SAS® Viya™ 3.2 National Language Support: Reference Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2017. *SAS® Viya™ 3.2 National Language Support: Reference Guide*. Cary, NC: SAS Institute Inc.

SAS® Viya™ 3.2 National Language Support: Reference Guide

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

March 2017

3.2-P1:nlsref

Contents

What's New in SAS Viya National Language Support: Reference Guide

Overview

One CALL Rroutine and seven functions are new. An appendix with encoding tables has been added. Several functions have been updated.

New CALL Routines, Functions and Appendixes

CALL Routines and Functions

These CALL routines and functions are new:

CALL KSCANX (p. 316)
Returns the position and length of the *n*th word from a character string.

KCOUNTC (p. 329)
Counts the number of words in a character string.

KCOUNTW (p. 331)
Counts the number of words in a character string.

KCOUNTX (p. 334)
Counts the number of words in a character string.

KFIND (p. 337)
Searches for a specific substring of characters within a character string.

KFINDC (p. 339)
Searches a string for any character in a list of characters.

KFINDW (p. 342)
Returns the character position of a word in a string or the number of the word in a string.

KSCANX (p. 365)
Selects a specified word from a character expression.

Appendix

This appendix is new:

Encodings and Their Aliases and Encoding Character Set Compatibility (p. 655)
The tables Encoding and Alias Names and Encoding Character Set Compatibility have been added.

Updated CALL Routines and Functions

These functions and CALL routines have been updated:

KLEFT (p. 350)
The *Unicode Spaces That Are Removed by KLEFT, KRIGHT, and KTRIM* table is new.

KSTRIP (p. 370)
Information about data normalization has been added.

Part 1

NLS Concepts

Chapter 1

National Language Support

Overview to National Language Support

National Language Support is a set of features that enable a software product to function properly in every global market for which the product is targeted. SAS contains NLS features to ensure that SAS applications can be written so that they conform to local language conventions. Typically, software that is written in the English language works well for users who speak the English language and use data that is formatted using the conventions that are observed in the United States. However, without NLS, these products might not work well for users in other regions of the world. NLS in SAS enables users in regions such as Asia and Europe to process data successfully in their native languages and environments.

NLS is applied to data that is moved between machines; for example, NLS ensures that the data is converted to the correct format for use on the target machine.

Text-string operations are sensitive to SAS settings for language and region. This action enables correct results for such operations as uppercasing and lowercasing characters, classifying characters, and scanning data. SAS provides features to ensure that national characters, which are characters specific to a particular nation or group of nations, are displayed and printed properly.

Software applications that incorporate NLS can avoid dependencies on language-specific or cultural-specific conventions for software features. Here are some examples:

- string manipulation
- character classifications
- character comparison rules
- code sets
- date and time formatting
- user interfaces
- message-text languages
- numeric and monetary formatting

- sort orders

Definition of Localization and Internationalization

Internationalization is the process of designing a software product without making assumptions that are based on a single language or locale, thereby facilitating localization. Internationalization ensures that international conventions (including rules for sorting strings and for formatting dates, times, numbers, and currencies) are supported. It also facilitates a consistent user experience across different language editions of a product. The abbreviation for internationalization is I18N.

Although the application logic might support cultural conventions (for example, the monetary and numeric formats of a particular region), only a localized version of the software presents user interfaces and system messages in the local language.

Localization is the process of adapting a product to meet the language, cultural, and other requirements of a specific target environment or market so that users can see results created in their own language and conventions when using the product. Translation of the user interface, system messages, and documentation is part of localization.

SAS NLS features are available for localizing and internationalizing your SAS applications. For more information see "Internationalization Compatibility for SAS String Functions" on page 279.

Chapter 2

Locale for NLS

Overview of Locale Concepts for NLS

A *locale* reflects the language, local conventions such as data formatting, and culture for a geographical region. Local conventions might include specific formatting rules for dates, times, and numbers and a currency symbol for the country or region. Collating sequence, paper size, postal addresses, and telephone numbers can also be included in a locale.

Dates have many representations, depending on the conventions that are accepted in a culture. The month might be represented as a number or as a name. The name might be fully spelled or abbreviated. The order of the month, day, and year might differ according to the locale.

For example, “the third day of October in the year 2016” would be displayed in a different way for each of these locales:

Germany
03.10.16

Italy
3/10/16

United States
10/03/16

Language is part of a locale, but it is not unique to any one locale. For example, Portuguese is spoken in Brazil as well as in Portugal, but the cultures are different. In Brazil and in Portugal, there are similarities in the formatting of data. Numbers are formatted using a comma (,) to separate integers from fractional values and a dot (.) to separate groups of digits to the left of the radix character. However, there are important differences, such as the currency symbols that are used in the two different locales. Portugal uses the euro and requires the euro symbol (€) while Brazil uses the real, which is represented by the two-character currency symbol R$.

Also, a country might have more than one official language. Canada has two official languages: English and French. Two values can be specified for the LOCALE= system option: English_Canada and French_Canada.

Numbers, including currency, can have different representations. For example, the decimal separator, or radix character, is a dot (.) in some regions and a comma (,) in other regions. The thousands separator can be a dot, a comma, or even a space in some regions. Monetary conventions likewise vary between locales; for example, a dollar sign or a yen sign might be attached to a monetary value.

Paper size and measurement are also locale considerations. Standard paper sizes include letter (8-1/2-by-11-inch paper) and A4 (210-by-297-millimeter paper). The letter paper size is mainly used by some English-speaking countries; A4 is used by most other locales. Most locales use centimeters, while some locales use inches.

Time can be represented in one English-speaking country or region by using the 12-hour notation. Other English speakers expect time values to be formatted using the 24-hour notation.

Specifying a Locale

How the Locale Is Specified at SAS Invocation

You can use the LOCALE= system option to specify the locale of the SAS session at SAS invocation. LOCALE= also implicitly sets the following SAS system options:

- DATESTYLE=
- DFLANG=
- ENCODING=
- LOCALEDATA
- MAPEBCDICTOASCII
- ODSLANGCHG
- PAPERSIZE=
- RSASIOTRANSERROR
- TIMEZONE
- URLENCODING

Note: Locale can also be specified using POSIX naming standards. For example, en_US is the POSIX equivalent for the SAS value English_UnitedStates.

Default values for the LOCALE= option are the same under each operating environment. For more information, see "LOCALE= Values for PAPERSIZE and DFLANG Options" on page 629.

The English_UnitedStates value for LOCALE= implicitly sets the following options to the specified default values at SAS invocation:

- **DATESTYLE=MDY**
- **DFLANG=English**
- **ENCODING=UTF-8**
- **PAPERSIZE=Letter**

At invocation, an explicitly set system option overrides any implicitly set option.

At invocation, the explicit setting PAPERSIZE=A4 overrides an implicit setting of the PAPERSIZE= option using the LOCALE= option. For more information, see "PAGESIZE= System Option" in *SAS Viya System Options: Reference* .

How the Locale Is Specified during a SAS Session

You can use the LOCALE= system option to specify the locale of the SAS session during the SAS session. However, only the values for these system options change implicitly to reflect the changed value of LOCALE=:

- DFLANG=
- PAPERSIZE=

Note: The PAPERSIZE= option is affected only if it is set to LOCALE. For more information see "PAGESIZE= System Option" in *SAS Viya System Options: Reference*.

The value for the ENCODING= system option does not change implicitly to reflect the changed value of LOCALE=.

The Italian_Italy value that is assigned to the LOCALE= option implicitly resets the following options during the SAS session to reflect the changed value of the LOCALE= system option:

- DFLANG=Italian
- PAPERSIZE=A4

Language Switching

SAS messages are displayed in the language that is specified by the settings in the SAS configuration file during startup. You can view SAS messages in another language by using the language switching feature. You can access the language switching feature with the ODSLANGCHG system option. If ODSLANGCHG is enabled, then the value of the LOCALE system option determines the language for procedure output, user interface elements, and ODS fonts. If ODSLANGCHG is disabled, then messages appear in the language that is set during startup. This feature is supported on any server as long as the characters of the messages are supported by the encoding. For more information, see the "ODSLANGCHG System Option" on page 585. The LSWLANG system option specifies the language for the language switching feature when the LOGLANGCHG or ODSLANGCHG system option is set at SAS invocation. The LSWLANG system option has higher priority over the LOCALE system option. If the LSWLANG option is set to a valid SAS language, the SAS log output is controlled by its value. Otherwise, the LOCALE= option determines the language of the SAS log.

The LOGLANGCHG system option controls whether language switching occurs for the SAS log messages. This option controls the language of message switching in SAS log output. If LOGLANGCHG is specified, the language of the SAS log depends on the LSWLANG or LOCALE= option. For more information, see "LOGLANGCHG System Option" on page 580.

The LSWLANG option specifies the language of messages if LOGLANGCHG or ODSLANGCHG are enabled. If LSWLANG is set to LOCALE, the LOCALE= option determines the language for switching. **`LSWLANG=LOCALE`** is the default. For more information, see "LSWLANG System Option" on page 582.

The LOGLANGENG option is a toggle option that overrides LOGLANGCHG and LSWLANG and sets them to **LOGLANGCHG=ON** and **LSWLANG=EN**. As a result, the LOG output is in English, and the NL format output does not change. This option changes the setting of ODSLANGCHG. If **ODSLANGCHG=OFF**, the system message language for ODS output is determined by the SAS configuration. If **ODSLANGCHG=ON**, all messages are in English because of the LSWLANG setting. For more information, see "LOGLANGENG System Option" on page 581.

For more information about language switching, see Multilingual Computing with SAS® 9.4.

Chapter 3

Encoding for NLS

Overview: Encoding for NLS

An *encoding* maps each character in a character set to a unique numeric representation, which results in a table of all code points. This table is referred to as a *code page*, which is an ordered set of characters in which a numeric index (code point value) is associated with each character. The position of a character on the code page determines its two-digit hexadecimal number.

UTF-8 is the only SAS session encoding supported by SAS Viya.

A *character set* is the set of characters and symbols that are used by a language or group of languages. A character set includes national characters (which are characters specific to a particular nation or group of nations), special characters (such as punctuation marks), the unaccented Latin characters A–Z, the digits 0–9, and control characters that are needed by the computer.

An *encoding method* is a set of rules that assign the numeric representations to the set of characters. These rules govern the size of the encoding (the number of bits used to store the numeric representation of the character) and the ranges in the code page where characters appear. The encoding methods result from the adherence to standards that

have been developed in the computing industry. An encoding method is often specific to the computer hardware vendor.

An encoding results from applying an encoding method to a character set.

An individual character can occupy a different position in a code page, depending on the code page used. An example is the euro symbol, €. The Unicode code point for the euro symbol is U+20AC.

Difference between Encoding and Transcoding

Encoding establishes the default working environment for your SAS session. UTF-8 is the only SAS session encoding supported by SAS Viya.

However, if you are working in an international environment (for example, you access SAS data that is encoded in German EBCDIC), the German EBCDIC code point for the uppercase letter Ä is 4A hexadecimal. For SAS Viya, which supports only UTF-8, to properly interpret a data set that is encoded in German EBCDIC, the data must be transcoded. *Transcoding* is the process of converting data from one encoding to another. For conceptual information, see Chapter 4, "Transcoding for NLS," on page 19.

Common Encoding Methods

An encoding results from applying an encoding method to a coded character set.

An *encoding method* is the application of established industry rules to a coded character set to produce an encoded character scheme. Such rules prescribe the number of bits required for storing the numeric representation of a specific character and its code position in the encoding. ISO 2022 and UTF-8 are examples of encoding methods. For more information, see "Standards Organizations for NLS Encodings" on page 13.

An encoding method is a set of rules that assign numeric representations to a set of characters. These rules govern the size of the encoding, the number of bits used to store the numeric representation of the characters, and the ranges in the encoding where characters appear.

UTF-8 is the only SAS session encoding supported by SAS Viya.

Common encoding methods are listed here:

Unicode is a common and popular character set.

Unicode
: provides up to 110,116 character combinations. Unicode can accommodate basically all of the world's languages. Unicode is a coded character set. HTTP and HTML protocols are based on Unicode. The following UTF encodings are part of the Unicode character set:

UTF-8
: is an MBCS encoding that contains the Latin-script languages, Greek, Cyrillic, Arabic, and Hebrew. It also includes East Asian languages such as Japanese, Chinese, and Korean. The characters in UTF-8 are of varying width, from 1 to 4 bytes. UTF-8 maintains ASCII compatibility by preserving the ASCII characters in code positions 1 through 128.

  UTF-8 is the only SAS session encoding supported by SAS Viya.

UTF-16
: is a 16-bit form that contains all of the most common characters in all modern writing systems. Most of the characters are uniformly represented with 2 bytes, although there is extended space, called *surrogate space*, for additional characters that require 4 bytes.

UTF-32
: is a 32-bit form whose characters each occupy 4 bytes.

ASCII (American Standard Code for Information Interchange)
: is a 7-bit encoding for the United States that provides 128 character combinations. The encoding contains characters for uppercase and lowercase English, American English punctuation, base 10 numbers, and a few control characters. This set of 128 characters is common to most other encodings. ASCII is used by personal and UNIX computers.

EBCDIC (Extended Binary Coded Decimal Interchange Code) family
: is an 8-bit encoding that provides 256 character combinations. There are multiple EBCDIC-based encodings. EBCDIC is used on IBM mainframes and most IBM mid-range computers. EBCDIC follows ISO 646 conventions to facilitate translations between EBCDIC encodings and 7-bit (and 8-bit) ASCII-based encodings. The 95 EBCDIC graphical characters include 82 invariant characters (including a blank space), which occupy the same code positions across most EBCDIC single-byte code pages. It also includes 13 variant graphic characters, which occupy varying code positions across most EBCDIC single-byte code pages. For more information about variant characters, see "Code Point Discrepancies among EBCDIC Encodings" on page 13.

  There are also multi-byte character set (MBCS) EBCDIC encodings.

ISO (International Organization for Standardization) 646 family
: is a 7-bit encoding that is an international standard and provides 128 character combinations. The ISO 646 family of encodings is similar to ASCII except that it has 12 code points for national variants. The 12 national variants represent specific characters that are needed for a particular language.

ISO 8859 family and Windows family
: is an 8-bit extension of ASCII that supports all of the ASCII code points and adds 12 more, providing 256 character combinations. Latin1, which is officially named ISO-8859-1, is the most frequently used member of the ISO 8859 family of encodings. In addition to the ASCII characters, Latin1 contains accented characters, other letters needed for languages of Western Europe, and some special characters.

Other encodings
: The ISO 8859 family has other members that are designed for other languages. The following table describes the other encodings that are approved by ISO.

Table 3.1 *Other Encodings Approved by ISO*

ISO Standard	Name of Encoding	Description
ISO 8859-1	Latin1	US and Western European
ISO 8859-2	Latin 2	Central and Eastern European
ISO 8859-3	Latin 3	Southern European, Maltese, and Esperanto

ISO Standard	Name of Encoding	Description
ISO 8859-4	Baltic	Northern European
ISO 8859-5	Cyrillic	Slavic languages
ISO 8859-6	Arabic	Arabic
ISO 8859-7	Greek	Modern Greek
ISO 8859-8	Hebrew	Hebrew and Yiddish
ISO 8859-9	Turkish	Turkish
ISO 8859-10	Latin 6	Nordic (Inuit, Sámi, Icelandic)
ISO 8859-11	Latin/Thai	Thai
ISO 8859-13	Latin 7	Baltic Rim
ISO 8859-14	Latin 8	Celtic
ISO 8859-15	Latin 9	Western European
ISO-8859-16	Latin 10	Central and Eastern European

Also, a number of encoding standards have been developed for East Asian languages. Some of these are listed in the following table.

Table 3.2 *Some East Asian Language Encodings Approved by ISO*

Standard	Name of Encoding	Description
GB 2312-80	Simplified Chinese	People's Republic of China
CNS 11643	Traditional Chinese	Taiwan
Big-5	Traditional Chinese	Taiwan
KS C 5601	Korean National Standard	Korea
JIS	Japan Industry Standard	Japan
Shift-JIS	Japan Industry Standard multi–byte encoding	Japan

There are other encodings in the standards for EBCDIC and Windows that support different languages and locales. See "Encodings and their Character Set Compatibility and Aliases" on page 655 for common encodings and their aliases.

Standards Organizations for NLS Encodings

Encodings that are supported by SAS are defined by the following standards organizations:

International Organization for Standardization (ISO)
: promotes the development of standardization and related activities to facilitate the free flow of goods and services between nations and to advocate for the exchange of intellectual, scientific, and technological information. ISO also establishes standards for encodings.

American National Standards Institute (ANSI)
: coordinates voluntary standards and conformity to those standards in the United States. ANSI works with ISO to establish global standards.

Unicode Consortium
: develops and promotes the Unicode standard, which provides a unique number for every character.

Code Point Discrepancies among EBCDIC Encodings

The following characters are considered variant because they can have different code positions in various EBCDIC variations: ! # $ @ \ [ ] ^ ` { } | ~ These characters exist in every encoding, but their hexadecimal values might change from one encoding to another, as shown in the following table.

Table 3.3 *Variant Characters*

Character	1047	838	870	1025	1141	1142	1143	1144	1145	1146	1147	1148
!	5A	5A	4F	4F	4F	4F	4F	4F	BB	5A	4F	4F
#	7B	7B	7B	7B	7B	4A	63	B1	69	7B	B1	7B
$	5B	5B	5B	5B	5B	67	67	5B	5B	4A	5B	5B
@	7C	7C	7C	7C	B5	80	EC	B5	7C	7C	44	7C
\	E0	E0	E0	E0	EC	E0	71	48	E0	E0	48	E0
[	AD	49	4A	4A	63	9E	B5	90	4A	B1	90	4A
]	BD	59	5A	5A	FC	9F	9F	51	5A	BB	B5	5A
^	5F	69	5F	5F	5F	5F	5F	5F	BA	BA	5F	5F
`	79	79	79	79	79	79	51	DD	79	79	A0	79
{	C0	C0	C0	C0	43	9	43	44	C0	C0	51	C0

Character	1047	838	870	1025	1141	1142	1143	1144	1145	1146	1147	1148
}	D0	D0	D0	D0	DC	47	47	54	D0	D0	54	D0
|	4F	4F	6A	6A	BB	BB	BB	BB	4F	4F	BB	BB
~	A1	A1	A1	A1	DC	DC	DC	58	BD	BC	BD	A1

Examples of characters that are commonly used in programming languages are { and $.

These characters are known as variant characters. For example, if a German mainframe user entered an ä, which occupies code point C0, an American compiler would interpret code point C0 as a {.

Collating Sequence

Overview to Collating Sequence

The *collating sequence* is the order in which characters are sorted. For example, when the SORT procedure is executed, the collating sequence determines the sort order (higher, lower, or equal to) of a particular character in relation to other characters.

The default collating sequence is binary collation, which sorts characters according to each character's location in the code page of the session encoding. The sort order corresponds directly to the arrangement of the code points within the code page. UTF-8 is the only SAS session encoding supported by SAS Viya.

Binary collation is the fastest type of collation because it is the most efficient for the computer. However, locating characters within a binary-collated report might be difficult if you are not familiar with this method. For example, a binary-collated report lists words beginning with uppercase characters separately from words beginning with lowercase characters, and it lists words beginning with accented characters after words beginning with unaccented characters. Therefore, for ASCII-based encodings, the capital letter **Z** precedes the lowercase letter **a**. Similarly, for EBCDIC-based encodings, the lowercase letter **z** precedes the capital letter **A**.

You can request an alternate collating sequence that overrides the binary collation. To request an alternate collating sequence, specify one of the following sequences:

- a translation table name
- an encoding value
- a linguistic collation

Table 3.4 on page 14 illustrates the results of using different collating sequences to sort a short list of words,

Table 3.4 *Results of Different Collating Sequences*

Binary	Translation Table	Encoding Value	Linguistic
Aaron	aardvark	Aaron	aardvark

Binary	Translation Table	Encoding Value	Linguistic
Aztec	azimuth	Aztec	Aaron
Zeus	Aaron	Zeus	azimuth
aardvark	Aztec	aardvark	Aztec
azimuth	cote	azimuth	cote
cote	coté	cote	côte
coté	côte	coté	coté
côte	côté	côte	côté
côté	zebra	côté	zebra
zebra	zèbre	zebra	zèbre
zèbre	Zeus	zèbre	Zeus

The first column shows the results of binary collation on characters that are represented in an ASCII-based encoding. The alphabetization is not consistent because of the separate grouping of words that begin with uppercase and lowercase characters. For example, the word Zeus appears before aardvark because of the code points that are assigned to the characters within the ASCII-based encoding.

The second column shows the results of specifying a translation table that alternates the ordering of lowercase and uppercase characters. If you use the translation table, the word aardvark appears before Zeus. However, the word azimuth appears before Aaron because the translation table assigns a weight value to the lowercase character **a** that is less than the weight value of the uppercase character **A**. In addition, accents are sorted from left to right. For example, coté comes before côte.

The last column shows the results of linguistic collation for the session locale fr_FR (French_France), which uses a collation algorithm to alphabetize words. The algorithm specifies that words beginning with lowercase characters appear before words beginning with uppercase characters. In addition, this linguistic collation sorts accents from right to left because of the French locale specification.

SAS has adopted the International Components for Unicode (ICU) to implement linguistic collation. The ICU and its implementation of the Unicode Collation Algorithm (UCA) have become a standard. The collating sequence is the default provided by the ICU for the specified locale.

Specifying Linguistic Collation

Linguistic collation sorts characters according to rules of language and produces results that are intuitive and culturally acceptable. The results are similar to the collation used in printed materials such as dictionaries, phone books, and book indexes. Linguistic collation is useful for generating reports or other data presentations and for achieving compatibility between systems.

SAS incorporates the International Components for Unicode (ICU), which is an open-source library that provides routines for linguistic collation that are compatible with the Unicode Collation Algorithm (UCA). The UCA is a standard by which Unicode strings can be compared and ordered.

To request linguistic collation, you must use the SORTSEQ= option in the PROC SORT statement because the SORTSEQ= system option does not support linguistic collation. For example, in the following statements, the SORT procedure collates linguistically, in accordance with the French_France locale:

```
options locale=fr_FR;

proc sort data=myfiles.test sortseq=linguistic;
   by name;
run;
```

When linguistic collation is requested, SAS uses the default linguistic collation algorithm that is provided by the ICU for the SAS session locale. This algorithm reflects the language, the local conventions such as data formatting, and the culture for a geographical region. You can modify the algorithm by specifying options in parentheses following the LINGUISTIC keyword. For example, you can specify a different locale, you can specify the CASE_FIRST= option to collate lowercase characters before uppercase characters or vice versa, and so on. Generally, it is not necessary to specify options, because the ICU associates defaults with the various languages and locales. For more information about the linguistic options, see the SORTSEQ= option in "Collating Sequence Option" on page 599 or the SORTSEQ= option in the PROC SORT statement in *SAS Viya Visual Data Management and Utility Procedures Guide*.

A SAS data set that is sorted linguistically contains a sort indicator that displays the collating sequence LINGUISTIC in the CONTENTS procedure output. Along with the sort indicator, the data set also records a complete description of the linguistic collating sequence in the file's descriptor information, which is also displayed in the CONTENTS procedure output.

Determining the Encoding of a SAS Data Set

Using PROC CONTENTS to Display the Encoding of a SAS Data Set

You can display the encoding of a data set by using PROC CONTENTS:

```
proc contents data=sashelp.class; run;
```

Here is the output from the PROC CONTENTS code. The encoding is US–ASCII.

```
The CONTENTS Procedure

      Data Set Name        SASHELP.CLASS                                  Observations          19
      Member Type          DATA                                           Variables             5
      Engine               V9                                             Indexes               0
      Created              01/17/2016 20:12:45                            Observation Length    40
      Last Modified        01/17/2016 20:12:45                            Deleted Observations  0
      Protection                                                          Compressed            NO
      Data Set Type                                                       Sorted                NO
      Label                Student Data
      Data Representation  SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64
      Encoding             us-ascii  ASCII (ANSI)
```

Encoding Behavior in a SAS Session

Encoding Support for Data Sets by SAS Release

Data sets that are created in SAS automatically have UTF-8 set in the encoding attribute, which is in the descriptor portion of the file.

Output Processing

When you create a data set in SAS, the encoding is determined as follows:

- If a new output file is created, the data is written to the current session encoding, UTF-8.
- If a new output file is created using the OUTREP= option, the data is written to the file using the default session encoding for the operating system that is specified by the OUTREP= value. For more information, see "OUTREP= Data Set Option" on page 44.
- If a new output file replaces an existing file, the new file inherits the encoding of the existing file. For output processing that replaces an existing file that is from another operating environment or if the existing file has no encoding specified, then the current session encoding is used.

Input Processing

For input (read) processing in SAS, the encoding behavior is as follows:

- Most users choose the default behavior that does not specify an encoding for the input file.
- If the session encoding and the encoding that is specified in the file are incompatible, the data is transcoded to the session encoding. For example, if the encoding specified for the file is LATIN1, SAS Viya transcodes the file from LATIN1 to its session encoding of UTF-8.
- If a file does not have an encoding specified, SAS transcodes the data only if the file's data representation is different from the current session.

Reading and Writing External Files

SAS reads and writes external files using the current session encoding. SAS assumes that the external file has the same encoding as the session encoding. SAS Viya assumes that the encoding of the file is UTF-8. For example, if you are creating a new SAS data set by reading an external file, SAS assumes that the encoding of the external file and the current session are the same. If the encodings are not the same, the external data could be written incorrectly to the new SAS data set. You need to specify an appropriate ENCODING option. The following example specifies the Shift-JIS encoding:

```
filename in 'external-file'
 encoding='Shift-JIS';
data mylib.contacts;
infile in;
```

```
length name $ 30 first $ 30 street $ 60 zip $ 10 city $ 30;
input name first street zip city;
run;
```

For details about the syntax for the SAS statements that perform input and output processing, see "SAS Options That Transcode SAS Data" on page 20.

Chapter 4

Transcoding for NLS

Overview to Transcoding

Transcoding is the process of converting data from one encoding to another. Transcoding is necessary when the SAS session encoding and the encoding of the data are different. Transcoding is often necessary when you move data between operating environments that use different locales and encoding. UTF-8 is the only SAS session encoding supported by SAS Viya.

The sort order corresponds directly to the arrangement of the code points within the code page.

For example, consider a file that was created under a Linux operating environment that uses the Latin1 encoding and then moved to SAS Viya, which uses the UTF-8 encoding. When the file is processed on SAS Viya, the data is remapped from the LATIN1 encoding to the UTF-8 encoding.

Transcoding does not translate between languages; transcoding remaps characters.

To dynamically transcode data between operating environments that use different encodings, an explicit encoding value must be specified. For more information, see Chapter 18, “Encoding Values in SAS Language Elements,” on page 643.

Common Reasons for Transcoding

Some situations where data might commonly be transcoded are:

- sharing data between two different SAS sessions that are running in different locales or in different operating environments
- reading an external file with an encoding other than UTF-8.

SAS Options That Transcode SAS Data

The following SAS options for various language elements enable you to transcode or to override the default encoding behavior. These elements enable you to specify a different encoding for a SAS file or a SAS application or to suppress transcoding.

Table 4.1 *SAS Options That Transcode SAS Data*

Option	Where Used
CHARSET=	ODS MARKUP statement
CORRECTENCODING=	MODIFY statement of the DATASETS procedure
ENCODING=	%INCLUDE, FILE, FILENAME, INFILE, ODS statements; FILE and INCLUDE commands
ENCODING=	DATA step
INENCODING=	LIBNAME statement
ODSCHARSET=	LIBNAME statement for XML
ODSTRANTAB=	LIBNAME statement for XML
OUTENCODING=	LIBNAME statement
XMLENCODING=	LIBNAME statement for XML

For a list of supported encoding values to use for these options, see "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

Transcoding Considerations

Although transcoding usually occurs without problems, there are situations that can affect your data and produce unsatisfactory results. For example:

- One encoding can conflict with another. That is, two encodings can use different code points for the same character or use the same code points for two different characters.
- Characters in one encoding might not be present in another encoding. For example, a specific encoding might not have a character for the dollar sign ($). Transcoding the data to an encoding that does not support the dollar sign would result in the character not printing or displaying.
- The number of bytes for a character in one encoding can be different from the number of bytes for the same character in another encoding. An example is transcoding from a double-byte character set (DBCS) to an SBCS. Therefore, transcoding can result in character value truncation.
- If an error occurs during transcoding so that the data cannot be transcoded back to its original encoding, data can be lost. That is, if you open a data set for update processing, the observation might not be updated. However, if you open the data set for input (read) processing and no output data set is open, SAS issues a warning that can be printed. Processing proceeds and allows a PRINT procedure or other Read operation to show the data that does not transcode.
- CEDA has some processing limitations. For example, CEDA does not support update processing.
- Incorrect encoding can be stamped on a SAS 7 or SAS 8 data set if it is copied or replaced in a SAS Viya session. The incorrect encoding stamp can be corrected with the CORRECTENCODING= option in the MODIFY statement in PROC DATASETS. If a character variable contains binary data, transcoding might corrupt the data.

Compatible and Incompatible Encodings

Overview to Compatible and Incompatible Encodings

An *incompatible file* is either a file that has a different data representation from the host architecture or an encoding value that differs from the SAS session encoding.

ASCII has been the foundation for most encodings, and it is used by most personal computers, minicomputers, and workstations. However, the IBM mainframe uses an EBCDIC encoding. Therefore, ASCII and EBCDIC machines and data are incompatible. Transcoding is necessary if some or all characters in one encoding are different from the characters in the other encoding.

UTF-8 is the only SAS session encoding supported by SAS Viya.

However, to avoid transcoding, you can create a data set and specify an encoding value that SAS does not transcode. For example, if you use the following values in either the

ENCODING= data set option or the INENCODING= or the OUTENCODING= option in the LIBNAME statement, transcoding is not performed:

- ANY specifies that no transcoding is desired, even between EBCDIC and ASCII encodings.

  Note: ANY is a synonym for binary. Because the data is binary, the actual encoding is irrelevant.

- ASCIIANY enables you to create a data set that is compatible with all ASCII-based encodings.
- EBCDICANY enables you to create a data set that is compatible with all EBCDIC-based encodings.

You might want to create a SAS data set that contains mixed encodings. An example is both Latin1 and Latin2. You do not want the data transcoded for either input or output processing. By default, data is transcoded to the current session encoding.

Data must be transcoded when the SAS file does not match the SAS Viya session encoding of UTF-8.

In some cases, transcoding is not required because the SAS file and the SAS session have compatible encodings.

Line-feed Characters and Transferring Data between EBCDIC and ASCII

Software that runs under ASCII operating environments requires that the end of the line be specified by the line-feed character. When data is transferred from z/OS to a machine that supports ASCII encodings, formatting problems can occur, particularly in HTML output, because the EBCDIC newline character is not recognized. SAS supports two sets of EBCDIC-based encodings for z/OS:

- The encodings that have EBCDIC in their names use the traditional mapping of an EBCDIC line-feed character to an ASCII line-feed character, which can cause data to appear as one stream.
- The encodings that have Open Edition in their names use the line-feed character as the end-of-line character. When the data is transferred to an operating environment that uses ASCII, the EBCDIC newline character maps to an ASCII line-feed character. This mapping enables ASCII applications to interpret the end of the line correctly, resulting in better formatting.

Preventing Transcoding

Some encoding values enable you to create a data set that SAS does not transcode. You might not want to transcode data for input or output processing. You might want to create a SAS library that contains data in mixed encodings instead. An example is both LATIN1 and LATIN2.

For example, you can avoid transcoding if you use the following values in either the ENCODING= data set option or the INENCODING= or OUTENCODING= option in the LIBNAME statement:

- ANY specifies that no transcoding is desired, even between EBCDIC and ASCII encodings.

Note: ANY is a synonym for binary. Because the data is binary, the actual encoding is irrelevant.

- ASCIIANY specifies that no transcoding is required between any ASCII-based encodings.
- EBCDICANY specifies that no transcoding is required between any EBCDIC-based encodings.

For more information, see "ENCODING= Data Set Option" on page 41 and "INENCODING= and OUTENCODING= Options" on page 615.

You can prevent transcoding for a specific column of data while the rest of the character data in the data set is transcoded by using the TRANSCODE= option. For more information, see "TRANSCODE= Column Modifier on PROC SQL" on page 618.

Avoiding Character Data Truncation By Using the CVP Engine

If your data set encoding does not match the encoding of your SAS session, the character data in your data set might need more space in the new encoding. For example, if your SAS session is using a UTF-8 session encoding and you are reading a data set with an encoding of Windows cp 1252 (WLATIN1), some of the characters that require 1 byte in WLATIN1 might require 2 or 3 bytes in UTF-8. To avoid data truncation, your character variables must expand to a width that is 1.5 times greater than the width of the original data.

When you process a SAS data file that requires transcoding, you can request that the CVP (character variable padding) engine expand character variable lengths so that character data truncation does not occur. (A variable's length is the number of bytes used to store each of the variable's values.)

Character data truncation can occur when the number of bytes for a character in one encoding is different from the number of bytes for the same character in another encoding, such as when a single-byte character set (SBCS) is transcoded to a double-byte character set (DBCS) or to a multi-byte character set (MBCS). An SBCS represents each character in 1 byte, and a DBCS represents each character in 1 or 2 bytes. An MBCS represents characters in a varying length from 1 to 4 bytes. For example, when transcoding from Wlatin2 to a Unicode encoding, such as UTF-8, the variable lengths (in bytes) might not be sufficient to hold the values, and the result is character data truncation.

Using the CVP engine, you specify an expansion amount so that variable lengths are expanded before transcoding, and then the data is processed. Think of the CVP engine as an intermediate engine that prepares the data for transcoding. After the lengths are increased, the primary engine, such as the default base engine, actually processes the file.

The CVP engine is a read-only engine for SAS data files only. You can request character variable expansion (for example, with the LIBNAME statement) in either of the following ways:

- explicitly specify the CVP engine and use the default expansion of 1.5 times the variable lengths.
- implicitly specifying the CVP engine with the LIBNAME options CVPBYTES= or CVPMULTIPLIER=. These options specify the expansion amount. In addition, you

can use the CVPENGINE= option to specify the primary engine to process the SAS file; the default is the default SAS engine.

For example, the following LIBNAME statement explicitly assigns the CVP engine. Character variable lengths are increased using the default expansion, which multiples the lengths by 1.5. For example, a character variable with a length of 10 has a new length of 15, and a character variable with a length of 100 has a new length of 150:

```
libname expand cvp ' SAS data-library';
```

Note: The expansion amount must be large enough to accommodate any expansion. Otherwise, truncation still occurs. An error is written to the SAS log.

Note: For processing that conditionally selects a subset of observations by using a WHERE expression, using the CVP engine might affect performance. Processing the file without using the CVP engine might be faster than processing the file using the CVP engine. For example, if the data set has indexes, the indexes are not used in order to optimize the WHERE expression if you use the CVP engine.

Note: The CVP engine does not adjust formats or informats automatically. If you have attached formats to your data variables, you might have truncation issues. Aadjust these formats manually or programmatically.

Note: CVP creates a read-only copy of the data. If you want to save a permanent copy of the data, you need to create a new data set.

For more information and examples, see "CVPBYTES=, CVPENGINE=, and CVPMULTIPLIER= Options" on page 605.

