "NO-LAST" character. |

**Table 1–9 Date Time Routines**

| Routine Name | Function |
|---|---|
| HSY$DX_DATE_TIME | Returns the date and time in local language format. |
| HSY$DX_TIME | Returns the date and time of the system time in local language format. |

**Table 1–10 Conversion Routines**

| Routine Name | Function |
|---|---|
| HSY$CHG_KEISEN | Converts ’0’ to ’9’ and ’-’ to multi-byte line drawing characters. |
| HSY$CHG_GENERAL | Performs general multi-byte conversion. |
| HSY$CHG_KANA_HIRA | Converts Katakana characters to Hiragana characters. |
| HSY$CHG_KANA_KATA | Converts Hiragana characters to Katakana characters. |
| HSY$CHG_KANA_KANA | Toggles Kana characters to Hiragana or Katakana characters. |
| HSY$CHG_ROM_FULL | Converts half form ASCII to full form ASCII. |
| HSY$CHG_ROM_HALF | Converts full form ASCII to half form ASCII equivalence. |
| HSY$CHG_ROM_SIZE | Toggles the form (full form or half form) of the input character. |
| HSY$CHG_ROM_UPPER | Converts one-byte and multi-byte letters to upper case. |
| HSY$CHG_ROM_LOWER | Converts one byte and multi-byte letters to lower case. |
| HSY$CHG_ROM_CASE | Toggles the casing of one-byte and multi-byte letters of the input character. |

(continued on next page)

**Table 1–10 (Cont.) Conversion Routines**

| Routine Name | Function |
|---|---|
| HSY$TRA_KANA_HIRA | Converts Katakana character strings to Hiragana character strings. |
| HSY$TRA_KANA_KATA | Converts Hiragana character strings to Katakana character strings. |
| HSY$TRA_KANA_KANA | Toggles Kana character strings to Hiragana or Katakana characters. |
| HSY$TRA_ROM_FULL | Converts half form ASCII to full form ASCII. |
| HSY$TRA_ROM_HALF | Converts full form ASCII to half form ASCII equivalence. |
| HSY$TRA_ROM_SIZE | Toggles the form (full form or half form) of the input string. |
| HSY$TRA_ROM_UPPER | Converts one-byte and multi-byte letters to upper case. |
| HSY$TRA_ROM_LOWER | Converts one-byte and multi-byte letters to lower case. |
| HSY$TRA_ROM_CASE | Toggles the casing of one-byte and multi-byte letters found in the string. |
| HSY$TRA_SYMBOL | Converts the sequence of a one-byte character to a string of multi-byte symbols. |
| HSY$DX_TRA_KANA_HIRA | Converts Katakana character strings to Hiragana character strings. |
| HSY$DX_TRA_KANA_KATA | Converts Hiragana character strings to Katakana character strings. |
| HSY$DX_TRA_KANA_KANA | Toggles Kana character strings to Hiragana or Katakana character strings. |
| HSY$DX_TRA_ROM_FULL | Converts half form ASCII to full form ASCII. |
| HSY$DX_TRA_ROM_HALF | Converts full form ASCII to half form ASCII equivalence. |
| HSY$DX_TRA_ROM_SIZE | Toggles the form (full form or half form) of the input string. |
| HSY$DX_TRA_ROM_UPPER | Converts one-byte and multi-byte letters to upper case. |
| HSY$DX_TRA_ROM_LOWER | Converts one-byte and multi-byte letters to lower case. |
| HSY$DX_TRA_ROM_CASE | Toggles the casing of one-byte and multi-byte letters found in the input string. |
| HSY$DX_TRA_SYMBOL | Converts the sequence of a one-byte character to a string of multi-byte symbols. |

## 1.2 Features of HSYSHR

HSYSHR provides the following features and capabilities:

- HSYSHR performs a wide range of general multi-byte processing operations. You can call the HSY$ routines instead of writing your own code to perform the operation.
- Routines in HSYSHR follow the OpenVMS Procedure Calling Standard. It allows you to call any HSY$ routines from any programming language support in OpenVMS/Hangul, thus increasing program flexibility.

- Because all routines are shared, they take up less virtual address space of a process.
- When new versions of the HSYSHR are installed, you do not need to revise your calling program, and generally do not need to relink.

## 1.3 Linking with HSYSHR

Routines in HSYSHR execute entirely in the mode of the caller and are intended to be called in the user mode. To link your application that contains explicit calls to HSYSHR, use the following link command:

```
$ LINK program, SYS$LIBRARY:HSYIMGLIB.OLB/LIBRARY
```

# 2
# MULTI-BYTE CHARACTER CONCEPTS

This chapter describes some important concepts of multi-byte character that are used throughout the documentation.

## 2.1 What is Multi-byte Character?

DEC Hangul character set is implemented as a multi-byte character set containing Korean characters, punctuation marks and various kinds of symbols. Each multi-byte character refers to a two-byte character with the most significant bit of the first byte always set. In OpenVMS/Hangul operating system, the DEC Hangul character set is adopted, and all Korean characters are represented as multi-byte characters from the character set. For detailed discussion of the DEC Hangul character set, please refer to *OpenVMS/Hangul User Guide*.

## 2.2 Proper Character Boundary

In HSYSHR, most of the routines use characters as a processing entity contrary to conventional byte by byte processing. Some routines require the input character pointer pointing at the proper character boundary in the user buffer. "Pointing at the proper character boundary" means the character pointer should not point to the non-first-byte position of a multi-byte character.

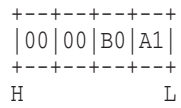
## 2.3 Full Form and Half Form Character

In the DEC Hangul character set, there is a set of two-byte ASCII characters. To distinguish them from the conventional one-byte 7-bit ASCII characters, the terms "full form" and "half form" characters are used. Full form characters refer to two-byte ASCII characters whereas half form characters refer to one-byte 7-bit ASCII characters. Conversion services between full form and half form characters are provided by the conversion routines in HSYSHR. In some applications where character matching requires treating the full form and half form characters equivalent, the user can call the searching routines in HSYSHR and specify the conversion flag argument. Note that uppercasing and lowercasing can both be applied to these full form characters.

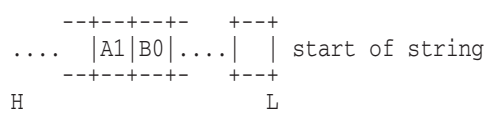
## 2.4 Multi-byte Character Unsigned Longword Representation

In HSYSHR, multi-byte character representation in single character argument is different from that found in the character string argument. Single character argument uses unsigned longword integer representation whereas characters in the string argument use the normal character string representation. An example is as follows. The two-byte character B0A1(hex) is represented differently in the following two cases.

Single character argument: (VMS Usage - longword_unsigned)




In a string argument: (VMS Usage - char_string)




The read routines in HSYSHR read the buffer with character string format and return the character read in unsigned longword format. The write routines write the character in unsigned longword format to the buffer. The character written will be in character string format.

# HSY$ Reference Section

This section provides detailed discussions of the routines provided in the Korean Processing Run Time Library HSYSHR.

# HSY$CH_MOVE

HSY$CH_MOVE moves a substring from a specified source buffer to a specified destination buffer.

## Format

HSY$CH_MOVE len,src,dst

## Arguments

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

The length in bytes of the substring to be moved.

**src**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the source buffer.

**dst**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the destination buffer.

## Description

This routine is multi-byte insensitive. If **len** is not specifying the proper multi-byte character boundary, e.g. it indicates the second byte of a two-byte character, then only half of the multi-byte character is moved to the last character of the destination string.

# HSY$DX_TRIM

HSY$DX_TRIM trims trailing one-byte and multi-byte spaces and TAB characters.

## Format

HSY$DX_TRIM dst,src,[len]

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | longword (unsigned) |
| access: | write only |
| mechanism: | by value |

## Arguments

**dst**

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | write only |
| mechanism: | by descriptor |

The destination string to store the trimmed string.

**src**

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | read only |
| mechanism: | by descriptor |

The source string that is to be converted.

**len**

| | |
|---|---|
| VMS Usage: | word_signed |
| type: | word integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the trimmed string. If this optional argument is not supplied, no length information of the trimmed string will be returned to the caller.

## Description

**dst** and **src** can contain one-byte and multi-byte characters.

## CONDITION VALUES RETURNED

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |

| | |
|---|---|
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRUNC

HSY$DX_TRUNC truncates the input string to the specified length.