Chapter 5

Time Zones

Overview of SAS Time Zone IDs

SAS must process local and international time and date values to accommodate international customers. This chapter explains time zones and how SAS processes the following:

- time zones
- Daylight Saving Time (DST)
- local time
- SAS datetime values

Specifying Time Zones in SAS

Definitions

Daylight Saving Time (DST)
Also referred to as *Summer Time*, *Daylight Saving Time (DST)* is a way of making better use of the daylight in the evenings by setting the clocks forward one hour during the longer days of summer and back again in the fall. DST begins in the northern hemisphere between March–April and ends between September–November. Standard time begins in the northern hemisphere between September–November and ends between March–April. DST begins in the southern hemisphere between September–November and ends between March–April. Standard time begins in the southern hemisphere between March–April and ends between September–November.

Many of the countries in the northern hemisphere observe DST. DST is not a standard for all countries.

Local time
: Specifies the local time when a time zone is not specified by the TIMEZONE= system option.

Time zone
: An area of the earth's surface in which an identical and standard time is used. It is usually referred to as the local time. Time zones have political and geographical boundaries and might be adjusted for the convenience of local populations. Some geographically large countries, such as India and China, use only one time zone, but other large countries, such as Russia and the United States, have more than one time zone.

Time zone ID
: Specifies a region and area separated by a forward slash (/). Asia/Tokyo is a time zone ID. Time zone IDs are compatible with Java time zones. For a list of time zone IDs, see Appendix 2, "Time Zone IDs and Time Zone Names," on page 663.

Time zone name
: Specifies three or four characters that indicate a time zone. For a list of time zone names, see Appendix 2, "Time Zone IDs and Time Zone Names," on page 663.

Time zone offset
: Specifies the number of hours and minutes that a time zone is off from the Universal Coordinated Time (UTC) in the form +|-*hh:mm* or +|-*hhmm*.

User local time
: Specifies the local time for the time zone that is specified by the TIMEZONE= system option.

Universal Coordinated Time (UTC)
: Specifies the time at the zero meridian, near Greenwich, England. UTC is a datetime value that uses the ISO 8601 basic form *yyyymmdd*T*hhmmss*+|-*hhmm* or the ISO 8601 extended form *yyyy-mm-dd*T*hh:mm:ss*+|-*hh:mm*.

About Time Zones in SAS

SAS datetime values are measured in seconds, beginning with 01Jan1960 00:00:00 local time. For example, the SAS datetime value for 01Jan1960 00:00:00 in England is 0. In Japan, the SAS datetime value for 01Jan1960 00:00:00 is also 0. There is a nine-hour difference between England and Japan. At the same point in time, the datetime in England and the datetime in Japan cannot be 0. In order to work with absolute time and datetime values, SAS supports the UTC date, time, and datetime values in time zones.

You can name a time zone either by specifying a time zone ID or a time zone name. Time zone IDs name a region and an area in the form *region/area*. An example of a time zone ID is America/New_York. A time zone name specifies the time zone. An example of a time zone name is EST for Eastern Standard Time. When a specific time zone changes the time for DST or summer time, a separate time zone name is available for that alternate time. PST is Pacific Standard Time. PDT is Pacific Daylight Time. EET is Eastern European Time. EEST is Eastern European Summer Time. When you specify a time zone ID, SAS determines the time using DST. You do not need to determine whether the region/area uses alternate times. For a list of time zone IDs and time zone names, see Appendix 2, "Time Zone IDs and Time Zone Names," on page 663.

When you specify a time zone, SAS adds a time-zone-specific timestamp to data sets and SAS catalogs when the data set or catalog is created or modified. The time-zone-

specific timestamp is also added to the SAS log when SAS starts and to the output at execution. The SAS timestamp constant enables you to specify a timestamp using the ISO 8601 standard. Here is a timestamp using the SAS timestamp constant:

```
tstamp='2013-05-17T09:15:30-05:00'dt;
```

When you specify a SAS timestamp constant, SAS converts the timestamp to a local datetime value by using the current time zone and the time zone offset.

Time zone settings affect the TIME( ) function, the TODAY( ) function, the DATE( ) function, and the DATETIME( ) function. When a time zone is set, the date and datetime values use the current time zone.

Tasks and SAS Language Elements for Time Zone Processing

Set a Time Zone

To set a time zone, use the TIMEZONE= system option:

```
options timezone='asia/tokyo';
```

Use time zone names or time zone IDs for the value of the option. Enclose the value in quotation marks. For time zone names and time zone IDs, see Appendix 2, "Time Zone IDs and Time Zone Names," on page 663. For more information, see "TIMEZONE= System Option" on page 588.

Determine a Time Zone ID Offset

You use the TZONEOFF( ) function to determine a time zone name or time zone ID offset:

- The TZONEOFF( ) function returns the time zone offset for the current time zone.
- The TZONEOFF('*time-zone-ID*') function returns the time zone offset for the '*time-zone-ID*'.

This program returns the time zone offset for the current time zone (EST) and for Tokyo:

```
data _null_;
   o1=tzoneoff();
   o2=tzoneoff('asia/tokyo');
   put o1 time.;
   put o2 time.;
run;
```

Here is the output in the SAS log:

```
-5:00:00
 9:00:00
```

To find the difference between two time zones, you can use the ABS( ) function:

```
diff=abs(tzoneoff('america/new_york') - tzoneoff('asia/tokyo'));
```

For more information, see "TZONEOFF Function" on page 414.

Determine a Time Zone ID or Time Zone Name

Use these functions to determine a time zone name or time zone ID:

- The TZONEID( ) function returns the current time zone ID.

- The TZONENAME( ) function returns the current time zone name based on a time zone ID and a SAS datetime value or based on a SAS datetime value only.

Here are some examples of using these functions. March 10 of 2013 is the first day of DST:

```
options timezone='America/Chicago';
data _null_;
   tzid=tzoneid();
   put 'Current time zone is ' tzid;
   tzn=tzonename('america/los_angeles');
   put 'Time zone for Los Angeles: ' tzn;
   tznST=tzonename('america/los_angeles','10mar2013:01:00:00'dt);
   put 'Time zone for Los Angeles standard time: ' tznST;
   tznDT=tzonename('america/los_angeles','10mar2013:02:00:00'dt);
   put 'Time zone for Los Angeles daylight time: ' tznDT;
   tznSDT=tzonename('10mar2013:02:00:00'dt);
   put 'Time zone name for this SAS datetime: ' tznSDT;
run;
```

Here is the output:

```
Current time zone is AMERICA/CHICAGO
Time zone for Los Angeles: PST
Time zone for Los Angeles standard time: PST
Time zone for Los Angeles daylight time: PDT
Time zone name for this SAS datetime: CDT
```

For more information, see "TZONEID Function" on page 411 and "TZONENAME Function" on page 412.

Convert Datetime Values between SAS and UTC

These functions convert SAS datetime values to UTC and UTC to SAS datetime values:

- The TZONES2U( ) converts a SAS datetime value to a UTC datetime value.
- The TZONEU2S( ) function converts a UTC datetime value to a SAS datetime value.

Here are some examples of using these functions:

```
options timezone='est';
data _null_;
   put ' The time zone is EST';
   diff=abs(tzoneoff('america/new_york') - tzoneoff('europe/london'));
   put ' New York-London difference:' diff time.;
   diff=abs(tzoneoff('america/new_york') - tzoneoff('asia/tokyo'));
   put ' New York-Tokyo difference:' diff time.;
   put ' The SAS datetime is 2013-03-15T09:15:00+00:00 ';
   put ' ';
   put ' Change a SAS datetime to a UTC value ';
   put ' ';
   put ' The time zone offset +00:00 is for London ';
   put ' Subtract the 5 hours for the EST time zone offset';
   stu1=tzones2u('2013-03-15T09:15:00+00:00'dt);
   put ' STU1 Using E8601DX:' stu1 e8601dx.;
   put ' ';
   put ' 2013-03-15 9:15 AM in Tokyo is 2013-03-14 7:15 PM in New York';
   put ' Subtract the 5 hours for the EST time zone offset';
```

```
   stu2=tzones2u('2013-03-15T09:15:00+00:00'dt, 'Asia/Tokyo');
   put ' STU2 Using E8601DX:' stu2 e8601dx.;
   put ' ';
   put ' Change a UTC to a SAS datetime value. ';
   put ' ';
   put ' +00:00 is the time zone offset for London.';
   put ' Subtract the 5 hours for the EST time zone offset';
   uts1=tzoneu2s('2013-03-15T09:15:00+00:00'dt);
   put ' UTS1 Using DATETIME:' uts1 datetime.;
   put ' ';
   put ' 9:15:00+00:00 is 18:15:00 in Tokyo. ';
   put ' Subtract the 5 hours for the EST time zone offset';
   uts2=tzoneu2s('2013-03-15T09:15:00+00:00'dt, 'Asia/Tokyo');
   put ' UTS2 Using DATETIME:' uts2 datetime.;
run;
```

Here is the output:

```
The time zone is EST
 New York-London difference: 5:00:00
 New York-Tokyo difference:14:00:00
 The SAS datetime is 2013-03-15T09:15:00+00:00

 Change a SAS datetime to a UTC value

 The time zone offset +00:00 is for London
 Subtract the 5 hours for the EST time zone offset
 STU1 Using E8601DX: 2013-03-15T04:15:00-05:00

 2013-03-15 9:15 AM in Tokyo is 2013-03-14 7:15 PM in New York
 Subtract the 5 hours for the EST time zone offset
 STU2 Using E8601DX: 2013-03-14T14:15:00-05:00

 Change a UTC to a SAS datetime value.

 +00:00 is the time zone offset for London.
 Subtract the 5 hours for the EST time zone offset
 UTS1 Using DATETIME:14MAR13:23:15:00

 9:15:00+00:00 is 18:15:00 in Tokyo.
 Subtract the 5 hours for the EST time zone offset
 UTS2 Using DATETIME:15MAR13:13:15:00
```

For more information, see "TZONES2U Function" on page 415.

Write ISO 8601 Time and Datetime Values Based on the Time Zone

These formats write ISO 8601 time and datetime values using basic and extended notation. The time, the UTC offset, or both are based on the value of the TIMEZONE= system option:

- The B8601TX. and E8601TX. formats convert a UTC time value to a user's local time and write the time value with a time zone offset.
- The B8601DX. and E8601DX. formats convert a UTC datetime value to a user's local time and write the datetime value with a time zone offset.
- The B8601LX. and E8601LX. formats write a local datetime value using the UTC offset for the user's local time zone.

This program formats time and datetime values based on the time zone ID Australia/Sydney:

```
options timezone='Australia/Sydney';
data _null_;
   st='18:33:40't;
   sdt='2013-03-17T14:30:22+00:00'dt;
   put 'Time B8601TX:' st b8601tx.;
   put 'Time E8601TX:' st e8601tx.;
   put 'UTC B8601DX:' sdt b8601dx.;
   put 'UTC E8601DX:' sdt e8601dx.;
   put 'UTC B8601LX:' sdt b8601lx.;
   put 'UTC E8601LX:' sdt e8601lx.;
run;
```

Here is the output:

```
Time B8601TX:   043340+1000
Time E8601TX:04:33:40+10:00
UTC B8601DX:      20130318T123022+1100
UTC E8601DX: 2013-03-18T12:30:22+11:00
UTC B8601LX:      20130318T013022+1100
UTC E8601LX: 2013-03-18T01:30:22+11:00
```

For more information, see "B8601DX Format" on page 82, "B8601LX Format" on page 84, "B8601TX Format" on page 85, "E8601DX Format" on page 91, "E8601LX Format" on page 93, and "E8601TX Format" on page 94.

Write SAS Datetime Values Using a Time Zone Offset

These formats write SAS datetime values using a UTC offset based on the value of the TIMEZONE= system option:

- The NLDATMZ. format writes a SAS datetime value using the form *ddmmmyyyy:hh:mm:ss* +|-*hhmm*.
- The NLDATMTZ. format writes a SAS time value using the form *hh:mm:ss* +|-*hhmm*.
- The NLDATMWZ. format writes a SAS datetime value as the day of the week, the month, the day, the year, and AM | PM in the form *day-of-week*, *month-name dd*, *yyyy* AM|PM +*hhmm*.

This program formats SAS time and datetime values using a time zone offset:

```
options timezone='Indian/Maldives';
data _null_;
   st='18:33:40't;
   sdt='2013-03-17T14:30:22+00:00'dt;
   put 'Time NLDATMTZ:' st nldatmtz.;
   put 'SAS datetime NLDATMZ:' sdt nldatmz.;
   put 'SAS datetime NLDATMWZ:' sdt nldatmwz.;
run;
```

Here is the output:

```
Time NLDATMTZ:18:33:40 +0500
SAS datetime NLDATMZ:17Mar2013:19:30:22 +0500
SAS datetime NLDATMWZ:Sunday, March 17, 2013 07:30:22 PM +0500
```

For more information, see "NLDATMTZ Format" on page 138, "NLDATMZ Format" on page 151, and "NLDATMWZ Format" on page 141.

Time Zone Example

This example shows the arrival time of a flight from Los Angeles to Tokyo:

```
/* Set the time zone */
options timezone='America/Los_Angeles';
data depart;
/* Set the departure time */
   depart='2013-05-17T09:15:00-08:00'dt;
   put 'Depart Los Angeles: ' depart nldatmwz.;
/* Set the flight time */
   ftime='13:00't;
   put 'Flight time=' ftime time.;
   utc=depart+ftime;
   put 'Arrive PST=' utc nldatmwz.;
   put 'Arrive UTC=' utc nldatmwz.;
run;
/* Set the time zone for Tokyo */
options timezone='Asia/Tokyo';
data arrive;
   set depart;
   put 'Arrive in Tokyo ' utc nldatmwz.;
run;
```

Here is the output:

```
39   /* Set the time zone */
40   options timezone='America/Los_Angeles';
41   data depart;
42   /* Set the departure time */
43      depart='2013-05-17T09:15:00-08:00'dt;
44      put 'Depart Los Angeles: ' depart nldatmwz.;
45   /* Set the flight time */
46      ftime='13:00't;
47      put 'Flight time=' ftime time.;
48      utc=depart+ftime;
49      put 'Arrive PST=' utc nldatmwz.;
50      put 'Arrive UTC=' utc nldatmwz.;
51   run;

Depart Los Angeles: Friday, May 17, 2013 10:15:00 AM -0700
Flight time=13:00:00
Arrive PST=Friday, May 17, 2013 11:15:00 PM -0700
Arrive UTC=Friday, May 17, 2013 11:15:00 PM -0700
NOTE: The data set WORK.DEPART has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


52   /* Set the time zone for Tokyo */
53   options timezone='Asia/Tokyo';
54   data arrive;
55      set depart;
56      put 'Arrive in Tokyo ' utc nldatmwz.;
57   run;

Arrive in Tokyo Friday, May 17, 2013 11:15:00 PM +0900
NOTE: There were 1 observations read from the data set WORK.DEPART.
NOTE: The data set WORK.ARRIVE has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Part 2

Autocall Macros for NLS

Chapter 6

Dictionary of Autocall Macros for NLS

Autocall Macro Entries by Category

The following table provides brief descriptions of the SAS NLS autocall macros. For more detailed descriptions, see the NLS entry for each macro.

Category	Language Elements	Description
DBCS	%KLOWCASE and %QKLOWCAS Autocall Macros (p. 35)	Change uppercase characters to lowercase.
	%KTRIM and %QKTRIM Autocall Macros (p. 36)	Trim trailing blanks.
	%KVERIFY Autocall Macro (p. 36)	Returns the position of the first character unique to an expression.

Dictionary

%KLOWCASE and %QKLOWCAS Autocall Macros

Change uppercase characters to lowercase.

Category: DBCS

Requirement: MAUTOSOURCE system option

Syntax

%KLOWCASE (text | text expression)

%QKLOWCAS (text | text expression)

Details

The %KLOWCASE and %QKLOWCAS macros change uppercase alphabetic characters to their lowercase equivalents. If the argument might contain a special character or mnemonic operator, listed below, use %QKLOWCAS.

%KLOWCASE returns a result without quotation marks, even if the argument has quotation marks. %QKLOWCAS produces a result with the following special characters and mnemonic operators masked so that the macro processor interprets them as text instead of as elements of the macro language:

& % ' " ( ) + - * / < > = ¬ ^ ~ ; , blank AND OR NOT EQ NE LE LT GE GT IN

Autocall macros are included in a SAS library. This library might not be installed at your site or might be a site-specific version. If you cannot access this macro or if you want to find out if the library is a site-specific version, see your on-site SAS support personnel.

%KTRIM and %QKTRIM Autocall Macros

Trim trailing blanks.

Category: DBCS

Requirement: MAUTOSOURCE system option

Syntax

%KTRIM (text | text expression)

%QKTRIM (text | text expression)

Details

The KTRIM macro and the QKTRIM macro trim trailing blanks. If the argument contains a special character or mnemonic operator, listed below, use %QKTRIM.

QKTRIM produces a result with the following special characters and mnemonic operators masked so that the macro processor interprets them as text instead of as elements of the macro language:

& % ' " ( ) + - * / < > = ¬ ? ~ ; , # blank AND OR NOT EQ NE LE LT GE GT IN

Autocall macros are included in a SAS library. This library might not be installed at your site or might be a site-specific version. If you cannot access this macro or if you want to find out if the library is a site-specific version, see your on-site SAS support personnel.

%KVERIFY Autocall Macro

Returns the position of the first character unique to an expression.

Category: DBCS

Requirement: MAUTOSOURCE system option

Syntax

%KVERIFY (source, excerpt)

Syntax

source
is text or a text expression that you want to examine for characters that do not exist in excerpt.

excerpt
is text or a text expression that defines the set of characters that %KVERIFY uses to examine source.

Details

%KVERIFY returns the position of the first character in source that is not also present in excerpt. If all characters in source are present in excerpt, %KVERIFY returns 0.

Autocall macros are included in a SAS library. This library might not be installed at your site or might be a site-specific version. If you cannot access this macro or if you want to find out if the library is a site-specific version, see your on-site SAS support personnel.

Part 3

Data Set Options for NLS

Chapter 7

Dictionary of Data Set Options for NLS

Data Set Options by Category

NLS affects the data set control category of options for selected data set options. The following table provides brief descriptions of the data set options. For more detailed descriptions, see the dictionary entry for each data set option.

Category	Language Elements	Description
Data Set Control	ENCODING= Data Set Option (p. 41)	Overrides the encoding to use for reading or writing a SAS data set.
	OUTREP= Data Set Option (p. 44)	Specifies the data representation for the output SAS data set.

Dictionary

ENCODING= Data Set Option

Overrides the encoding to use for reading or writing a SAS data set.

Valid in: DATA step and PROC steps

Category: Data Set Control

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

ENCODING= ANY | ASCIIANY | EBCDICANY | *encoding-value*

Syntax Description

ANY
specifies that no transcoding occurs.

Note ANY is a synonym for binary. Because the data is binary, the actual encoding is irrelevant.

ASCIIANY
specifies that no transcoding occurs when the mixed encodings are ASCII encodings.

EBCDICANY
specifies that no transcoding occurs when the mixed encodings are EBCDIC encodings.

encoding-value
specifies an encoding value.

See Chapter 3, "Encoding for NLS," on page 9

Details

The value for ENCODING= indicates that the SAS data set has a different encoding from the current session encoding. When you read data from a data set, SAS transcodes the data from the specified encoding to the session encoding. When you write data to a data set, SAS transcodes the data from the session encoding to the specified encoding.

UTF-8 is the only SAS session encoding supported by SAS Viya.

Input Processing

By default, encoding for input processing is determined as follows:

- If the session encoding and the encoding that is specified in the file are different, SAS transcodes the data to the session encoding.
- If a file has no encoding specified, but the file's data representation is different from the encoding of the current session, then SAS transcodes the data to the current session.

Output Processing

By default, encoding for output processing is determined as follows:

- Data is written to a file using the encoding of the current session, except when a different output representation is specified using the OUTREP= data set option, the OUTENCODING= option in the LIBNAME statement, or the ENCODING= data set option.
- If a new file replaces an existing file, then the new file inherits the encoding of the existing file.
- If an existing file is replaced by a new file that was created under a different operating environment or that has no encoding specified, the new file uses the encoding of the current session.

Note: Character metadata and data output appears garbled if you specify a different encoding from where the data set was created. In this example, the data set to be printed is internally encoded as ASCII. However, the data set option specifies an

EBCDIC encoding. SAS attempts to transcode the data from EBCDIC to ASCII, but the data is already in ASCII. The result is garbled data.

```
data a;
x=1;
abc='abc';
run;
proc print data=a (encoding="ebcdic");
run;
```

Note: The following values for ENCODING= are invalid:

- UCS2
- UCS4
- UTF16
- UTF32

Comparisons

- Session encoding is specified using the ENCODING= system option or the LOCALE= system option, with each operating environment having a default encoding.UTF-8 is the only SAS session encoding supported by SAS Viya.
- You can specify encoding for a SAS library by using the LIBNAME statement's INENCODING= option (for input files) and the OUTENCODING= option (for output files). If both the LIBNAME statement option and the ENCODING= data set option are specified, SAS uses the data set option.

Examples

Example 1: Creating a SAS Data Set with Mixed Encodings and with Transcoding Suppressed

By specifying the data set option ENCODING=ANY, you can create a SAS data set that contains mixed encodings, and suppress transcoding for either input or output processing.

In this example, the new data set MYFILES.MIXED contains some data that uses the Latin1 encoding, and some data that uses the Latin2 encoding. When the data set is processed, no transcoding occurs. For example, the correct Latin1 characters in a Latin1 session encoding and correct Latin2 characters in a Latin2 session encoding are displayed.

```
libname myfiles 'SAS data-library';
data myfiles.mixed (encoding=any);
   set work.latin1;
   set work.latin2;
run;
```

Example 2: Creating a SAS Data Set with a Particular Encoding

For output processing, you can override the current session encoding. This action might be necessary, for example, if the data set is being created for use in a SAS environment other than SAS Viya.

For example, if the current session encoding is UTF-8, you can specify ENCODING=WLATIN2 in order to create the data set that uses the encoding Wlatin2. The following statements tell SAS to write the data to the new data set using the Wlatin2

encoding instead of the session encoding. The encoding is also specified in the descriptor portion of the file.

```
libname myfiles 'SAS data-library';
data myfiles.difencoding (encoding=wlatin2);
      .
      .
      .
run;
```

See Also

- Conceptual discussion in Chapter 3, "Encoding for NLS," on page 9

Options in Statements and Commands:

- "ENCODING= Option" on page 612
- "INENCODING= and OUTENCODING= Options" on page 615

System Options:

- "ENCODING System Option" on page 577
- "LOCALE System Option" on page 578

OUTREP= Data Set Option

Specifies the data representation for the output SAS data set.

Valid in: DATA step and PROC steps

Category: Data Set Control

See: "OUTREP= Data Set Option" in *SAS Viya Data Set Options: Reference*.

Syntax

OUTREP= *format*

See Also

"OUTREP= Data Set Option" in *SAS Viya Data Set Options: Reference*

Part 4

Formats for NLS

Chapter 8

Overview to NLS Formats

International Date and Datetime Formats

SAS supports international formats that are equivalent to some of the most commonly used English-language date formats. In each case the format works like the corresponding English-language format. Only the maximum, minimum, and default widths are different.

Table 8.1 *International Date and Datetime Formats*

Language	English Format	International Format	Min	Max	Default
Afrikaans (AFR)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	10
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9

Language	English Format	International Format	Min	Max	Default
	WORDDATX.	NLDATE.	10	200	20
Catalan (CAT)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Croatian (CRO)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Czech (CSY)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Danish (DAN)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9

Language	English Format	International Format	Min	Max	Default
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Dutch (NLD)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WORDDATX.	NLDATE.	10	200	20
	WEEKDAY.	NLDATEWN.	4	200	9
Finnish (FIN)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
French (FRA)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
German (DEU)	DATE.	NLDATE.	105	200	20

Language	English Format	International Format	Min	Max	Default
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Hungarian (HUN)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Italian (ITA)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Macedonian (MAC)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29

Language	English Format	International Format	Min	Max	Default
	WEEKDDATX.	EURDFWDX.	1	32	1
	WORDDATX.	NLDATEWN.	4	200	9
Norwegian (NOR)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Polish (POL)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	20	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Portuguese (PTG)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATEWX.	10	200	20
Russian (RUS)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9

Language	English Format	International Format	Min	Max	Default
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Spanish (ESP)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Slovenian (SLO)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20
Swedish (SVE)	DATE.	NLDATE.	10	200	20
	DATETIME.	NLDATM.	10	200	30
	DOWNAME.	NLDATEWN.	4	200	9
	MONNAME.	NLDATEMN.	4	200	9
	MONYY.	NLDATEYM.	6	200	16
	WEEKDATX.	NLDATEW.	10	200	29
	WEEKDAY.	NLDATEWN.	4	200	9
	WORDDATX.	NLDATE.	10	200	20

Currency Representation

Overview to Currency

Currency is the medium of exchange, which is specific to a country. SAS provides formats and informats for reading and writing currency.

U.S. Dollars

The DOLLAR*w.d* formats and informats were first introduced to read and write American currency. DOLLARw.d

- uses the dollar sign ($) currency symbol to precede U.S. currency
- uses a comma (,) as the thousands separator and a dot (.) as the decimal separator.

Example:

```
$12,345.00
```

DOLLARXw.d also writes currency with a leading dollar sign ($), but it uses a dot (.) as the thousands separator and a comma (,) as the decimal separator. The reversal of the dot and comma for currency formatting is a convention used in many European countries.

Example:

```
 $12.345,00
```

Limitations of the DOLLAR formats and informats are

- lack of support for all currency symbols is a limitation
- reversal of the dot and comma for currency formatting is not used by all European countries
- appearance of the currency symbol varies by computer (an EBCDIC-based computer and an ASCII-based computer render characters differently).

Euros

The EUROw.d formats and informats were introduced to support the euro currency that was established by the European Monetary Union (EMU), which was formed in 1999. EUROw.d

Example:

```
options locale=English_UnitedKingdom;
data _null_;
x=12345;
put x euro10.2;
run;
```

Output:

```
 E12,345.00
```

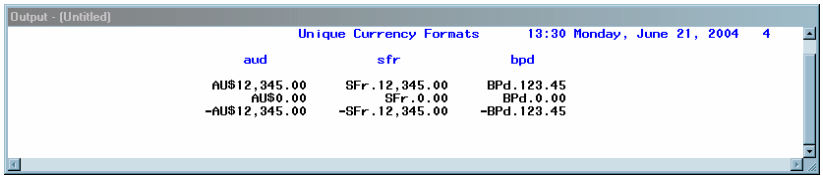
Euro format is not locale–sensitive. It uses a comma(,) as the thousands separator and a dot (.) as the decimal separator.

Customized Currency Representations

To create a customized currency representation, you can use the FORMAT procedure. The following example shows the creation of unique formats for the Australian dollar, the Swiss franc, and the British pound. For more information about the FORMAT procedure, see *SAS Viya Visual Data Management and Utility Procedures Guide*.

Example Code 8.1 *SAS Code that Customizes Currency Representations*

```
proc format;

   picture aud low-<0='0,000,000,009.00'
               (prefix='-AU$' mult=100)
               0-high='0,000,00,009.00 '
               (prefix='AU$' mult=100);

   picture sfr low-<0='0,000,000,009.00'
               (prefix='-SFr.' mult=100)
               0-high='0,000,00,009.00 '
               (prefix='-SFr.' mult=100);

   picture bpd low-<0='0,000,000,009.00'
               (prefix='-BPd.' mult=100)
               0-high='0,000,00,009.00 '
               (prefix='BPd.' mult=100);

run;
data currency;
input aud sfr bpd 12.2;
datalines;
12345 12345 12345
0 0 0
-12345 -12345 -12345
;

proc print data=currency noobs;
   var aud sfr bpd;
   format aud aud. sfr sfr. bpd bpd.;
   title 'Unique Currency Formats';
run;
```



Output - (Untitled)

Unique Currency Formats 13:30 Monday, June 21, 2004 4

aud	sfr	bpd
AU$12,345.00	SFr.12,345.00	BPd.123.45
AU$0.00	SFr.0.00	BPd.0.00
-AU$12,345.00	-SFr.12,345.00	-BPd.123.45

Customizing currency representations offers flexibility, but they require a programming solution.

Localized National and International Currency Representations

The NLMNYw.d and NLMNYIw.d formats and informats were introduced to represent localized currency in two forms:

Localized national currency representation
reflects the customs and conventions of the locale. National formats are specified using the NLMNYw.d formats and informats. You must also use the LOCALE= option to specify the locale when using the NLMNYw.d formats and informats.

Example:

```
options locale=english_UnitedStates;
data _null_;
   x=12345;
   put x nlmny15.2;
run;
```

Output:

```
$12,345.00
```

Selected national currency representations follow:

Table 8.2 *Localized National Currency Representations*

LOCALE=	Currency	National Representation
English_UnitedStates	U.S. dollars	$12,345.00
French_Canada	Canadian dollars	12 345,00 $
French_France	Euros	12 345,00 €
French_Switzerland	Swiss francs	SFr. 12'345.00
German_Germany	Euros	12.345,00 €
German_Luxembourg	Euros	12.345,00 €
Spanish_Spain	Euros	12.345,00 €
Spanish_Venezuela	Venezuelan bolivars	VEF12.345,00

The localized renderings show the native customs for representing currency. For example, although these selected EMU countries might use the same euro currency, their depiction of the currency varies. French_France uses no thousands separator but uses a comma as a decimal separator, while German_Germany and Spanish_Spain use a dot as a thousands separator and a comma as a decimal separator.

Localized international currency representation
conforms to ISO standard 4217. International forms are specified using the NLMNYIw.d formats and informats. International forms are commonly used to show a comparison of world currencies (for example, airline ticket, trade, and stock market pricing). You must also use the LOCALE= option to specify the locale when using

the NLMNYIw.d formats and informats. The letter "I," which signifies "International," is appended to the format and informat names. Here is an example:

```
options locale=english_UnitedStates;
data _null_;
 x=12345; put x nlmnyi15.2;
run;

Output:
USD12,345.00
```

Example:

Selected international currency representations follow:

Table 8.3 *International Currency Representations by Locale (ISO Standard 4217)*

LOCALE=	Currency	International Representation
English_UnitedStates	U.S. dollars	USD12,345.00
French_Canada	Canadian dollars	12 345,00 CAD
French_France	Euros	12 345,00 EUR
French_Luxembourg	Euros	12.345,00 EUR
German_Germany	Euros	12.345,00 EUR
German_Switzerland	Swiss francs	CHF 12'345.00
Spanish_Spain	Euros	12.345,00 EUR
Spanish_Venezuela	Venezuelan bolivars	VEF12.345,00

The international renderings also reflect native customs for representing currency. For example, not every locale uses a comma as the thousands separator and a dot as the decimal separator. When used, the ISO currency code placement varies by locale. The EMU countries put the currency code after the currency, while English_UnitedStates, German_Switzerland, and Spanish_Venezuela precede the currency with the ISO code.

For a complete list of the ISO standard 4217 currency codes, see http://www.xe.com/iso4217.php.

A primary limitation of using localized national and international currency representations is their dependence on a value for the LOCALE= system option.

Unique National and International Monetary Representations

The NLMNL*ISO*w.d and NLMNI*ISOw.d* formats and informats were introduced to uniquely represent each currency without having to also use the LOCALE= option. Each currency is specified by a unique ISO standard 4217 currency code.

Unique national monetary representation
is specified by the unique ISO currency code. National formats are specified using the NLMNL*ISO*w.d formats and informats. In the following example, USD is the ISO currency code for American dollars.

Note: When using the NLMNL*ISO*w.d formats and informats, the LOCALE= option is optional.

Example: **`data _null_; x=12345; put x nlmnlusd15.2; run;`**

Output:

`US$12,345.00` Selected unique national currency representations follow:

In this example, locale is specified as fr_FR.

data _null_; x=12345; put x nlmnleur15.2; run;

Output:

`€12,345.00`

Table 8.4 *Unique Currency Representations by ISO Currency Code*

ISO Currency Code	Currency	National Representation
USD	U.S. dollars	US$12,345.00
CAD	Canadian dollars	CA$12,345.00
EUR	Euros	€12,345.00
CHF	Swiss francs	SFr.12,345.00
EUR	Euros	€12,345.00
EUR	Euros	€12,345.00
EUR	Euros	€12,345.00
VEB	Venezuelan bolivars	BsF.12,345.00

A currency symbol or a currency code precedes most currencies. Also used are a comma as the thousands separator and a dot as the decimal separator. If the currency symbol of the local currency is not supported in the current SAS session encoding, the NLMNLxxxw.d format formats the value with the three-letter ISO currency code.

Unique international monetary representation
is specified by the unique ISO currency code. International formats are specified using the NLMNI*ISO*w.d formats and informats. International formats are commonly used to show a comparison of world currencies (for example, airline ticket, trade, and stock market pricing). The letter “I”, which signifies “International”, is appended to the format and informat names. In the following example, USD is the ISO currency code for American dollars.

Note: When using the NLMNI*ISO*w.d formats and informats, do not use the LOCALE= option to specify the locale.

Example: **`data _null_; x=12345; put x nlmniusd15.2; run;`** Output: **`USD12,345.00`**

Selected international currency representations follow:

Table 8.5 *International Currency Representations by ISO Currency Code*

ISO Currency Code	Currency	International Representation
USD	U.S. dollars	USD12,345.00
CAD	Canadian dollars	CAD12,345.00
EUR	Euros	EUR12,345.00
CHF	Swiss francs	CHF12,345.00
EUR	Euros	EUR12,345.00
EUR	Euros	EUR12,345.00
EUR	Euros	EUR12,345.00
VEB	Venezuelan bolivars	Not found

The international renderings precede the currency with the appropriate ISO code. Also used are a comma as the thousands separator and a dot as the decimal separator.

Example: Representing Currency in National and International Formats

This SAS program uses the exchange rates for selected Asia-Pacific countries against the U.S. dollar. In the output, each country's currency is represented using a national format and an international format.

Example Code 8.2 *SAS Code that Formats National and International Currency Formats*

```
data curr;

input ex_date mmddyy. usd aud hkd jpy sgd 12.2;

datalines;

061704 1.00000 1.45349 7.79930 110.110 1.71900  1


;

proc print data=curr noobs label;

        var ex_date usd aud hkd jpy sgd;
```

```
        format ex_date mmddyy. usd nlmnlusd15.2 aud nlmnlaud15.2 hkd
nlmnlhkd15.2
                          jpy nlmnljpy15.2 sgd nlmnlsgd15.2;
```

2

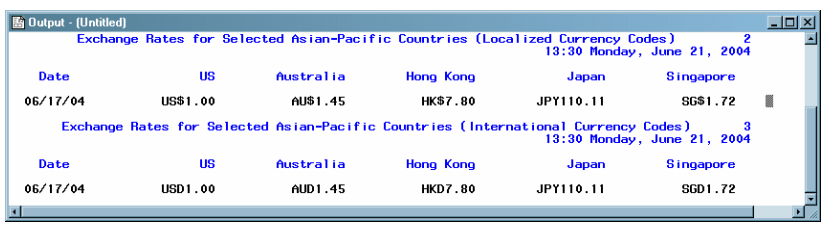
```
        label ex_date='Date' usd="US" aud='Australia' hkd='Hong Kong'

                            jpy='Japan' sgd='Singapore';

        title 'Exchange Rates for Selected Asian-Pacific Countries
                (Localized Currency Codes)';



proc print data=curr noobs label;

        var ex_date usd aud hkd jpy sgd;

        format ex_date mmddyy. usd nlmniusd15.2 aud nlmniaud15.2 hkd
nlmnihkd15.2
                          jpy nlmnijpy15.2 sgd nlmnisgd15.2;
```

3

```
        label ex_date='Date' usd="US" aud='Australia' hkd='Hong Kong'

                            jpy='Japan' sgd='Singapore';

        title 'Exchange Rates for Selected Asian-Pacific Countries
                (International Currency Codes)';

run;
```

1. These exchange rates, which were effective June 17, 2004, are specified as data in the SAS program.
2. These NLMNL*ISO* formats are applied to each of the numeric data items that are specified in the INPUT statement. These formats show currencies in the appropriate national formats.
3. These NLMNI*ISO* formats are applied to each of the numeric data items that are specified in the INPUT statement. These formats show currencies in the appropriate international formats.

Figure 8.1 *National and International Format Output*



Output - (Untitled)

Exchange Rates for Selected Asian-Pacific Countries (Localized Currency Codes) 2
13:30 Monday, June 21, 2004

Date	US	Australia	Hong Kong	Japan	Singapore
06/17/04	US$1.00	AU$1.45	HK$7.80	JPY110.11	SG$1.72

Exchange Rates for Selected Asian-Pacific Countries (International Currency Codes) 3
13:30 Monday, June 21, 2004

Date	US	Australia	Hong Kong	Japan	Singapore
06/17/04	USD1.00	AUD1.45	HKD7.80	JPY110.11	SGD1.72

Exceptions for Date and Time Default Widths

This table lists the recommended widths for UTF-8.

Encoding	Locale	Format	Default Width	Recommended Width (>=)
UTF-8	CS_CS	nltimap	10	11
	EN_GB	nldatmw	30	32
		nltimap	10	16
	EN_US	nltimap	10	11
	JA_JP	nldatmw	30	43
		nldateyq	16	20
		nldatmyq	16	20
		nldatmwz	40	49
		nltimap	10	21
	KO_KR	nldatew	20	21
		nldatmw	30	46
		nldatmwz	40	52
		nltimap	10	22
	RU_RU	nldatew	20	21
		nldatmw	30	32
	ZH_CN	nldatew	20	25
		nldatmw	30	49
		nldatmwz	40	55
		nltimap	10	21
	ZT_HK	nldatew	20	25
		nldatmw	30	49
		nldatmwz	40	55
		nltimap	10	21
	ZT_TW	nldatew	20	25

Encoding	Locale	Format	Default Width	Recommended Width (>=)
		nldatmw	30	49
		nldatmwz	40	55
		nltimap	10	21

Chapter 9

Dictionary of Formats for NLS

Categories of NLS Formats

Some formats run in SAS only, and some formats run in SAS and on the CAS engine. If CAS is specified for the format category, then the format runs in SAS and on the CAS server. If CAS is not specified for the format category, then the format runs in SAS only. For example, the NLSDATE format runs in SAS and on the CAS server, so CAS is specified as a category. The $UNCR format runs on SAS only, so CAS is not specified as a category.

The following categories relate to NLS issues:

Category	Description
BIDI Text Handling	Instructs SAS to write bidirectional data values from data variables.
CAS	Instructs SAS that these formats run on the CAS server.
Character	Instructs SAS to write character data values from character variables.
Currency Conversion	Instructs SAS to convert an amount from one currency to another currency.

Category	Description
DBCS	Instructs SAS to translate double-byte character sets that are used in Asian languages.
Hebrew Text Handling	Instructs SAS to read Hebrew data from data variables.
International Date and Time	Instructs SAS to write data values from variables that represent dates, times, and datetimes.
Numeric	Instructs SAS to write numeric data values from numeric variables.

Category	Language Elements	Description
CAS	EURO Format (p. 96)	Writes numeric values with a leading euro symbol (E), a comma that separates every three digits, and a period that separates the decimal fraction.
	EUROX Format (p. 99)	Writes numeric values with a leading euro symbol (E), a period that separates every three digits, and a comma that separates the decimal fraction.
	NENGO Format (p. 102)	Writes date values as Japanese dates in the form e.yymmdd.
	NLBEST Format (p. 104)	Writes the best numerical notation based on the locale.
	NLDATE Format (p. 106)	Converts a SAS date value to the date value of the specified locale, and then writes the date value as a date.
	NLDATEL Format (p. 107)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form, date, month, year.
	NLDATEM Format (p. 108)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date.
	NLDATEMD Format (p. 109)	Converts the SAS date value to the date value of the specified locale, and then writes the value as the name of the month and the day of the month.
	NLDATEMDL Format (p. 110)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.
	NLDATEMDM Format (p. 111)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.

Category	Language Elements	Description
	NLDATEMDS Format (p. 111)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.
	NLDATEMN Format (p. 112)	Converts a SAS date value to the date value of the specified locale, and then writes the value as the name of the month.
	NLDATES Format (p. 113)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date string.
	NLDATEW Format (p. 114)	Converts a SAS date value to the date value of the specified locale, and then writes the value as the date and the day of the week.
	NLDATEWN Format (p. 116)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the day of the week.
	NLDATEYM Format (p. 117)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the name of the month.
	NLDATEYML Format (p. 118)	Converts a SAS date value to the date string of the specified locale, and then writes the month and year.
	NLDATEYMM Format (p. 119)	Converts a SAS date value to the date string of the specified locale, and then writes the date values as the month and year with abbreviations.
	NLDATEYMS Format (p. 120)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and year.
	NLDATEYQ Format (p. 121)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the quarter.
	NLDATEYQL Format (p. 122)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (Q1–Q4) using abbreviations.
	NLDATEYQM Format (p. 123)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (Q1–Q4) using abbreviations.
	NLDATEYQS Format (p. 124)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (1–4) with numbers and delimiters.
	NLDATEYR Format (p. 125)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year.
	NLDATEYW Format (p. 126)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the week.
	NLDATM Format (p. 127)	Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime.

Category	Language Elements	Description
	NLDATMAP Format (p. 128)	Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime with a.m. or p.m.
	NLDATMDT Format (p. 129)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month, day of the month and year.
	NLDATML Format (p. 130)	Converts a SAS datetime value to the date string of the specified locale, and then writes the date value as a date in the form, month, date, year, and time.
	NLDATMM Format (p. 131)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and time with abbreviations for the month and time.
	NLDATMMD Format (p. 131)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month and the day of the month.
	NLDATMMDL Format (p. 132)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the full-length of the month and day of the month.
	NLDATMMDM Format (p. 133)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using abbreviations.
	NLDATMMDS Format (p. 134)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using numbers and delimiters.
	NLDATMMN Format (p. 135)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month.
	NLDATMS Format (p. 136)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form, MM/DD/YYYY .
	NLDATMTM Format (p. 137)	Converts the time portion of a SAS datetime value to the time-of-day value of the specified locale, and then writes the value as a time of day.
	NLDATMTZ Format (p. 138)	Converts the time portion of the SAS datetime of the locale to the time of day and time zone.
	NLDATMW Format (p. 139)	Converts SAS datetime values to the locale sensitive datetime string as the day of the week and the datetime.
	NLDATMWN Format (p. 140)	Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as the day of the week.
	NLDATMWZ Format (p. 141)	Converts SAS date values of the specified locale to a day-of-week, datetime, and time zone value.

Category	Language Elements	Description
	NLDATMYM Format (p. 142)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the name of the month.
	NLDATMYML Format (p. 143)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.
	NLDATMYMM Format (p. 144)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.
	NLDATMYMS Format (p. 144)	Converts a SAS date value to the date string of the specified locale, and then writes the month and year with numbers and delimiters.
	NLDATMYQ Format (p. 145)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the quarter of the year.
	NLDATMYQL Format (p. 146)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year's quarter value (1–4) and the year.
	NLDATMYQM Format (p. 147)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year's quarter (1–4) and then the year.
	NLDATMYQS Format (p. 148)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the quarter (1-4) using numbers and delimiters.
	NLDATMYR Format (p. 149)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year.
	NLDATMYW Format (p. 150)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the name of the week.
	NLDATMZ Format (p. 151)	Converts SAS datetime values to the locale-sensitive datetime string as time zone and datetime.
	NLMNY Format (p. 226)	Writes the monetary format of the local expression in the specified locale using local currency.
	NLMNYI Format (p. 227)	Writes the monetary format of the international expression in the specified locale.
	NLNUM Format (p. 228)	Writes the numeric format of the local expression in the specified locale.
	NLNUMI Format (p. 230)	Writes the numeric format of the international expression in the specified locale.
	NLPCT Format (p. 231)	Writes percentage data of the local expression in the specified locale.

Category	Language Elements	Description
	NLPCTI Format (p. 232)	Writes percentage data of the international expression in the specified locale.
	NLPCTN Format (p. 234)	Produces percentages, using a minus sign for negative values.
	NLPCTP Format (p. 235)	Writes locale-specific numeric values as percentages.
	NLPVALUE Format (p. 236)	Writes p-values of the local expression in the specified locale.
	NLSTRMON Format (p. 237)	Writes the month name in the specified locale.
	NLSTRQTR Format (p. 238)	Writes a numeric value as the quarter-of-the-year in the specified locale.
	NLSTRWK Format (p. 239)	Writes a numeric value as the day-of-the-week in the specified locale.
	NLTIMAP Format (p. 241)	Converts a SAS time value to the time value of a specified locale, and then writes the value as a time value with a.m. or p.m. NLTIMAP also converts SAS date-time values.
	NLTIME Format (p. 242)	Converts a SAS time value to the time value of the specified locale, and then writes the value as a time value. NLTIME also converts SAS date-time values.
	YEN Format (p. 271)	Writes numeric values with yen signs, commas, and decimal points.
	YYWEEKU Format (p. 272)	Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.
	YYWEEKV Format (p. 273)	Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.
	YYWEEKW Format (p. 275)	Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.
Character	$EBCDIC Format (p. 90)	Converts native format character data to EBCDIC representation.
	$UCS2B Format (p. 243)	Processes a character string that is in the encoding of the current SAS session, and then writes the character string in big-endian, 16-bit, UCS2, Unicode encoding.
	$UCS2BE Format (p. 244)	Processes a character string that is in big-endian, 16-bit, UCS2, Unicode encoding, and then writes the character string in the encoding of the current SAS session.
	$UCS2L Format (p. 245)	Processes a character string that is in the encoding of the current SAS session, and then writes the character string in little-endian, 16-bit, UCS2, Unicode encoding.
	$UCS2LE Format (p. 247)	Processes a character string that is in little-endian, 16-bit, UCS2, Unicode encoding, and then writes the character string in the encoding of the current SAS session.

Category	Language Elements	Description
	$UCS2X Format (p. 248)	Processes a character string that is in the encoding of the current SAS session, and then writes the character string in native-endian, 16-bit, UCS2, Unicode encoding.
	$UCS2XE Format (p. 249)	Processes a character string that is in native-endian, 16-bit, UCS2, Unicode encoding, and then writes the character string in the encoding of the current SAS session.
	$UCS4B Format (p. 250)	Processes a character string that is in the encoding of the current SAS session, and then writes the character string in big-endian, 32-bit, UCS4, Unicode encoding.
	$UCS4BE Format (p. 251)	Processes a character string that is in big-endian, 32-bit, UCS4, Unicode encoding, and then writes the character string in the encoding of the current SAS session.
	$UCS4L Format (p. 252)	Processes a character string that is in the encoding of the current SAS session, and then writes the character string in little-endian, 32-bit, UCS4, Unicode encoding.
	$UCS4LE Format (p. 254)	Processes a character string that is in little-endian, 32-bit, UCS4, Unicode encoding, and then writes the character string in the encoding of the current SAS session.
	$UCS4X Format (p. 255)	Processes a character string that is in the encoding of the current SAS session, and then writes the character string in native-endian, 32-bit, UCS4, Unicode encoding.
	$UCS4XE Format (p. 256)	Processes a character string that is in native-endian, 32-bit, UCS4, Unicode encoding, and then writes the character string in the encoding of the current SAS session.
	$UESC Format (p. 257)	Processes a character string that is encoded in the current SAS session, and then writes the character string in Unicode escape (UESC) representation.
	$UESCE Format (p. 259)	Processes a character string that is in Unicode escape (UESC) representation, and then writes the character string in the encoding of the current SAS session.
	$UNCR Format (p. 260)	Processes a character string that is encoded in the current SAS session, and then writes the character string in numeric character representation (NCR).
	$UNCRE Format (p. 261)	Processes a character string that is in numeric character representation (NCR), and then writes the character string in the encoding of the current SAS session.
	$UPAREN Format (p. 262)	Processes a character string that is encoded in the current SAS session, and then writes the character string in Unicode parenthesis (UPAREN) representation.
	$UPARENE Format (p. 263)	Processes a character string that is in Unicode parenthesis (UPAREN), and then writes the character string in the encoding of the current SAS session.

Category	Language Elements	Description
	$UTF8X Format (p. 264)	Processes a character string that is in the encoding of the current SAS session, and then writes the character string in universal transformation format (UTF-8) encoding.
	$UTF8XE Format (p. 265)	Processes a character string that is in universal transformation format (UTF-8), and then writes the character string in the encoding of the current SAS session.
Date and Time	B8601DX Format (p. 82)	Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+hhmm.
	B8601LX Format (p. 84)	Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation yyyymmddThhmmss+|–hhmm.
	B8601TX Format (p. 85)	Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss+|-hhmm.
	E8601DX Format (p. 91)	Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+hh:mm.
	E8601LX Format (p. 93)	Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss+|–hh:mm.
	E8601TX Format (p. 94)	Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 extended time notation hh:mm:ss+|–hh:mm.
	MINGUO Format (p. 101)	Writes date values as Taiwanese dates in the form yyyymmdd.
	NENGO Format (p. 102)	Writes date values as Japanese dates in the form e.yymmdd.
	NLDATE Format (p. 106)	Converts a SAS date value to the date value of the specified locale, and then writes the date value as a date.
	NLDATEL Format (p. 107)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form, date, month, year.
	NLDATEM Format (p. 108)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date.
	NLDATEMD Format (p. 109)	Converts the SAS date value to the date value of the specified locale, and then writes the value as the name of the month and the day of the month.
	NLDATEMDL Format (p. 110)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.

Category	Language Elements	Description
	NLDATEMDM Format (p. 111)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.
	NLDATEMDS Format (p. 111)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.
	NLDATEMN Format (p. 112)	Converts a SAS date value to the date value of the specified locale, and then writes the value as the name of the month.
	NLDATES Format (p. 113)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date string.
	NLDATEW Format (p. 114)	Converts a SAS date value to the date value of the specified locale, and then writes the value as the date and the day of the week.
	NLDATEWN Format (p. 116)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the day of the week.
	NLDATEYM Format (p. 117)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the name of the month.
	NLDATEYML Format (p. 118)	Converts a SAS date value to the date string of the specified locale, and then writes the month and year.
	NLDATEYMM Format (p. 119)	Converts a SAS date value to the date string of the specified locale, and then writes the date values as the month and year with abbreviations.
	NLDATEYMS Format (p. 120)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and year.
	NLDATEYQ Format (p. 121)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the quarter.
	NLDATEYQL Format (p. 122)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (Q1–Q4) using abbreviations.
	NLDATEYQM Format (p. 123)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (Q1–Q4) using abbreviations.
	NLDATEYQS Format (p. 124)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (1–4) with numbers and delimiters.
	NLDATEYR Format (p. 125)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year.

Category	Language Elements	Description
	NLDATEYW Format (p. 126)	Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the week.
	NLDATM Format (p. 127)	Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime.
	NLDATMAP Format (p. 128)	Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime with a.m. or p.m.
	NLDATMDT Format (p. 129)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month, day of the month and year.
	NLDATML Format (p. 130)	Converts a SAS datetime value to the date string of the specified locale, and then writes the date value as a date in the form, month, date, year, and time.
	NLDATMM Format (p. 131)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and time with abbreviations for the month and time.
	NLDATMMD Format (p. 131)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month and the day of the month.
	NLDATMMDL Format (p. 132)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the full-length of the month and day of the month.
	NLDATMMDM Format (p. 133)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using abbreviations.
	NLDATMMDS Format (p. 134)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using numbers and delimiters.
	NLDATMMN Format (p. 135)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month.
	NLDATMS Format (p. 136)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form, MM/DD/YYYY .
	NLDATMTM Format (p. 137)	Converts the time portion of a SAS datetime value to the time-of-day value of the specified locale, and then writes the value as a time of day.
	NLDATMTZ Format (p. 138)	Converts the time portion of the SAS datetime of the locale to the time of day and time zone.
	NLDATMW Format (p. 139)	Converts SAS datetime values to the locale sensitive datetime string as the day of the week and the datetime.

Category	Language Elements	Description
	NLDATMWN Format (p. 140)	Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as the day of the week.
	NLDATMWZ Format (p. 141)	Converts SAS date values of the specified locale to a day-of-week, datetime, and time zone value.
	NLDATMYM Format (p. 142)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the name of the month.
	NLDATMYML Format (p. 143)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.
	NLDATMYMM Format (p. 144)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.
	NLDATMYMS Format (p. 144)	Converts a SAS date value to the date string of the specified locale, and then writes the month and year with numbers and delimiters.
	NLDATMYQ Format (p. 145)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the quarter of the year.
	NLDATMYQL Format (p. 146)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year's quarter value (1–4) and the year.
	NLDATMYQM Format (p. 147)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year's quarter (1–4) and then the year.
	NLDATMYQS Format (p. 148)	Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the quarter (1-4) using numbers and delimiters.
	NLDATMYR Format (p. 149)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year.
	NLDATMYW Format (p. 150)	Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the name of the week.
	NLDATMZ Format (p. 151)	Converts SAS datetime values to the locale-sensitive datetime string as time zone and datetime.
	NLTIMAP Format (p. 241)	Converts a SAS time value to the time value of a specified locale, and then writes the value as a time value with a.m. or p.m. NLTIMAP also converts SAS date-time values.
	NLTIME Format (p. 242)	Converts a SAS time value to the time value of the specified locale, and then writes the value as a time value. NLTIME also converts SAS date-time values.

Category	Language Elements	Description
	WEEKU Format (p. 266)	Writes a week number in decimal format by using the U algorithm.
	WEEKV Format (p. 267)	Writes a week number in decimal format by using the V algorithm.
	WEEKW Format (p. 269)	Writes a week number in decimal format by using the W algorithm.
	YYWEEKU Format (p. 272)	Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.
	YYWEEKV Format (p. 273)	Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.
	YYWEEKW Format (p. 275)	Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.
Hebrew Text Handling	$CPTDW Format (p. 88)	Processes a character string that is in Hebrew text, encoded in IBM-PC (cp862), and then writes the character string in Windows Hebrew encoding (cp 1255).
	$CPTWD Format (p. 89)	Processes a character string that is encoded in Windows (cp1255), and then writes the character string in Hebrew DOS (cp862) encoding.
ISO 8601	B8601DX Format (p. 82)	Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+hhmm.
	B8601LX Format (p. 84)	Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation yyyymmddThhmmss+|–hhmm.
	B8601TX Format (p. 85)	Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss+|-hhmm.
	E8601DX Format (p. 91)	Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+hh:mm.
	E8601LX Format (p. 93)	Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss+|–hh:mm.
	E8601TX Format (p. 94)	Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 extended time notation hh:mm:ss+|–hh:mm.
Numeric	BESTDOTX Format (p. 87)	Specifies that SAS choose the best notation and use a dot as a decimal separator.

Category	Language Elements	Description
	EURO Format (p. 96)	Writes numeric values with a leading euro symbol (E), a comma that separates every three digits, and a period that separates the decimal fraction.
	EUROX Format (p. 99)	Writes numeric values with a leading euro symbol (E), a period that separates every three digits, and a comma that separates the decimal fraction.
	NLBEST Format (p. 104)	Writes the best numerical notation based on the locale.
	NLMNIAED Format (p. 152)	Writes the monetary format of the international expression for the United Arab Emirates.
	NLMNIAUD Format (p. 153)	Writes the monetary format of the international expression for Australia.
	NLMNIBGN Format (p. 154)	Writes the monetary format of the international expression for Bulgaria.
	NLMNIBRL Format (p. 155)	Writes the monetary format of the international expression for Brazil.
	NLMNICAD Format (p. 156)	Writes the monetary format of the international expression for Canada.
	NLMNICHF Format (p. 157)	Writes the monetary format of the international expression for Liechtenstein and Switzerland.
	NLMNICNY Format (p. 158)	Writes the monetary format of the international expression for China.
	NLMNICZK Format (p. 159)	Writes the monetary format of the international expression for the Czech Republic.
	NLMNIDKK Format (p. 160)	Writes the monetary format of the international expression for Denmark, Faroe Island, and Greenland.
	NLMNIEEK Format (p. 161)	Writes the monetary format of the international expression for Estonia.
	NLMNIEGP Format (p. 162)	Writes the monetary format of the international expression for Egypt.
	NLMNIEUR Format (p. 163)	Writes the monetary format of the international expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNIGBP Format (p. 164)	Writes the monetary format of the international expression for the United Kingdom.
	NLMNIHKD Format (p. 165)	Writes the monetary format of the international expression for Hong Kong.

Category	Language Elements	Description
	NLMNIHRK Format (p. 166)	Writes the monetary format of the international expression for Croatia.
	NLMNIHUF Format (p. 167)	Writes the monetary format of the international expression for Hungary.
	NLMNIIDR Format (p. 168)	Writes the monetary format of the international expression for Indonesia.
	NLMNIILS Format (p. 169)	Writes the monetary format of the international expression for Israel.
	NLMNIINR Format (p. 170)	Writes the monetary format of the international expression for India.
	NLMNIJPY Format (p. 171)	Writes the monetary format of the international expression for Japan.
	NLMNIKRW Format (p. 172)	Writes the monetary format of the international expression for South Korea.
	NLMNILTL Format (p. 173)	Writes the monetary format of the international expression for Lithuania.
	NLMNILVL Format (p. 174)	Writes the monetary format of the international expression for Latvia.
	NLMNIMOP Format (p. 175)	Writes the monetary format of the international expression for Macau.
	NLMNIMXN Format (p. 176)	Writes the monetary format of the international expression for Mexico.
	NLMNIMYR Format (p. 177)	Writes the monetary format of the international expression for Malaysia.
	NLMNINOK Format (p. 178)	Writes the monetary format of the international expression for Norway.
	NLMNINZD Format (p. 179)	Writes the monetary format of the international expression for New Zealand.
	NLMNIPLN Format (p. 180)	Writes the monetary format of the international expression for Poland.
	NLMNIRUB Format (p. 181)	Writes the monetary format of the international expression for Russia.
	NLMNISEK Format (p. 182)	Writes the monetary format of the international expression for Sweden.
	NLMNISGD Format (p. 183)	Writes the monetary format of the international expression for Singapore.

Category	Language Elements	Description
	NLMNITHB Format (p. 184)	Writes the monetary format of the international expression for Thailand.
	NLMNITRY Format (p. 185)	Writes the monetary format of the international expression for Turkey.
	NLMNITWD Format (p. 186)	Writes the monetary format of the international expression for Taiwan.
	NLMNIUSD Format (p. 187)	Writes the monetary format of the international expression for Puerto Rico and the United States.
	NLMNIZAR Format (p. 188)	Writes the monetary format of the international expression for South Africa.
	NLMNLAED Format (p. 189)	Writes the monetary format of the local expression for the United Arab Emirates.
	NLMNLAUD Format (p. 190)	Writes the monetary format of the local expression for Australia.
	NLMNLBGN Format (p. 191)	Writes the monetary format of the local expression for Bulgaria.
	NLMNLBRL Format (p. 192)	Writes the monetary format of the local expression for Brazil.
	NLMNLCAD Format (p. 193)	Writes the monetary format of the local expression for Canada.
	NLMNLCHF Format (p. 194)	Writes the monetary format of the local expression for Liechtenstein and Switzerland.
	NLMNLCNY Format (p. 195)	Writes the monetary format of the local expression for China.
	NLMNLCZK Format (p. 196)	Writes the monetary format of the local expression for the Czech Republic.
	NLMNLDKK Format (p. 197)	Writes the monetary format of the local expression for Denmark, Faroe Island, and Greenland.
	NLMNLEEK Format (p. 198)	Writes the monetary format of the local expression for Estonia.
	NLMNLEGP Format (p. 199)	Writes the monetary format of the local expression for Egypt.
	NLMNLEUR Format (p. 200)	Writes the monetary format of the local expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNLGBP Format (p. 201)	Writes the monetary format of the local expression for the United Kingdom.
	NLMNLHKD Format (p. 202)	Writes the monetary format of the local expression for Hong Kong.
	NLMNLHRK Format (p. 203)	Writes the monetary format of the local expression for Croatia.

Category	Language Elements	Description
	NLMNLHUF Format (p. 204)	Writes the monetary format of the local expression for Hungary.
	NLMNLIDR Format (p. 205)	Writes the monetary format of the local expression for Indonesia.
	NLMNLILS Format (p. 206)	Writes the monetary format of the local expression for Israel.
	NLMNLINR Format (p. 207)	Writes the monetary format of the local expression for India.
	NLMNLJPY Format (p. 208)	Writes the monetary format of the international expression for Japan.
	NLMNLKRW Format (p. 209)	Writes the monetary format of the local expression for South Korea.
	NLMNLLTL Format (p. 210)	Writes the monetary format of the local expression for Lithuania.
	NLMNLLVL Format (p. 211)	Writes the monetary format of the local expression for Latvia.
	NLMNLMOP Format (p. 212)	Writes the monetary format of the local expression for Macau.
	NLMNLMXN Format (p. 213)	Writes the monetary format of the local expression for Mexico.
	NLMNLMYR Format (p. 214)	Writes the monetary format of the local expression for Malaysia.
	NLMNLNOK Format (p. 215)	Writes the monetary format of the local expression for Norway.
	NLMNLNZD Format (p. 216)	Writes the monetary format of the local expression for New Zealand.
	NLMNLPLN Format (p. 217)	Writes the monetary format of the local expression for Poland.
	NLMNLRUB Format (p. 218)	Writes the monetary format of the local expression for Russia.
	NLMNLSEK Format (p. 219)	Writes the monetary format of the local expression for Sweden.
	NLMNLSGD Format (p. 220)	Writes the monetary format of the local expression for Singapore.
	NLMNLTHB Format (p. 221)	Writes the monetary format of the local expression for Thailand.
	NLMNLTRY Format (p. 222)	Writes the monetary format of the local expression for Turkey.
	NLMNLTWD Format (p. 223)	Writes the monetary format of the local expression for Taiwan.
	NLMNLUSD Format (p. 224)	Writes the monetary format of the local expression for Puerto Rico and the United States.
	NLMNLZAR Format (p. 225)	Writes the monetary format of the local expression for South Africa.
	NLMNY Format (p. 226)	Writes the monetary format of the local expression in the specified locale using local currency.

Category	Language Elements	Description
	NLMNYI Format (p. 227)	Writes the monetary format of the international expression in the specified locale.
	NLNUM Format (p. 228)	Writes the numeric format of the local expression in the specified locale.
	NLNUMI Format (p. 230)	Writes the numeric format of the international expression in the specified locale.
	NLPCT Format (p. 231)	Writes percentage data of the local expression in the specified locale.
	NLPCTI Format (p. 232)	Writes percentage data of the international expression in the specified locale.
	NLPCTN Format (p. 234)	Produces percentages, using a minus sign for negative values.
	NLPCTP Format (p. 235)	Writes locale-specific numeric values as percentages.
	NLPVALUE Format (p. 236)	Writes p-values of the local expression in the specified locale.
	NLSTRMON Format (p. 237)	Writes the month name in the specified locale.
	NLSTRQTR Format (p. 238)	Writes a numeric value as the quarter-of-the-year in the specified locale.
	NLSTRWK Format (p. 239)	Writes a numeric value as the day-of-the-week in the specified locale.
	YEN Format (p. 271)	Writes numeric values with yen signs, commas, and decimal points.

Dictionary

B8601DX Format

Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation *yyyymmdd*T*hhmmss +hhmm*.

Categories: Date and Time
ISO 8601

Alignment: Left

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

B8601DX*w*.

Syntax Description

w
specifies the width of the output field.

Default	26
Range	20–35

Details

UTC values specify a date and a time that are based on the zero meridian in Greenwich, England. Using this format, SAS converts a datetime value to the UTC value and determines the user local date and time by using the value of the TIMEZONE= system option. If the TIMEZONE= option is not set, then the user local date and time are based on the local date and time. The B8601DX format writes SAS datetime values by using the following ISO 8601 basic datetime notation:

- *yyyymmdd*T*hhmmss*+*hhmm*

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

hh
is a two-digit hour (zero padded) between 00 and 23.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
is a two-digit second (zero padded) between 00 and 59.

+|–*hhmm*
is an hour and minute signed offset from zero meridian time. The offset must be +|–*hhmm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Example

The first example uses the local time to determine the time and the time zone offset. The second example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

Statement	Result

`data _null_ ;`
`t='01Feb2013T12:34:56'dt ;`
`put t b8601dx.;`
`run;`
	`20130201T073456-0500`

`options timezone='America/Adak';`
`data _null_ ;`
`t='01Feb2013T12:34:56'dt ;`
`put t b8601dx.;`
`run;`
	`20130201T023456-1000`

See Also

"Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations" in *SAS Viya Formats and Informats: Reference*

B8601LX Format

Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation *yyyymmdd*T*hhmmss*+|–*hhmm*.

Categories: Date and Time
ISO 8601

Alignment: Right

Supports: ISO 8601 Elements 5.3.3 and 5.3.4.2

Syntax

B8601LX*w*

Syntax Description

w
specifies the width of the output field.

Default 26

Range 20–35

Details

The B8601LX format writes datetime values without making any adjustments, and appends the UTC time zone offset for the local SAS session by using the ISO 8601 basic datetime notation:

- *yyyymmdd*T*hhmmss*+|–*hhmm*

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

hh
is a two-digit hour (zero padded) between 00 and 23.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
is a two-digit second (zero padded) between 00 and 59.

+|–*hhmm*
is an hour and minute signed offset from zero meridian time. The offset must be +|–*hhmm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Example

This PUT statement writes the time for the Eastern Standard time zone:

```
blx='01Feb2013T12:34:56'dt;
put blx b8601lx.;
```

Value of blx	Result
1675341296	20130201T123456-0500

See Also

"Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations" in *SAS Viya Formats and Informats: Reference*

B8601TX Format

Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 basic time notation *hhmmss*+|-*hhmm*.

Categories: Date and Time
ISO 8601

Alignment: Right

Supports: ISO 8601 Elements 5.3.3 and 5.3.4

Syntax

B8601TX*w*.

Syntax Description

w
specifies the width of the output field.

Default 14

Range 9–20

Details

UTC values specify a time based on the zero meridian in Greenwich, England. Using this format, SAS converts a time value to the UTC value and determines the user local time by using the TIMEZONE= system option. If the TIMEZONE= option is not set, then the user local time is based on the local time. The B8601TX format writes SAS datetime values by using the following ISO 8601 basic time notation:

- *hhmmss*+|–*hhmm*

hh
is a two-digit hour (zero padded) between 00 and 23.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
is a two-digit second (zero padded) between 00 and 59.

+|–*hhmm*
is an hour and minute signed offset from zero meridian time. The offset must be +|–*hhmm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

When SAS reads a UTC time by using the B8601TZ informat, and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 000000 and 240000. If the B8601TX format attempts to format a time outside of this time range, the time is formatted with asterisks to indicate that the value is out of range.

Example

The first example uses the local time to determine the time and the time zone offset. The second example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

Statement	Result

`data _null_;`
`t='12:34:56't;`
`put t b8601tx.;`
`run;`
	`073456-0500`

`options timezone='America/Adak';`
`data _null_ ;`
`t='12:34:56't;`
`put t b8601tx.;`
`run;`
	`023456-1000`

See Also

"Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations" in *SAS Viya Formats and Informats: Reference*

BESTDOTX Format

Specifies that SAS choose the best notation and use a dot as a decimal separator.

Category: Numeric

Alignment: Right

Syntax

BESTDOTX*w*.

Syntax Description

w

specifies the width of the output field.

Default 12

Range 1–32

Tip If you print numbers between 0 and .01 exclusively, use a field width of at least 7 to avoid excessive rounding. If you print numbers between 0 and -01 exclusively, use a field width of at least 8.

Details

If the NLDECSEPARATOR system option is disabled, the BEST*w* and BESTDOTX*w* formats process data the same way. If the NLDECSEPARATOR system option is enabled, then the results from the BEST and BESTDOTX formats are different. See the following table to understand the differences:

LOCALE option	Default decimal separator character for the locale	NLDECSEPARATOR option	Separator character used by BEST*w.*	Separator character used by BESTDOTX*w.*
en_US	Dot	Disabled (default)	Dot	Dot
		Enabled	Dot	Dot
fr_FR	Comma	Disabled (default)	Dot	Dot
		Enabled	Comma	Dot

Example

The following code produces results using the BESTDOTX format with the NLSDECSEPARATOR option enabled and the locale set to fr_FR.

Statements	Results

`options nld locale=fr_FR;`
`data _null_;`
`  x=1.2;`
`  call symput('macx', put(x, BESTDOTX.));`
`  put x; /* Result is printed as "1,2" */`
 	
`y=put(x, BESTDOTX.); /*Result is printed as "1.2 */`
`run;`
	`1,2`
	`1.2`

`&put &macx;`	`1.2`

`data _null_;`
`bx=&macx; /* Succeeds because the value of &macx is 1.2 */`
`put bx;`
`run;`
	`1,2`

See Also

- "BEST Format" in *SAS Viya Formats and Informats: Reference*
- "NLDECSEPARATOR System Option" on page 584

$CPTDW Format

Processes a character string that is in Hebrew text, encoded in IBM-PC (cp862), and then writes the character string in Windows Hebrew encoding (cp 1255).

Category: Hebrew Text Handling

Alignment: Left

Syntax

$CPTDW*w*.

Syntax Description

w

specifies the width of the output field.

Default 200

Range 1–32767

Comparisons

The $CPTDW*w*. format performs processing that is the opposite of the $CPTWD*w*. format.

Example

The following example uses the input value of “808182x.”

Statement	Result
	`----+----1----+`
`put text $cptdw3.;`	אבג

See Also

Format:

- “$CPTWD Format” on page 89

Informats:

- “$CPTDW Informat” on page 442
- “$CPTWD Informat” on page 443

$CPTWD Format

Processes a character string that is encoded in Windows (cp1255), and then writes the character string in Hebrew DOS (cp862) encoding.

Category: Hebrew Text Handling

Alignment: Left

Syntax

$CPTWD*w*.

Syntax Description

w

specifies the width of the output field.

Default 200

Range 1–32767

Comparisons

The $CPTWD*w*. format performs processing that is the opposite of the $CPTDW*w*. format.

Example

The following example uses the input value of "ㄱㄱㄴ".

Statement	Result
	`----+----1----+----2----+`
`put text $cptwd3.;`	€□,

See Also

Format:

- "$CPTDW Format" on page 88

Informats:

- "$CPTDW Informat" on page 442
- "$CPTWD Informat" on page 443

$EBCDIC Format

Converts native format character data to EBCDIC representation.

Category: Character

Alignment: Left

Note: UTF-8 is the only supported session encoding. $EBCDIC format works only for 7bit ASCII characters.

Syntax

$EBCDIC*w.*

Syntax Description

w

specifies the width of the output field.

Default 1

Range 1–32767

Details

If EBCDIC is the native format, no conversion occurs.

On ASCII systems, the $EBCDIC*w*. format is based on the default encoding value of the LOCALE= option that is specified when SAS starts. For example, if the locale was set to en_US locale, the default encoding that is used by the $EBCDIC*w*. format is Open_ed-1047. If the locale is de_DE (German_Germany), the default encoding that is used by the $EBCDIC*w*. format is Open_ed–1141. For a list of locales and encoding values, see Table 17.2 on page 636.

You can specify the translation table that is used to map characters between EBCDIC and ASCII by using the MAPEBCDIC2ASCII system option. For more information, see "MAPEBCDIC2ASCII= System Option" on page 583.

Comparisons

- On ASCII systems, $EBCDIC*w*. converts ASCII character data to EBCDIC.
- On all other systems, $EBCDIC*w*. behaves like the $CHAR*w*. format.

Example