## Format

HSY$DX_TRUNC dst,src,offset,[len]

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | longword (unsigned) |
| access: | write only |
| mechanism: | by value |

## Arguments

### dst

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | write only |
| mechanism: | by descriptor |

The specified destination string to store the truncated string.

### src

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | read only |
| mechanism: | by descriptor |

The specified source string to be truncated.

### offset

| | |
|---|---|
| VMS Usage: | word_signed |
| type: | word integer (signed) |
| access: | read only |
| mechanism: | by reference |

The offset in bytes from the starting position of the source string which indicates the position of the first character just after the truncated string. Note that this offset may not be on the proper character boundary, e.g. it may point to the second byte of a two-byte character.

### len

| | |
|---|---|
| VMS Usage: | word_signed |
| type: | word integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the truncated string. If this optional argument is not supplied, no length information of the truncated string will be returned to the caller.

## Description

The value returned in **len** may not necessarily be equal to the value specified in **offset** since **offset** may not be pointing at the first byte of a multi-byte character. In any case, the character indicated by **offset** will be treated as the first character that follows the truncated string.

## CONDITION VALUES RETURNED

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. Truncated string is further truncated due to insufficient space allocated in the destination string buffer. |
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$TRIM

HSY$TRIM trims trailing one-byte and multi-byte spaces and TAB characters.

## Format

HSY$TRIM str,len

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The offset in bytes from the starting position of the input string which indicates the position of the terminating character of the trimmed string. If the terminating character is a multi-byte character, the returned offset will be pointing to the first byte of the multi-byte character.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the input string to be trimmed.

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

The length in bytes of the input string.

## Description

**str** can contain one-byte and multi-byte characters.

# HSY$TRUNC

HSY$TRUNC returns the position of the first character that follows the truncated string.

## Format

HSY$TRUNC str,len,offset

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The offset in bytes which indicates the position of the first character just follows the truncated string. If this character is a multi-byte character, the offset will be pointing at the first byte of the multi-byte character.

## Arguments

### str

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the input string.

### len

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

The length in bytes of the input string.

### offset

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

The offset in bytes of the character just follows the truncated string. It may not be on the proper character boundary, e.g. it can point to the second byte of a two-byte character.

## Description

**str** can contain one-byte and multi-byte characters. This routine helps you to position **offset** to the proper character boundary. Its function is similar to routine **HSY$CH_CURR** but with different parameter interface.

# HSY$CH_GCHAR

HSY$CH_GCHAR reads the current character.

## Format

HSY$CH_GCHAR cur,end

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The current character.

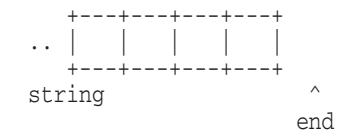
## Arguments

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the specified current character. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**end**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the string terminating position plus one as illustrated below:




## Description

This routine reads a character with end of buffer checking. FFFF (hex) will be returned when read past the end of buffer. If the current character is a one-byte 7-bit control character or one-byte 8-bit character (e.g. an 8-bit character followed by a 7-bit control character), the one-byte 7-bit or 8-bit character will be returned. No updating of current pointer is done since **cur** is passed by value.

# HSY$CH_GNEXT

HSY$CH_GNEXT reads the current character.

## Format

HSY$CH_GNEXT cur,end

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The current character.

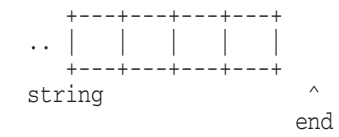
## Arguments

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: modify
mechanism: by reference

The address of the current position of the specified current character. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**end**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the string terminating position plus one as illustrated below:




## Description

This routine reads a character with end of buffer checking. FFFF (hex) will be returned when read past the end of buffer. If the current character is a one-byte 7-bit control character or one-byte 8-bit character (e.g. an 8-bit character followed by a 7-bit control character), the one-byte 7-bit or 8-bit character will be returned. Updating of the current pointer is done. After the read action, **cur** will be updated to the next character position pointing at the proper character boundary. This routine is useful for successive character reading.

# HSY$CH_NEXTG

HSY$CH_NEXTG reads the next character, skipping the current character.

## Format

HSY$CH_NEXTG cur,end

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The next character.

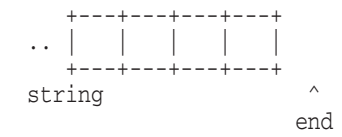
## Arguments

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: modify
mechanism: by reference

The address of the current position of the specified current character. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**end**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the string terminating position plus one as illustrated below:




## Description

This routine reads the next character, skipping the current character. FFFF (hex) will be returned when read past the end of buffer. If the next character is a one-byte 7-bit control character or one-byte 8-bit character (e.g. an 8-bit character followed by a 7-bit control character), the one-byte 7-bit or 8-bit character will be returned. Updating of the current pointer is done. After the read action, **cur** will be updated to the next character position pointing at the proper character boundary.

# HSY$CH_PCHAR

HSY$CH_PCHAR writes a specified character to the current position of a buffer.

## Format

HSY$CH_PCHAR chr,cur,end

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

Either 1 or 0 is returned as status.

1 - Input character is successfully written to the specified position of the string.

0 - Input character is not written to the specified position of the string.

## Arguments

### chr

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

The character to be written to the specified current position of the string.

### cur

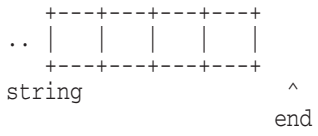
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the string where the input character is to be written to.

### end

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the string terminating position plus one as illustrated below:



## Description

This routine writes a character to a specified position. End of buffer checking is performed to make sure there is enough space in the buffer for the character to be written since **chr** can be a multi-byte character.

# HSY$CH_PNEXT

HSY$CH_PNEXT writes a specified character to the current position of a buffer.

## Format

HSY$CH_PNEXT chr,cur,end

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

Either 1 or 0 is returned as status.

1 - Input character is successfully written to the specified position of the string.

0 - Input character is not written to the specified position of the string.

## Arguments

### chr

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

The character to be written to the specified current position of the string. Note that the input character can either be one- or two-byte character.

### cur

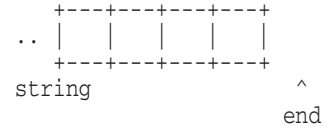
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: modify
mechanism: by reference

The address of the current position of the string where the input character is to be written to.

### end

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the string terminating position plus one as illustrated below:



## Description

This routine writes a character to a specified position. End of buffer checking is performed to make sure there is enough space in the buffer for the character to be written since **chr** can be a multi-byte character. Note that **cur** is updated. It points to the next character position after the write action. This routine is useful for successive writing of character to a buffer.

# HSY$CH_RCHAR

HSY$CH_RCHAR reads the current character.

## Format

HSY$CH_RCHAR cur

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The current character.

## Arguments

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the specified current character. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

## Description

This routine reads the current character. If the current character is a one-byte 7-bit control character or one-byte 8-bit character (e.g. an 8-bit character followed by a 7-bit control character), the one-byte 7-bit or 8-bit character will be returned.

# HSY$CH_RNEXT

HSY$CH_RNEXT reads the current character.

## Format

HSY$CH_RNEXT cur

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The current character.

## Arguments

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: modify
mechanism: by reference

The address of the current position of the specified current character. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

## Description

This routine reads the current character. If the current character is a one-byte 7-bit control character or one-byte 8-bit character (e.g. an 8-bit character followed by a 7-bit control character), the one-byte 7-bit or 8-bit character will be returned. Note that the read pointer is updated to the next character position after the read action. This routine is useful in successive reading of characters.

# HSY$CH_RPREV

HSY$CH_RPREV reads the previous character.

## Format

HSY$CH_RPREV str,cur

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The previous character read.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: modify
mechanism: by reference

The address of the current position of the specified current character. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

## Description

This routine reads the previous character. Note that the current character pointer is updated. It points to the previous character position after the read action.

# HSY$CH_WCHAR

HSY$CH_WCHAR writes a specified character to the current position of a buffer.

## Format

HSY$CH_WCHAR chr,cur

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The character to be written to the specified current position of the string.

**cur**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the string where the input character is to be written to.

## Description

This routine writes a specified character to the current position. It does not perform checking of writing past the end of buffer.

# HSY$CH_WNEXT

HSY$CH_WNEXT writes a specified character to the current position of a buffer.

## Format

HSY$CH_WNEXT chr,cur

## Arguments

**chr**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

The character to be written to the specified current position of the string.

**cur**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | modify |
| mechanism: | by reference |

The address of the current position of the string where the input character is to be written to.