```
put name $ebcdic3.;
```

Value of `name`	Locale Value Is en_US	Locale Value Is fr_FR	Locale Value Is de_DE
ABC	C1C2C3	C1C2C3	C1C2C3
{ä}	C043D0	514354	43C0DC
[@]	AD7CBD	9044B5	63B5FC

The results are shown as hexadecimal representations of EBCDIC codes for characters. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one character.

E8601DX Format

Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone extended notation *yyyy-mm-dd*T*hh:mm:ss+hh:mm*.

Categories: Date and Time
ISO 8601

Alignment: Left

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

E8601DX*w*.

Syntax Description

w

specifies the width of the output field.

Default 26

Range 20–35

Details

UTC values specify a date and time that are based on the zero meridian in Greenwich, England. Using this format, SAS converts a datetime value to the UTC value and determines the user local date and time by using the value of the TIMEZONE= system option. If the TIMEZONE= option is not set, the user local date and time are based on the local date and time. The E8601DX format writes SAS datetime values by using the following ISO 8601 basic datetime notation:

- *yyyy-mm-dd*T*hh:mm:ss+hh:mm*

yyyy

is a four-digit year.

mm

is a two-digit month (zero padded) between 01 and 12.

dd

is a two-digit day of the month (zero padded) between 01 and 31.

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

+|–*hh:mm*

is an hour and minute signed offset from zero meridian time. The offset must be +|–*hh:mm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +02:00 indicates a two-hour time difference to the east of the zero meridian, and –06:00 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Example

The first example uses the local time to determine the time. The second example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

Statement	Result

`data _null_ ;`
`t='01Feb2013T12:34:56'dt ;`
`put t e8601dx.;`
`run;`
	`2013-02-01T07:34:56-05:00`

`options timezone='America/Adak';`
`data _null_ ;`
`t='01Feb2013T12:34:56'dt ;`
`put t e8601dx.;`
`run;`
	`2013-02-01T02:34:56-10:00`

See Also

"Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations" in *SAS Viya Formats and Informats: Reference*

E8601LX Format

Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation *yyyy-mm-dd*T*hh:mm:ss*+|–*hh:mm*.

Categories: Date and Time
ISO 8601

Alignment: Right

Supports: ISO 8601 Elements 5.3.3 and 5.3.4.2

Syntax

E8601LX*w*

Syntax Description

w
specifies the width of the output field.

Default 26

Range 20–35

Details

The E8601LX format writes datetime values without making any adjustments, and appends the UTC time zone offset for the local SAS session by using the ISO 8601 basic datetime notation:

- *yyyy-mm-dd*T*hh:mm:ss+hh:mm*

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

hh
is a two-digit hour (zero padded) between 00 and 23.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
is a two-digit second (zero padded) between 00 and 59.

+|–*hh:mm*
is an hour and minute signed offset from zero meridian time. The offset must be +|–*hh:mm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +02:00 indicates a two-hour time difference to the east of the zero meridian, and –06:00 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Example

This PUT statement writes the time for the Eastern time zone:

```
blx='01Feb2013T12:34:56'dt;
put blx e8601lx.;
```

Value of blx	Result
1675341296	2013-02-01T12:34:56-05:00

See Also

"Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations" in *SAS Viya Formats and Informats: Reference*

E8601TX Format

Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 extended time notation *hh:mm:ss*+|–*hh:mm*.

Categories: Date and Time
ISO 8601

Alignment: Right

Supports: ISO 8601 Elements 5.3.3 and 5.3.4

Syntax

E8601TX*w*.

Syntax Description

w
specifies the width of the output field.

Default 14

Range 9–20

Details

UTC values specify a time based on the zero meridian in Greenwich, England. Using this format, SAS converts a time value to the UTC value and determines the user local time by using the value of the TIMEZONE= system option. If the TIMEZONE= option is not set, the user local time is based on the local time. The E8601TX format writes SAS datetime values by using the following ISO 8601 basic time notation:

- *hh:mm:ss*+|–*hh:mm*

hh
is a two-digit hour (zero padded) between 00 and 23.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
is a two-digit second (zero padded) between 00 and 59.

+|–*hh:mm*
is an hour and minute signed offset from zero meridian time. The offset must be +|–*hhmm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|-*hh* is not supported.

When SAS reads a UTC time by using the E8601TZ informat, and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 000000 and 240000. If the E8601TX format attempts to format a time outside of this time range, the time is formatted with asterisks to indicate that the value is out of range.

Example

The first example uses the local time to determine the time and the time zone offset. The second example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

Statement	Result

`data _null_ ;`
`t='12:34:56't;`
`put t e8601tx.;`
`run;`
	`07:34:56-05:00`

`options timezone='America/Adak';`
`data _null_ ;`
`t='12:34:56't;`
`put t e8601tx.;`
`run;`
	`02:34:56-10:00`

See Also

"Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations" in *SAS Viya Formats and Informats: Reference*

EURO Format

Writes numeric values with a leading euro symbol (E), a comma that separates every three digits, and a period that separates the decimal fraction.

Categories: Numeric
CAS

Alignment: Right

Syntax

EURO*w.d*

Syntax Description

w

specifies the width of the output field.

Default 6

Range 1-32

Tip If you want the euro symbol to be part of the output, be sure to choose an adequate width.

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 0

Range 0-31

Requirement must be less than *w*

Comparisons

- The EURO*w.d* format is similar to the EUROX*w.d* format, but EUROX*w.d* format reverses the roles of the decimal point and the comma. This convention is common in European countries.
- The EURO*w.d* format is similar to the DOLLAR*w.d* format, except that DOLLAR*w.d* format writes a leading dollar sign instead of the euro symbol.

Example

These examples use 1254.71 as the value of amount.

Statements	Results
	----+----1----+----2----+----3
put amount euro10.2;	E1,254.71
put amount euro5.;	E1255
put amount euro9.2;	E1,254.71
put amount euro15.3;	E1,254.710

```
data _null_;
   input x;
   put x euro10.2;
   put x euro5.;
   put x euro9.2;
   put x euro15.3;
   datalines;
1254.71
;
run;
SAS Log:
 E1,254.71
E1255
E1,254.71
     E1,254.710
```

```
   /* This code determines the default length. */
   data _null_;
      input x;
      put x euro.;
      datalines;
   1
   22
   333
   4444
   55555
   666666
   7777777
   88888888
   999999999
```

```
   1234561234
   ;run;
   SAS Log:
        datalines;
       E1
      E22
     E333
   E4,444
   55,555
   666666
   7.78E6
   8.89E7
      1E9
   1.23E9
   NOTE: At least one W.D format was too small for the number to be printed.
   The decimal may be shifted by the "BEST" format.
```

```
/* This code determines the range. */
data _null_;
   input x;
   put x euro5.;
   put x euro6.;
   put x euro7.;
   put x euro8.;
   put x euro9.;
   put x euro9.2;
   put x euro10.;
   put x euro10.2;
   put x euro10.4;
   put x euro11.;
   put x euro11.3;
   put x euro12.;
   put x euro12.2;
   put x euro13.;
   put x euro13.2;
   datalines;
333
4444
55555
666666
7777777
88888888
999999999
1234561234
;run;
```

See Also

Format:

- “EUROX Format” on page 99

Informats:

- “EURO Informat” on page 444
- “EUROX Informat” on page 445

EUROX Format

Writes numeric values with a leading euro symbol (E), a period that separates every three digits, and a comma that separates the decimal fraction.

Categories: Numeric
CAS

Alignment: Right

Syntax

EUROX*w.d*

Syntax Description

w

specifies the width of the output field.

Default	6
Range	1-32
Tip	If you want the euro symbol to be part of the output, be sure to choose an adequate width.

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default	0
Range	0-31
Requirement	must be less than *w*

Comparisons

- The EUROX*w.d* format is similar to the EURO*w.d* format, but EURO*w.d* format reverses the roles of the comma and the decimal point. This convention is common in English–speaking countries.
- The EUROX*w.d* format is similar to the DOLLARX*w.d* format, except that DOLLARX*w.d* format writes a leading dollar sign instead of the euro symbol.

Example

These examples use 1254.71 as the value of amount.

Statements	Results
	`----+----1----+----2----+----3`
`put amount eurox10.2;`	`E1.254,71`

Statements	Results
put amount eurox5.;	E1255
put amount eurox9.2;	E1.254,71
put amount eurox15.3;	E1.254,710

```
data _null_;
   input x;
   put x eurox10.2;
   put x eurox5.;
   put x eurox9.2;
   put x eurox15.3;
   datalines;
1254.71
; run;
SAS Log:
 E1.254,71
E1255
E1.254,71
     E1.254,710
```

```
/* This code determines the default length. */
data _null_;
   input x;
   put x eurox.;
   datalines;
1
22
333
4444
55555
666666
7777777
88888888
999999999
1234561234
;run;
SAS Log:
    E1
   E22
  E333
E4.444
55.555
666666
7.78E6
8.89E7
   1E9
1.23E9
```

Note: At least one W.D format was too small for the number to be printed. The decimal might be shifted by the "BEST" format.

See Also

Format:

- "EURO Format" on page 96

Informats:

- "EURO Informat" on page 444
- "EUROX Informat" on page 445

MINGUO Format

Writes date values as Taiwanese dates in the form *yyyymmdd*.

Category: Date and Time

Alignment: Left

Syntax

MINGUO*w*.

Syntax Description

w
specifies the width of the output field.

Default 8

Range 1–10

Details

The MINGUO*w*. format writes SAS date values in the form *yyyymmdd*, where

yyyy
is an integer that represents the year.

mm
is an integer that represents the month.

dd
is an integer that represents the day of the month.

The Taiwanese calendar uses 1912 as the base year (01/01/01 is January 1, 1912). Dates before 1912 appear as a series of asterisks. Year values do not roll around after 100 years. Instead, they continue to increase.

Example

The example table uses the following input values:

- 12054 is the SAS date value that corresponds to January 1, 1993.
- 18993 is the SAS date value that corresponds to January 1, 2012.

- -20088 is the SAS date value that corresponds to January 1, 1905.

Statements	Results
`x=put(12054,minguo7.);` `put x=;`	`x=820101`
`x=put(12054,minguo9.);` `put x=;`	`x=82/01/01`
`x=put(12054,minguo10.);` `put x=;`	`x=0082/01/01`
`x=put(18993,minguo7.);` `put x=;`	`x=1000101`
`x=put(18993,minguo9.);` `put x=;`	`x=100/01/01`
`x=put(18993,minguo10.);` `put x=;`	`x=0101/01/01`
`x=put(-20088,minguo7.);` `put x=;`	`**********`
`x=put(-20088,minguo9.);` `put x=;`	`**********`
`x=put(-20088,minguo10.);` `put x=;`	`**********`

See Also

Informat:

- "MINGUO Informat" on page 447

NENGO Format

Writes date values as Japanese dates in the form *e.yymmdd*.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NENGO*w.*

Syntax Description

w
specifies the width of the output field.

Default 10

Range 2–10

Details

The NENGO*w*. format writes SAS date values in the form *e.yymmdd*, where

e
is the first letter of the name of the emperor (Meiji, Taisho, Showa, or Heisei).

yy
is an integer that represents the year.

mm
is an integer that represents the month.

dd
is an integer that represents the day of the month.

If the width is too small, SAS omits the period.

Example

The example table uses the input value of 15342, which is the SAS date value that corresponds to January 2, 2002.

```
data _null_;
   date=15342;
   put date nengo3.;
   put date nengo6.;
   put date nengo8.;
   put date nengo9.;
   put date nengo10.;
run
```

Statements	Results
	----+----1
put date nengo3.;	H14
put date nengo6.;	H14/01
put date nengo8.;	H.140102
put date nengo9.;	H14/01/02
put date nengo10.;	H.14/01/02

See Also

Informat:

- "NENGO Informat" on page 449

NLBEST Format

Writes the best numerical notation based on the locale.

Categories: Numeric
CAS

Alignment: Right

Syntax

NLBEST*w.*

Syntax Description

w
specifies the width of the output field.

Default	12
Range	1–32
Tip	If you print numbers between 0 and .01 exclusively, then use a field width of at least 7 to avoid excessive rounding. If you print numbers between 0 and -.01 exclusively, use a field width of at least 8.

Details

The NLBEST format writes the best numerical value based on the locale's decimal point and the sign mark's location. NLBEST is similar to the BEST format. For more information, see the BEST format in the *SAS Viya Formats and Informats: Reference*.

Example

The following code produces results based on the locale:

```
x=-1257000
 put x nlbest6.;
 put x nlbest3.;
 put "=====";
x=-0.1
 put x nlbest6.;
 put x nlbest3.;
 put "=====";
x=0.1
 put x nlbest6.;
 put x nlbest3.;
 put "=====";
```

```
x=1257000
 put x nlbest6.;
 put x nlbest3.;
```

Locales	Results
locale=English_UnitedStates	-126E4 *** ===== -0.1 -.1 ===== 0.1 0.1 ===== 1.26E6 1E6
locale=German_Germany	-126E4 *** ===== -0,1 -,1 ===== 0,1 0,1 ===== 1,26E6 1E6
locale=ar_BH	126E4- *** ===== 0.1- .1- ===== 0.1 0.1 ===== 1.26E6 1E6

NLDATE Format

Converts a SAS date value to the date value of the specified locale, and then writes the date value as a date.

Categories: Date and Time
CAS

Alignment: Left

Note: Use the NLDATE format in the CAS server instead of the WORDDATE format.

Syntax

NLDATE*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Note: The format pattern, long, medium, short, is used according to the specified width.

Default 20

Range 10–200

Comparisons

NLDATE*w*. is similar to DATE*w*. and WORDDATE*w*. except that NLDATE*w*. is locale-specific.

Example

These examples use the input value of 15760, which is the SAS date value that corresponds to February 24, 2003.

Statements	Results
	`----+----1----+----2`
`options locale=English_UnitedStates;` `put day nldate.;`	`February 24, 2003`
`options locale=German_Germany;` `put day nldate.;`	`24. Februar 2003`

See Also

Formats:

- "NLDATEMN Format" on page 112
- "NLDATEW Format" on page 114
- "NLDATEWN Format" on page 116

NLDATEL Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form, date, month, year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEL*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default	18
Range	2–220

Details

NLDATEL writes the date in a long-uniform pattern with the full length of the month and week names.

Example

This example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEL min=2 default=18`
` max=200 ---+";`
`  put dy nldatel.;`
`  put dy nldatel10.;`
`  put dy nldatel12.;`
`  put dy nldatel18.;`
`  put dy nldatel200.;`
`run;`
	`+--- NLDATEL min=2 default=18 max=200 ---+`
	`November 19, 2012`
	`11/19/2012`
	`Nov 19, 2012`
	`November 19, 2012`
	`November 19, 2012`

NLDATEM Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEM*w*

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 14

Range 2–200

Details

NLDATEM writes the date in a medium-uniform pattern such as Nov 19, 2012.

Example

This example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEM min=2 default=14`
` max=200 ---+";`
`  put dy nldatem.;`
`  put dy nldatem8.;`
`  put dy nldatem14.;`
`  put dy nldatem200.;`
`run;`
	`+--- NLDATEM min=2 default=14 max=200 ---+`
	`Nov 19, 2012`
	`11/19/12`
	`Nov 19, 2012`
	`Nov 19, 2012`

NLDATEMD Format

Converts the SAS date value to the date value of the specified locale, and then writes the value as the name of the month and the day of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEMD*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 6-200

Example

This example uses the en_US locale option.

Statement	Result
`put 1 nldatemd.;`	`January 02`

See Also

Format:

- "NLDATEYM Format" on page 117

NLDATEMDL Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEMDL*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 12

Range 5–200

Details

NLDATEMDL writes the date in a long-uniform pattern with full length of the month and the day, such as November 19.

Example

The following example uses the date of November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEMDL min=5 default=12 max=200 ---+";`
`  put dy nldatemdl.;`
`  put dy nldatemdl5.;`
`  put dy nldatemdl9.;`
`  put dy nldatemdl12.;`
`  put dy nldatemdl200.;`
`run;`
	`+--- NLDATEMDL min=5 default=12 max=200 ---+`
	`November 19`
	`11/19`
	`Nov 19`
	`November 19`
	`November 19`

NLDATEMDM Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEMDM*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 9

Range 5–200

Details

NLDATEMDM writes the date in a medium-uniform pattern with abbreviation of the month and the day using numbers and delimiters, such as Nov 19.

Example

The following example uses the date of November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEMDM min=5 default=9`
`  max=200 ---+";`
`  put dy nldatemdm.;`
`  put dy nldatemdm5.;`
`  put dy nldatemdm9.;`
`  put dy nldatemdm200.;`
`run;`
	`+--- NLDATEMDM min=5 default=9 max=200 ---+`
	`Nov 19`
	`11/19`
	`Nov 19`
	`Nov 19`

NLDATEMDS Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEMDS*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 5

Range 5–200

Details

NLDATEMDS writes the date in a short-uniform pattern in full length of the month and the day using numbers and delimiters, such as MM/dd.

Example

The following example uses the date of November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEMDS min=5 default=5`
`  max=200 ---+";`
`  put dy nldatemds.;`
`  put dy nldatemds5.;`
`  put dy nldatemds5.;`
`  put dy nldatemds200.;`
`run;`
	`+--- NLDATEMDS min=5 default=5 max=200 ---+`
	`11/19`
	`11/19`
	`11/19`
	`11/19`

NLDATEMN Format

Converts a SAS date value to the date value of the specified locale, and then writes the value as the name of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEMN*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the name of the month to fit the format width.

Default 9

Range 4–200

Comparisons

NLDATEMN*w*. is similar to MONNAME*w*. except that NLDATEMN*w*. is locale-specific.

Example

These examples use the input value of 15760, which is the SAS date value that corresponds to February 24, 2003.

Statements	Results
	`----+----1`
`options locale=English_UnitedStates;` `put month nldatemn.;`	`February`
`options locale=German_Germany;` `put month nldatemn.;`	`Februar`

See Also

Formats:

- "NLDATE Format" on page 106
- "NLDATEW Format" on page 114
- "NLDATEWN Format" on page 116

NLDATES Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date string.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATES*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 10

Range 2–200

Details

NLDATES writes the date in a short-uniform pattern that contains only numbers and delimiters, such as mm/dd/yyyy.

Example

This example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATES min=2 default=10`
` max=200 ---+";`
`  put dy nldates.;`
`  put dy nldates8.;`
`  put dy nldates10.;`
`  put dy nldates200.;`
`run;`
	`+--- NLDATES min=2 default=10 max=200 ---+`
	`11/19/2012`
	`11/19/12`
	`11/19/2012`
	`11/19/2012`

NLDATEW Format

Converts a SAS date value to the date value of the specified locale, and then writes the value as the date and the day of the week.

Categories: Date and Time
CAS

Alignment: Left

Note: Use the NLDATEW format in the CAS server instead of the WEEKDATE format.

Syntax

NLDATEW*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date and the day of the week to fit the format width.

Default 29

Range 10–200

Details

The NLDATEW format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths on page 60 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

Comparisons

NLDATEW*w*. is similar to WEEKDATE*w*. except that NLDATEW*w*. is locale specific.

Example

These examples use the input value of 15760, which is the SAS date value that corresponds to February 24, 2003.

Statements	Results
	`----+----1----+----2`

`options locale=English_UnitedStates;`
`date=15760;`
`x=put(date,nldatew.);`
`y=put(date,nldatew20.);`
`z=put(date,nldatew200.);`
`run;`
	`Monday, February 24, 2003`
	`Mon, Feb 24, 2003`
	`Monday, February 24, 2003`

`options locale=German_Germany;`
`date=15760;`
`x=put(date,nldatew.);`
`y=put(date,nldatew20.);`
`z=put(date,nldatew200.);`
`run;`
	`Mo., 24. Feb 2003`
	`Mo., 24. Feb 2003`
	`Montag, 24. Februar 2003`

See Also

Formats:

- "NLDATE Format" on page 106
- "NLDATEMN Format" on page 112
- "NLDATEWN Format" on page 116

NLDATEWN Format

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the day of the week.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEWN*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the day of the week to fit the format width.

Default 9

Range 4–200

Comparisons

NLDATEWN*w*. is similar to DOWNAME*w*. except that NLDATEWN*w*. is locale-specific.

Example

These examples use the input value of 15760, which is the SAS date value that corresponds to February 24, 2003.

Statements	Results
	`----+----1`
`options locale=English_UnitedStates;` `put date nldatewn.;`	`Monday`

Statements	Results
`options locale=German_Germany;` `put date nldatewn.;`	`Montag`

See Also

Formats:

- "NLDATE Format" on page 106
- "NLDATEMN Format" on page 112
- "NLDATEW Format" on page 114

NLDATEYM Format

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the name of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYM*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 6–200

Details

If you specify a width of 6, but your data is larger than 6, your output contains asterisks: **d=******. To remove the asterisks, you can use PROC LOCALEDATA. The following example uses PROC LOCALEDATA to write the date without the asterisks:

```
PROC LOCALEDATA;
  LOAD SASLOCALE;
  MODIFY key=DATE_YYMM_SHORT_FORMAT value='%b %y' ;
  SAVE REGISTRY / _ALL_ syntax=SAS;

data _null_;
 format d nldateym6.;
 d = '17OCT14'd;
 put d=;
```

```
run;
```

```
d=Oct 14
```

Example

This example uses the spanish_Spain locale option, and the date of August 2010.

Statement	Result
`options locale=spanish_Spain;`	
`data_null_;`	`agosto de 2010`
`dy=today();`	`ago de 10`
`x=put(dy, nldateym.);`	`agosto de 2010`
`y=put(dy, nldateym12.);`	
`z=put(dy, nldateym200.);`	
`run;`	

See Also

Format:

- "NLDATEMD Format" on page 109

NLDATEYML Format

Converts a SAS date value to the date string of the specified locale, and then writes the month and year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYML*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 14

Range 5–200

Details

NLDATEYML writes the date in a long-uniform pattern with abbreviations for the month and year, such as November 2012.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEYML min=5 default=14 max=200 ---+";`
`  put dy nldateyml.;`
`  put dy nldateyml7.;`
`  put dy nldateyml11.;`
`  put dy nldateyml14.;`
`  put dy nldateyml200.;`
`run;`
	`+--- NLDATEYML min=5 default=14 max=200 ---+`
	`November 2012`
	`11/2012`
	`Nov 2012`
	`November 2012`
	`November 2012`

NLDATEYMM Format

Converts a SAS date value to the date string of the specified locale, and then writes the date values as the month and year with abbreviations.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYMM*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 11

Range 5–200

Details

NLDATEYMS writes the date in a medium-uniform pattern with abbreviations for the month and year, such as Nov 2012.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEYMM min=5 default=11 max=200 ---+";`
`  put dy nldateymm.;`
`  put dy nldateymm7.;`
`  put dy nldateymm11.;`
`  put dy nldateymm200.;`
`run;`
	`+--- NLDATEYMM min=5 default=11 max=200 ---+`
	`Nov 2012`
	`11/2012`
	`Nov 2012`
	`Nov 2012`

NLDATEYMS Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYMS*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 7

Range 5–200

Details

NLDATEYMS writes the date in a short-uniform pattern with numbers and delimiters such as mm/yyy.

Example

This example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEYMS min=5 default=7 max=200 ---+";`
`  put dy nldateyms.;`
`  put dy nldateyms5.;`
`  put dy nldateyms7.;`
`  put dy nldateyms200.;`
`run;`
	`+--- NLDATEYMS min=5 default=7 max=200 ---+`
	`11/2012`
	`11/12`
	`11/2012`
	`11/2012`

NLDATEYQ Format

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the quarter.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYQ*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 4–200

Details

The NLDATEYQ format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths on page 60 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

Example

This example uses the fr_FR locale option.

Statements	Results

options locale=fr_FR;
data _null_;
dy=today();
dt=datetime();
put "+--- NLDATEYQ min=4 default=16 max=200 ---+";
put ' 16' +5 dy nldateyq.;
put ' 4' +5 dy nldateyq4.;
put ' 14' +5 dy nldateyq14.;
put ' 32' +5 dy nldateyq32.;
put '200' +5 dy nldateyq200.;
run;
	+--- NLDATEYQ min=4 default=16 max=200 ---+
	16 T3 08
	4 ****
	14 T3 08
	32 3e trimestre 2008
	200
	3e trimestre 2008

NLDATEYQL Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (Q1–Q4) using abbreviations.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYQL*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 18

Range 4–200

Details

NLDATEYQL writes the date in a long-uniform pattern with full length for the year and year's quarter value, such as 4th quarter 2012.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEYQL min=4 default=18 max=200 ---+";`
`  put dy nldateyql.;`
`  put dy nldateyql6.;`
`  put dy nldateyql7.;`
`  put dy nldateyql18.;`
`  put dy nldateyql200.;`
`run;`
	`+--- NLDATEYQL min=4 default=18 max=200 ---+`
	`4th quarter 2013`
	`2013/4`
	`Q4 2013`
	`4th quarter 2013`
	`4th quarter 2013`

NLDATEYQM Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (Q1–Q4) using abbreviations.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYQM*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 7

Range 4–200

Details

NLDATEYQM writes the date in a medium-uniform pattern with abbreviations for the year and year's quarter value, such as Q4 2012.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEYQM min=4 default=7 max=200 ---+";`
`  put dy nldateyqm.;`
`  put dy nldateyqm6.;`
`  put dy nldateyqm7.;`
`  put dy nldateyqm200.;`
`run;`
	`+--- NLDATEYQM min=4 default=7 max=200 ---+`
	`Q4 2012`
	`2012/4`
	`Q4 2012`
	`Q4 2012`

NLDATEYQS Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (1–4) with numbers and delimiters.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYQS*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 6

Range 4–200

Details

NLDATEYQS writes the date in a short-uniform pattern with numbers and delimiters for the year and year's quarter value, such as 2012/4.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATEYQS min=4 default=6 max=200 ---+";`
`  put dy nldateyqs.;`
`  put dy nldateyqs4.;`
`  put dy nldateyqs6.;`
`  put dy nldateyqs200.;`
`run;`
	`+--- NLDATEYQS min=4 default=6 max=200 ---+`
	`2012/4`
	`12/4`
	`2012/4`
	`2012/4`

NLDATEYR Format

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYR*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 2–200

Example

This example uses the fr_FR locale option.

Statements	Results

options locale=fr_FR;
data _null_;
dy=today();
dt=datetime();
put "+--- NLDATEYR min=2 default=16 max=200 ---+";
put dy nldateyr.;
put dy nldateyr2.;
put dy nldateyr8.;
put dy nldateyr200.;
run;
	+--- NLDATEYR min=2 default=16 max=200 ---+
	2008
	08
	2008
	2008

NLDATEYW Format

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the week.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATEYW*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 5–200

Example

This example uses the fr_FR locale option.

Statements	Results

options locale=fr_FR;
data _null_;
dy=today();
dt=datetime();
put "+--- NLDATEYW min=5 default=16 max=200 ---+";
put ' 16' +5 dy nldateyw.;
put ' 5' +5 dy nldateyw5.;
put ' 8' +5 dy nldateyw8.;
put ' 32' +5 dy nldateyw32.;
put '200' +5 dy nldateyw200.;
run;
	16 Week 33 2008
	5 *****
	8 W33 08
	32 Week 33 2008
	200
	Week 33 2008

NLDATM Format

Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATM*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the datetime value to fit the format width.

Default 30

Range 10–200

Comparisons

The NLDATM*w*. format is similar to the DATETIME*w*. format except that the NLDATM*w*. format is locale-specific.

Example

These examples use the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003.

Statements	Results
	`----+----1----+----2----+----3`
`options locale=English_UnitedStates;` `put day nldatm.;`	`24Feb2003:12:39:43`
`options locale=German_Germany;` `put day nldatm.;`	`24. Februar 2003 12.39 Uhr`

See Also

Formats:

- "NLDATMAP Format" on page 128
- "NLDATMTM Format" on page 137
- "NLDATMW Format" on page 139

NLDATMAP Format

Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime with a.m. or p.m.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMAP*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date-time value to fit the format width.

Default 32

Range 16–200

Comparisons

The NLDATMAP*w*. format is similar to DATEAMPM*w*. except that the NLDATMAP*w*. format is locale-specific.

Example

These examples use the input value of 1361709583, which is the SAS date-time value that corresponds to 12:39:43 p.m. on February 24, 2003.

Statements	Results
	`----+----1----+----2----+----3`
`options locale=English_UnitedStates;` `put event nldatmap.;`	`February 24, 2003 12:39:43 PM`
`options locale=Spanish_Mexico;` `put event nldatmap.;`	`24/02/2003 12:39:43 PM`

See Also

Formats:

- "NLDATM Format" on page 127
- "NLDATMTM Format" on page 137
- "NLDATMW Format" on page 139

NLDATMDT Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month, day of the month and year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMDT*w*.

Syntax Description

w
specifies the width of the output field

Default 18

Range 10-200

Example

This example uses the en_US locale option.

Statements	Results

`options locale=en_US;`
`data _null_;`
`x=86400;`
`put x nldatmdt.;`
`run;`
`options locale=en_US;`
`data _null_;`
`x=86400;`
`put x dtdate.;`
`run;`
	`January 02, 1960`
	`02JAN60`

See Also

Formats:

- "NLDATMMD Format" on page 131

NLDATML Format

Converts a SAS datetime value to the date string of the specified locale, and then writes the date value as a date in the form, month, date, year, and time.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATML*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 30

Range 9–200

Details

NLDATML writes the date in a long-uniform pattern with the full length of the month date, year, and time, such as, November 19, 2012 02:57:44 PM.

Example

This example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATML min=9 default=30 max=200 ---+";`
`  put dt nldatml.;`
`  put dt nldatml19.;`
`  put dt nldatml24.;`
`  put dt nldatml30.;`
`  put dt nldatml200.;`
`run;`
	`+--- NLDATML min=9 default=30 max=200 ---+`
	`November 19, 2012 02:57:44 PM`
	`11/19/2012 14:57:44`
	`Nov 19, 2012 02:57:44 PM`
	`November 19, 2012 02:57:44 PM`
	`November 19, 2012 02:57:44 PM`

NLDATMM Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and time with abbreviations for the month and time.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMM*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 24

Range 9–200

Details

NLDATMM writes the date in a medium-uniform pattern with abbreviations of the month and week names, such as Nov 19, 2012 02:51:40 PM.

NLDATMMD Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month and the day of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMMD*w*.

Syntax Description

w

specifies the width of the output field.

Default 16

Range 6–200

Example

This example uses the en_US locale option.

Statement	Result

`options locale=en_US;`
`data _null_`
`x=put(86400,nldatmmd.);`
`put x=;`
`run;`
	`January 02`

See Also

Format:

- "NLDATMYM Format" on page 142

NLDATMMDL Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the full-length of the month and day of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMMDL*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 12

Range 5–200

Details

NLDATMMDL writes the date in a long-uniform pattern with full-length of the month and the day, such as November 19.

Example

The following example uses the date of November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMMDL min=5 default=12 max=200 ---+";`
`  put dt nldatmmdl.;`
`  put dt nldatmmdl5.;`
`  put dt nldatmmdl9.;`
`  put dt nldatmmdl12.;`
`  put dt nldatmmdl200.;`
`run;`
	`+--- NLDATMMDL min=5 default=12 max=200 ---+`
	`November 19`
	`11/19`
	`Nov 19`
	`November 19`
	`November 19`

NLDATMMDM Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using abbreviations.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMMDM*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 9

Range 5–200

Details

NLDATMMDM writes the date in a medium-uniform pattern with abbreviations of the month and the day, such as Nov 19.

Example

The following example uses the date of November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMMDM min=5 default=9 max=200 ---+";`
`  put dt nldatmmdm.;`
`  put dt nldatmmdm5.;`
`  put dt nldatmmdm9.;`
`  put dt nldatmmdm200.;`
`run;`
	`+--- NLDATMMDM min=5 default=9 max=200 ---+`
	`Nov 19`
	`11/19`
	`Nov 19`
	`Nov 19`

NLDATMMDS Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using numbers and delimiters.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMMDS*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 5

Range 5–200

Details

NLDATMMDS writes the date in a short-uniform pattern with numbers and delimiters of the month and the day, such as 11/19.

Example

The following example uses the date of November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMMDS min=5 default=5 max=200 ---+";`
`  put dt nldatmmds.;`
`  put dt nldatmmds5.;`
`  put dt nldatmmds5.;`
`  put dt nldatmmds200.;`
`run;`
	`+--- NLDATMMDS min=5 default=5 max=200 ---+`
	`11/19`
	`11/19`
	`11/19`
	`11/19`

NLDATMMN Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMMN*w*.

Syntax Description

w
specifies the width of the output field.

Default 9

Range 4–200

Example

This example uses the en_US locale option.

Statements	Results

data _null_;
dt = datetime();
dy = date();
put "+--- NLDATEMN min=4 default=10 max=200 ---+";
put dt nldatmmn.;
put dt nldatmmn4.;
put dt nldatmmn10.;
put dt nldatmmn200.;
run;
	+--- NLDATMMN min=4 default=10 max=200 ---+
	October
	Oct
	October
	October

NLDATMS Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form, MM/DD/YYYY .

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMS*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 19

Range 9–200

Details

NLDATMS writes the date in a short-uniform pattern with number and delimiters, such as MM/DD/YYYY hh:mm:ss.

Example

This example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMS min=9 default=19 max=200 ---+";`
`  put dt nldatms.;`
`  put dt nldatms10.;`
`  put dt nldatms19.;`
`  put dt nldatms200.;`
`run;`
	`+--- NLDATMS min=9 default=19 max=200 ---+`
	`11/19/2012 14:38:52`
	`111912 14`
	`11/19/2012 14:38:52`
	`11/19/2012 14:38:52`

NLDATMTM Format

Converts the time portion of a SAS datetime value to the time-of-day value of the specified locale, and then writes the value as a time of day.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMTM*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 16–200

Comparisons

The NLDATMTM*w*. format is similar to the TOD*w*. format except that the NLDATMTM*w*. format is locale-specific.

Example

These examples use the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003.

Statements	Results
	`----+----1`

Statements	Results
`options locale=English_UnitedStates;` `put event nldatmtm.;`	`12:39:43`
`options locale=German_Germany;` `put event nldatmtm.;`	`12.39 Uhr`

See Also

Formats:

- "NLDATM Format" on page 127
- "NLDATMAP Format" on page 128
- "NLDATMW Format" on page 139

NLDATMTZ Format

Converts the time portion of the SAS datetime of the locale to the time of day and time zone.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMTZ*w*.

Syntax Description

w
specifies the width of the output field.

Default 32

Range 16–200

Example

This example uses the current datetime value.

Statements	Result

`options locale=fr_FR;`
`data test;`
`x=datetime();`
`put x=nldatmtz.;`
`run;`
	`x=10 h 40 -0400`

NLDATMW Format

Converts SAS datetime values to the locale sensitive datetime string as the day of the week and the datetime.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMW*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the day of week and datetime to fit the format width.

Default 41

Range 16–200

Details

The NLDATMW format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths on page 60 for information about recommended widths for locale and encoding combinations. You might need to use the suggested width for the NLDATMW format.

Comparisons

The NLDATMW*w*. format is similar to the TWMDY*w*. format except that the NLDATMW*w*. format is locale-specific.

Example

These examples use the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003.

Statements	Results
	`----+----1----+----2----+----3`

Statements	Results

options locale=English_UnitedStates;
data _null_;
x=put(1361709583,nldatmw.);
y=put(1361709583,nldatmw30.);
z=put(1361709583,nldatmw200.);
run;
	Mon, Feb 24, 2003 12:39:43 PM
	Mon, Feb 24, 2003 12:39:43 PM
	Monday, February 24, 2003 12:39:43 PM

options locale=german_germany;
data _null_;
x=put(1361709583,nldatmw.);
y=put(1361709583,nldatmw30.);
z=put(1361709583,nldatmw200.);
run;
	Mo, 24. Feb 2003 12.39 Uhr
	Mo, 24. Feb 2003 12.39 Uhr
	Montag, 24. Februar 2003 12.39 Uhr

See Also

Formats:

- "NLDATM Format" on page 127
- "NLDATMAP Format" on page 128
- "NLDATMTM Format" on page 137

NLDATMWN Format

Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as the day of the week.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMWN*w*.

Syntax Description

w
specifies the width of the output field.

Default 9

Range 4–200

Example

This example writes the SAS datetime value as a day of the week.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMWN min=4 default=9`
`  max=200 ---+";`
`  put dt nldatmwn.;`
`  put dt nldatmwn4.;`
`  put dt nldatmwn9.;`
`  put dt nldatmwn200.;`
`run;`
	`+--- NLDATMWN min=4 default=9 max=200 ---+`
	`Monday`
	`Mon`
	`Monday`
	`Monday`

NLDATMWZ Format

Converts SAS date values of the specified locale to a day-of-week, datetime, and time zone value.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMWZ*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the day of week and datetime to fit the format width.

Default 40

Range 16–200

Details

The NLDATMWZ format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths on page 60 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

Example

This example uses the current datetime value.

Statements	Result

`options locale=fr_FR;`
`data test;`
`x=datetime();`
`put x=nldatmwz.;`
`run;`
	`x=vendredi 18 mars 2011 10 h 40 -0400`

NLDATMYM Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the name of the month.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYM*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 6–200

Example

This example uses the en_US locale option.

Statement	Result
`options locale=en_US;`	`January 1960`
`data _null_;`	`January 1960`
`x=put(86400,nldatmym.);`	
`y=put(86400,nldatmym12.);`	
`put x=;`	
`put y=;`	
`run;`	

See Also

Format:

- "NLDATMMD Format" on page 131

NLDATMYML Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYML*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 14

Range 5–200

Details

NLDATMYML writes the date in a long-uniform pattern with full length of the month and year, such as November 2012.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMYML min=5 default=14 max=200 ---+";`
`  put dt nldatmyml.;`
`  put dt nldatmyml7.;`
`  put dt nldatmyml11.;`
`  put dt nldatmyml14.;`
`  put dt nldatmyml200.;`
`run;`
	`+--- NLDATMYML min=5 default=14 max=200 ---+`
	`November 2012`
	`11/2012`
	`Nov 2012`
	`November 2012`
	`November 2012`

NLDATMYMM Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYMM*w*.

Syntax Description

w
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 11

Range 5–200

Details

NLDATMYMM writes the date in a medium-uniform pattern with abbreviations of the month and year, such as Nov 2012.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMYMM min=5 default=11 max=200 ---+";`
`  put dt nldatmymm.;`
`  put dt nldatmymm7.;`
`  put dt nldatmymm11.;`
`  put dt nldatmymm200.;`
`run;`
	`+--- NLDATMYMM min=5 default=11 max=200 ---+`
	`Nov 2012`
	`11/2012`
	`Nov 2012`
	`Nov 2012`

NLDATMYMS Format

Converts a SAS date value to the date string of the specified locale, and then writes the month and year with numbers and delimiters.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYMS*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 7

Range 5–200

Details

NLDATMYMS writes the date in a short-uniform pattern with numbers and delimiters for the month and year, such as 11/2012.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMYMS min=5 default=7`
`  max=200 ---+";`
`  put dt nldatmyms.;`
`  put dt nldatmyms5.;`
`  put dt nldatmyms7.;`
`  put dt nldatmyms200.;`
`run;`
	`+--- NLDATMYMS min=5 default=7 max=200 ---+`
	`11/2012`
	`11/12`
	`11/2012`
	`11/2012`

NLDATMYQ Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the quarter of the year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYQ*w*.

Syntax Description

w

specifies the width of the output field.

Default 16

Range 4–200

Details

The NLDATMYQ format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths on page 60 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

Example

This example uses the fr_FR locale option.

Statements	Results

options locale=fr_FR;
data _null_;
dy=today();
dt=datetime();
put "+--- NLDATMYQ min=4 default=16 max=200 ---+";
put ' 16' +5 dt nldatmyq.;
put ' 4' +5 dt nldatmyq4.;
put ' 14' +5 dt nldatmyq14.;
put ' 32' +5 dt nldatmyq32.;
put '200' +5 dt nldatmyq200.;
run;
	+--- NLDATMYQ min=4 default=16 max=200 ---+
	16 T3 08
	4 ****
	14 T3 08
	32 3e trimestre 2008
	200 3e trimestre 2008

NLDATMYQL Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year's quarter value (1–4) and the year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYQL*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 18

Range 4–200

Details

NLDATMYQL writes the date in a long uniform pattern in full length of the year's quarter and then the year, such as 4th quarter 2012.

Example

The following example uses the date of November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMYQL min=4 default=18 max=200 ---+";`
`  put dt nldatmyql.;`
`  put dt nldatmyql6.;`
`  put dt nldatmyql7.;`
`  put dt nldatmyql18.;`
`  put dt nldatmyql200.;`
`run;`
	`+--- NLDATMYQL min=4 default=18 max=200 ---+`
	`4th quarter 2012`
	`2012/4`
	`Q4 2012`
	`4th quarter 2012`
	`4th quarter 2012`

NLDATMYQM Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year's quarter (1–4) and then the year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYQM*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 7

Range 4–200

Details

NLDATMYQM writes the date in a medium uniform pattern of the year's quarter and then the year, such as Q4 2012.

Example

The following example uses the date of November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMYQM min=4 default=7 max=200 ---+";`
`  put dt nldatmyqm.;`
`  put dt nldatmyqm6.;`
`  put dt nldatmyqm7.;`
`  put dt nldatmyqm200.;`
`run;`
	`+--- NLDATMYQM min=4 default=7 max=200 ---+`
	`Q4 2012`
	`2012/4`
	`Q4 2012`
	`Q4 2012`

NLDATMYQS Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the quarter (1-4) using numbers and delimiters.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYQS*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 6

Range 4–200

Details

NLDATMYQS writes the date in a short-uniform pattern of the year and year's quarter value using numbers and delimiters , such as 2012/4.

Example

The following example uses the date November 19, 2012.

Statements	Results

`data _null_;`
`  dt = datetime();`
`  dy = date();`
`  put "+--- NLDATMYQS min=4 default=6 max=200 ---+";`
`  put dt nldatmyqs.;`
`  put dt nldatmyqs4.;`
`  put dt nldatmyqs6.;`
`  put dt nldatmyqs200.;`
`run;`
	`+--- NLDATMYQS min=4 default=6 max=200 ---+`
	`2012/4`
	`12/4`
	`2012/4`
	`2012/4`

NLDATMYR Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYR*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 2–200

Example

This example uses the en_US locale option.

Statements	Results

options locale=fr_FR;
data _null_;
dy=today();
dt=datetime();
put "+--- NLDATMYR min=2 default=16 max=200 ---+";
put dt nldatmyr.;
put dt nldatmyr2.;
put dt nldatmyr32.;
put dt nldatmyr200.;
run;
	+--- NLDATMYR min=2 default=16 max=200 ---+
	2008
	08
	2008
	2008

NLDATMYW Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year and the name of the week.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMYW*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 5–200

Example

This example uses the fr_FR locale option.

Statements	Results

options locale=fr_FR;
data _null_;
dy=today();
dt=datetime();
put "+--- NLDATMYW min=5 default=16 max=200 ---+";
put ' 16' +5 dt nldatmyw.;
put ' 5' +5 dt nldatmyw5.;
put ' 8' +5 dt nldatmyw8.;
put ' 32' +5 dt nldatmyw32.;
put '200' +5 dt nldatmyw200.;
run;
	+--- NLDATMYW min=5 default=16 max=200 ---+
	16 Week 33 2008
	5 *****
	8 W33 08
	32 Week 33 2008
	200
	Week 33 2008

NLDATMZ Format

Converts SAS datetime values to the locale-sensitive datetime string as time zone and datetime.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLDATMZ*w*.

Syntax Description

w
specifies the width of the output field.

Default 40

Range 16–200

Example

This example uses the current datetime value.

Statements	Result

`options locale=fr_FR;`
`data test;`
`x=datetime();`
`put x=nldatmz.;`
`run;`
	`x=18 mars 2011 10 h 40 -0400`

NLMNIAED Format

Writes the monetary format of the international expression for the United Arab Emirates.

Category: Numeric

Alignment: Left

Syntax

NLMNIAED*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 3

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniaed32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(AED1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLAED Format" on page 189

NLMNIAUD Format

Writes the monetary format of the international expression for Australia.

Category: Numeric

Alignment: Left

Syntax

NLMNIAUD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniaud32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(AUD1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

• "NLMNLAUD Format" on page 190

NLMNIBGN Format

Writes the monetary format of the international expression for Bulgaria.

Category: Numeric

Alignment: Left

Syntax

NLMNIBGN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnibgn32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	----+----1----+
put x=;	(BGN1,234.57)
put y=;	$-1,234.57

See Also

Format:

• "NLMNLBGN Format" on page 191

NLMNIBRL Format

Writes the monetary format of the international expression for Brazil.

Category: Numeric

Alignment: Left

Syntax

NLMNIBRL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnibrl32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(BRL1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLBRL Format" on page 192

NLMNICAD Format

Writes the monetary format of the international expression for Canada.

Category: Numeric

Alignment: Left

Syntax

NLMNICAD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnicad32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(CAD1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLCAD Format" on page 193

NLMNICHF Format

Writes the monetary format of the international expression for Liechtenstein and Switzerland.

Category: Numeric

Alignment: Left

Syntax

NLMNICHF*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnichf32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(CHF1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLCHF Format" on page 194

NLMNICNY Format

Writes the monetary format of the international expression for China.

Category: Numeric

Alignment: Left

Syntax

NLMNICNY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 02

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnicny32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(CNY1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLCNY Format" on page 195

NLMNICZK Format

Writes the monetary format of the international expression for the Czech Republic.

Category: Numeric

Alignment: Left

Syntax

NLMNICZK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniczk32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(CZK1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLCZK Format" on page 196

NLMNIDKK Format

Writes the monetary format of the international expression for Denmark, Faroe Island, and Greenland.

Category: Numeric

Alignment: Left

Syntax

NLMNIDKK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnidkk32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(DKK1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLDKK Format" on page 197

NLMNIEEK Format

Writes the monetary format of the international expression for Estonia.

Category: Numeric

Alignment: Left

Syntax

NLMNIEEK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnieek32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	----+----1----+
put x=;	(EEK1,234.57)
put y=;	$-1,234.57

See Also

Format:

- "NLMNLEEK Format" on page 198

NLMNIEGP Format

Writes the monetary format of the international expression for Egypt.

Category: Numeric

Alignment: Left

Syntax

NLMNIEGP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 3

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniegp32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(EGP1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLEGP Format" on page 199

NLMNIEUR Format

Writes the monetary format of the international expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.

Category: Numeric

Alignment: Left

Syntax

NLMNIEUR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to **`Locale=German_Germany`**.