## Description

This routine writes a specified character to the current position. It does not perform checking of writing past the end of buffer. Note that the write pointer **cur** is updated to the next character position after the write action.

# HSY$DX_RCHAR

HSY$DX_RCHAR reads the current character.

## Format

HSY$DX_RCHAR str,[pos]

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

FFFF (hex) - Routine completed unsuccessfully.
non FFFF (hex) - The current character.

## Arguments

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

The input string to be read.

**pos**
VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by reference

Byte position from the starting position of the specified string which is used to indicate the current position. Note that this position must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

## Description

This routine reads a character at the current character position as specified by **pos**. If **pos** is not specified, the first character of the string will be read. FFFF (hex) will be returned if **pos** is less than 1 or an invalid descriptor is specified by **str**.

# HSY$DX_RNEXT

HSY$DX_RNEXT reads the current character.

## Format

HSY$DX_RNEXT str,[pos]

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

FFFF (hex) - Routine completed unsuccessfully.
non FFFF (hex) - The current character.

## Arguments

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

The input string to be read.

**pos**
VMS Usage: longword_signed
type: longword integer (signed)
access: modify
mechanism: by reference

Byte position from the starting position of the specified string which is used to indicate the current position. Note that this position must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

## Description

This routine reads a character at the current character position as specified by **pos**. If **pos** is not specified, the first character of the string will be read. FFFF (hex) will be returned if **pos** is less than 1 or an invalid descriptor is specified by **str**. Note that **pos** is updated to the next character position after the read action. This routine is useful for successive reading of characters.

# HSY$DX_WCHAR

HSY$DX_WCHAR writes a specified character.

## Format

HSY$DX_WCHAR chr,str,[pos]

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return status.

| | | |
|---|---|---|
| 0 | - | Unsuccessful write caused by either invalid descriptor specified or **pos** less than 1. |
| SS$_NORMAL | - | Successful write. |

## Arguments

### chr

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by reference

The character to be written.

### str

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

The specified string.

### pos

VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by reference

Byte position from the starting position of the specified string which indicates the position to where the input character is written.

## Description

This routine writes a specified character to the current character position in the buffer. If **pos** is not specified, the character will be written to the start of the input string.

# HSY$DX_WNEXT

HSY$DX_WNEXT writes a specified character.

## Format

HSY$DX_WNEXT chr,str,[pos]

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return status.

| | | |
|---|---|---|
| 0 | - | Unsuccessful write caused by either invalid descriptor specified or **pos** less than 1. |
| SS$_NORMAL | - | Successful write. |

## Arguments

### chr

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by reference

The character to be written.

### str

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

The specified string.

### pos

VMS Usage: longword_signed
type: longword integer (signed)
access: modify
mechanism: by reference

Byte position from the starting position of the specified string which indicates the position to where the input character is written.

## Description

This routine writes a specified character to the current character position in the buffer. If **pos** is not specified, the character will be written to the start of the input string. Note that **pos** is updated after the write action. It points to the next character position after writing. This routine is useful for successful writing of character.

# HSY$CH_CURR

HSY$CH_CURR points to the first byte of the current character.

## Format

HSY$CH_CURR str,cur

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The address of the first byte of the current character.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the input string. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the specified current character. Note that **cur** may not point to the first byte of a multi-byte character if the current character is a multi-byte character.

## Description

This routine provides the function of locating a character pointer on the proper character boundary if the current character is a multi-byte character. It checks if **cur** is specifying a position before **str**. It also checks if the current character is a 7-bit or 8-bit control character. In both cases, **cur** will be returned to the caller.

**str** can contain one-byte and multi-byte characters.

# HSY$CH_NEXT

HSY$CH_NEXT points to the first byte of the next character.

## Format

HSY$CH_NEXT cur

## Returns

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | write only |
| mechanism: | by value |

The address of the first byte of the next character.

## Arguments

**cur**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

The address of the first byte of the current character. Note that this character pointer must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

## Description

This routine does not check if the next character position contains garbage or if it is passing beyond the buffer end since no buffer end position is specified.

# HSY$CH_PREV

HSY$CH_PREV points to the first byte of the previous character.

## Format

HSY$CH_PREV str,cur

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The address of the first byte of the previous character.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the input string. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the current character. Note that this character pointer must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

## Description

This routine checks if the previous character position appears before the starting position of the input string. It also checks if the previous character is a 7-bit or 8-bit control character. In both cases, **cur** will be returned to the caller.

**str** can contain one-byte and multi-byte characters.

# HSY$DX_POS_CURR

HSY$DX_POS_CURR points to the first byte of the current character.

## Format

HSY$DX_POS_CURR str,pos

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return byte position.

0 - Procedure completed unsuccessfully.

Non-zero - Byte position from the starting position of the input string that points to the first byte of the current character.

## Arguments

**str**

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

Input string.

**pos**

VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by reference

Byte position from the starting position of the input string which indicates the position of the current character. Note that this position may not be on the proper character boundary, e.g. it may point to the second byte of a two-byte character.

## Description

**HSY$DX_POS_CURR** lets you position the character pointer at the first byte of a multi-byte character. E.g. if **pos** is pointing to the second byte of a two-byte character, **HSY$DX_POS_CURR** will return the byte position of the first byte of the two-byte character.

**str** can contain one-byte and multi-byte characters.

# HSY$DX_POS_NEXT

HSY$DX_POS_NEXT points to the first byte of the next character.

## Format

HSY$DX_POS_NEXT str,pos

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return byte position.

0 - Procedure completed unsuccessfully.

Non-zero - Byte position from the starting position of the input string that points to the first byte of the next character.

## Arguments

**str**

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

Input string.

**pos**

VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by reference

Byte position from the starting position of the input string which indicates the position of the current character. Note that this position need not be on the proper character boundary, e.g. it may point to the second byte of a two-byte character.

## Description

This routine allows **pos** to point at a non-character boundary position. If **pos** is pointing to the second byte of a two-byte character, the two-byte character will be treated as the current character and the next character will be returned to the caller. This routine does not check if **pos** is beyond the end of the string.

**str** can contain one-byte and multi-byte characters.

# HSY$DX_POS_PREV

HSY$DX_POS_PREV points to the first byte of the previous character.

## Format

HSY$DX_POS_PREV str,pos

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return byte position.

0 - Procedure completed unsuccessfully.

Non-zero - Byte position from the starting position of the specified input string that points to the first byte of the previous character.

## Arguments

**str**

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

Input string.

**pos**

VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by reference

Byte position from the starting position of the specified input string which indicates the position of the current character. Note that this position need not be on the proper character boundary, e.g. it may point to the second byte of a two-byte character.

## Description

This routine allows **pos** to point at a non-character boundary position. If **pos** is pointing to the second byte of a two-byte character, the two-byte character will be treated as the current character and the previous character will be returned to the caller. This routine does not check if **pos** is beyond the end of the string.

**str** can contain one-byte and multi-byte characters.

# HSY$DX_SKPC

HSY$DX_SKPC skips a specified character.

## Format

HSY$DX_SKPC chr,str

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return byte position.

0 - Either all characters in the input string are equal to the specified character or procedure is completed unsuccessfully due to corrupted input descriptor.

Non-zero - Byte position from the starting position of the input string that points to the first character that does not match **chr**.

## Arguments

### chr

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

The specified input character to be skipped.

### str

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

Input string.

## Description

This routine skips the specified character **chr** at the start of the input string **str**. The position of the first character that does not match with the specified character **chr** will be returned to the caller.

**str** and **chr** can contain one-byte and multi-byte characters. If **chr** contains more than one characters, only the first character specified by the descriptor will be used.

# HSY$POS_CURR

HSY$POS_CURR points to the first byte of the current character.

## Format

HSY$POS_CURR str,cur,end

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The address of the first byte of the current character.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

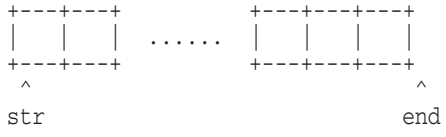
The address of the starting position of the input string. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the specified current character. Note that this character pointer may not be on the proper character boundary, e.g. it may point to the second byte of a two-byte character.

**end**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the string terminating position plus one as illustrated below:



## Description

This routine provides the function of locating a character pointer on the proper character boundary of a multi-byte character.

**str** can contain one-byte and multi-byte characters.

# HSY$POS_NEXT

HSY$POS_NEXT points to the first byte of the next character.

## Format

HSY$POS_NEXT cur,end

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The address of the first byte of the next character.