```
x=put(-1234.56789,nlmnieur32.2);
y=put(-1234.56789,nlmnleur32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`x=-1.234,57 EUR`
`put y=;`	y=-1.234,57 €

See Also

Format:

- "NLMNLEUR Format" on page 200

NLMNIGBP Format

Writes the monetary format of the international expression for the United Kingdom.

Category: Numeric

Alignment: Left

Syntax

NLMNIGBP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnigbp32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	----+----1----+
put x=;	(GBP1,234.57)
put y=;	$-1,234.57

See Also

Format:

- "NLMNLGBP Format" on page 201

NLMNIHKD Format

Writes the monetary format of the international expression for Hong Kong.

Category: Numeric

Alignment: Left

Syntax

NLMNIHKD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnihkd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(HKD1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLHKD Format" on page 202

NLMNIHRK Format

Writes the monetary format of the international expression for Croatia.

Category: Numeric

Alignment: Left

Syntax

NLMNIHRK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnihrk32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(HRK1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLHRK Format" on page 203

NLMNIHUF Format

Writes the monetary format of the international expression for Hungary.

Category: Numeric

Alignment: Left

Syntax

NLMNIHUF*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnihuf32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(HUF1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLHUF Format" on page 204

NLMNIIDR Format

Writes the monetary format of the international expression for Indonesia.

Category: Numeric

Alignment: Left

Syntax

NLMNIIDR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniidr32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(IDR1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLIDR Format" on page 205

NLMNIILS Format

Writes the monetary format of the international expression for Israel.

Category: Numeric

Alignment: Left

Syntax

NLMNIILS*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniils32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(ILS1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLILS Format" on page 206

NLMNIINR Format

Writes the monetary format of the international expression for India.

Category: Numeric

Alignment: Left

Syntax

NLMNIINR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniinr32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(INR1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLINR Format" on page 207

NLMNIJPY Format

Writes the monetary format of the international expression for Japan.

Category: Numeric

Alignment: Left

Syntax

NLMNIJPY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 0

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnijpy32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(JPY1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLJPY Format" on page 208

NLMNIKRW Format

Writes the monetary format of the international expression for South Korea.

Category: Numeric

Alignment: Left

Syntax

NLMNIKRW*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 0

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnikrw32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(KRW1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLKRW Format" on page 209

NLMNILTL Format

Writes the monetary format of the international expression for Lithuania.

Category: Numeric

Alignment: Left

Syntax

NLMNILTL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniltl32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(LTL1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLLTL Format" on page 210

NLMNILVL Format

Writes the monetary format of the international expression for Latvia.

Category: Numeric

Alignment: Left

Syntax

NLMNILVL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnilvl32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(LVL1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLLVL Format" on page 211

NLMNIMOP Format

Writes the monetary format of the international expression for Macau.

Category: Numeric

Alignment: Left

Syntax

NLMNIMOP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnimop32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(MOP1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLMOP Format" on page 212

NLMNIMXN Format

Writes the monetary format of the international expression for Mexico.

Category: Numeric

Alignment: Left

Syntax

NLMNIMXN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnimxn32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(MXN1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLMXN Format" on page 213

NLMNIMYR Format

Writes the monetary format of the international expression for Malaysia.

Category: Numeric

Alignment: Left

Syntax

NLMNIMYR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnimyr32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(MYR1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLMYR Format" on page 214

NLMNINOK Format

Writes the monetary format of the international expression for Norway.

Category: Numeric

Alignment: Left

Syntax

NLMNINOK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmninok32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(NOK1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLNOK Format" on page 215

NLMNINZD Format

Writes the monetary format of the international expression for New Zealand.

Category: Numeric

Alignment: Left

Syntax

NLMNINZD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmninzd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(NZD1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLNZD Format" on page 216

NLMNIPLN Format

Writes the monetary format of the international expression for Poland.

Category: Numeric

Alignment: Left

Syntax

NLMNIPLN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnipln32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(PLN1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLPLN Format" on page 217

NLMNIRUB Format

Writes the monetary format of the international expression for Russia.

Category: Numeric

Alignment: Left

Syntax

NLMNIRUB*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnirub32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(RUB1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLRUB Format" on page 218

NLMNISEK Format

Writes the monetary format of the international expression for Sweden.

Category: Numeric

Alignment: Left

Syntax

NLMNISEK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnisek32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(SEK1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLSEK Format" on page 219

NLMNISGD Format

Writes the monetary format of the international expression for Singapore.

Category: Numeric

Alignment: Left

Syntax

NLMNISGD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnisgd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(SGD1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLSGD Format" on page 220

NLMNITHB Format

Writes the monetary format of the international expression for Thailand.

Category: Numeric

Alignment: Left

Syntax

NLMNITHB*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnithb32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(THB1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLTHB Format" on page 221

NLMNITRY Format

Writes the monetary format of the international expression for Turkey.

Category: Numeric

Alignment: Left

Syntax

NLMNITRY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnitry32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(TRY1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLTRY Format" on page 222

NLMNITWD Format

Writes the monetary format of the international expression for Taiwan.

Category: Numeric

Alignment: Left

Syntax

NLMNITWD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnitwd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(TWD1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLTWD Format" on page 223

NLMNIUSD Format

Writes the monetary format of the international expression for Puerto Rico and the United States.

Category: Numeric

Alignment: Left

Syntax

NLMNIUSD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmniusd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(USD1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLUSD Format" on page 224

NLMNIZAR Format

Writes the monetary format of the international expression for South Africa.

Category: Numeric

Alignment: Left

Syntax

NLMNIZAR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnizar32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(ZAR1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNLZAR Format" on page 225

NLMNLAED Format

Writes the monetary format of the local expression for the United Arab Emirates.

Category: Numeric

Alignment: Left

Syntax

NLMNLAED*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 3

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlaed32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(AED1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIAED Format" on page 152

NLMNLAUD Format

Writes the monetary format of the local expression for Australia.

Category: Numeric

Alignment: Left

Syntax

NLMNLAUD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlaud32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	----+----1----+
put x=;	(AU$1,234.57)
put y=;	$-1,234.57

See Also

Format:

- "NLMNIAUD Format" on page 153

NLMNLBGN Format

Writes the monetary format of the local expression for Bulgaria.

Category: Numeric

Alignment: Left

Syntax

NLMNLBGN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlbgn32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(BGN1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIBGN Format" on page 154

NLMNLBRL Format

Writes the monetary format of the local expression for Brazil.

Category: Numeric

Alignment: Left

Syntax

NLMNLBRL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlbrl32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(R$1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIBRL Format" on page 155

NLMNLCAD Format

Writes the monetary format of the local expression for Canada.

Category: Numeric

Alignment: Left

Syntax

NLMNLCAD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlcad32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	----+----1----+
put x=; put y=;	(CA$1,234.57) $-1,234.57

See Also

Format:

- "NLMNICAD Format" on page 156

NLMNLCHF Format

Writes the monetary format of the local expression for Liechtenstein and Switzerland.

Category: Numeric

Alignment: Left

Syntax

NLMNLCHF*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlchf32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`SFr.1,234.57`
`put y=;`	`$-1,234.57`

See Also

Format:

- “NLMNICHF Format” on page 157

NLMNLCNY Format

Writes the monetary format of the local expression for China.

Category: Numeric

Alignment: Left

Syntax

NLMNLCNY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlcny32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(RMB1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNICNY Format" on page 158

NLMNLCZK Format

Writes the monetary format of the local expression for the Czech Republic.

Category: Numeric

Alignment: Left

Syntax

NLMNLCZK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlczk32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(CZK1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNICZK Format" on page 159

NLMNLDKK Format

Writes the monetary format of the local expression for Denmark, Faroe Island, and Greenland.

Category: Numeric

Alignment: Left

Syntax

NLMNLDKK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnldkk32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(kr1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIDKK Format" on page 160

NLMNLEEK Format

Writes the monetary format of the local expression for Estonia.

Category: Numeric

Alignment: Left

Syntax

NLMNLEEK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnleek32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(Kr1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIEEK Format" on page 161

NLMNLEGP Format

Writes the monetary format of the local expression for Egypt.

Category: Numeric

Alignment: Left

Syntax

NLMNLEGP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 3

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlegp32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(EGP1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIEGP Format" on page 162

NLMNLEUR Format

Writes the monetary format of the local expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.

Category: Numeric

Alignment: Left

Syntax

NLMNLEUR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to German_Germany.

```
x=put(-1234.56789,nlmnieur32.2);
y=put(-1234.56789,nlmnleur32.2);
```

Statements	Results
	----+----1----+----2----+
`put x=;`	x=-1.234,57 EUR
`put y=;`	y=-1.234,57 €

See Also

Format:

- “NLMNIEUR Format” on page 163

NLMNLGBP Format

Writes the monetary format of the local expression for the United Kingdom.

Category: Numeric

Alignment: Left

Syntax

NLMNLGBP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlgbp32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(£1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIGBP Format" on page 164

NLMNLHKD Format

Writes the monetary format of the local expression for Hong Kong.

Category: Numeric

Alignment: Left

Syntax

NLMNLHKD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlhkd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(HK$1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIHKD Format" on page 165

NLMNLHRK Format

Writes the monetary format of the local expression for Croatia.

Category: Numeric

Alignment: Left

Syntax

NLMNLHRK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlhrk32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(Kn1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIHRK Format" on page 166

NLMNLHUF Format

Writes the monetary format of the local expression for Hungary.

Category: Numeric

Alignment: Left

Syntax

NLMNLHUF*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlhuf32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(Ft1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIHUF Format" on page 167

NLMNLIDR Format

Writes the monetary format of the local expression for Indonesia.

Category: Numeric

Alignment: Left

Syntax

NLMNLIDR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value..

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlidr32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;` `put y=;`	`(Rp1,234.57)` `$-1,234.57`

See Also

Format:

- “NLMNIIDR Format” on page 168

NLMNLILS Format

Writes the monetary format of the local expression for Israel.

Category: Numeric

Alignment: Left

Syntax

NLMNLILS*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlils32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(ILS1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIILS Format" on page 169

NLMNLINR Format

Writes the monetary format of the local expression for India.

Category: Numeric

Alignment: Left

Syntax

NLMNLINR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlinr32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(INR1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIINR Format" on page 170

NLMNLJPY Format

Writes the monetary format of the international expression for Japan.

Category: Numeric

Alignment: Left

Syntax

NLMNLJPY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 0

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnljpy32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(JPY1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIJPY Format" on page 171

NLMNLKRW Format

Writes the monetary format of the local expression for South Korea.

Category: Numeric

Alignment: Left

Syntax

NLMNLKRW*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 0

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlkrw32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(KRW1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIKRW Format" on page 172

NLMNLLTL Format

Writes the monetary format of the local expression for Lithuania.

Category: Numeric

Alignment: Left

Syntax

NLMNLLTL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlltl32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(LT1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNILTL Format" on page 173

NLMNLLVL Format

Writes the monetary format of the local expression for Latvia.

Category: Numeric

Alignment: Left

Syntax

NLMNLLVL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnllvl32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(Ls1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- “NLMNILVL Format” on page 174

NLMNLMOP Format

Writes the monetary format of the local expression for Macau.

Category: Numeric

Alignment: Left

Syntax

NLMNLMOP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlmop32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(P1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIMOP Format" on page 175

NLMNLMXN Format

Writes the monetary format of the local expression for Mexico.

Category: Numeric

Alignment: Left

Syntax

NLMNLMXN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlmxn32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(MX$1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIMXN Format" on page 176

NLMNLMYR Format

Writes the monetary format of the local expression for Malaysia.

Category: Numeric

Alignment: Left

Syntax

NLMNLMYR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlmyr32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(R1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIMYR Format" on page 177

NLMNLNOK Format

Writes the monetary format of the local expression for Norway.

Category: Numeric

Alignment: Left

Syntax

NLMNLNOK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlnok32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(kr1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNINOK Format" on page 178

NLMNLNZD Format

Writes the monetary format of the local expression for New Zealand.

Category: Numeric

Alignment: Left

Syntax

NLMNLNZD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlnzd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(NZ$1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNINZD Format" on page 179

NLMNLPLN Format

Writes the monetary format of the local expression for Poland.

Category: Numeric

Alignment: Left

Syntax

NLMNLPLN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlpln32.2);
y=put(-1234.56789,dollar32.2)
```

Statements	Results
	`----+----1----+`
`put x=;`	`(PLN1,234.57`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIPLN Format" on page 180

NLMNLRUB Format

Writes the monetary format of the local expression for Russia.

Category: Numeric

Alignment: Left

Syntax

NLMNLRUB*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlrub32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(RUB1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIRUB Format" on page 181

NLMNLSEK Format

Writes the monetary format of the local expression for Sweden.

Category: Numeric

Alignment: Left

Syntax

NLMNLSEK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlsek32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(kr1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNISEK Format" on page 182

NLMNLSGD Format

Writes the monetary format of the local expression for Singapore.

Category: Numeric

Alignment: Left

Syntax

NLMNLSGD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlsgd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(SG$1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNISGD Format" on page 183

NLMNLTHB Format

Writes the monetary format of the local expression for Thailand.

Category: Numeric

Alignment: Left

Syntax

NLMNLTHB*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlthb32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(THB1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNITHB Format" on page 184

NLMNLTRY Format

Writes the monetary format of the local expression for Turkey.

Category: Numeric

Alignment: Left

Syntax

NLMNLTRY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnltry32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(YTL1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNITRY Format" on page 185

NLMNLTWD Format

Writes the monetary format of the local expression for Taiwan.

Category: Numeric

Alignment: Left

Syntax

NLMNLTWD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnltwd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(NT$1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNITWD Format" on page 186

NLMNLUSD Format

Writes the monetary format of the local expression for Puerto Rico and the United States.

Category: Numeric

Alignment: Left

Syntax

NLMNLUSD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlusd32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(US$1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIUSD Format" on page 187

NLMNLZAR Format

Writes the monetary format of the local expression for South Africa.

Category: Numeric

Alignment: Left

Syntax

NLMNLZAR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlzar32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(R1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNIZAR Format" on page 188

NLMNY Format

Writes the monetary format of the local expression in the specified locale using local currency.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLMNY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 9

Range 1–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 0

Range 0–31

Details

The NLMNY*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLMNY*w.d* format writes numeric values by using the currency symbol, the thousands separator, and the decimal separator that is used by the locale.

Note: The NLMNY*w.d* format does not convert currency format. Therefore, the value of the formatted number should equal the currency of the current locale value.

Comparisons

The NLMNY*w.d* and NLMNYI*w.d* formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNYI*w.d* format uses three-letter international currency codes, such as USD, while NLMNY*w.d* format uses local currency symbols, such as $.

The NLMNY*w.d* format is similar to the DOLLAR*w.d* format except that the NLMNY*w.d* format is locale-specific.

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmny32.2);
y=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`($1,234.57)`
`put y=;`	`$-1,234.57`

See Also

Format:

- "NLMNYI Format" on page 227

Informats:

- "NLMNY Informat" on page 529
- "NLMNYI Informat" on page 530

NLMNYI Format

Writes the monetary format of the international expression in the specified locale.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLMNYI*w.d*

Syntax Description

w

specifies the width of the output field.

Default 9

Range 1–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 0

Range 0–31

Details

The NLMNYI*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLMNYI*w.d* format writes numeric values by using the international currency code, and locale-dependent

thousands and decimal separators. The position of international currency code is also locale dependent.

Note: The NLMNYI*w.d* format does not convert currency format. Therefore, the value of the formatted number should equal the currency of the current locale value.

Comparisons

The NLMNY*w.d* and NLMNYI*w.d* formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNYI*w.d* format uses three-letter international currency codes, such as USD, while NLMNY*w.d* format uses local currency symbols, such as $.

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnyi32.2);
y=put(-1234.56789,nlmny32.2);
z=put(-1234.56789,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`(USD1,234.57)`
`put y=;`	`($1,234.57)`
`put z=;`	`$-1,234.57`

See Also

Format:

- "NLMNY Format" on page 226

Informats:

- "NLMNY Informat" on page 529
- "NLMNYI Informat" on page 530

NLNUM Format

Writes the numeric format of the local expression in the specified locale.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLNUM*w.d*

Syntax Description

w

specifies the width of the output field.

Default 6

Range 1–32

d

specifies to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Details

The NLMUM*w.d* format reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLNUM*w.d* format writes numeric values by using the thousands separator and the decimal separator that is used by the locale.

Comparisons

The NLNUM*w.d* format writes the numeric value with locale-dependent thousand and decimal separators. The NLNUMI*w.d* format writes the numeric value with a comma (,) as thousands separator and a period (.) as a decimal separator.

If the *w* or *d* values are not large enough to generate a formatted number, the NLNUM*w.d* format uses an algorithm that prints the thousands-separator characters whenever possible, even if some decimal precision is lost.

Example

```
x=put(-1234356.7891,nlnum32.2);
```

Statements	Results
	`----+----1----+`
`options LOCALE=English_UnitedStates;` `put x=;`	`-1,234,356.79`
`options LOCALE=German_Germany;` `put x=;`	`-1.234.356,79`

See Also

Format:

- "NLNUMI Format" on page 230

Informats:

- "NLNUM Informat" on page 531
- "NLNUMI Informat" on page 533

NLNUMI Format

Writes the numeric format of the international expression in the specified locale.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLNUMI*w.d*

Syntax Description

w

specifies the width of the output field.

Default 6

Range 1–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 0

Range 0–31

Details

The NLNUMI*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLNUMI*w.d* format writes numeric values by using a comma (,) as thousands separator and a period (.) as a decimal separator for all locales.

Comparisons

The NLNUMI*w.d* format writes the numeric data of the international expression in the specified locale. The NLNUMI*w.d* format writes the numeric value with a comma (,) as thousands separator and a period (.) as a decimal separator.

If the *w* or *d* values are not large enough to generate a formatted number, the NLNUM*w.d* format uses an algorithm that prints the thousands-separator characters whenever possible, even if some decimal precision is lost.

Example

```
x=put(-1234356.7891,nlnumi32.2);
```

Statements	Results
	`----+----1----+`
`options LOCALE=English_UnitedStates;` `put x=;`	`-1,234,356.79`
`options LOCALE=German_Germany;` `put x=;`	`-1,234,356.79`

See Also

Format:

- "NLNUM Format" on page 228

Informats:

- "NLNUM Informat" on page 531
- "NLNUMI Informat" on page 533

NLPCT Format

Writes percentage data of the local expression in the specified locale.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLPCT*w.d*

Syntax Description

w

specifies the width of the output field.

Default 6

Range 4–32

d

specifies to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Comparisons

The NLPCT*w.d* format writes percentage data of the local expression in the specified locale. The NLPCT*w.d* format writes the percentage value with locale-dependent thousand and decimal separators. The NLPCTI*w.d* format writes the percentage value with a comma (,) as thousands separator and a period (.) as a decimal separator.

The NLPCT*w.d* format is similar to the PERCENT*w.d* format except the NLPCT*w.d* format is locale-specific.

Example

```
x=put(-12.3456789,nlpct32.2);
y=put(-12.3456789,nlpcti32.2);
z=put(-12.3456789,percent32.2);
```

Statements	Results
	`----+-----------1`

`options LOCALE=English_UnitedStates;`
`put x=;`
`put y=;`
`put z=;`
	`-1,234.57%`
	`-1,234.57%`
	`(  1234.57%)`

`options LOCALE=German_Germany;`
`put x=;`
`put y=;`
`put z=;`
	`-1.234,57%`
	`-1,234.57%`
	`(  1234.57%)`

See Also

Format:

- “NLPCTI Format” on page 232

Informats:

- “NLPCT Informat” on page 534
- “NLPCTI Informat” on page 535

NLPCTI Format

Writes percentage data of the international expression in the specified locale.

Categories: Numeric

CAS

Alignment: Left

Syntax

NLPCTI*w.d*

Syntax Description

w

specifies the width of the output field.

Default 6

Range 4–32

d

specifies to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Comparisons

The NLPCTI*w.d* format writes percentage data of the international expression in the specified locale. The NLPCT*w.d* format writes the percentage value with locale-dependent thousand and decimal separators. The NLPCTI*w.d* format writes the percentage value with a comma (,) as thousands separator and a period (.) as a decimal separator.

The NLPCT*w.d* format is similar to the PERCENT*w.d* format except the NLPCT*w.d* format is locale-specific.

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-12.3456789,nlpcti32.2);
y=put(-12.3456789,percent32.2);
```

Statements	Results
	`----+--------1`
`put x=;`	`-1,234.57%`
`put y=;`	`(  1234.57)`

See Also

Format:

- "NLPCT Format" on page 231

Informats:

- "NLPCT Informat" on page 534
- "NLPCTI Informat" on page 535

NLPCTN Format

Produces percentages, using a minus sign for negative values.

Categories: Numeric
CAS

Alignment: Right

Syntax

NLPCTN*w.d*

Syntax Description

w

specifies the width of the output field.

Default	6
Range	4–32
Tip	The width of the output field must account for the minus sign (–), the percent sign (%), and a trailing blank, whether the number is negative or positive.

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Default	0
Range	0–31
Requirement	must be less than *w*

Details

The NLPCTN*w.d* format multiplies negative values by 100, adds a minus sign to the beginning of the value, and adds a percent sign (%) to the end of the formatted value.

Example

```
x=-0.02;
```

Statements	Results

put x nlpctn6.;	x=-2%
put x percentn6.;	x=-2%

NLPCTP Format

Writes locale-specific numeric values as percentages.

Categories: Numeric
CAS

Alignment: Right

Syntax

NLPCTP*w.d*

Syntax Description

w

specifies the width of the output field.

Default	6
Range	4–32
Tip	The width of the output field must account for the percent sign (%).

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional. The thousands separator and decimal symbol for the NLPCTP format is locale-specific.

Default	0
Range	0–31
Requirement	must be less than *w*

Details

The NLPCTP*w.d* format multiplies values by 100, formats them, and adds a percent sign (%) to the end of the formatted value. The NLPCTP*w.d* format is similar to the The PERCENT*w.d* format except that the thousands separator and decimal symbol for the NLPCTP*w.d* format is locale-specific.

Example

```
x=-0.02;
```

Statements	Results

put x nlpctp6.;	–2%
put x percent6.;	(2%)

NLPVALUE Format

Writes p-values of the local expression in the specified locale.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLPVALUE*w.d*

Syntax Description

w

specifies the width of the output field.

Default 6

Range 3–32

d

specifies to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 4

Range 1–30

Example

This example uses the german_Germany locale option.

Statements:

```
options locale=german_germany;
data _null_;
   put "+--- nlpvalue min=3 default=6 max=32 ---+";
   x=0.1248;
   put x= +5 x pvalue.    +5 x nlpvalue.;
   put x= +5 x pvalue3.1  +5 x nlpvalue3.1;
   put x= +5 x pvalue20.2 +5 x nlpvalue20.2;
   put x= +5 x pvalue32.3 +5 x nlpvalue32.3;
run;
```

Results:

```
+--- nlpvalue min=3 default=6 max=32 ---+
x=0.1248   0.1248     0,1248
x=0.1248   0.1     0,1
x=0.1248                  0.12                     0,12
x=0.1248                              0.125                          0,125
```

See Also

Format:

- "PVALUE Format" in *SAS Viya Formats and Informats: Reference*

NLSTRMON Format

Writes the month name in the specified locale.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLSTRMON*w.d*

Syntax Description

w

specifies the width of the output field

Default 20

Range 1-200

d

specifies the following:

- 00000001: write abbreviated form.
- 00000010: write capitalized form.

Default 0

Range 0-3

Details

The NLSTRMONw.d format writes a SAS value, 1–12 as the name-of-the-month in the specified locale. The following examples use the English_UnitedStates locale.

- 1 = the first month (January)
- 2 = the second month (February)

- 3 = the third month (March)
- 4 = the fourth month (April)
- 5 = the fifth month (May)
- 6 = the sixth month (June)
- 7 = the seventh month (July)
- 8 = the eight month (August)
- 9 = the ninth month (September)
- 10 = the tenth month (October)
- 11 = the eleventh month (November)
- 12 = the twelfth month (December)

Example

This example uses the English_UnitedStates session encoding.

Statements	Results
Data _null_ ; monnum = 1 ; /* January=1, December=12 */ put monnum NLSTRMON20. ; put monnum NLSTRMON20.1; /* decimal .1 specified use abbreviation. */ put monnum NLSTRMON20.2; put monnum NLSTRMON20.3; run;	January Jan JANUARY JAN

NLSTRQTR Format

Writes a numeric value as the quarter-of-the-year in the specified locale.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLSTRQTR*w.d*

Syntax Description

w
specifies the width of the output field

Default 20

Range 1–200

d

specifies the following:

- 00000001: write abbreviated form.
- 00000010: write capitalized form.

Default 0

Range 0–3

Details

The NLSTRQTRw.d format writes a SAS value, 1–4 as the name-of-the-quarter for the year in the specified locale. The following examples use the English_UnitedStates locale.

- 1 = 1st quarter
- 2 = 2nd quarter
- 3 = 3rd quarter
- 4 = 4th quarter

Example

This example uses the English_UnitedStates session encoding.

Statements	Results
Data _null_ ; qtrnum = 1 ; /* January=1, December=12 */ put qtrnum NLSTRQTR20. ; put qtrnum NLSTRQTR20.1; /* decimal .1 specified use abbreviation. */ put qtrnum NLSTRQTR20.2; put qtrnum NLSTRQTR20.3; run;	1st quarter Q1 1ST QUARTER Q1

NLSTRWK Format

Writes a numeric value as the day-of-the-week in the specified locale.

Categories: Numeric
CAS

Alignment: Left

Syntax

NLSTRWK*w.d*

Syntax Description

w

specifies the width of the output field

Default 20

Range 1–200

d

specifies the following:

- 00000001: write abbreviated form.
- 00000010: write capitalized form.

Default 0

Range 0–3

Details

The NLSTRWKw.d format writes a SAS value, 1–7 as the name-of-the-week in the specified locale. The following examples use the English_UnitedStates locale.

- 1 = First day-of-week (Monday)
- 2 = Second day-of-week (Tuesday)
- 3 = Third day-of-week (Wednesday)
- 4 = Fourth day-of-week (Thursday)
- 5 = Fifth day-of-week (Friday)
- 6 = Sixth day-of-week (Saturday)
- 7 = Seventh day-of-week (Sunday)

Example

This example uses the English_UnitedStates session encoding.

Statements	Results
Data _null_ ; wknum = 1 ; /* Sunday=1, Saturday=7 */ put wknum NLSTRWK20. ; put wknum NLSTRWK20.1; /* decimal .1 specified use abbreviation. */ put wknum NLSTRWK20.2; put wknum NLSTRWK20.3; run;	Sunday Sun SUNDAY SUN

NLTIMAP Format

Converts a SAS time value to the time value of a specified locale, and then writes the value as a time value with a.m. or p.m. NLTIMAP also converts SAS date-time values.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLTIMAP*w*.

Syntax Description

w
specifies the width of the output field.

Default 10

Range 4–200

Details

The NLTIMAP format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths on page 60 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

Comparisons

The NLTIMAP*w*. format is similar to the TIMEAMPM*w*. format except that the NLTIMAP*w*. format is locale-specific.

Example

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

Statements	Results
	`----+----1----+`
`options locale=English_UnitedStates;` `put time nltimap.;`	`4:24:43 PM`
`options locale=German_Germany;` `put time nltimap14.;`	`4:24:43 nachm`

See Also

Format:

- "NLTIME Format" on page 242

NLTIME Format

Converts a SAS time value to the time value of the specified locale, and then writes the value as a time value. NLTIME also converts SAS date-time values.

Categories: Date and Time
CAS

Alignment: Left

Syntax

NLTIME*w*.

Syntax Description

w
specifies the width of the input field.

Default 20

Range 10–200

Comparisons

The NLTIME*w*. format is similar to the TIME*w*. format except that the NLTIME*w*. format is locale-specific.

Example

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

Statements	Results
	`----+----1----+`
`options locale=English_UnitedStates;` `put time nltime.;`	`4:24:43`
`options locale=German_Germany;` `put time nltime.;`	`16.24`

See Also

Format:

- "NLTIMAP Format" on page 241

$UCS2B Format

Processes a character string that is in the encoding of the current SAS session, and then writes the character string in big-endian, 16-bit, UCS2, Unicode encoding.

Category: Character

Alignment: Left

Syntax

$UCS2B*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Details

The $UCS2B*w*. format writes a character string in big-endian, 16-bit, UCS2 (universal character set code in two octets), Unicode encoding. It processes character strings that are in the encoding of the current SAS session.

Comparisons

The $UCS2B*w*. format performs processing that is the opposite of the $UCS2BE*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1`

Statements	Results

`data _null_;`
`x='大';`
`y=put(x,$ucs2b2.);`
`put y $hex.;`
`run;`
	`5927`

See Also

Formats:

- "$UCS2L Format" on page 245
- "$UCS2X Format" on page 248
- "$UTF8X Format" on page 264
- "$UCS2BE Format" on page 244

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2BE Informat" on page 539
- "$UCS2L Informat" on page 540
- "$UCS2X Informat" on page 543
- "$UTF8X Informat" on page 557

$UCS2BE Format

Processes a character string that is in big-endian, 16-bit, UCS2, Unicode encoding, and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UCS2BE*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Details

The $UCS2BE*w*. format writes a character string in the encoding of the current SAS session. It processes character strings that are in big-endian, 16-bit, UCS2 (universal character set code in two octets), Unicode encoding.

Comparisons

The $UCS2BE*w*. format performs processing that is the opposite of the $UCS2B*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1`
`x ='592700410042'x;` `put x $ucs2be.;`	大AB

See Also

Formats:

- "$UCS2B Format" on page 243

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2BE Informat" on page 539

$UCS2L Format

Processes a character string that is in the encoding of the current SAS session, and then writes the character string in little-endian, 16-bit, UCS2, Unicode encoding.

Category: Character

Alignment: Left

Syntax

$UCS2L*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default	8
Range	2–32767

Details

The $UCS2L*w*. format writes a character string in little-endian, 16-bit, UCS2 (universal character set code in two octets), Unicode encoding. It processes character strings that are in the encoding of the current SAS session.

Comparisons

The $UCS2L*w*. format performs processing that is the opposite of the $UCS2LE*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`

`data _null_;`
`x ='大';`
`y=put(x,$ucs2l2.);`
`put y $hex.;`
`run;`
	`2759`

See Also

Formats:

- "$UCS2B Format" on page 243
- "$UCS2LE Format" on page 247
- "$UCS2X Format" on page 248
- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UCS2LE Informat" on page 542
- "$UCS2X Informat" on page 543
- "$UTF8X Informat" on page 557

$UCS2LE Format

Processes a character string that is in little-endian, 16-bit, UCS2, Unicode encoding, and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UCS2LE*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Details

The $UCS2LE*w*. format writes a character string in the encoding of the current SAS session. It processes character strings that are in little-endian, 16-bit, UCS2 (universal character set code in two octets), Unicode encoding.

Comparisons

The $UCS2LE*w*. format performs processing that is the opposite of the $UCS2L*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`
`x ='275941004200'x;` `put x $ucs2le.;`	大**AB**

See Also

Format:

- "$UCS2L Format" on page 245

Informats:

- "$UCS2L Informat" on page 540
- "$UCS2LE Informat" on page 542

$UCS2X Format

Processes a character string that is in the encoding of the current SAS session, and then writes the character string in native-endian, 16-bit, UCS2, Unicode encoding.

Category: Character

Alignment: Left

Syntax

$UCS2X*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Details

The $UCS2X*w*. format writes a character string in 16-bit, UCS2 (universal character set code in two octets), Unicode encoding, by using byte order that is native to the operating environment.

Comparisons

The $UCS2X*w*. format performs processing that is the opposite of the $UCS2XE*w*. format. If you are exchanging data within the same operating environment, use the $UCS2X*w*. format. If you are exchanging data with a different operating environment, use the $UCS2B*w*. format or $UCS2L*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`
`x ='大';` `put x $ucs2x2.;`	`'5927'x (binary) or '2759'x (little endian)`

See Also

Formats:

- "$UCS2B Format" on page 243
- "$UCS2XE Format" on page 249
- "$UCS2L Format" on page 245
- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UCS2X Informat" on page 543
- "$UCS2XE Informat" on page 544
- "$UTF8X Informat" on page 557

$UCS2XE Format

Processes a character string that is in native-endian, 16-bit, UCS2, Unicode encoding, and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UCS2XE*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2-32767

Details

The $UCS2XE*w*. format writes a character string in the encoding of the current SAS session. It processes character strings that are in native-endian, 16-bit, UCS2 (universal character set code in two octets), Unicode encoding.

Comparisons

The $UCS2XE*w*. format performs processing that is the opposite of the $UCS2X*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`
`x ='e5a4a7'x;` `put x $utf8xe10.;`	大

See Also

Format:

- "$UCS2X Format" on page 248

Informats:

- "$UCS2X Informat" on page 543
- "$UCS2XE Informat" on page 544

$UCS4B Format

Processes a character string that is in the encoding of the current SAS session, and then writes the character string in big-endian, 32-bit, UCS4, Unicode encoding.

Category: Character

Alignment: Left

Syntax

$UCS4B*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default 8

Range 4–32767

Details

The $UCS4B*w*. format writes a character string in big-endian, 32-bit, UCS4 (universal character set code in four octets), Unicode encoding. It processes character strings that are in the encoding of the current SAS session.

Comparisons

The $UCS4B*w*. format performs processing that is the opposite of the $UCS4BE*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`
`x ='大';` `put x $ucs4b4.;`	`'00005927'x (binary)`

See Also

Formats:

- "$UCS2L Format" on page 245
- "$UCS2X Format" on page 248
- "$UCS4BE Format" on page 251
- "$UCS4L Format" on page 252
- "$UCS4X Format" on page 255
- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UCS2X Informat" on page 543
- "$UCS4B Informat" on page 545
- "$UCS4L Informat" on page 546
- "$UCS4X Informat" on page 547
- "$UTF8X Informat" on page 557

$UCS4BE Format

Processes a character string that is in big-endian, 32-bit, UCS4, Unicode encoding, and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UCS4BE*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default 8

Range 4–32767

Details

The $UCS4BE*w*. format writes a character string in the encoding of the current SAS session. It processes character strings that are in big-endian, 32-bit, UCS4 (universal character set code in four octets), Unicode encoding.

Comparisons

The $UCS4BE*w*. format performs processing that is the opposite of the $UCS4B*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`
`x ='00005927000004100000042'x;` `put x $ucs4be.;`	大 **AB**

See Also

Format:

- "$UCS4B Format" on page 250

Informat:

- "$UCS4B Informat" on page 545

$UCS4L Format

Processes a character string that is in the encoding of the current SAS session, and then writes the character string in little-endian, 32-bit, UCS4, Unicode encoding.

Category: Character

Alignment: Left

Syntax

$UCS4L*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default 8

Range 4–32767

Details

The $UCS4L*w*. format writes a character string in little-endian, 32-bit, UCS4 (universal character set code in four octets), Unicode encoding. It processes character strings that are in the encoding of the current SAS session.

Comparisons

The $UCS4L*w*. format performs processing that is the opposite of the $UCS4LE*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`

`data _null_;`
`x = '大';`
`y=put(x,$ucs4l4.);`
`put y $hex.;`
`run;`
	`2759`

See Also

Formats:

- "$UCS2B Format" on page 243
- "$UCS2X Format" on page 248
- "$UCS4B Format" on page 250
- "$UCS4LE Format" on page 254
- "$UCS4X Format" on page 255

- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UCS2X Informat" on page 543
- "$UCS4B Informat" on page 545
- "$UCS4L Informat" on page 546
- "$UCS4X Informat" on page 547
- "$UTF8X Informat" on page 557

$UCS4LE Format

Processes a character string that is in little-endian, 32-bit, UCS4, Unicode encoding, and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UCS4LE*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default	8
Range	4–32767

Details

The $UCS4LE*w*. format writes a character string in the encoding of the current SAS session. It processes character strings that are in little-endian, 32-bit, UCS4 (universal character set code in four octets), Unicode encoding.

Comparisons

The $UCS4LE*w*. format performs processing that is the opposite of the $UCS4L*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`
`x ='275900004100000042000000'x;` `put x $ucs4le.;`	大AB

See Also

Format:

- "$UCS4L Format" on page 252

Informat:

- "$UCS4L Informat" on page 546

$UCS4X Format

Processes a character string that is in the encoding of the current SAS session, and then writes the character string in native-endian, 32-bit, UCS4, Unicode encoding.

Category: Character

Alignment: Left

Syntax

$UCS4X*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default 8

Range 4–32767

Details

The $UCS4X*w*. format writes a character string in 32-bit, UCS4 (universal character set code in two octets), Unicode encoding, by using byte order that is native to the operating environment.

Comparisons

The $UCS4X*w*. format performs processing that is the opposite of the $UCS4XE*w*. format. If you are exchanging data within the same operating environment, use the $UCS4X*w*. format. If you are exchanging data with a different operating environment, use the $UCS4B*w*. format or $UCS4L*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1`
`x = '大';` `put x $ucs4x4.;`	`'00005927'x (binary) or '27590000'x (little endian)`

See Also

Formats:

- "$UCS2L Format" on page 245
- "$UCS4XE Format" on page 256
- "$UCS2X Format" on page 248
- "$UCS4B Format" on page 250
- "$UCS4L Format" on page 252
- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UCS2X Informat" on page 543
- "$UCS4B Informat" on page 545
- "$UCS4B Format" on page 250
- "$UCS4L Informat" on page 546
- "$UCS4X Informat" on page 547
- "$UTF8X Informat" on page 557

$UCS4XE Format

Processes a character string that is in native-endian, 32-bit, UCS4, Unicode encoding, and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UCS4XE*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default 8

Range 4–32767

Details

The $UCS4XE*w*. format writes a character string in the encoding of the current SAS session. It processes character strings that are in native-endian, 32-bit, UCS4 (universal character set code in four octets), Unicode encoding.

Comparisons

The $UCS4XE*w*. format performs processing that is the opposite of the $UCS4X*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1`
`x ='27590000410000004200000 0'x;` `put x $ucs4be4.;`	大**AB (little endian)**

See Also

Format:

- "$UCS4X Format" on page 255

Informat:

- "$UCS4X Informat" on page 547

$UESC Format

Processes a character string that is encoded in the current SAS session, and then writes the character string in Unicode escape (UESC) representation.

Category: Character

Alignment: Left

Syntax

$UESC*w*.

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Details

If the characters are not available on all operating environments, for example, 0–9, a–z, A–Z, they must be represented in UESC. $UESC*w*. can be nested.

Comparisons

The $UESC*w*. format performs processing that is opposite of the $UESCE*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1----+----2`

`x='大' ;`
`y='u5927'`
`z='uu5927';`
`put x = $uesc10. ;`
`put y = $uesc10. ;`
`put z = $uesc10. ;`
	¥u5927
	¥uu5927
	¥uuu5927

See Also

Formats:

- “$UESCE Format” on page 259

Informats:

- “$UESC Informat” on page 549
- “$UESCE Informat” on page 550

$UESCE Format

Processes a character string that is in Unicode escape (UESC) representation, and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UESCE*w*.

Syntax Description

w

specifies the width of the output field.

Default 8

Range 1–32767

Details

If the data is not supported by the encoding of the current SAS session, the data remains in UESC.

Comparisons

The $UESCE*w*. format performs processing that is the opposite of the $UESC*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1----+----2`
`x=put('¥u5927',$uesce10.) ;`	`x=¥u5927`
`x=put('¥uu5927',$uesce10.) ;`	`x=¥uu5927`
`x=put('¥uuu5927',$uesce10.) ;`	`x=¥uuu5927`

See Also

Format:

- "$UESC Format" on page 257

Informats:

- "$UESC Informat" on page 549
- "$UESCE Informat" on page 550

$UNCR Format

Processes a character string that is encoded in the current SAS session, and then writes the character string in numeric character representation (NCR).

Category: Character

Alignment: Left

Syntax

$UNCR*w*.

Syntax Description

w

specifies the width of the output field.

Default 8

Range 1–32767

Comparisons

The $UNCR*w*. format performs processing that is the opposite of the $UNCRE*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1----+----2`

`x='91E5'x ; /* Japanese '大' in Shift-JIS */`
`y='abc';`
`put x $uncr10.;`
`put y $uncr10.;`
	`&#22823`
	`abc`

See Also

Formats:

- "$UNCRE Format" on page 261

Informats:

- "$UNCR Informat" on page 551
- "$UNCRE Informat" on page 552

$UNCRE Format

Processes a character string that is in numeric character representation (NCR), and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UNCRE*w*.

Syntax Description

w

specifies the width of the output field.

Default 8

Range 1–32767

Details

National characters should be represented in NCR.

Comparisons

The $UNCRE*w*. format performs processing that is the opposite of the $UNCR*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1`
`x='大abc';` `put x $uncr10.;`	大abc

See Also

Formats:

- "$UNCR Format" on page 260

Informats:

- "$UNCR Informat" on page 551
- "$UNCRE Informat" on page 552

$UPAREN Format

Processes a character string that is encoded in the current SAS session, and then writes the character string in Unicode parenthesis (UPAREN) representation.

Category: Character

Alignment: Left

Syntax

$UPAREN*w.*

Syntax Description

w

specifies the width of the output field.

Default 8

Range 7–32767

Details

The character string is encoded with parentheses and Unicode hexadecimal representation.

Comparisons

The $UPAREN*w*. format performs processing that is the opposite of the $UPARENE*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1----+----2----+----3----+`

`x='大';`
`y='abc3';`
`put x $uparen7.;`
`put y $uparen28.;`
	`<u5927>`
	`<u0061> <u0062> <u0063> <u0033>`

See Also

Formats:

- "$UPARENE Format" on page 263

Informats:

- "$UPAREN Informat" on page 553
- "$UPARENE Informat" on page 555

$UPARENE Format

Processes a character string that is in Unicode parenthesis (UPAREN), and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UPARENE*w.*

Syntax Description

w

specifies the width of the output field.

Default 8

Range 1–32767

Comparisons

The $UPARENE*w*. format performs processing that is the opposite of the $UPAREN*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+`
`x='<u0061><u0062><u0063><u0033>';` `put x $uparene4.;`	`abc3`

See Also

Formats:

- "$UPAREN Format" on page 262

Informats:

- "$UPAREN Informat" on page 553
- "$UPARENE Informat" on page 555

$UTF8X Format

Processes a character string that is in the encoding of the current SAS session, and then writes the character string in universal transformation format (UTF-8) encoding.

Category: Character

Alignment: Left

Syntax

$UTF8X*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to include all of the characters in the variable. The width of the characters is dependent on the code point value of the individual characters.

Default 8

Range 1–32767

Comparisons

This example uses UTF-8 encoding.

Statements	Results
	`----+----1`
`x ='91E5'x; ; /* Japanese '大' in Shift-JIS */` `put x $utf8x10.;`	`x='e5a4a7'x`

See Also

Formats:

- "$UCS2B Format" on page 243

- "$UCS2L Format" on page 245
- "$UCS2X Format" on page 248

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UCS2X Informat" on page 543

$UTF8XE Format

Processes a character string that is in universal transformation format (UTF-8), and then writes the character string in the encoding of the current SAS session.

Category: Character

Alignment: Left

Syntax

$UTF8XE*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to include all of the characters in the variable. The width of the characters is dependent on the code point value of the individual characters.

Default 8

Range 1–32767

Comparisons

The $UTF8XE*w*. format performs processing that is the opposite to the $UTF8X*w*. format.

Example

This example uses UTF-8 encoding.

Statements	Results
	`----+----1`
x = unicode('u5927'); `put x $utf8xe10.;`	大

See Also

Formats:

- "$UTF8X Format" on page 264

Informats:

- "$UTF8X Informat" on page 557

WEEKU Format

Writes a week number in decimal format by using the U algorithm.

Category: Date and Time

Alignment: Left

Syntax

WEEKU*w*.

Syntax Description

w

specifies the width of the output field.

Default 11

Range 2–200

Details

The WEEKU*w*. format writes a week-number format. The WEEKU*w*. format writes the various formats depending on the specified width. Algorithm U calculates the SAS date value by using the number of the week within the year (Sunday is considered the first day of the week). The number-of-the-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

For more information about widths, formats, and examples see the following table:

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	03W01
7-8	yyWwwdd	03W0101
9-10	yyyyWwwdd	2003W0101
11-200	yyyy-Www-dd	2003-W01-01

Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53, with weeks that begin on a Monday and week 1 of the year including both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. The WEEKW*w*. format writes the week number of the year as a decimal number in the range 00–53, with Monday as the first day of week 1. The WEEKU*w*. format writes the week number of the year (Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading zero.

Example

```
sasdate = '01JAN2003'd;
```

Statements	Results
	`----+----1----+`

`v=put(sasdate,weeku3.);`
`w=put(sasdate,weeku5.);`
`x=put(sasdate,weeku7.);`
`y=put(sasdate,weeku9.);`
`z=put(sasdate,weeku11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	`W00`
	`03W00`
	`03W0004`
	`2003W0004`
	`2003-W00-04`

See Also

Formats:

- "WEEKV Format" on page 267
- "WEEKW Format" on page 269

WEEKV Format

Writes a week number in decimal format by using the V algorithm.

Category: Date and Time

Alignment: Left

Syntax

WEEKV*w*.

Syntax Description

w

specifies the width of the output field.

Default 11

Range 2–200

Details

The WEEKV*w*. format writes the various formats depending on the specified width. Algorithm V calculates the SAS date value, and the number-of-the-week value is represented as a decimal number in the range 01–53, with a leading zero and maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. For example, the fifth week of the year would be represented as 06.

For more information about widths, formats, and examples see the following table:

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	03W01
7-8	yyWwwdd	03W0101
9-10	yyyyWwwdd	2003W0101
11-200	yyyy-Www-dd	2003-W01-01

Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53, with weeks that begin on a Monday and week 1 of the year including both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. The WEEKW*w*. format writes the week number of the year as a decimal number in the range 00–53, with Monday as the first day of week 1. The WEEKU*w*. format writes the week number of the year (Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading zero.

Example

```
sasdate='01JAN2003'd;
```

Statements	Results
	`----+----1----+`

Statements	Results
`v=put(sasdate,weekv3.);`	`W01`
`w=put(sasdate,weekv5.);`	`03W01`
`x=put(sasdate,weekv7.);`	`03W0103`
`y=put(sasdate,weekv9.);`	`2003W0103`
`z=put(sasdate,weekv11.);`	`2003-W01-03`
`put v;`	
`put w;`	
`put x;`	
`put y;`	
`put z;`	

See Also

Formats:

- "WEEKU Format" on page 266
- "WEEKW Format" on page 269

WEEKW Format

Writes a week number in decimal format by using the W algorithm.

Category: Date and Time

Alignment: Left

Syntax

WEEKW*w*.

Syntax Description

w

specifies the width of the output field.

Default 11

Range 2–200

Details

The WEEKW*w*. format writes the various formats depending on the specified width. Algorithm W calculates the SAS date value using the number of the week within the year (Monday is considered the first day of the week). The number-of-the-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

For more information about widths, formats, and examples see the following table:

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	03W01
7-8	yyWwwdd	03W0101
9-10	yyyyWwwdd	2003W0101
11-200	yyyy-Www-dd	2003-W01-01

Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53. Weeks beginning on a Monday and on week 1 of the year include both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. The WEEKW*w*. format writes the week number of the year as a decimal number in the range 00–53, with Monday as the first day of week 1. The WEEKU*w*. format writes the week number of the year (Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading zero.

Example

```
sasdate = '01JAN2003'd;
```

Statements	Results
	`----+----1----+`

`v=put(sasdate,weekw3.);`
`w=put(sasdate,weekw5.);`
`x=put(sasdate,weekw7.);`
`y=put(sasdate,weekw9.);`
`z=put(sasdate,weekw11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	`W03`
	`03W03`
	`03W0003`
	`2003W0003`
	`2003-W00-03`

See Also

Formats:

- “WEEKU Format” on page 266
- “WEEKV Format” on page 267

YEN Format

Writes numeric values with yen signs, commas, and decimal points.

Categories: Numeric
CAS

Alignment: Right

Syntax

YEN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 8

Range 1–32

d

specifies the number of digits to the right of the decimal point in the numeric value.

Default 0

Range 0–31

Details

The YEN*w.d* format writes numeric values with a leading yen sign and with a comma that separates every three digits of each value.

The hexadecimal representation of the code for the yen sign character is 5B on EBCDIC systems and 5C on ASCII systems. The monetary character these codes represent might be different in other countries.

Example

```
put cost yen10.2;
```

```
data _null_;
   value=1254.71;
   put value yen10.2;
  run;
```

Cost	Result
	----+----1
1254.71	¥1,254.71

YYWEEKU Format

Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.

Categories: Date and Time
CAS

Alignment: Left

Syntax

YYWEEKU*w*.

Syntax Description

w
specifies the width of the output field.

Default 7

Range 2-8

Details

The YYWEEKU*w*. format writes a week-number format. The YYWEEKU*w*. format writes the various formats depending on the specified width. Algorithm U calculates the SAS date value by using the number of the week within the year (Sunday is considered the first day of the week).

For more information about widths, formats, and examples see the following table:

Widths	Formats	Examples
3-4	Www	W01
5-6	yyWww	07W01
7	yyyyWww	2007W01
8	yyyy-Www	2007-W01
9-above	invalid	invalid

Comparisons

The YYWEEKU*w*. format is similar to the WEEKU*w*. format except that the YYWEEKU*w*. format does not specify the day-of-week information. Also, the YYWEEKU*w*. format does not accept any width that is greater than 8.

Example

```
sasdate = '01JAN2007'd;
```

Statements	Results
	----+----1----+

u=put(sasdate,yyweeku3.);
v=put(sasdate,yyweeku4.);
w=put(sasdate,yyweeku5.);
x=put(sasdate,yyweeku6.);
y=put(sasdate,yyweeku7.);
z=put(sasdate,yyweeku8.);
put u;
put v;
put w;
put x;
put y;
put z;
	W00
	W00
	07W00
	07W00
	2007W00
	2007-W00

See Also

Format:

- "WEEKU Format" on page 266

YYWEEKV Format

Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.

Categories: Date and Time
CAS

Alignment: Left

Syntax

YYWEEKV*w*.

Syntax Description

w
specifies the width of the output field.

Default 7

Range 2–8

Details

The YYWEEKV*w*. format writes the various formats depending on the specified width. Algorithm V calculates the SAS date value, and the number-of-the-week value is

represented as a decimal number in the range 01–53, with a leading zero and maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. For example, the fifth week of the year would be represented as 06.

For more information about widths, formats, and examples see the following table:

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	07W01
7	yyyyWww	2007W01
8	yyyy-Www	2007-W01
9-above	invalid	invalid

Comparisons

The YYWEEKV*w*. format is similar to the WEEKV*w*. format except that the YYWEEKV*w*. format does not specify the day-of-week information. Also, the YYWEEKV*w*. format does not accept a width that is greater than 8.

Example

```
sasdate = '01JAN2007'd;
```

Statements	Results
	`----+----1----+`

`u=put(sasdate,yyweekv3.);`
`v=put(sasdate,yyweekv4.);`
`w=put(sasdate,yyweekv5.);`
`x=put(sasdate,yyweekv6.);`
`y=put(sasdate,yyweekv7.);`
`z=put(sasdate,yyweekv8.);`
`put u;`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	`W01`
	`W01`
	`07W01`
	`07W01`
	`2007W01`
	`2007-W01`

See Also

Format:

- “WEEKV Format” on page 267

YYWEEKW Format

Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.

Categories: Date and Time
CAS

Alignment: Left

Syntax

YYWEEKW*w*.

Syntax Description

w
specifies the width of the output field.

Default 7

Range 2–8

Details

The YYWEEKW*w*. format writes the various formats depending on the specified width. Algorithm W calculates the SAS date value using the number of the week within the year.

For more information about widths, formats, and examples see the following table:

Widths	Formats	Examples
3-4	Www	W01
5-6	yyWww	07W01
7	yyyyWww	2007W01
8	yyyy-Www	2007-W01
9-above	invalid	invalid

Comparisons

The YYWEEKW*w*. format is similar to the WEEKW*w*. format except that the YYWEEKW*w*. format does not specify the day-of-week information. Also, the YYWEEKW*w*. format does not accept any width that is greater than 8.

Example

```
sasdate = '01JAN2007'd
```

Statements	Results
	`----+----1----+`

`u=put(sasdate,yyweekw3.);`
`v=put(sasdate,yyweekw4.);`
`w=put(sasdate,yyweekw5.);`
`x=put(sasdate,yyweekw6.);`
`y=put(sasdate,yyweekw7.);`
`z=put(sasdate,yyweekw8.);`
`put u;`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	`W01`
	`W01`
	`07W01`
	`07W01`
	`2007W01`
	`2007-W01`

See Also

Format:

- “WEEKW Format” on page 269

Part 5

Functions for NLS

Chapter 10

Internationalization Compatibility for SAS String Functions

Internationalization Compatibility for SAS String Functions

SAS provides string functions and CALL routines that enable you to easily manipulate your character data. Many of the original SAS string functions assume that the size of one character is always 1 byte. This process works well for data in a single-byte character set (SBCS). However, when some of these functions and CALL routines are used with data in a double-byte character set (DBCS) or a multi-byte character set (MBCS) such as UTF-8, the data is often handled improperly, and the string functions produce incorrect results.

To solve this problem, SAS introduced a set of string functions and CALL routines, called *K functions*, for those string manipulations where DBCS or MBCS data must be handled carefully. The K functions do not make assumptions about the size of a character string. The Table 10.1 on page 280 shows the level of I18N compatibility for each SAS string function. I18N is the abbreviation for internationalization. Compatibility indicates whether a program using a particular string function can be adapted to different languages and locales without program changes.

It is important to understand the difference between byte-based offset-length and character-based offset-length in order to use the K functions properly.

A byte-based offset assumes that the starting position specified for a character is the byte position of that character in the string. For SBCS data, where one character is always 1 byte in length, you can assume that the second character in the string begins in byte 2. However, if the data in the string is multi-byte data, the data in byte 2 might be one of the following, depending on the data and the encoding of the data:

- the second character in the string
- the second byte of a 2-byte character
- the first byte of the first character in the string.

A byte-based length represents the number of bytes in the string.

A character-based offset assumes that the position specified is the position of the character in the string. For all encodings, a character-based position of 2 is always the second character in the string. You cannot assume that you know the size of the characters in the string.

A character-based length represents the number of characters in the string.

K functions use a character-based offset or length, which does not take into consideration the byte position of the character in the string. For SAS Viya, you should consider using the K functions to process your UTF-8 data.

Here is an example of the functionality of a SAS string function and its corresponding K function.

In the UTF8 encoding, the STR='E282AC313233'x string has 6 bytes and represents four characters. Table 10.1 on page 280

Table 10.1 *String Representation in UTF-8*

Hexadecimal Representation	E282AC	31	32	33
	€	1	2	3

The following example uses this same hexadecimal string in the SUBSTR function, which returns an unexpected value in UTF-8. The expected returned string would be the first character, the euro symbol. However, the result is an invalid character. The LENGTH function is byte-oriented, so it reads the individual bytes of data. The SUBSTR function returns the first byte of the first character, which is an incomplete value because the first character is encoded on 3 bytes.

Table 10.2 *Code Example Using Regular Functions*

Statements	Results

`/* SAS program to submit in a UTF-8 SAS session */`
`data _null_ ;`
`  str= "€123" ;`
`  s=substr(str,1,1) ;`
`  sl=length(s);`
`  l=length(str) ;`
`  put str= $hex16. /s= sl= / s= $hex. /l=;`
`run ;`
	`str=E282AC313233`
	`s=  sl=1 s=E22020202020`
	`l=6`

The code in Table 10.3 on page 281 solves this issue by substituting the SUBSTR() function, which only works on single-byte data, with the KSUBSTR() function, which works on single– and multi-byte data. Direct substitutions can also be made for other string functions, such as changing the LENGTH() or INDEX() functions to use the KLENGTH() or KINDEX() functions. Because the two functions read the data differently, you get different results when running this code in a multi-byte environment.

Table 10.3 *Code Example Using K Functions*

Statements	Results

`/* SAS program to submit in a UTF-8 SAS session */`
`data _null_ ;`
`  str= "€123" ;`
`  s=ksubstr(str,1,1) ;`
`  sl=klength(s);`
`  l=klength(str) ;`
`  put str= $hex16. /s= sl= / s= $hex. /l=;`
`run ;`
	`str=E282AC313233`
	`s=€`
	`l=4`

Before replacing all of the original SAS string-handling functions with K functions, examine your SAS program. If the string function processes data that contains only single-byte characters, K functions are not necessary. For example, strings containing XML tags do not require the use of K functions. Knowing the character data that is in your SAS programs and how it is processed can save unnecessary updates to your SAS code. The processing of binary data is not supported by the string-handling K functions, which expect strings to match the current session encoding. UTF-8 is the only SAS session encoding supported by SAS Viya.

String functions are assigned I18N levels depending on whether the functions can process MBCS or SBCS data . Here are descriptions of the levels:

I18N Level 0
: This function is designed for SBCS data. Do not use this function to process MBCS data.

I18N Level 1
: This function should be avoided, if possible, if you are processing MBCS data. The I18N Level 1 functions might not work correctly with DBCS or MBCS encodings under certain circumstances.

I18N Level 2
: This function can be used for SBCS and MBCS (UTF-8) data.