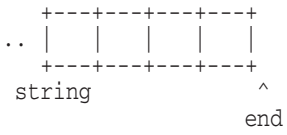
## Arguments

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the specified current character. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**end**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the string terminating position plus one as illustrated below:




## Description

This routine provides more checking than **HSY$CH_NEXT**. If **cur** is greater than or equal to **end**, **cur** will be returned to the caller. If **cur** is pointing at a single-byte 7-bit or 8-bit control character, **cur**+1 will be returned. If **cur** is pointing at a 8-bit character which is at the end of the input string, **end** will be returned. In general, if **cur** is pointing at an invalid character (e.g. a single 8-bit followed by a control character), it will skip the invalid character and return the next character positon.

**str** can contain one-byte and multi-byte characters.

# HSY$POS_PREV

HSY$POS_PREV points to the first byte of the previous character.

## Format

HSY$POS_PREV str,cur,end

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The address of the previous character of the specified current character.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

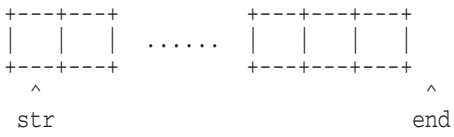
The address of the starting position of the input string. Note that this address must be on the proper byte boundary, e.g. it should not point to the second byte of a two-byte character.

**cur**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the current position of the string. Note that this character pointer must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

**end**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the string terminating position plus one as illustrated below:



## Description

This routine provides more checking than **HSY$CH_PREV**. If **cur** is outside the range of the string as specified by **str** and **end**, no previous character position will be returned. Instead, **cur** will be returned to the caller.

**str** can contain one-byte and multi-byte characters.

# HSY$SKPC

HSY$SKPC skips a specified character.

## Format

HSY$SKPC chr,str,len

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The return address.

0 - All characters in the input string are equal to the specified character.

Non-zero - The address of the position of the first character that does not match **chr**.

## Arguments

### chr

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The specified character to be skipped.

### str

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the input string. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

### len

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

The length in bytes of the input string.

## Description

This routine skips the specified character **chr** at the start of the input string **str**. The address of the first character that does not match with the specified character **chr** will be returned to the caller.

**str** can contain one-byte and multi-byte characters.

# HSY$COMPARE

HSY$COMPARE compares two specified strings.

## Format

HSY$COMPARE str1,len1,str2,len2

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return status.

-1 - **str1** is less than **str2**.
0 - **str1** is equal to **str2**.
1 - **str1** is greater than **str2**.

## Arguments

### str1

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the first string for comparison.

### len1

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

The length in bytes of **str1**.

### str2

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of second string for comparison.

### len2

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

The length in bytes of **str2**.

## Description

**str1** and **str2** can contain one-byte and multi-byte characters.

If the two input strings have different length, the shorter one is padded with space for comparison.

# HSY$DX_STR_EQUAL

HSY$DX_STR_EQUAL checks if two specified character strings are equal.

## Format

HSY$DX_STR_EQUAL str1,str2,[conv-flag]

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return status.

0 - The two strings are unequal.
1 - The two strings are equal.

## Arguments

### str1

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

First input string to be compared.

### str2

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

Second input string to be compared.

### conv-flag

VMS Usage: byte_signed
type: byte integer (signed)
access: read only
mechanism: by reference

Conversion flag indicating what conversion is done before comparing characters. Only bit 0 to bit 2 of this flag is used.

Bit 0 = 0: Performs uppercasing conversion. Uppercasing can be done to both full form and half form letters.

Bit 1 = 0: Performs full form to half form conversion.

Bit 2 = 0: Performs Kana to Hiragana conversion. This is only valid for Japanese Kana characters.

No conversion will be done prior to comparing the characters if this optional parameter is not specified.

## Description

**str1** and **str2** can contain one-byte and multi-byte characters.

If the number of characters in the two input strings are not equal, the shorter string is padded with space for comparison.

# HSY$STR_EQUAL

HSY$STR_EQUAL checks if two specified character strings are equal.

## Format

HSY$STR_EQUAL str1,len1,str2,len2,[conv-flag]

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return status.

0 - The two strings are unequal.
1 - The two strings are equal.

## Arguments

### str1

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the first input string to be compared.

### len1

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

The length in bytes of **str1**.

### str2

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the second input string to be compared.

### len2

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

The length in bytes of **str2**.

**conv-flag**

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

Conversion flag indicating what conversion is done before comparing characters. Only bit 0 to bit 2 of this flag is used.

Bit 0 = 0: Performs uppercasing conversion. Uppercasing can be done to both full form and half form letters.

Bit 1 = 0: Performs full form to half form conversion.

Bit 2 = 0: Performs Kana to Hiragana conversion. This is only valid for Japanese Kana characters.

No conversion will be done prior to comparing the characters if this optional parameter is not specified.

## Description

**str1** and **str2** can contain one-byte and multi-byte characters.

If the number of characters in the two input strings are not equal, the shorter string is padded with space for comparison.

# HSY$DX_LOCC

HSY$DX_LOCC locates the position of the first occurrence of the specified character.

## Format

HSY$DX_LOCC chr,str

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return byte position.

0 - Cannot locate the specified character in the specified string. It may be due to invalid descriptor specified or no such character found in the specified string.

Non-zero - Byte position from the starting position of the specified string which indicates the position of the first occurrence of the specified character.

## Arguments

### chr

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

The character to be located in the specified string.

### str

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

The specified string.

## Description

**chr** can either be one-byte or multi-byte character. If a character string is specified by **chr**, only the first character in the string will be used.

**str** can contain one-byte and multi-byte characters.

# HSY$DX_POSITION

HSY$DX_POSITION searches the first occurrence of a substring in a specified string.

## Format

HSY$DX_POSITION str,sub-str,[pos]

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return byte position.

0 - Procedure completed unsuccessfully.

Non-zero - Byte position from the starting position of the input string which indicates the position containing the first byte of the first character of the substring found.

## Arguments

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

Input string.

**sub-str**
VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The substring to be located in the input string.

**pos**
VMS Usage: word_signed
type: word integer (signed)
access: read only
mechanism: by reference

Byte position from the starting position of the input string which indicates the starting position for searching the substring. If this optional argument is not supplied, the searching will start from the beginning of the input string.

## Description

**str** and **sub-str** can contain one-byte and multi-byte characters.

# HSY$DX_STR_SEARCH

HSY$DX_STR_SEARCH searches the first occurrence of a specified substring in the input string.

## Format

HSY$DX_STR_SEARCH str,sub-str,[pos],[conv-flag]

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return byte position.

0 - Procedure completed unsuccessfully.

Non-zero - Byte position from the starting position of the input string which indicates the first byte of the first character of the substring found.

## Arguments

### str

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

Input string.

### sub-str

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

The specified substring to be searched.

### pos

VMS Usage: word_signed
type: word integer (signed)
access: read only
mechanism: by reference

Byte position from the starting position of the input string which indicates the starting position for searching the substring. If this optional argument is not supplied, the searching will start from the beginning of the input string.

### conv-flag

VMS Usage: byte_signed
type: byte integer (signed)
access: read only
mechanism: by reference

Conversion flag indicating what conversion is done before comparing the characters. Only bit 0 to bit 2 of this flag are used.

Bit 0 = 0: Performs uppercasing conversion. Uppercasing can be done to both full form and half form letters.

Bit 1 = 0: Performs full form to half form conversion.

Bit 2 = 0: Performs Kana to Hiragana conversion. This is only valid for Japanese Kana characters.

No conversion will be done prior to comparing the characters if this optional parameter is not specified.

## Description

**str** and **sub-str** can contain one-byte and multi-byte characters.

# HSY$DX_STR_START

HSY$DX_STR_START checks if the specified substring is found in another input string and starts from the first byte of the input string.

## Format

HSY$DX_STR_START str,sub-str,[conv-flag]

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The return status.

0 - The two strings are equal. Cannot find the substring starting from the start of the input string.

1 - Finds the substring starting from the start of the input string.

## Arguments

### str

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

Input string.

### sub-str

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

Substring to be located.

### conv-flag

VMS Usage: byte_signed
type: byte integer (signed)
access: read only
mechanism: by reference

Conversion flag indicating what conversion is done before comparing the characters. Only bit 0 to bit 2 of this flag are used.

Bit 0 = 0: Performs uppercasing conversion. Uppercasing can be done to both full form and half form letters.

Bit 1 = 0: Performs full form to half form conversion.

Bit 2 = 0: Performs Kana to Hiragana conversion. This is only valid for Japanese Kana characters.