Table 10.4 *SAS String Functions*

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"ANORM420 Function" (p. 310)	Returns a normalized string from an input string encoded in EBCDIC420.		X	
"ANYALNUM Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for an alphanumeric character, and returns the first position at which the character is found.		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"ANYALPHA Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for an alphabetic character, and returns the first position at which the character is found.		X	
"ANYCNTRL Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a control character, and returns the first position at which that character is found.			X
"ANYDIGIT Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a digit, and returns the first position at which the digit is found.			X
"ANYFIRST Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a character that is valid as the first character in a SAS variable name under **VALIDVARNAME=V7**, and returns the first position at which that character is found.			X
"ANYGRAPH Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a graphical character, and returns the first position at which that character is found.		X	
"ANYLOWER Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a lowercase letter, and returns the first position at which the letter is found.		X	
"ANYNAME Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a character that is valid in a SAS variable name under VALIDVARNAME= V7, and returns the first position at which that character is found.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"ANYPRINT Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a printable character, and returns the first position at which that character is found.		X	
"ANYPUNCT Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a punctuation character, and returns the first position at which that character is found.		X	
"ANYSPACE Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a white-space character (blank, horizontal and vertical tab, carriage return, line feed, and form feed). Returns the first position at which that character is found.		X	
"ANYUPPER Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for an uppercase letter, and returns the first position at which the letter is found.		X	
"ANYXDIGIT Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a hexadecimal character that represents a digit, and returns the first position at which that character is found.			X
"BASECHAR Function" (p. 314)	Converts characters to base characters.			X
"BYTE Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns one character in the ASCII or the EBCDIC collating sequence.	X		
"CAT Function" in *SAS Viya Functions and CALL Routines: Reference*	Does not remove leading or trailing blanks, and returns a concatenated character string.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"CATQ Function" in *SAS Viya Functions and CALL Routines: Reference*	Concatenates character or numeric values by using a delimiter to separate items and by adding quotation marks to strings that contain the delimiter.			X
"CATS Function" in *SAS Viya Functions and CALL Routines: Reference*	Removes leading and trailing blanks, and returns a concatenated character string.			X
"CATT Function" in *SAS Viya Functions and CALL Routines: Reference*	Removes trailing blanks, and returns a concatenated character string.			X
"CATX Function" in *SAS Viya Functions and CALL Routines: Reference*	Removes leading and trailing blanks, inserts delimiters, and returns a character string.			X
"CHAR Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a single character from a specified position in a character string.	X		
"CHOOSEC Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a character value that represents the results of choosing from a list of arguments.			X
"CHOOSEN Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a numeric value that represents the results of choosing from a list of arguments.			X
"COALESCEC Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the first nonmissing value from a list of numeric arguments.			X
"COLLATE Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a character string in ASCII or EBCDIC collating sequence.	X		

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"COMPARE Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the position of the leftmost character by which two strings differ, or returns 0 if there is no difference.	X		
"COMPBL Function" in *SAS Viya Functions and CALL Routines: Reference*	Removes multiple blanks from a character string.			X
"COMPGED Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the generalized edit distance between two strings.	X		
"COMPLEV Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the Levenshtein edit distance between two strings.	X		
"COMPRESS Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a character string with specified characters removed from the original string.	X		
"COUNT Function" in *SAS Viya Functions and CALL Routines: Reference*	Counts the number of times that a specified substring appears within a character string.	X		
"COUNTC Function" in *SAS Viya Functions and CALL Routines: Reference*	Counts the number of characters in a string that appear or do not appear in a list of characters.	X		
"COUNTW Function" in *SAS Viya Functions and CALL Routines: Reference*	Counts the number of words in a character string.	X		
"DEQUOTE Function" in *SAS Viya Functions and CALL Routines: Reference*	Removes matching quotation marks from a character string that begins with a quotation mark, and deletes all characters to the right of the closing quotation mark.		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"FIND Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches for a specific substring of characters within a character string.	X		
"FINDC Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a string for any character in a list of characters.	X		
"FINDW Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the character position of a word in a string, or returns the number of the word in a string.		X	
"FIRST Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the first character in a character string.	X		
"GETLOCENV Function" (p. 322)	Returns the current locale/language environment.			X
"GETPXLANGUAGE Function" (p. 323)	Returns the current two-letter language code.			X
"GETPXLOCALE Function" (p. 324)	Returns the POSIX locale value for a SAS locale.			X
"GETPXREGION Function" (p. 325)	Returns the current two-letter region code.			X
"HTMLDECODE Function" in *SAS Viya Functions and CALL Routines: Reference*	Decodes a string that contains HTML numeric character references or HTML character entity references, and returns the decoded string.			X
"HTMLENCODE Function" in *SAS Viya Functions and CALL Routines: Reference*	Encodes characters using HTML character entity references, and returns the encoded string.			X
"IFC Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a character value based on whether an expression is true, false, or missing.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"IFN Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a numeric value based on whether an expression is true, false, or missing.			X
"INDEX Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character expression for a string of characters, and returns the position of the string's first character for the first occurrence of the string.	X		
"INDEXC Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character expression for any of the specified characters, and returns the position of that character.	X		
"INDEXW Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character expression for a string that is specified as a word, and returns the position of the first character in the word.	X		
"KCOMPARE Function" (p. 326)	Returns the result of a comparison of character expressions.			X
"KCOMPRESS Function" (p. 327)	Removes specified characters from a character expression.			X
"KCOUNT Function" (p. 328)	Returns the number of double-byte characters in an expression.			X
"KCVT Function" (p. 335)	Converts data from one type of encoding data to another type of encoding data.		X	
"KCOUNTC Function" (p. 329)	Counts the number of words in a character string.			X
"KCOUNTW Function" (p. 331)	Counts the number of words in a character string.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
“KCOUNTX Function” (p. 334)	Counts the number of times that a specified substring appears within a character string.			X
“KFIND Function” (p. 337)	Searches for a specific substring of characters within a character string.			X
“KFINDC Function” (p. 339)	Searches a string for any character in a list of characters.			X
“KFINDW Function” (p. 342)	Returns the character position of a word in a string, or returns the number of the word in a string.			X
“KINDEX Function” (p. 346)	Searches a character expression for a string of characters.			X
“KINDEXB Function” (p. 347)	Searches a character expression for a string of characters.			X
“KINDEXCB Function” (p. 349)	Searches a character expression for specified characters.			X
“KINDEXC Function” (p. 348)	Searches a character expression for specified characters.			X
“KLEFT Function” (p. 350)	Left-aligns a character expression by removing unnecessary leading DBCS blanks and Shift out/Shift in (SO-SI) .			X
“KLENGTH Function” (p. 353)	Returns the length of an argument.			X
“KLOWCASE Function” (p. 354)	Converts all letters in an argument to lowercase.			X
“KPROPCASE Function” (p. 355)	Converts Chinese, Japanese, Korean, Taiwanese (CJKT) characters.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"KPROPCHAR Function" (p. 358)	Converts special characters to normal characters.			X
"KPROPDATA Function" (p. 359)	Removes or converts unprintable characters.			X
"KREVERSE Function" (p. 361)	Reverses a character expression.			X
"KRIGHT Function" (p. 362)	Right-aligns a character expression by trimming trailing DBCS blanks and SO-SI.			X
"KSCAN Function" (p. 363)	Selects a specified word from a character expression.			X
"KSCANX Function" (p. 365)	Selects a specified word from a character expression.			X
"KSTRCAT Function" (p. 369)	Concatenates two or more character expressions.			X
"KSTRIP Function" (p. 370)	Removes leading and trailing blanks from a character string.			X
"KSUBSTR Function" (p. 372)	Extracts a substring from an argument.			X
"KSUBSTRN Function" (p. 373)	Returns a substring, allowing a result with a length of 0.			X
"KSUBSTRB Function" (p. 375)	Extracts a substring from an argument according to the byte position of the substring in the argument.			X
"KTRIM Function" (p. 377)	Removes trailing DBCS blanks and SO-SI from character expressions.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"KTRUNCATE Function" (p. 379)	Truncates a character string to a specified length in byte units without breaking multibyte characters.			X
"KUPCASE Function" (p. 380)	Converts all letters in an argument to uppercase.			X
"KUPDATE Function" (p. 381)	Inserts, deletes, and replaces character value contents.			X
"KUPDATEB Function" (p. 383)	Inserts, deletes, and replaces the contents of the character value according to the byte position of the character value in the argument.			X
"KUPDATES Function" (p. 384)	Inserts, deletes, and replaces character value contents.			X
"KVERIFYB Function" (p. 387)	Returns the position of the first character that is unique to an expression.			X
"KVERIFY Function" (p. 386)	Returns the position of the first character that is unique to an expression.			X
"LEFT Function" in *SAS Viya Functions and CALL Routines: Reference*	Left-aligns a character string.			X
"LENGTH Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the length of a non-blank character string, excluding trailing blanks, and returns 1 for a blank character string.			X
"LENGTHC Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the length of a character string, including trailing blanks.			X
"LENGTHM Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the amount of memory (in bytes) that is allocated for a character string.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"LENGTHN Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the length of a character string, excluding trailing blanks.			X
"LOWCASE Function" in *SAS Viya Functions and CALL Routines: Reference*	Converts all letters in an argument to lowercase.			X
"MD5 Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the result of the message digest of a specified string.			X
"MISSING Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a numeric result that indicates whether the argument contains a missing value.			X
"MVALID Function" in *SAS Viya Functions and CALL Routines: Reference*	Checks the validity of a character string for use as a SAS member name.			X
"NLDATE Function" (p. 388)	Converts the SAS date value to the date value of the specified locale by using the date format descriptors.			X
"NLDATM Function" (p. 391)	Converts the SAS datetime value to the time value of the specified locale by using the datetime-format descriptors.			X
"NLTIME Function" (p. 394)	Converts the SAS time or the datetime value to the time value of the specified locale by using the NLTIME descriptors.			X
"NLITERAL Function" in *SAS Viya Functions and CALL Routines: Reference*	Converts a character string that you specify to a SAS name literal.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"NOTALNUM Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a non-alphanumeric character, and returns the first position at which the character is found.		X	
"NOTALPHA Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a non–alphabetic character, and returns the first position at which the character is found.		X	
"NOTCNTRL Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a character that is not a control character, and returns the first position at which that character is found.			X
"NOTDIGIT Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for any character that is not a digit, and returns the first position at which that character is found.			X
"NOTFIRST Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for an invalid first character in a SAS variable name under VALIDVARNAME=V7, and returns the first position at which that character is found.			X
"NOTGRAPH Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a non-graphical character, and returns the first position at which that character is found.		X	
"NOTLOWER Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a character that is not a lowercase letter, and returns the first position at which that character is found.		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"NOTNAME Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for an invalid character in a SAS variable name under VALIDVARNAME= V7, and returns the first position at which that character is found.			X
"NOTPRINT Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a non–printable character, and returns the first position at which that character is found.	X		
"NOTPUNCT Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a character that is not a punctuation character, and returns the first position at which that character is found.		X	
"NOTSPACE Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a character that is not a white-space character (blank, horizontal and vertical tab, carriage return, line feed, and form feed), and returns the first position at which that character is found.		X	
"NOTUPPER Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a character that is not an uppercase letter, and returns the first position at which that character is found.		X	
"NOTXDIGIT Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches a character string for a character that is not a hexadecimal character, and returns the first position at which that character is found.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"NVALID Function" in *SAS Viya Functions and CALL Routines: Reference*	Checks the validity of a character string for use as a SAS variable name.			X
"PROPCASE Function" in *SAS Viya Functions and CALL Routines: Reference*	Converts all words in an argument to proper case.			X
"PRXCHANGE Function" in *SAS Viya Functions and CALL Routines: Reference*	Performs a pattern-matching replacement.	X		
"PRXMATCH Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches for a pattern match, and returns the position at which the pattern is found.	X		
"PRXPAREN Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the last bracket match for which there is a match in a pattern.	X		
"PRXPARSE Function" in *SAS Viya Functions and CALL Routines: Reference*	Compiles a Perl regular expression (PRX) that can be used for pattern matching of a character value.	X		
"PRXPOSN Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a character string that contains the value for a capture buffer.	X		
"PUT Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a value using a specified format.			X
"PUTC Function" in *SAS Viya Functions and CALL Routines: Reference*	Enables you to specify a character format at run time.			X
"PUTN Function" in *SAS Viya Functions and CALL Routines: Reference*	Enables you to specify a numeric format at run time.			X
"QUOTE Function" in *SAS Viya Functions and CALL Routines: Reference*	Adds double quotation marks to a character value.			X
"RANK Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the position of a character in the ASCII or EBCDIC collating sequence.	X		

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"REPEAT Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a character value that consists of the first argument repeated *n*+1 times.			X
"REVERSE Function" in *SAS Viya Functions and CALL Routines: Reference*	Reverses a character string.	X		
"RIGHT Function" in *SAS Viya Functions and CALL Routines: Reference*	Right-aligns a character expression.			X
"SASMSG Function" (p. 395)	Specifies a message from a data set. The returned message is based on the current locale and a specified key.			X
"SASMSGL Function" (p. 398)	Specifies a message from a data set. The message is based on a specified locale value and a specified key value.			X
"SCAN Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the *n*th word from a character string.	X		
"SETLOCALE Function" (p. 404)	Specifies the locale keys for the current SAS locale.			X
"SHA256 Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the result of the message digest of a specified string.			X
"SORTKEY Function" (p. 401)	Creates a linguistic sort key.			X
"SOUNDEX Function" in *SAS Viya Functions and CALL Routines: Reference*	Encodes a string to facilitate searching.	X		
"SPEDIS Function" in *SAS Viya Functions and CALL Routines: Reference*	Determines the likelihood of two words matching, expressed as the asymmetric spelling distance between the two words.	X		

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"STRIP Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a character string with all leading and trailing blanks removed.		X	
"SUBPAD Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a substring that has a length that you specify, using blank padding if necessary.	X		
"SUBSTR (left of =) Function" in *SAS Viya Functions and CALL Routines: Reference*	Replaces character value contents.	X		
"SUBSTR (right of =) Function" in *SAS Viya Functions and CALL Routines: Reference*	Extracts a substring from an argument.	X		
"SUBSTRN Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a substring, allowing a result with a length of 0.	X		
"TRANSLATE Function" in *SAS Viya Functions and CALL Routines: Reference*	Replaces specific characters in a character string.	X		
"TRANSTRN Function" in *SAS Viya Functions and CALL Routines: Reference*	Replaces or removes all occurrences of a substring in a character string.			X
"TRANTAB Function" on page 420	Transcodes data by using the specified translation table.	X		
"TRANWRD Function" in *SAS Viya Functions and CALL Routines: Reference*	Replaces or removes all occurrences of a substring in a character string.			X
"TRIM Function" in *SAS Viya Functions and CALL Routines: Reference*	Removes trailing blanks from a character string, and returns one blank if the string is missing.			X
"TRIMN Function" in *SAS Viya Functions and CALL Routines: Reference*	Removes trailing blanks from character expressions, and returns a string with a length of 0 if the expression is missing.			X
"TZONEDSTNAME Function" (p. 416)	Returns a daylight savings time name.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"TZONEDSTOFF Function" (p. 417)	Returns the time zone offset value for the specified daylight savings time.			X
"TZONEID Function" (p. 411)	Returns the current time zone ID.			X
"TZONENAME Function" (p. 412)	Returns the current standard or daylight savings time time zone name.			X
"TZONEOFF Function" (p. 414)	Returns the user time zone offset.			X
"TZONES2U Function" (p. 415)	Converts a SAS datetime value to a UTC datetime value.			X
"TZONESTTNAME Function" (p. 418)	Returns a standard time zone name.			X
"TZONESTTOFF Function" (p. 419)	Returns the time zone offset value for the specified standard time.			X
"TZONEU2S Function" (p. 419)	Converts a UTC datetime value to a SAS datetime value.			X
"UNICODE Function" (p. 421)	Converts Unicode characters to the current SAS session encoding.			X
"UNICODEC Function" (p. 423)	Converts characters in the current SAS session encoding to Unicode characters.			X
"UNICODELEN Function" (p. 424)	Specifies the length of the character unit for the Unicode data.			X
"UNICODEWIDTH Function" (p. 425)	Specifies the length of a display unit for the Unicode data.			X
"UPCASE Function" in *SAS Viya Functions and CALL Routines: Reference*	Converts all letters in an argument to uppercase.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"URLDECODE Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a string that was decoded using the URL escape syntax.		X	
"URLENCODE Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns a string that was encoded using the URL escape syntax.		X	
"VERIFY Function" in *SAS Viya Functions and CALL Routines: Reference*	Returns the position of the first character in a string that is not in any of several other strings.			X
"WHICHC Function" in *SAS Viya Functions and CALL Routines: Reference*	Searches for a character value that is equal to the first argument, and returns the index of the first matching value.			X

Chapter 11

VARCHAR Support in Functions

VARCHAR Data Type in String Functions

SAS Viya supports the VARCHAR data type for character data. Variables created using the VARCHAR data type vary in width and use character semantics, rather than being fixed-width and using byte semantics of the original SAS character data type. When a VARCHAR variable is passed to any string function, including the traditional string functions or the K functions, SAS assumes that the length represents the number of characters. An offset for a VARCHAR variable represents the position of a character in the string. The fixed-width character data type is better suited for binary data.

VARCHAR is supported in CAS tables and is used in a data_NULL_step. Some procedures support VARCHAR columns. Procedures that do not support VARCHAR columns convert the columns to a fixed-width character column. VARCHAR is not supported in data sets.

Here is an example of the data types using the SUBSTR function **`substr(X, 10, 2)`**.

If the variable *X* is defined as a fixed-width character variable, SUBSTR returns the 10th and 11th bytes in the string. If X is defined as a VARCHAR variable, SUBSTR returns the 10th and 11th *characters* in the string.

A variable that is assigned a VARCHAR data type of length 10 can hold up to 10 characters.

This table shows the differences between fixed-width and VARCHAR data types. The fixed-width character data type uses byte semantics and VARCHAR uses character semantics.

Data Type	Traditional String Function	K Function
Fixed-width character	byte	character
VARCHAR	character	character

The SAS Viya session encoding, UTF-8, supports data that can be 1 or multiple bytes. An example of a multi-byte UTF-8 character is the euro (€) character, which has 3 bytes. Here is a table that shows the representation of the string *€123*. The € character has 3 bytes and each number has 1 byte.

Character	€			1	2	3
Hexadecimal value	E2	82	AC	31	32	33

The following examples compare the fixed-width data type and the VARCHAR data type. Table 11.1 on page 300 shows results from the fixed-width variable using the data *€123*. Table 11.2 on page 301 shows results using VARCHAR.

K functions should be used for fixed-width character data that might contain multi-byte characters:

1. The example in the first row assigns the string to a fixed-width character column. The PUT statement displays the characters and then displays the hexadecimal value for those characters.
2. The example in the second row uses the SUBSTR function with a position of 2. Because the variable x is a fixed-width character column, the position interpreted as the second byte of the string in x, which is also the second byte of the € character. The result is that the first 2 bytes assigned to xsub are invalid UTF-8 data. The results displayed here show garbage (or the Unicode replacement character) as the first 2 bytes of xsub.
3. The example in the third row uses the INDEX function to locate the 1 in the string. Because the variable x is a fixed-width character column, the result returned by the INDEX function is the byte position of the character in the string. The result is **`xidx=4`** because the 1 is located at the fourth byte in x.

Table 11.1 *Fixed-Width Character Example*

Statements	Results

`data _null_;`
`  x='€123';`
`  put x=`
`  x= $hex6.;`
 	
`  xsub=substr(x,2);`
`  put xsub=`
`  xsub= $hex.;`
 	
`  xidx=index(x,'1');`
`  put xidx=;`
`run;`
	`x=€123 x=E282AC`
	`xsub=¬123 xsub=82AC31323320`
	`xidx=4`

1. The example in the first row assigns the string to a VARCHAR column. The PUT statement displays the characters and then the hexadecimal value for those characters.
2. The example in the second row uses the SUBSTR function with a position of 2. In this example, the variable v is defined as a VARCHAR so that the position is interpreted as the second character of the string in v. The result assigned to vsub is the string '123', which contains valid UTF-8 characters.
3. The example in the third row uses the INDEX function to locate the 1 in the string. Because the variable v is a VARCHAR column, the result returned by the INDEX

function is the character position of the character in the string. The result is **`vidx=2`** because 1 is the second character in v.

Table 11.2 *VARCHAR Character Example*

Statements	Results

`data _null_;`
`  length v VARCHAR(6);`
`  v='€123';`
`  put v=`
`  v= $hex.;`
 	
`  vsub=substr(v,2);`
`  put vsub=`
`  vsub= $hex.;`
 	
`  vidx=index(v,'1');`
`  put vidx=;`
`run;`
	`v=€123 v=E282AC313233`
	
	`vsub=123 vsub=313233202020202020202020202020202020202020202020`
	
	`vidx=2`

Chapter 12

Dictionary of NLS Call Routines and Functions

Functions by Category

The following categories relate to NLS issues:

Some functions run in SAS only, and some functions run in SAS and on the CAS engine. If CAS is specified for the function category, then the function runs in SAS and on the CAS server. If CAS is not specified for the function category, then the function runs in SAS only. For example, the BASECHAR function runs in SAS and on the CAS server, so CAS is specified as a category. The KCVT function runs on SAS only, so CAS is not specified as a category.

Table 12.1 *Categories of NLS Functions*

Category	Description
CAS	Functions that run on the CAS server.
Character	Processes character data.

Currency Conversion	Converts one currency to another currency.
DBCS	Processes double-byte character set.
Date and Time	Processes data and time data.
Locale	Processes data based on the specified locale.
Variable Information	Processes variable information.

Category	Language Elements	Description
CAS	BASECHAR Function (p. 314)	Converts characters to base characters.
	CALL KSCANX Routine (p. 316)	Returns the position and length of the nth word from a character string.
	KCOMPRESS Function (p. 327)	Removes specified characters from a character expression.
	KCOUNT Function (p. 328)	Returns the number of double-byte characters in an expression.
	KCOUNTC Function (p. 329)	Counts the number of words in a character string.
	KCOUNTW Function (p. 331)	Counts the number of words in a character string.
	KCOUNTX Function (p. 334)	Counts the number of times that a specified substring appears within a character string.
	KFIND Function (p. 337)	Searches for a specific substring of characters within a character string.
	KFINDC Function (p. 339)	Searches a string for any character in a list of characters.
	KFINDW Function (p. 342)	Returns the character position of a word in a string or returns the number of the word in a string.
	KINDEX Function (p. 346)	Searches a character expression for a string of characters.
	KINDEXC Function (p. 348)	Searches a character expression for specified characters.
	KLEFT Function (p. 350)	Left-aligns a character expression by removing unnecessary leading DBCS blanks and SO/SI.
	KLENGTH Function (p. 353)	Returns the length of an argument.
	KLOWCASE Function (p. 354)	Converts all single-width English alphabet letters in an argument to lowercase.
	KREVERSE Function (p. 361)	Reverses a character expression.
	KRIGHT Function (p. 362)	Right-aligns a character expression by trimming trailing DBCS blanks and SO/SI.

Category	Language Elements	Description
	KSCAN Function (p. 363)	Selects a specified word from a character expression.
	KSCANX Function (p. 365)	Selects a specified word from a character expression.
	KSTRCAT Function (p. 369)	Concatenates two or more character expressions.
	KSTRIP Function (p. 370)	Removes leading and trailing blanks from a character string.
	KSUBSTR Function (p. 372)	Extracts a substring from an argument.
	KTRANSLATE Function (p. 376)	Replaces specific characters in a character expression.
	KTRIM Function (p. 377)	Removes trailing DBCS blanks and SO/SI from character expressions.
	KUPCASE Function (p. 380)	Converts all single-width English alphabet letters in an argument to uppercase.
	KUPDATE Function (p. 381)	Inserts, deletes, and replaces character value contents.
	KUPDATES Function (p. 384)	Inserts, deletes, and replaces character value contents.
	KVERIFY Function (p. 386)	Returns the position of the first character that is unique to an expression.
	VTRANSCODE Function (p. 427)	Returns a value that indicates whether transcoding is enabled for the specified character variable.
	VTRANSCODEX Function (p. 428)	Returns a value that indicates whether transcoding is enabled for the specified argument.
Character	ANORM420 Function (p. 310)	Returns a normalized string from an input string encoded in EBCDIC420.
	BASECHAR Function (p. 314)	Converts characters to base characters.
	KCVT Function (p. 335)	Converts data from one type of encoding data to another type of encoding data.
	KSTRIP Function (p. 370)	Removes leading and trailing blanks from a character string.
	TRANTAB Function (p. 420)	Transcodes data by using the specified translation table.
	UNICODE Function (p. 421)	Converts Unicode characters to the current SAS session encoding.
	UNICODEC Function (p. 423)	Converts characters in the current SAS session encoding to Unicode characters.
	UNICODELEN Function (p. 424)	Specifies the length of the character unit for the Unicode data.

Category	Language Elements	Description
	UNICODEWIDTH Function (p. 425)	Specifies the length of a display unit for the Unicode data.
Date and Time	NLDATE Function (p. 388)	Converts the SAS date value to the date value of the specified locale by using the date format descriptors.
	NLDATM Function (p. 391)	Converts the SAS datetime value to the time value of the specified locale by using the datetime-format descriptors.
	NLTIME Function (p. 394)	Converts the SAS time or the datetime value to the time value of the specified locale by using the NLTIME descriptors.
	TZONEID Function (p. 411)	Returns the current time zone ID.
	TZONENAME Function (p. 412)	Returns the current standard or daylight savings time, time zone name.
	TZONEOFF Function (p. 414)	Returns the user time zone offset.
	TZONES2U Function (p. 415)	Converts a SAS date time value to a UTC date time value.
	TZONEDSTNAME Function (p. 416)	Returns a daylight savings time name.
	TZONEDSTOFF Function (p. 417)	Returns the time zone offset value for the specified daylight savings time.
	TZONESTTNAME Function (p. 418)	Returns a standard time zone name.
	TZONESTTOFF Function (p. 419)	Returns the time zone offset value for the specified standard time.
	TZONEU2S Function (p. 419)	Converts a UTC date time value to a SAS date time value.
DBCS	CALL KSCANX Routine (p. 316)	Returns the position and length of the nth word from a character string.
	KCOMPARE Function (p. 326)	Returns the result of a comparison of character expressions.
	KCOMPRESS Function (p. 327)	Removes specified characters from a character expression.
	KCOUNT Function (p. 328)	Returns the number of double-byte characters in an expression.
	KCOUNTC Function (p. 329)	Counts the number of words in a character string.
	KCOUNTW Function (p. 331)	Counts the number of words in a character string.
	KCOUNTX Function (p. 334)	Counts the number of times that a specified substring appears within a character string.

Category	Language Elements	Description
	KFIND Function (p. 337)	Searches for a specific substring of characters within a character string.
	KFINDC Function (p. 339)	Searches a string for any character in a list of characters.
	KFINDW Function (p. 342)	Returns the character position of a word in a string or returns the number of the word in a string.
	KINDEX Function (p. 346)	Searches a character expression for a string of characters.
	KINDEXB Function (p. 347)	Searches a character expression for a string of characters.
	KINDEXC Function (p. 348)	Searches a character expression for specified characters.
	KINDEXCB Function (p. 349)	Searches a character expression for specified characters.
	KLEFT Function (p. 350)	Left-aligns a character expression by removing unnecessary leading DBCS blanks and SO/SI.
	KLENGTH Function (p. 353)	Returns the length of an argument.
	KLOWCASE Function (p. 354)	Converts all single-width English alphabet letters in an argument to lowercase.
	KPROPCASE Function (p. 355)	Converts Chinese, Japanese, Korean, Taiwanese (CJKT) characters.
	KPROPCHAR Function (p. 358)	Converts special characters to normal characters.
	KPROPDATA Function (p. 359)	Removes or converts unprintable characters.
	KREVERSE Function (p. 361)	Reverses a character expression.
	KRIGHT Function (p. 362)	Right-aligns a character expression by trimming trailing DBCS blanks and SO/SI.
	KSCAN Function (p. 363)	Selects a specified word from a character expression.
	KSCANX Function (p. 365)	Selects a specified word from a character expression.
	KSTRCAT Function (p. 369)	Concatenates two or more character expressions.
	KSUBSTR Function (p. 372)	Extracts a substring from an argument.
	KSUBSTRN Function (p. 373)	Returns a substring, allowing a result with a length of zero.
	KSUBSTRB Function (p. 375)	Extracts a substring from an argument according to the byte position of the substring in the argument.
	KTRANSLATE Function (p. 376)	Replaces specific characters in a character expression.

Category	Language Elements	Description
	KTRIM Function (p. 377)	Removes trailing DBCS blanks and SO/SI from character expressions.
	KTRUNCATE Function (p. 379)	Truncates a string to a specified length in byte unit without breaking multibyte characters.
	KUPCASE Function (p. 380)	Converts all single-width English alphabet letters in an argument to uppercase.
	KUPDATE Function (p. 381)	Inserts, deletes, and replaces character value contents.
	KUPDATEB Function (p. 383)	Inserts, deletes, and replaces the contents of the character value according to the byte position of the character value in the argument.
	KUPDATES Function (p. 384)	Inserts, deletes, and replaces character value contents.
	KVERIFY Function (p. 386)	Returns the position of the first character that is unique to an expression.
	KVERIFYB Function (p. 387)	Returns the position of the first character that is unique to an expression.
Encoding	ENCODCOMPAT Function (p. 320)	Verifies the transcoding compatibility between two encodings.
	ENCODISVALID Function (p. 321)	Verifies a valid encoding name.
Locale	GETLOCENV Function (p. 322)	Returns the current locale/language environment.
	GETPXLANGUAGE Function (p. 323)	Returns the current two-letter language code.
	GETPXLOCALE Function (p. 324)	Returns the POSIX locale value for a SAS locale.
	GETPXREGION Function (p. 325)	Returns the current two-letter region code.
	SASMSG Function (p. 395)	Specifies a message from a data set. The returned message is based on the current locale and a specified key.
	SASMSGL Function (p. 398)	Specifies a message from a data set. The message is based on a specified locale value and a specified key value.
	SORTKEY Function (p. 401)	Creates a linguistic sort key.
	SETLOCALE Function (p. 404)	Specifies the locale keys for the current SAS locale.
Variable Information	VARTRANSCODE Function (p. 426)	Returns the transcode attribute of a SAS data set variable.

Category	Language Elements	Description
	VTRANSCODE Function (p. 427)	Returns a value that indicates whether transcoding is enabled for the specified character variable.
	VTRANSCODEX Function (p. 428)	Returns a value that indicates whether transcoding is enabled for the specified argument.

Dictionary

ANORM420 Function

Returns a normalized string from an input string encoded in EBCDIC420.

Category: Character

Restriction: This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility on page 279 .

Syntax

ANORM420 (*string*,<*modifiers*>)

Required Argument

string
specifies an input string that is encoded in EBCDIC420.

Optional Argument

modifiers
are character constants, variables, or expressions that specify one or more modifiers. The following modifiers can be in uppercase or lowercase:

g ignores the ligature

i ignores the Arabic-Indic numbers

p ignores mapping of shaped characters

s ignores the addition of space

t ignores transcoding

z ignores mapping of the zero-length-space character

Details

The ANORM420 function processes data that is encoded in EBCDIC420.

Refer to the following tables for information about the modifier variables:

The ANORM420 function replaces the following character and code points with the appropriate ligature unless the modifier *g* is specified.

Table 12.2 *Modifier g: Arabic Ligatures*

Character	Unicode Value	From	To	Description
ﻵ	(U+FEF5)	0xB2	0xB1 + 0x47	Arabic Ligature Lam with Aleph with Madda above - Isolated form
ﻶ	(U+FEF6)	0xB3	0xB1 + 0x47	Arabic Ligature Lam with Aleph with Madda above - Final form
ﻷ	(U+FEF7)	0xB4	0xB1 + 0x49	Arabic Ligature Lam with Aleph with Hamza above - Isolated form
ﻸ	(U+FEF8)	0xB5	0xB1 + 0x49	Arabic Ligature Lam with Aleph with Hamza above - Final form
ﻻ	(U+FEFC)	0xB8	0xB1 + 0x56	Arabic Ligature Lam with Aleph - Isolated form
ﻺ	(U+FEFD)	0xB9	0xB1 + 0x56	Arabic Ligature Lam with Aleph - Final form

The ANORM420 function remaps the Arabic-Indic numbers to the digit unless the modifier *i* is specified and no transcoding occurs.

Table 12.3 *Modifier i: Arabic-Indic Numbers*

From	٠ U+0660	١ U+0661	٢ U+0662	٣ U+0663	٤ U+0664	٥ U+0665	٦ U+0666	٧ U+0667	٨ U+0668	٩ U+0669
Hexadecimal	0xDF	0xEA	0xEB	0xED	0xEE	0xEF	0xFB	0xFC	0xFD	0xFE
To	0	1	2	3	4	5	6	7	8	9
Hexadecimal	0xF0	0xF1	0xF2	0xF3	0xF4	0xF5	0xF6	0xF7	0xF8	0xF9

The ANORM420 function remaps the shaped characters to their unshaped equivalent unless the modifier p is specified.

Table 12.4 *Modifier p: Shaped Characters Mapping*

From		To	
0x43	U+FE7D	0x42	U+0651
0x48	FE82	0x47	U+0622

From		To	
0x51	U+FE84	0x49	U+0623
0x57	U+FE8E	0x56	U+0627
0x59	U+FE91	0x58	U+0628
0x64	U+FE97	0x63	U+062A
0x66	U+FE9B	0x65	U+062B
0x68	U+FE9F	0x67	U+062C
0x70	U+FEA3	0x69	U+062D
0x72	U+FEA7	0x71	U+062E
0x78	U+FEB3	0x77	U+0633
0x8A	U+FEB7	0x80	U+0634
0x8C	U+FEBB	0x8B	U+0635
0x8E	U+FEBF	0x8D	U+0636
0x9B	U+FECA	0x9A	U+0639
0x9C	U+FECB	0x9A	U+0639
0x9D	U+FECC	0x9A	U+0639
0x9F	U+FECE	0x9E	U+063A
0xA0	U+FECF	0x9E	U+063A
0xAA	U+FED0	0x9E	U+063A
0xAC	U+FED3	0xAB	U+0641
0xAE	U+FED7	0xAD	U+0642
0xB0	U+FEDB	0xAF	U+0643
0xBA	U+FEDF	0xB1	U+0644
0xBC	U+FEE3	0xBB	U+0645
0xBE	U+FEE7	0xBD	U+0646
0xCB	U+FEEB	0xBF	U+0647

From		To	
0xCD	U+FEEC	0xBF	U+0647
0xDB	U+FEF0	0xDA	U+061F
0xDD	U+FEF2	0xDC	U+064A
0xDE	U+FEF3	0xDC	U+064A

The ANORM420 function adds a space after the following characters unless the modifier *s* is specified.

Table 12.5 *Modifier s: ignores the addition of a space*

ب	0x58	(U+0628)	Arabic Letter BEH
ة	0x62	(U+0629)	Arabic Letter TEH MARBUTA
ت	0x63	(U+062A)	Arabic Letter TEH
ث	0x65	(U+062B)	Arabic Letter THEH
ج	0x67	(U+062C)	Arabic Letter JEEM
ح	0x69	(U+062D)	Arabic Letter HAH
خ	0x71	(U+062E)	Arabic Letter KHAH
س	0x77	(U+0633)	Arabic Letter SEEN
ش	0x80	(U+0634)	Arabic Letter SHEEN
ص	0x8B	(U+0635)	Arabic Letter SAD
ض	0x8D	(U+0636)	Arabic Letter DAD
ع	0x9A	(U+0639)	Arabic Letter AIN
ع	0x9B	(U+FECA)	Arabic Letter AIN final form
غ	0x9E	(U+063A)	Arabic Letter GHAIN
غ	0x9F	(U+FECE)	Arabic Letter GHAIN final form
ف	0xAB	(U+0641)	Arabic Letter FEH

ق	0xAD	(U+0642)	Arabic Letter QAF
ك	0xAF	(U+0643)	Arabic Letter KAF
ل	0xB1	(U+0644)	Arabic Letter LAM
م	0xBB	(U+0645)	Arabic Letter MEEM
ن	0xBD	(U+0646)	Arabic Letter NOON
ه	0xBF	(U+0647)	Arabic Letter HEH
ى	0xDA	(U+0649)	Arabic Letter ALEF MAKSURA
ي	0xDC	(U+064A)	Arabic Letter YEH
ى	0xDB	(U+FEF0)	Arabic Letter ALEF MAKSURA final form
ي	0xDD	(U+FEF2)	Arabic Letter YEH final form

The ANORM420 function transcodes the normalized string to the session encoding unless the modifier *t* is specified.