No conversion will be done prior to comparing the characters if this optional parameter is not specified.

## Description

**str** and **sub-str** can contain one-byte and multi-byte characters.

# HSY$LOCC

HSY$LOCC locates the position of the first occurrence of the specified character.

## Format

HSY$LOCC chr,str,len

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The returned character pointer.

0 - No such character found in the specified string.

Non-zero - The address of the position of the first occurrence of the specified character found in the input string.

## Arguments

**chr**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The specified character to be located in the input string.

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: modify
mechanism: by value

The address of the starting position of the input string.

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

The length in bytes of the input string.

## Description

**chr** can either be one-byte or multi-byte character.

# HSY$POSITION

HSY$POSITION searches the first occurrence of a specified substring in the input string.

## Format

HSY$POSITION str,str-len,sub-str,sub-str-len

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The return address.

0 - The specified substring is not found in the input string.

Non-zero - The address of the starting position of the substring located in the input string.

## Arguments

### str

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the input string.

### str-len

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

The length in bytes of **str**.

### sub-str

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified substring to be located.

### sub-str-len

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

The length in bytes of **sub-str**.

## Description

**str** and **sub-str** can contain one-byte and multi-byte characters.

# HSY$STR_SEARCH

HSY$STR_SEARCH searches the first occurrence of a specified substring in the input string with conversion performed prior to comparing the characters.

## Format

```
HSY$STR_SEARCH  str,str-len,sub-str,sub-str-len,
                [conv-flag]
```

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The return address.

| | | |
|---|---|---|
| 0 | - | The specified substring is not found in the input string. |
| Non-zero | - | The address of the starting position of the substring located in the input string. |

## Arguments

### str

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the input string.

### str-len

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

The length in bytes of **str**.

### sub-str

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified substring.

### sub-str-len

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

The length in bytes of **sub-str**.

**conv-flag**

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

Conversion flag indicating what conversion is done before comparing the characters. Only bit 0 to bit 2 of this flag are used.

Bit 0 = 0: Performs uppercasing conversion. Uppercasing can be done to both full form and half form letters.

Bit 1 = 0: Performs full form to half form conversion.

Bit 2 = 0: Performs Kana to Hiragana conversion. This is only valid for Japanese Kana characters.

No conversion will be done prior to comparing the characters if this optional parameter is not specified.

## Description

**str** and **sub-str** can contain one-byte and multi-byte characters.

# HSY$STR_START

HSY$STR_START checks if the specified substring is found in another input string and starts from the first byte of the input string.

## Format

```
HSY$STR_START   str,str-len,sub-str,sub-str-len,
                [conv-flag]
```

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned status.

0 - Cannot find the substring starting from the start of the input string.

1 - Finds the substring starting from the start of the input string.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the input string.

**str-len**
VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by value

The length in bytes of the **str**.

**sub-str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the substring.

**sub-str-len**
VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by value

The length in bytes of **sub-str**.

**conv-flag**

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

Conversion flag indicating what conversion is done before comparing the characters. Only bit 0 to bit 2 of this flag is used.

Bit 0 = 0: Performs uppercasing conversion. Uppercasing can be done to both full form and half form letters.

Bit 1 = 0: Performs full form to half form conversion.

Bit 2 = 0: Performs Kana to Hiragana conversion. This is only valid for Japanese Kana characters.

No conversion will be done prior to comparing the characters if this optional parameter is not specified.

## Description

**str** and **sub-str** can contain one-byte and multi-byte characters.

# HSY$CH_NBYTE

HSY$CH_NBYTE counts the number of bytes of a character string.

## Format

HSY$CH_NBYTE str,nof-chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The number of bytes counted in the specified number of characters.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string.

**nof-chr**
VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by value

The number of characters to be scanned from the starting position of the input string for counting the number of bytes.

## Description

The routine accepts the number of characters and returns the number of bytes contained in the string of characters.

# HSY$CH_NCHAR

HSY$CH_NCHAR returns the number of characters in a specified string.

## Format

HSY$CH_NCHAR str,len

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The number of characters in the specified string.

## Arguments

**str**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string. Note that this address must be on the proper character boundary, e.g. it should not point to the second byte of a two-byte character.

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

The length in byte of the specified string.

## Description

This routine returns the number of characters found in the input string up to the position specified by **len**. All one-byte 7-bit control characters and one-byte 8-bit characters (e.g. an 8-bit character followed by a 7-bit control character) are treated as a character. If the last character specified by **len** is a multi-byte character with its last byte located beyond the input string terminating position as specified by **len**, this multi-byte character is also counted by the routine.

# HSY$CH_SIZE

HSY$CH_SIZE tells the byte length of the specified character.

## Format

HSY$CH_SIZE chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The size in bytes of the specified character.

## Arguments

**chr**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

This routine returns the number of bytes of the specified character. If the character is an ASCII character, 1 will be returned. If it is a multi-byte character, the number of bytes of the character will be returned.

# HSY$DX_NOF_BYTE

HSY$DX_NOF_BYTE counts the number of bytes of a character string.

## Format

HSY$DX_NOF_BYTE str,nof-chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned number of bytes.

0 - Procedure completed unsuccessfully due to either invalid descriptor specified or **nof-chr** less than 1.

Non-zero - The number of bytes.

## Arguments

### str

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

Input string.

### nof-chr

VMS Usage: longword_signed
type: longword integer (signed)
access: read only
mechanism: by reference

The number of characters to be scanned from the starting position of the input string for counting the number of bytes.

## Description

This routine accepts the number of characters and returns the number of bytes contained in the string of characters.

# HSY$DX_NOF_CHAR

HSY$DX_NOF_CHAR returns the number of characters in a specified number of bytes.

## Format

HSY$DX_NOF_CHAR str

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned number of characters.

0 - Procedure completed unsuccessfully due to either invalid descriptor specified or **nof-chr** less than 1.

Non-zero - The number of characters.

## Arguments

**str**

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

Input string.

## Description

This routine returns the number of characters found in the input string up to the position specified by the length field of the descriptor. All one-byte 7-bit control characters and one-byte 8-bit characters (e.g. an 8-bit character followed by a 7-bit control character) are treated as a character. If the last character specified by the length field of the descriptor is a multi-byte character with its last byte located beyond the input string terminating position, this multi-byte character is also counted by the routine.

# HSY$IS_ALPHA

HSY$IS_ALPHA checks if the input character is a Greek, Russian or Roman letter.

## Format

HSY$IS_ALPHA chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a letter character.

1 - The input character is a letter character.

Note that "letter character" here means one of the following:

(1) one-byte English letter

(2) multi-byte English letter

(3) multi-byte Greek letter

(4) multi-byte Russian letter

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_DESCRIPTION

HSY$IS_DESCRIPTION checks if the input character is a multi-byte local language punctuation (excluding parenthesis, bracket and quote).

## Format

HSY$IS_DESCRIPTION chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte local language punctuation (excluding parenthesis, bracket and quote).

1 - The input character is a multi-byte local language punctuation (excluding parenthesis, bracket and quote).

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_DIGIT

HSY$IS_DIGIT checks if the input character is a one-byte or multi-byte numeric digit.

## Format

HSY$IS_DIGIT chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a digit symbol.

1 - The input character is a digit symbol.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer(unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_GENERAL

HSY$IS_GENERAL checks if the input character is a multi-byte general symbol character (that does not belong to any of the above catagories).

## Format

HSY$IS_GENERAL chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte general symbol character.

1 - The input character is a multi-byte general symbol character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_GREEK

HSY$IS_GREEK checks if the input character is a multi-byte Greek letter.

## Format

HSY$IS_GREEK chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte Greek letter.
1 - The input character is a multi-byte Greek letter.

## Arguments

**chr**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_HIRAGANA

HSY$IS_HIRAGANA checks if the input character is a multi-byte Japanese Hiragana character.

## Format

HSY$IS_HIRAGANA chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a Hiragana character.

1 - The input character is a Hiragana character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_IDEOGRAPH

HSY$IS_IDEOGRAPH checks if the input multi-byte character is an ideographic multi-byte character (excluding all multi-byte Roman, Greek, Russian letters, Japanese characters and all multi-byte symbols).

## Format

HSY$IS_IDEOGRAPH chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not an ideographic multi-byte character.

1 - The input character is an ideographic multi-byte character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_KANA

HSY$IS_KANA checks if the input character is a multi-byte Japanese Kana character.

## Format

HSY$IS_KANA chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a Kana character.

1 - The input character is a Kana character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

Japanese Kana characters can either be Hiragana or Katakana characters.

# HSY$IS_KATAKANA

HSY$IS_KATAKANA checks if the input character is a multi-byte Japanese Katakana character.

## Format

HSY$IS_KATAKANA chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a Katakana character.

1 - The input character is a Katakana character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_LEFT_PARENTHESIS

HSY$IS_LEFT_PARENTHESIS checks if the input character is a multi-byte left parenthesis symbol character.

## Format

HSY$IS_LEFT_PARENTHESIS chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte left parenthesis symbol character.

1 - The input character is a multi-byte left parenthesis symbol character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_LINE_DRAWING

HSY$IS_LINE_DRAWING checks if the input character is a multi-byte line drawing symbol character.

## Format

HSY$IS_LINE_DRAWING chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte line drawing symbol character.

1 - The input character is a multi-byte line drawing symbol character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_LOWER

HSY$IS_LOWER checks if the input character is a lower case Greek, Russian or Roman letter.

## Format

HSY$IS_LOWER chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a lower case letter character.

1 - The input character is a lower case letter character.

Note that "letter character" here means one of the following:

(1) one-byte English letter

(2) multi-byte English letter

(3) multi-byte Greek letter

(4) multi-byte Russian letter

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_NO_FIRST

HSY$IS_NO_FIRST checks if the input character is a multi-byte "NO FIRST" character.

## Format

HSY$IS_NO_FIRST chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte "NO FIRST" character.

1 - The input character is a multi-byte "NO FIRST" character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

"NO FIRST" multi-byte characters include right parenthesis, right bracket, right quote as well as some multi-byte punctuations that should not appear at the start of a line.

# HSY$IS_NO_LAST

HSY$IS_NO_LAST checks if the input character is a multi-byte "NO-LAST" character.

## Format

HSY$IS_NO_LAST chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte "NO LAST" character.

1 - The input character is a multi-byte "NO LAST" character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

"NO LAST" multi-byte characters include left parenthesis, left bracket and left quote.

# HSY$IS_PARENTHESIS

HSY$IS_PARENTHESIS checks if the input character is a multi-byte parenthesis symbol character.

## Format

HSY$IS_PARENTHESIS chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte parenthesis symbol character.

1 - The input character is a multi-byte parenthesis symbol character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

Multi-byte parenthesis symbols include left and right multi-byte parentheses.

# HSY$IS_RIGHT_PARENTHESIS

HSY$IS_RIGHT_PARENTHESIS checks if the input character is a multi-byte right parenthesis symbol character.

## Format

HSY$IS_RIGHT_PARENTHESIS chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte right parenthesis symbol character.

1 - The input character is a multi-byte right parenthesis symbol character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_ROMAN

HSY$IS_ROMAN checks if the input character is a one-byte or multi-byte English letter.

## Format

HSY$IS_ROMAN chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a one-byte or multi-byte English letter.

1 - The input character is a one-byte or multi-byte English letter.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_RUSSIAN

HSY$IS_RUSSIAN checks if the input character is a multi-byte Russian letter.

## Format

HSY$IS_RUSSIAN chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte Russian letter.
1 - The input character is a multi-byte Russian letter.

## Arguments

**chr**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_TECHNICAL

HSY$IS_TECHNICAL checks if the input character is a scientific or mathematical multi-byte symbol character.

## Format

HSY$IS_TECHNICAL chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a technical multi-byte symbol character.

1 - The input character is a technical multi-byte symbol character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_UNIT

HSY$IS_UNIT checks if the input character is a multi-byte standard unit symbol character.

## Format

HSY$IS_UNIT chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a multi-byte standard unit symbol character.

1 - The input character is a multi-byte standard unit symbol character.

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_UPPER

HSY$IS_UPPER checks if the input character is an upper case Greek, Russian or Roman letter.

## Format

HSY$IS_UPPER chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not an upper case letter character.

1 - The input character is an upper case letter character.

Note that "letter character" here means one of the following:

(1) one-byte English letter

(2) multi-byte English letter

(3) multi-byte Greek letter

(4) multi-byte Russian letter

## Arguments

**chr**

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

None.

# HSY$IS_VALID

HSY$IS_VALID checks if the input character is a valid multi-byte character.

## Format

HSY$IS_VALID chr

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

The returned flag.

0 - The input character is not a valid multi-byte character.
1 - The input character is a valid multi-byte character.

## Arguments

**chr**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

Valid multi-byte characters are those found in the DEC supported local language character set.

# HSY$DX_DATE_TIME

HSY$DX_DATE_TIME returns the date and time in local language format (the time can either be the system time or the user-specified date).

## Format

HSY$DX_DATE_TIME dst,[flag],[time-addr]

## Returns

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

The same condition value returned by OpenVMS Run Time Library routine LIB$SCOPY_R_DX.

## Arguments

### dst

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

The specified destination string to store the resulting time string in local language format.

### flag

VMS Usage: byte_signed
type: byte integer (signed)
access: read only
mechanism: by reference

Bit 0: 0 - 12 hour format (default)
1 - 24 hour format
Bit 1: 0 - Full date and time (default)
1 - Time only

If this argument is not specified, 0 will be used which means 12 hour format with full date and time display.

### time-addr

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the quadword that contains the user-specified date and time in 64-bit time format. If this argument is not specified, the current system time will be used.

## Description

None.

# HSY$DX_TIME

HSY$DX_TIME returns the date and time of the system time in local language format.

## Format

HSY$DX_TIME dst,[flag]

## Returns

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

The same condition value returned by OpenVMS Run Time Library routine LIB$SCOPY_R_DX.

## Arguments

### dst

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

The specified destination string to store the resulting displaying string in local language format.

### flag

VMS Usage: byte_signed
type: byte integer (signed)
access: read only
mechanism: by reference

Bit 0: 0 - 12 hour format (default)
1 - 24 hour format
Bit 1: 0 - Full date and time (default)
1 - Time only

If this argument is not specified, 0 will be used which means 12 hour format with full date and time display.

## Description

None.

# HSY$CHG_GENERAL

HSY$CHG_GENERAL converts specified characters in one of the three following ways:

(1) From lower case letters to upper case letters (including English, Greek and Russian letters)

(2) From full form ASCII to half form ASCII

(3) From Kana to Hiragana (for Japanese characters only)

## Format

HSY$CHG_GENERAL chr,conv-flag

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The converted character.

## Arguments

### chr

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The specified character to be converted.

### conv-flag

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

Conversion flag indicating what conversion is to be done. Only bit 0 to bit 2 of this flag is used.

Bit 0 = 0: Performs uppercasing conversion. Uppercasing can be done to both full form and half form letters.

Bit 1 = 0: Performs full form to half form conversion.

Bit 2 = 0: Performs Kana to Hiragana conversion. This is only valid for Japanese Kana characters.

## Description

If **chr** is not applicable to a particular conversion, e.g. **chr** is not a letter and uppercasing conversion is specified by **conv-flag**, then no conversion will be done and **chr** will be returned.

# HSY$CHG_KANA_HIRA

HSY$CHG_KANA_HIRA converts Katakana characters to Hiragana characters. (For Japanese characters only.)

## Format

```
HSY$CHG_KANA_HIRA  kana
```

## Returns

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | write only |
| mechanism: | by value |

The converted Hiragana character.

## Arguments

**kana**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

Input Kana character.

## Description

If **kana** is not a Katakana character, **kana** will be returned with no conversion done.

# HSY$CHG_KANA_KANA

HSY$CHG_KANA_KANA toggles Kana characters to Hiragana or Katakana characters. (For Japanese characters only.)

## Format

HSY$CHG_KANA_KANA kana

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The converted Kana character which can either be Hiragana or Katakana.

## Arguments

**kana**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input Kana character.

## Description

If **kana** is a Hiragana character, it will be converted to a Katakana character. If **kana** is a Katakana character, it will be converted to a Hiragana character.

If **kana** is not a Hiragana or Katakana character, **kana** will be returned with no conversion done.

# HSY$CHG_KANA_KATA

HSY$CHG_KANA_KATA converts Hiragana characters to Katakana characters. (For Japanese characters only.)

## Format

HSY$CHG_KANA_KATA kana

## Returns

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | write only |
| mechanism: | by value |

The converted Katakana character.

## Arguments

**kana**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

Input Kana character.

## Description

If **kana** is not a Hiragana character, **kana** will be returned with no conversion done.

# HSY$CHG_KEISEN

HSY$CHG_KEISEN converts '0' to '9' and '-' to multi-byte line drawing characters.

## Format

HSY$CHG_KEISEN chr

## Returns

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | write only |
| mechanism: | by value |

Returned multi-byte line drawing character.

## Arguments

**chr**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

The input character code (see description).

## Description

The conversion table is as follows:

If **chr** is not from '0' to '9' or '-', **chr** will be returned with no conversion done.

# HSY$CHG_ROM_CASE

HSY$CHG_ROM_CASE toggles the casing of one-byte and multi-byte letters (English letters, Greek letters and Russian letters) of the input character.

## Format

```
HSY$CHG_ROM_CASE  chr
```

## Returns

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | write only |
| mechanism: | by value |

Converted character with its case toggled.

## Arguments

**chr**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

Input character.

## Description

One-byte and multi-byte English letters, multi-byte Greek letters and multi-byte Russian letters contain both upper and lower case characters. This routine converts upper case characters to lower case characters and vice versa.

If **chr** is not a one-byte or multi-byte letter as stated above, **chr** will be returned and no conversion will be done.

# HSY$CHG_ROM_FULL

HSY$CHG_ROM_FULL converts one-byte ASCII (half form ASCII) to multi-byte equivalence (full form ASCII).

## Format

```
HSY$CHG_ROM_FULL  chr
```

## Returns

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | write only |
| mechanism: | by value |

The corresponding full form ASCII character.

## Arguments

**chr**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

Input character.

## Description

If **chr** is not a half form ASCII character, **chr** will be returned and no conversion will be done.

# HSY$CHG_ROM_HALF

HSY$CHG_ROM_HALF converts multi-byte ASCII (full form ASCII) to one-byte (half form ASCII) equivalence.

## Format

HSY$CHG_ROM_HALF chr

## Returns

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | write only |
| mechanism: | by value |

The corresponding half form ASCII character.

## Arguments

**chr**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

Input character.

## Description

If **chr** is not a full form character, **chr** will be returned and no conversion will be done.

# HSY$CHG_ROM_LOWER

HSY$CHG_ROM_LOWER converts one-byte and multi-byte letters (English letters, Greek letters and Russian letters) to lower case.

## Format

HSY$CHG_ROM_LOWER chr

## Returns

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | write only |
| mechanism: | by value |

The corresponding lowercase character.

## Arguments

**chr**

| | |
|---|---|
| VMS Usage: | longword_unsigned |
| type: | longword integer (unsigned) |
| access: | read only |
| mechanism: | by value |

Input character.

## Description

If **chr** is not an upper case letter (English letter, Greek letter and Russian letter), **chr** will be returned and no conversion will be done.

# HSY$CHG_ROM_SIZE

HSY$CHG_ROM_SIZE toggles the form (full form or half form) of the input character.

## Format

HSY$CHG_ROM_SIZE chr

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

Toggled character.

## Arguments

**chr**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

Full form and half form conversions only apply to one-byte ASCII (half form ASCII) and multi-byte ASCII (full form ASCII). There are no half form equivalence of other multi-byte characters such as Greek letters.

If **chr** is not a full form or half form character, **chr** will be returned and no conversion will be done.

# HSY$CHG_ROM_UPPER

HSY$CHG_ROM_UPPER converts one-byte and multi-byte letters (English letters, Greek letters and Russian letters) to upper case.

## Format

HSY$CHG_ROM_UPPER chr

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

The corresponding uppercase character.

## Arguments

**chr**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

Input character.

## Description

If **chr** is not a lower case letter (English letter, Greek letter and Russian letter), **chr** will be returned and no conversion will be done.

# HSY$DX_TRA_KANA_HIRA

HSY$DX_TRA_KANA_HIRA converts Katakana character strings to Hiragana character strings. (For Japanese characters only.)

## Format

HSY$DX_TRA_KANA_HIRA dst,src,[len]

## Returns

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

## Arguments

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

The destination string that stores the result of the conversion.

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

The source string that is to be converted.

**len**
VMS Usage: word_signed
type: word integer (signed)
access: write only
mechanism: by reference

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

Characters in the input string which are not Katakana characters will be copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |

| | |
|---|---|
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_KANA_KANA

HSY$DX_TRA_KANA_KANA toggles Kana character strings to Hiragana or Katakana character strings. (For Japanese characters only.)

## Format

HSY$DX_TRA_KANA_KANA dst,src,[len]

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | longword (unsigned) |
| access: | write only |
| mechanism: | by value |

## Arguments

**dst**

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | write only |
| mechanism: | by descriptor |

The destination string that stores the result of the conversion.

**src**

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | read only |
| mechanism: | by descriptor |

The source string that is to be converted.

**len**

| | |
|---|---|
| VMS Usage: | word_signed |
| type: | word integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

All Hiragana characters found are converted to Katakana and all Katakana characters found are converted to Hiragana.

Characters in the input string that are not Hiragana or Katakana are copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_KANA_KATA

HSY$DX_TRA_KANA_KATA converts Hiragana character strings to Katakana character strings. (For Japanese characters only.)

## Format

HSY$DX_TRA_KANA_KATA dst,src,[len]

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | longword (unsigned) |
| access: | write only |
| mechanism: | by value |

## Arguments

**dst**

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | write only |
| mechanism: | by descriptor |

The destination string that stores the result of the conversion.

**src**

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | read only |
| mechanism: | by descriptor |

The source string that is to be converted.

**len**

| | |
|---|---|
| VMS Usage: | word_signed |
| type: | word integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

Characters in the input string which are not Hiragana characters will be copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |

| | |
|---|---|
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_ROM_CASE

HSY$DX_TRA_ROM_CASE toggles the casing of one-byte and multi-byte letters (English letters, Greek letters and Russian letters) found in the input string.

## Format

HSY$DX_TRA_ROM_CASE dst,src,[len]

## Returns

| | |
|---|---|
| VMS Usage: | cond_value |
| type: | longword (unsigned) |
| access: | write only |
| mechanism: | by value |

## Arguments

**dst**

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | write only |
| mechanism: | by descriptor |

The destination string that stores the result of the conversion.

**src**

| | |
|---|---|
| VMS Usage: | char_string |
| type: | character string |
| access: | read only |
| mechanism: | by descriptor |

The source string that is to be converted.

**len**

| | |
|---|---|
| VMS Usage: | word_signed |
| type: | word integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

One-byte and multi-byte English letters, multi-byte Greek letters and multi-byte Russian letters all contain both upper case and lower case characters. This routine converts all upper case characters to lower case and all lower case characters to upper case.

Characters in the input string that are not upper case or lower case characters are copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_ROM_FULL

HSY$DX_TRA_ROM_FULL converts one-byte ASCII (half form ASCII) to multi-byte equivalence (full form ASCII).

## Format

HSY$DX_TRA_ROM_FULL dst,src,[len]

## Returns

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

## Arguments

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

The destination string that stores the result of the conversion.

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

The source string that is to be converted.

**len**
VMS Usage: word_signed
type: word integer (signed)
access: write only
mechanism: by reference

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

Characters in the input string that are not half form characters are copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |

| | |
|---|---|
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_ROM_HALF

HSY$DX_TRA_ROM_HALF converts multi-byte ASCII (full form ASCII) to one-byte (half form ASCII) equivalence.

## Format

HSY$DX_TRA_ROM_HALF dst,src,[len]

## Returns

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

## Arguments

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

The destination string that stores the result of the conversion.

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

The source string that is to be converted.

**len**
VMS Usage: word_signed
type: word integer (signed)
access: write only
mechanism: by reference

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

Characters in the input string that are not full form characters are copied to the corresponding position in the output string with no conversion done.

## CONDITION VALUES RETURNED

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |

| | |
|---|---|
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_ROM_LOWER

HSY$DX_TRA_ROM_LOWER converts one-byte and multi-byte letters (English letters, Greek letters and Russian letters) to lower case.

## Format

HSY$DX_TRA_ROM_LOWER dst,src,[len]

## Returns

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

## Arguments

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

The destination string that stores the result of the conversion.

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

The source string that is to be converted.

**len**
VMS Usage: word_signed
type: word integer (signed)
access: write only
mechanism: by reference

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

Characters in the input string that are not upper case letters are copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |

| | |
|---|---|
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_ROM_SIZE

HSY$DX_TRA_ROM_SIZE toggles the form (full form or half form) of the input string.

## Format

HSY$DX_TRA_ROM_SIZE dst,src,[len]

## Returns

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

## Arguments

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

The destination string that stores the result of the conversion.

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

The source string that is to be converted.

**len**
VMS Usage: word_signed
type: word integer (signed)
access: write only
mechanism: by reference

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

Full form and half form conversions only apply to one-byte ASCII (half form ASCII) and multi-byte ASCII (full form ASCII). There is no half form equivalence of other multi-byte characters such as multi-byte Greek letters.

Characters in the input string that are not full form or half form characters are copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_ROM_UPPER

HSY$DX_TRA_ROM_UPPER converts one-byte and multi-byte letters (English letters, Greek letters and Russian letters) to upper case.

## Format

HSY$DX_TRA_ROM_UPPER dst,src,[len]

## Returns

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

## Arguments

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

The destination string that stores the result of the conversion.

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

The source string that is to be converted.

**len**
VMS Usage: word_signed
type: word integer (signed)
access: write only
mechanism: by reference

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

Characters in the input string that are not lower case letters are copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |

| | |
|---|---|
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$DX_TRA_SYMBOL

HSY$DX_TRA_SYMBOL converts the sequence of a one-byte character to a string of multi-byte symbols.

## Format

HSY$DX_TRA_SYMBOL dst,src,[len]

## Returns

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

The return status.

## Arguments

### dst

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

The destination string that stores the result of the conversion.

### src

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

The source string that is to be converted.

### len

VMS Usage: word_signed
type: word integer (signed)
access: write only
mechanism: by reference

The length in bytes of the converted string. If this argument is not supplied, no length information of the converted string will be returned to the caller.

## Description

This routine provides conversion of sequences of ASCII characters to corresponding multi-byte symbols and multi-byte characters as stated in the following table.

If the characters in the input string are not applicable for conversion, they will be copied to the corresponding position in the output string with no conversion done.

## condition values returned

| | |
|---|---|
| LIB$_INVSTRDES | Invalid string descriptor. A string descriptor has an invalid value in its DSC$B_CLASS field. |
| LIB$_STRTRU | Procedure successfully completed. String truncated. |
| LIB$_FATERRLIB | Fatal internal error. An internal consistency check has failed. |
| LIB$_INSVIRMEM | Insufficient virtual memory. |
| SS$_NORMAL | Procedure successfully completed. |

# HSY$TRA_KANA_HIRA

HSY$TRA_KANA_HIRA converts Katakana character strings to Hiragana character strings. (For Japanese characters only.)

## Format

HSY$TRA_KANA_HIRA ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

**ip**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

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

The length in bytes of the input string.

**op**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

If the characters in the input string are not Katakana characters, they will be copied to the corresponding position in the output string with no conversion done.

# HSY$TRA_KANA_KANA

HSY$TRA_KANA_KANA toggles Kana character strings to Hiragana or Katakana characters. (For Japanese characters only.)

## Format

HSY$TRA_KANA_KANA ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

**ip**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

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

The length in bytes of the input string.

**op**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

All Hiragana characters found are converted to Katakana and all Katakana characters found are converted to Hiragana.

If the characters in the input string are not Hiragana or Katakana, they will be copied to the corresponding position in the output string with no conversion done.

# HSY$TRA_KANA_KATA

HSY$TRA_KANA_KATA converts Hiragana character strings to Katakana character strings. (For Japanese characters only.)

## Format

HSY$TRA_KANA_KATA ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

**ip**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

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

The length in bytes of the input string.

**op**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

If the characters in the input string are not Hiragana characters, they will be copied to the corresponding position in the output string with no conversion done.

# HSY$TRA_ROM_CASE

HSY$TRA_ROM_CASE toggles the casing of one-byte and multi-byte letters (English letters, Greek letters and Russian letters) found in the string.

## Format

HSY$TRA_ROM_CASE ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

**ip**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

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

The length in bytes of the input string.

**op**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

One-byte and multi-byte English letters, multi-byte Greek letters and multi-byte Russian letters contain both upper and lower case characters. This routine converts all upper case characters to lower case and all lower case characters to upper case.

If the characters in the input string are not one-byte or multi-byte letters, they will be copied to the corresponding position in the output string with no conversion done.

# HSY$TRA_ROM_FULL

HSY$TRA_ROM_FULL converts one-byte ASCII (half form ASCII) to multi-byte equivalence (full form ASCII).

## Format

HSY$TRA_ROM_FULL ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

### ip

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

### il

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

The length in bytes of the input string.

### op

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

### ol

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

If the characters in the input string are not half form characters, they will be copied to the corresponding position in the output string with no conversion done.

# HSY$TRA_ROM_HALF

HSY$TRA_ROM_HALF converts multi-byte ASCII (full form ASCII) to one-byte (half form ASCII) equivalence.

## Format

HSY$TRA_ROM_HALF ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

### ip

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

### il

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

The length in bytes of the input string.

### op

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

### ol

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

If the characters in the input string are not full form characters, they will be copied to the corresponding position in the output string with no conversion done.

# HSY$TRA_ROM_LOWER

HSY$TRA_ROM_LOWER converts one-byte and multi-byte letters (English letters, Greek letters and Russian letters) to lower case.

## Format

HSY$TRA_ROM_LOWER ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

**ip**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

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

The length in bytes of the input string.

**op**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

If the characters in the input string are not upper case letters, they will be copied to the corresponding position in the output string with no conversion done.

# HSY$TRA_ROM_SIZE

HSY$TRA_ROM_SIZE toggles the form (full form or half form) of the input string.

## Format

HSY$TRA_ROM_SIZE ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

### ip

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

### il

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

The length in bytes of the input string.

### op

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

### ol

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

Full form and half form conversions only apply to one-byte ASCII (half form ASCII) and multi-byte ASCII (full form ASCII). There is no half form equivalence of other multi-byte characters such as Greek letters.

If the characters in the input string are not full form or half form characters, they will be copied to the corresponding position in the output string with no conversion done.

# HSY$TRA_ROM_UPPER

HSY$TRA_ROM_UPPER converts one-byte and multi-byte letters (English letters, Greek letters and Russian letters) to upper case.

## Format

HSY$TRA_ROM_UPPER ip,il,op,ol,rl

## Returns

VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: write only
mechanism: by value

1 - The input string is successfully converted.

0 - A truncated input string is converted due to insufficient output space of the output string allocated by the caller.

## Arguments

**ip**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the starting position of the specified string which is the input for conversion.

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

The length in bytes of the input string.

**op**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read onlys
mechanism: by value

The address of the starting position of the specified string which stores the output of conversion.

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

The length in bytes of the output string.

**rl**

| | |
|---|---|
| VMS Usage: | longword_signed |
| type: | longword integer (signed) |
| access: | write only |
| mechanism: | by reference |

The length in bytes of the converted string.

## Description

If the characters in the input string are not lower case letters, they will be copied to the corresponding positions in the output string with no conversion done.

# HSY$TRA_SYMBOL

HSY$TRA_SYMBOL converts the sequence of a one-byte character to a string of multi-byte symbols.

## Format

HSY$TRA_SYMBOL ip,il,op,ol,rl

## Returns

VMS Usage: longword_signed
type: longword integer (signed)
access: write only
mechanism: by value

0 - The conversion completed unsuccessfully.

1 - The conversion completed successfully.

## Arguments

**ip**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the specified input string that is to be converted.

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

The length of the input string that is specified by the argument **ip**.

**op**
VMS Usage: longword_unsigned
type: longword integer (unsigned)
access: read only
mechanism: by value

The address of the specified output string that stores the converted string.

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

The length of the output string that is specified by the argument **op**.

**rl**
VMS Usage: longword_signed
type: longword integer (signed)
access: write only

mechanism: by reference

The actual length of the converted string.

## Description

This routine provides conversion of sequences of ASCII characters to corresponding multi-byte symbols and multi-byte characters as stated in the following table.

| | |
|---|---|
| > = | ≥ |
| > < | ☆ |
| > - | → |
| : - | ÷ |
| + - | ± |
| x x | × |
| = / | ≠ |
| = \| | ∵ |
| , . | … |
| . . | ‥ |
| . ; | ∴ |
| . c | ℃ |
| ` ` | “ |
| ´ ´ | ” |
| c / | ¢ |
| l - | £ |
| s s | § |
| o o | ∞ |
| o > | ♂ |
| o + | ♀ |
| K > < | ★ |
| K ( ) | ● |
| K < > | ◆ |
| K [ ] | ■ |
| K < \ | ▲ |

If the characters in the input string are not applicable for conversion, they will be copied to the corresponding position in the output string with no conversion done.