The ANORM420 function remaps the zero-length-break-space character (U+200B), also abbreviated ZWSP, to a space character unless the modifier *z* is specified. The zero-length character is located at code position 0x45.

Example

The following example demonstrates the ANORM420 function:

Statements	Results

`data _null_ ;`
`a = '59CD57BC577745'x  ;`
`  s1 = anorm420(a) ;`
`/* Turn off addition of space and mapping of`
`Arabic-Indic numbers */`
`s2 = anorm420(a,"si") ;`
`/* Turn off transcoding */`
`s3 = anorm420(a,'t') ;`
`  put s1= $hex20. / s2= $hex20. / s3=$hex20.;`
`run;`
	`s1=C8E5C7E3C7D320A02020`
	`s2=C8E5C7E3C7D3A0202020`
	`s3=59CD57BC577740454040`

BASECHAR Function

Converts characters to base characters.

Categories: Character
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

STR=BASECHAR(*<instr>*(,*<Unicode type>*))

Required Arguments

str
data string that is converted.

instr
input data string.

Unicode type
If one of the following Unicode character formats is specified, national characters are represented in the specified format:

ESC	Unicode Escape (\u00cd).
NCR	Numeric Character Representation (Å).
PAREN	Unicode Parenthesis Escape (<u00c5>).

Details

The BASECHAR function reads characters and converts them to base characters. Some characters consist of a base character and one or more accents or combining characters. The BASECHAR function reads the characters in the string and converts them to the base character form without the accents based on Unicode specifications. When Unicode type is specified, national characters are represented in the specified format. For example, the data string Mühlenflieβ is converted to Muhlenfließ.

Example

The following examples demonstrate using the Unicode types:

Statements	Results

Note: The SAS session encoding is UTF-8.
`data cities;length name$24 name_utf8$24;`
`  input name;`
`  name_utf8 = basechar(name); put name_utf8=;`
`cards;`
`Mühlenfließ`
`LaUña`
`ZielonaGóra`
`run;`
	Note: The BASECHAR function uses Unicode character decomposition mappings, so the sharp s, ß, is valid. http://www.unicode.org/reports/tr44/#Character_Decomposition_Mappings
	`Muhlenfließ`
	`LaUna`
	`ZielonaGora`

Statements	Results

Note: The SAS session encoding is UTF-8 and the ESC option is specified.
`data cities;length name$24 name_utf8$80;`
`  input name;`
`  name_utf8 = basechar(name, "ESC"); put name_utf8=;`
`cards;`
`Mühlenfließ`
`LaUña`
`ZielonaGóra`
`run;`
	`Muhlenflie\u00df`
	`LaUna`
	`ZielonaGora`

Note: The SAS session encoding is UTF-8 and the NCR option is specified.
`data cities;length name$24 name_utf8$80;`
`  input name;`
`  name_utf8 = basechar(name, "NCR"); put name_utf8=;`
`cards;`
`Mühlenfließ`
`LaUña`
`ZielonaGóra`
`run;`
	`Muhlenflie&#x00df`
	`LaUna`
	`ZielonaGora`

CALL KSCANX Routine

Returns the position and length of the *n*th word from a character string.

Categories: DBCS
CAS

Notes: CALL KSCANX is designed to process multi-byte data. However, if the first argument, *string,* has single-byte characters, then the CALL KSCANX routine processes the SBCS data. For more information, see Internationalization Compatibility on page 279.

Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL KSCANX(<*string*>,*count*,*position*,*length* <,<*character-list*><,<*modifier(s)*>>>);

Required Arguments

string
specifies a character constant, variable, or expression.

count
is a nonzero numeric constant, variable, or expression that has an integer value. The integer value specifies the number of the word in the character string that you want KSCAN to select. For example, a value of 1 indicates the first word, a value of 2 indicates the second word, and so on. The following rules apply:

- If *count* is positive, SCAN counts words from left to right in the character string.
- If *count* is negative, SCAN counts words from right to left in the character string.

position
specifies a numeric variable in which the position of the word is returned. If *count* exceeds the number of words in the string, then the value that is returned in *position* is zero. If *count* is zero or missing, then the value that is returned in *position* is missing.

length
specifies a numeric variable in which the length of the word is returned. If *count* exceeds the number of words in the string, then the value that is returned in *length* is zero. If *count* is zero or missing, then the value that is returned in *length* is missing.

Optional Arguments

character-list
specifies an optional character expression that initializes a list of characters. This list determines which characters are used as the delimiters that separate words. The following rules apply:

- By default, all characters in *character-list* are used as delimiters.
- If you specify the K modifier in the *modifier* argument, then all characters that are not in *character-list* are used as delimiters.

Tip You can add more characters to *character-list* by using other modifiers.

modifier(s)
specifies a character constant, variable, or expression in which each non-blank character modifies the actions of the SCAN function. Blanks are ignored. Use the following characters as modifiers:

a or A	adds alphabetic characters to the list of characters.
b or B	scans backward, from right to left instead of from left to right, regardless of the sign of the *count* argument.
c or C	adds control characters to the list of characters.
d or D	adds digits to the list of characters.
f or F	adds an underscore and English letters (that is, valid first characters in a SAS variable name using VALIDVARNAME=V7) to the list of characters.
g or G	adds graphic characters to the list of characters. Graphic characters are those that, when printed, produce an image on paper.
h or H	adds a horizontal tab to the list of characters.
i or I	ignores the case of the characters.
k or K	treats all characters that are not in the list of characters as delimiters. That is, if K is specified, then characters that are in the list of characters are kept in the returned value rather than being omitted because they are delimiters. If K is not specified, then all characters that are in the list of characters are treated as delimiters.
l or L	adds lowercase letters to the list of characters.
m or M	specifies that multiple consecutive delimiters and delimiters at the beginning or end of the *string* argument refer to words that have a

length of zero. If the M modifier is not specified, then multiple consecutive delimiters are treated as one delimiter, and delimiters at the beginning or end of the *string* argument are ignored.

n or N adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name using VALIDVARNAME=V7) to the list of characters.

o or O processes the *character-list* and *modifier* arguments only once, rather than every time the CALL KSCANX routine is called. Using the O modifier in the DATA step can make CALL KSCANX run faster when you call it in a loop where the *character-list* and *modifier* arguments do not change. The O modifier applies separately to each instance of the CALL KSCANX routine in your SAS code. It does not cause all instances of the CALL KSCANX routine to use the same delimiters and modifiers.

p or P adds punctuation marks to the list of characters.

q or Q ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of the *string* argument contains unmatched quotation marks, then scanning from left to right produces different words than scanning from right to left.

s or S adds space characters to the list of characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed).

t or T trims trailing blanks from the *string* and *character-list* arguments. If you want to remove trailing blanks from just one character argument instead of both character arguments, then use the TRIM function instead of the CALL KSCANX routine with the T modifier.

u or U adds uppercase letters to the list of characters.

w or W adds printable (writable) characters to the list of characters.

x or X adds hexadecimal characters to the list of characters.

Tip If the *modifier* argument is a character constant, then enclose it in quotation marks. Specify multiple modifiers in a single set of quotation marks. A *modifier* argument can also be expressed as a character variable or expression.

Details

Definition of Delimiter and Word

A *delimiter* is any of several characters that are used to separate words. You can specify the delimiters in the *character-list* and *modifier* arguments.

If you specify the Q modifier, then delimiters inside substrings that are enclosed in quotation marks are ignored.

In the CALL KSCANX routine, *word* refers to a substring that has all of the following characteristics:

- is bounded on the left by a delimiter or the beginning of the string
- is bounded on the right by a delimiter or the end of the string
- contains no delimiters

A word can have a length of zero if there are delimiters at the beginning or end of the string or if the string contains two or more consecutive delimiters. However, the CALL

KSCANX routine ignores words that have a length of zero unless you specify the M modifier.

Using Default Delimiters in ASCII and EBCDIC Environments

If you use the CALL KSCANX routine with only four arguments, then the default delimiters depend on whether your computer uses ASCII or EBCDIC characters.

- If your computer uses ASCII characters, then the default delimiters are as follows:
- blank ! $ % & ( ) * + , - . / ; < ^ ¦

  In ASCII environments that do not contain the ^ character, the CALL KSCANX routine uses the ~ character instead.
- If your computer uses EBCDIC characters, then the default delimiters are as follows:

  blank ! $ % & ( ) * + , - . / ; < ¬ | ¢ ¦

If you use the *modifier* argument without specifying any characters as delimiters, then the only delimiters used are those that are defined by the *modifier* argument. In this case, the lists of default delimiters for ASCII and EBCDIC environments are not used. In other words, modifiers add to the list of delimiters that are explicitly specified by the *character-list* argument. Modifiers do not add to the list of default modifiers.

Using the CALL KSCANX Routine with the M Modifier

If you specify the M modifier, then the number of words in a string is defined as one plus the number of delimiters in the string. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you specify the M modifier, the CALL KSCANX routine returns a positive position and a length of zero if one of the following conditions is true:

- The string begins with a delimiter and you request the first word.
- The string ends with a delimiter and you request the last word.
- The string contains two consecutive delimiters and you request the word that is between the two delimiters.

If you specify a count that is greater in absolute value than the number of words in the string, then the CALL KSCANX routine returns a position and length of zero.

Using the CALL KSCANX Routine without the M Modifier

If you do not specify the M modifier, then the number of words in a string is defined as the number of maximal substrings of consecutive non-delimiters. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you do not specify the M modifier, then the CALL KSCANX routine does the following:

- ignores delimiters at the beginning or end of the string
- treats two or more consecutive delimiters as if they were a single delimiter

If the string contains no characters other than delimiters or if you specify a count that is greater in absolute value than the number of words in the string, then the CALL KSCANX routine returns a position and length of zero.

Finding the Word as a Character String

To find the designated word as a character string after calling the CALL KSCANX routine, use the SUBSTRN function with the *string*, *position*, and *length* arguments:

```
substrn(string, position, length);
```

Because CALL KSCANX can return a length of zero, using the SUBSTR function can cause an error.

Using Null Arguments

The CALL KSCANX routine allows character arguments to be null. Null arguments are treated as character strings with a length of zero. Numeric arguments cannot be null.

Example

This example uses Chinese characters:

```
data _null_;
xyz='SAS是全球最大的软件公司之一，sas  是全球商业智能和分析软件与服务领袖。';
call kscanx(xyz, 2, pos, len,  '', 'sp'); /* Use spaces and punctuations as delimiter. */
put pos= len=;
run;
```

Here is the result from the CALL KSCANX routine example:

```
pos=17 len=3
```

ENCODCOMPAT Function

Verifies the transcoding compatibility between two encodings.

Category: Encoding

Syntax

ENCODCOMPAT(*source1,<source2>*)

Required Arguments

source1
a character string that represents an encoding.

source2
a character string that represents an encoding. This argument is optional.

Details

If you specify one encoding, the function verifies the compatibility of the specified encoding with the current SAS session encoding.

If you specify two encodings, the function verifies the compatibility of the two encodings.

The function compares two encoding identifiers and determines whether the data needs to be transcoded. *Source1* is the source encoding. *Source2* is the destination encoding. Transcoding 7-bit ASCII to another type of ASCII is compatible, but transcoding ASCII to 7-bit ASCII might not be compatible.

The ENCODCOMPAT function specifies the following values:

–1 *Source1* is not a valid encoding name.

–2 *Source2* is not a valid encoding name.

0 The encodings are not compatible. Transcoding is needed.

1 The encodings are compatible. Transcoding is not needed.

2 A newline character is detected.

Example

The following examples demonstrate the ENCODCOMPAT features:

Statements	Results

`/* session encoding is wlatin1 */`
`isCompat= EncodCompat("xyz");`
`put isCompat;`
	`-1`

`/* session encoding is wlatin1 */`
`isCompat= EncodCompat ("cp1252");`
`put isCompat;`
	`1`

`isCompat= EncodCompat`
`("ebcdic1149","open_ed-1149");`
`put isCompat;`
	`2`

`isCompat= EncodCompat`
`("cp1251","ebcdic1149");`
`put isCompat;`
	`0`

ENCODISVALID Function

Verifies a valid encoding name.

Category: Encoding

Syntax

ENCODISVALID(*source*)

Required Argument

source
a character string that represents an encoding name.

Details

The ENCODISVALID function returns the following values:

0 the character string is not a valid encoding name.

1 the character string is a valid short encoding name.

2 the character string is a valid long encoding name.

3 the character string is a valid alias encoding name.

Example

The following examples demonstrate the ENCODISVALID features:

SAS Statements	Results
`isValid= EncodIsValid("xyz");` `put isValid;`	0
`isValid= EncodIsValid("wlt2");` `put isValid;`	1
`isValid= EncodIsValid("wlatin2");` `put isValid;`	2

`isValid= EncodIsValid("cp1250");`
 	
`put isValid;`
	3

GETLOCENV Function

Returns the current locale/language environment.

Category: Locale

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

GETLOCENV( )

Details

The GETLOCENV function returns the locale/language environment value for a valid SAS locale. The following environment values are possible:

SBCS
: The SAS session encoding is SBCS (Single-Byte Character Set). SASWZSD is loaded for string manipulation.

DBCS
: The SAS session encoding is DBCS (Double-Byte Character Set). SASWZSD is loaded for string manipulation.

MBCS
The SAS session encoding is Unicode(UTF8). SASWZSU is loaded for string manipulation.

If you receive a blank value, then the WZSS subsystem is not available. This action suggests a configuration or installation error.

Example

In the following example, the LOCALE= system option is set to French_France.

Statements	Results

option locale=french_france;
environ=getlocenv();
put environ;
	SBCS

GETPXLANGUAGE Function

Returns the current two-letter language code.

Category: Locale

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

GETPXLANGUAGE()

Details

The GETPXLANGUAGE function returns the two-letter language code based on the current value of the LOCALE= SAS system option. The length of the language name is two characters. If the size of the variable that receives the value is less than two characters, the value is truncated.

Example

In the first example, the LOCALE= system option is set to French_France. The second example is set to German. The third example is set to English_United States.

Statements	Results

option locale=french_france;
lang=getpxLanguage();
put lang;
	fr

option locale=German;
lang=getpxLanguage();
put lang;
	de

option locale=en_US;
lang=getpxLanguage();
put lang;
	en

See Also

System Options:

- "LOCALE System Option" on page 578

Functions:

- "GETPXREGION Function" on page 325
- "GETPXLOCALE Function" on page 324

GETPXLOCALE Function

Returns the POSIX locale value for a SAS locale.

Category: Locale

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

GETPXLOCALE(*<source>*)

Required Argument

<source>
is an optional argument that specifies a locale name.

Details

The GETPXLOCALE function returns the POSIX locale value for a valid SAS locale name. If you specify an invalid locale name, then a null string is returned. If you do not specify a value for the *<source>* argument, then the function returns the POSIX name for the current SAS session. The length of the POSIX locale name is five characters. If the size of the variable that receives the value is less than five characters, the value is truncated.

Example

In the first example, the LOCALE= system option is set to French_France. In the second example, the <*source*> argument is set to German_Germany. In the third example, the <*source*> argument is set to English_United States.

Statements	Results

option locale=french_france;
locale=getpxLocale();
put locale;
	fr_FR

locale=getpxLocale("german_germany"); put locale;	de_DE
locale=getpxLocale("english_unitedstates"); put locale;	en_US

See Also

System Options:

- "LOCALE System Option" on page 578

Functions:

- "GETPXLANGUAGE Function" on page 323
- "GETPXREGION Function" on page 325

GETPXREGION Function

Returns the current two-letter region code.

Category: Locale

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

GETPXREGION()

Details

The GETPXREGION function returns the two-letter region code based on the current LOCALE= SAS system option. The length of the region name is two characters. If the size of the variable that receives the value is less than two characters, the value is truncated.

Example

In the first example the LOCALE= system option is set to French_France. The second example is set to German. The third example is set to English_United States.

Statements	Results

option locale=french_france;
region=getpxRegion();
put region;
	FR

option locale=German;
region=getpxRegion();
put region;
	DE

option locale=en_US;
region=getpxRegion();
put region;
	US

See Also

System Options:

- "LOCALE System Option" on page 578

Functions:

- "GETPXLOCALE Function" on page 324
- "GETPXLANGUAGE Function" on page 323

KCOMPARE Function

Returns the result of a comparison of character expressions.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Tip: Non-DBCS equivalent function is the "COMPARE Function" in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KCOMPARE(*source*,<*pos*, <*count*,> > *findstr*)

Required Arguments

source
specifies the character expression to be compared.

pos
specifies the starting position in *source* to begin the comparison. If *pos* is omitted, the entire *source* is compared. If *pos* is less than 0, *source* is assumed as extended DBCS data that does not contain any SO/SI characters.

count
specifies the number of bytes to compare. If *count* is omitted, all of *source* that follows *pos* is compared, except for any trailing blanks.

findstr
specifies the character expression to compare to *source*.

Details

KCOMPARE returns values as follows:

- a negative value if *source* is less than *findstr*
- 0 if *source* is equal to *findstr*
- a positive value if *source* is greater than *findstr*

Example

The following example uses Japanese characters.

Statements

```
options pageno=1 nodate ls=80 ps=60;

data test;
  rc1 = kcompare('漢字のテスト','漢字');
  rc2 = kcompare('漢字のテスト',7,'テスト');
  rc3 = kcompare('漢字のテスト',1,6,'漢字');
  rc4 = kcompare('漢字のテスト','ABC');
  rc5 = kcompare('ABCDEF','ABC');
run;

proc print data=test; run;
```

Results

The SAS System

Obs	rc1	rc2	rc3	rc4	rc5
1	3	0	3	1	4

KCOMPRESS Function

Removes specified characters from a character expression.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is COMPRESS in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KCOMPRESS(*source*,<*characters-to-remove*>)

Required Arguments

source
specifies a character expression that contains the characters to be removed. When only *source* is specified, KCOMPRESS returns this expression with all of the single and double-byte blanks removed.

characters-to-remove
specifies the character or characters that KCOMPRESS removes from the character. If characters-to-remove is omitted, KCOMPRESS removes all blanks.expression.

Tip Enclose a literal string of characters in quotation marks.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  result=kcompress('漢字のテスト','の');`
`  put result=;`
`run;`
	`result=漢字テスト`

See Also

Functions:

- "KLEFT Function" on page 350
- "KTRIM Function" on page 377

KCOUNT Function

Returns the number of double-byte characters in an expression.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Syntax

KCOUNT(*source*)

Required Argument

source
specifies the character expression to count.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  text='abcあいう';`
`  result=kcount(text);`
`  put result=;`
`run;`
	`result=3`

`data casuser.encodcompat;`
`text='abcわいう';`
`result=kcount(text);`
`put result=;`
`run;`
	`result=3`

KCOUNTC Function

Counts the number of words in a character string.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and is designed for DBCS data. However, if the first argument, *string*, has single-byte characters, then the KCOUNTC function processes the SBCS data. For more information, see Internationalization Compatibility on page 279.

Syntax

KCOUNTC(*string*,*character-list* <, *modifier(s)*>)

Required Arguments

string

specifies a character constant, variable, or expression in which characters are counted.

Tip Enclose a literal string of characters in quotation marks.

character-list

specifies a character constant, variable, or expression that initializes a list of characters. The KCOUNTC function counts characters in this list, provided that you do not specify the V modifier in the *modifier* argument. If you specify the V modifier, then all characters that are not in this list are counted. You can add more characters to the list by using other modifiers.

Tips Enclose a literal string of characters in quotation marks.

If there are no characters in the list after processing the modifiers, the KCOUNTC function returns a 0.

Optional Argument

modifier(s)

specifies a character constant, variable, or expression in which each non-blank character modifies the actions of the KCOUNTC function. Blanks are ignored. The following characters, in uppercase or lowercase, can be used as modifiers:

blank	is ignored.
a or A	adds alphabetic characters to the list of characters.
b or B	scans *string* from right to left instead of from left to right.
c or C	adds control characters to the list of characters.
d or D	adds digits to the list of characters.
f or F	adds an underscore and English letters (that is, the characters that can begin a SAS variable name using VALIDVARNAME=V7) to the list of characters.
g or G	adds graphic characters to the list of characters.
h or H	adds a horizontal tab to the list of characters.
i or I	ignores case.
l or L	adds lowercase letters to the list of characters.
n or N	adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name using VALIDVARNAME=V7) to the list of characters.
o or O	processes the *character-list* and *modifier* arguments only once, at the first call to this instance of the KCOUNTC function. If you change the value of *character-list* or *modifier* in subsequent calls, the change might be ignored by the KCOUNTC function.

p or P adds punctuation marks to the list of characters.

s or S adds space characters to the list of characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed).

t or T trims trailing blanks from *string* and *character-list*. If you want to remove trailing blanks from only one character argument instead of both (or all) character arguments, use the TRIM function instead of the KCOUNTC function with the T modifier.

u or U adds uppercase letters to the list of characters.

v or V counts characters that do not appear in the list of characters. If you do not specify this modifier, then the KCOUNTC function counts characters that do appear in the list of characters.

w or W adds printable characters to the list of characters.

x or X adds hexadecimal characters to the list of characters.

Tip If *modifier* is a constant, enclose it in quotation marks. Specify multiple constants in a single set of quotation marks.

Details

The Basics

The KCOUNTC function allows character arguments to be null. Null arguments are treated as character strings with a length of zero. If there are no characters in the list of characters to be counted, the KCOUNTC function returns a zero.

Comparisons

The KCOUNTC function counts individual characters in a character string, whereas the KCOUNTX function counts substrings of characters in a character string.

Example

This example uses Chinese characters:

```
data _null_;
xyz='SAS是全球最大的软件公司之一，是全球商业智能和分析软件与服务领袖。';
howmanythis=kcountc(xyz, '软件'); /* Count '软' and '件' occurrence */
put howmanythis = ;
run;
```

Here is the result from the KCOUNTC function example:

```
howmanythis=4
```

KCOUNTW Function

Counts the number of words in a character string.

Categories: DBCS

CAS

Restriction: This function is assigned an I18N Level 2 status and is designed for DBCS data. However, if the first argument, *string*, has single-byte characters, then the KCOUNTW function processes the SBCS data. For more information, see Internationalization Compatibility on page 279.

Syntax

KCOUNTW(<*string*><,<*character-list*> <,<*modifier(s)*>>>)

Optional Arguments

string
specifies a character constant, variable, or expression in which words are counted.

character-list
specifies an optional character constant, variable, or expression that initializes a list of characters. The characters in this list are delimiters that separate words. However, if you specify the K modifier in the *modifier* argument, then all characters, including the characters in the list, are treated as delimiters. To add more characters to the list, use other modifiers. *Delimiter* refers to any of several characters that you can specify to separate words.

Tip Character arguments can be null. Null arguments are treated as character strings with a length of zero. Numeric arguments cannot be null.

modifier
specifies a character constant, variable, or expression in which each non-blank character modifies the actions of the KCOUNTW function. The following characters can be used as modifiers:

blank	is ignored.
a or A	adds alphabetic characters to the list of characters.
b or B	counts from right to left instead of from left to right. Right-to-left counting makes a difference only when you use the Q modifier and the string contains unbalanced quotation marks.
c or C	adds control characters to the list of characters.
d or D	adds digits to the list of characters.
f or F	adds an underscore and English letters (that is, the characters that can begin a SAS variable name using VALIDVARNAME=V7) to the list of characters.
g or G	adds graphic characters to the list of characters.
h or H	adds a horizontal tab to the list of characters.
i or I	ignores the case of the characters.
k or K	treats all characters that are not in the list as delimiters. If K is not specified, all characters that are in the list are treated as delimiters.
l or L	adds lowercase letters to the list of characters.
m or M	specifies that multiple consecutive delimiters and delimiters at the beginning or end of the *string* argument refer to words that have a length of zero. If the M modifier is not specified, multiple consecutive

delimiters are treated as one delimiter, and delimiters at the beginning or end of the *string* argument are ignored.

n or N — adds digits, an underscore, and English letters (that is, the characters that can appear after the first character in a SAS variable name using VALIDVARNAME=V7) to the list of characters.

o or O — processes the *character-list* and *modifier* arguments only once rather than every time the KCOUNTW function is called. Using the O modifier in the DATA step (excluding WHERE clauses) or in the SQL procedure can make the KCOUNTW function run faster when you call it in a loop where *chars* and *modifier* arguments do not change.

p or P — adds punctuation marks to the list of characters.

q or Q — ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of *string* contains unmatched quotation marks, scanning from left to right produces different words than scanning from right to left.

s or S — adds space characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed) to the list of characters.

t or T — trims trailing blanks from the *string* and *character-list* arguments.

u or U — adds uppercase letters to the list of characters.

w or W — adds printable characters to the list of characters.

x or X — adds hexadecimal characters to the list of characters.

Details

Definition of Word

In the KCOUNTW function, *word* refers to a substring that has one of these characteristics:

- is bounded on the left by a delimiter or the beginning of the string
- is bounded on the right by a delimiter or the end of the string
- contains no delimiters, except if you use the Q modifier and the delimiters are within substrings that have quotation marks.

Using the KCOUNTW Function in ASCII and EBCDIC Environments

If you use the KCOUNTW function with only two arguments, the default delimiters are different depending on whether your computer uses ASCII or EBCDIC characters.

- If your computer uses ASCII characters, here are the default delimiters:

  blank ! $ % & ( ) * + , - . / ; < ^ |

  In ASCII environments that do not contain the ^ character, the SCAN function uses the ~ character instead.

- If your computer uses EBCDIC characters, here are the default delimiters:

  blank ! $ % & ( ) * + , - . / ; < ¬ | ¢

Using the M Modifier

If you do not use the M modifier, then a word must contain at least one character. If you use the M modifier, then a word can have a length of zero. In the latter case, the number

of words is one plus the number of delimiters in the string, not the number of delimiters inside strings that are enclosed in quotation marks when you use the Q modifier.

Example

This example uses Chinese characters:

```
data _null_;
xyz='SAS是全球最大的软件公司之一，是全球商业智能和分析软件与服务领袖。';
howmanythis=kcountw(xyz, ' ', 'p'); /* Count words, use punctuations as delimiter. */
put howmanythis = ;
run;
```

Here is the result from the KCOUNTW function example:

```
howmanythis=2
```

KCOUNTX Function

Counts the number of times that a specified substring appears within a character string.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and is designed for DBCS data. However, if the first argument, *string*, has single-byte characters, then the KCOUNTX function processes the SBCS data. For more information, see Internationalization Compatibility on page 279.

Syntax

KCOUNTX(*string*,*substring* <,*modifier(s)*>)

Required Arguments

string
specifies a character constant, variable, or expression in which substrings are counted.

Tip Enclose a literal string of characters in quotation marks.

substring
is a character constant, variable, or expression that specifies the substring of characters to count in *string*.

Tip Enclose a literal string of characters in quotation marks.

Optional Argument

modifier

is a character constant, variable, or expression that specifies one or more modifiers. The following *modifiers* can be in uppercase or lowercase:

i ignores character case during the count. If this modifier is not specified, the KCOUNTX function counts only character substrings with the same case as the characters in *substring*.

t trims trailing blanks from *string* and *substring*.

Tip If the *modifier* is a constant, enclose it in quotation marks. Specify multiple constants in a single set of quotation marks. *Modifier* can also be expressed as a variable or as an expression.

Details

The Basics

The KCOUNTX function searches *string*, from left to right, for the number of occurrences of the specified *substring* and returns that number of occurrences. If the substring is not found in *string*, the KCOUNTX function returns a value of 0.

CAUTION:

If two occurrences of the specified substring overlap in the string, the result is undefined. For example, `count('boobooboo', 'booboo')` might return either a 1 or a 2.

Example

This example uses Chinese characters:

```
data _null_;
xyz='SAS是全球最大的软件公司之一，是全球商业智能和分析软件与服务领袖。';
howmanythis=kcountx(xyz, '软件'); /* Count '软件' occurrence */
put howmanythis = ;
run;
```

Here is the result from the KCOUNTX function example:

```
howmanythis=2
```

KCVT Function

Converts data from one type of encoding data to another type of encoding data.

Category: Character

Restriction: This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility on page 279.

Syntax

KCVT(*text*, *intype*, *outtype*, <*options*,…>)

Required Arguments

text
specifies the character variable to be converted.

intype
specifies the encoding of the data. The encoding of the text must match the input data's encoding. For valid values, see "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

ASCIIANY and EBCIDICANY are invalid encoding values.

outtype
specifies the encoding to be converted into character data. For valid values, see "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

ASCIIANY and EBCIDICANY are invalid encoding values.

options
specifies character data options. Here are the available options:

NOSOSI | NOSHIFT	No shift code or Hankaku characters.
INPLACE	Replaces character data by conversion. The INPLACE option is specified to secure the same location between different hosts whose lengths of character data are not identical. For example, the INPLACE option converts data from the host that requires Shift-Codes into the other host, which does not require Shift Codes. Truncation occurs when the length of the character data that is converted into *outtype* for Shift-Codes is longer than the length that is specified in *intype*.
KANA	Includes Hankaku katakana characters in columns of character data.
UPCASE	Converts a 2-byte alphabet to uppercase characters.
LOWCASE	Converts a 2-byte alphabet to lowercase characters.
KATA2HIRA	Converts katakana data to hiragana.
HIRA2KATA	Converts hiragana data to katakana.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KCVT function converts SBCS, DBCS, and MBCS character strings into encoding data. For example, the KCVT function can convert the following: ASCII code data to UCS2 encoding data, Greek code data to UTF-8, and Japanese SJIS code data to another Japanese code data. You can specify the following types for the INTYPE and OUTTYPE options: UCS2, UCS2L, UCS2B, and UTF8. To enable the DBCS mode, specify the DBCS option in the configuration file or from the command line.

If the KCVT function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

Example

The following code converts IBM PC codes into DEC codes for the external text file specified as *my-input-file* and writes the results in OUTDD.

```
data _null_;
   infile 'my-input-file';
   file outdd noprint;
   input @1 text $char80.;
   text = kcvt(text, 'pcibm', 'dec');
   put @1 text $char80.;
run;
```

KFIND Function

Searches for a specific substring of characters within a character string.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and is designed for DBCS data. However, if the first argument, *string,* has multi-byte characters, then the KFIND function processes the multi-byte data. For more information, see Internationalization Compatibility on page 279

Syntax

KFIND(*string*,*substring* <,*start-position*> <,*modifier(s)*>)

Required Arguments

string
specifies a character constant, variable, or expression that is searched for substrings.

Tip Enclose a literal string of characters in quotation marks.

substring
is a character constant, variable, or expression that specifies the substring of characters to search for in *string*.

Tip Enclose a literal string of characters in quotation marks.

Optional Arguments

modifier(s)
is a character constant, variable, or expression that specifies one or more modifiers. The following modifiers are valid:

t or T
trims trailing blanks from *string* and *substring*.

Note: If you want to remove trailing blanks from only one character argument instead of both (or all) character arguments, use the TRIM function instead of the KFIND function with the T modifier.

TIP If *modifier* is a constant, enclose it in quotation marks. Specify multiple constants in a single set of quotation marks. *Modifier* can also be expressed as a variable or as an expression.

start-position
is a numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction of the search.

Details

The Basics

The KFIND function searches *string* for the first occurrence of the specified *substring*, and returns the position of that substring. If the substring is not found in *string*, the KFIND function returns a value of 0.

If *start-position* is not specified, the KFIND function starts the search at the beginning of the *string* and searches the *string* from left to right. If *start-position* is specified, the absolute value of *start-position* determines the position at which to start the search. The sign of *start-position* determines the direction of the search.

Value of *startpos*	Action
Greater than 0	Starts the search at position *start-position* and searches to the right. If *start-position* is greater than the length of *string*, KFIND returns a value of 0.
Less than 0	Starts the search at position *start-position* and searches to the left. If *start-position* is greater than the length of *string*, the search starts at the end of *string*.
Equal to 0	Returns a value of 0.

Example

This example uses Chinese characters:

```
data _null_;
xyz='SAS是全球最大的软件公司之一，是全球商业智能和分析软件与服务领袖。';
pos_of=kfind(xyz, '软件');  /* Find first position of '软件' */
put pos_of = ;
run;
```

Here is the result from the KFIND function example:

```
pos_of=10
```

KFINDC Function

Searches a string for any character in a list of characters.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and is designed for DBCS data. However, if the first argument, *string*, has single-byte characters, then the KFINDC function processes the SBCS data. For more information, see Internationalization Compatibility on page 279

Note: This function supports the varchar type.

Syntax

KFINDC(*string <,character-list>*)

KFINDC(*string,character-list<,modifier(s)>*

KFINDC(*string,character-list,modifier(s) <,start-position>*)

KFINDC(*string,character-list,<start-position>,<modifier(s)>*)

Required Arguments

string
is a character constant, variable, or expression that specifies the character string to search.

Tip Enclose a literal string of characters in quotation marks.

character-list
is a constant, variable, or character expression that initializes a list of characters. The KFINDC function searches for the characters in this list provided that you do not specify the K modifier in the *modifier* argument. If you specify the K modifier, the KFINDC function searches for all characters that are not in this list of characters. You can add more characters to the list by using other modifiers.

modifier(s)
is a character constant, variable, or expression in which each character modifies the actions of the KFINDC function. The following characters, in uppercase or lowercase, can be used as modifiers:

blank
is ignored.

a or A
adds alphabetic characters to the list of characters.

b or B
searches from right to left instead of from left to right, regardless of the sign of the *start-position* argument.

c or C
adds control characters to the list of characters.

d or D
adds digits to the list of characters.

f or F
adds an underscore and English letters (that is, the characters that can begin a SAS variable name using VALIDVARNAME=V7) to the list of characters.

g or G
adds graphic characters to the list of characters.

h or H
adds a horizontal tab to the list of characters.

i or I
ignores character case during the search.

k or K
searches for any character that does not appear in the list of characters. If you do not specify this modifier, then the KFINDC function searches for any character that appears in the list of characters. This modifier has the same functionality as the v or V modifier.

l or L
adds lowercase letters to the list of characters.

n or N
adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name using VALIDVARNAME=V7) to the list of characters.

o or O
processes the *character-list* and the *modifier* arguments only once rather than every time the KFINDC function is called. Using the O modifier in the DATA step (excluding WHERE clauses) or in the SQL procedure can make the KFINDC function run faster when you call it in a loop where the *character-list* and the *modifier* arguments do not change.

p or P
adds punctuation marks to the list of characters.

s or S
adds space characters to the list of characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed).

t or T
trims trailing blanks from the *string* and *character-list* arguments.

Note: If you want to remove trailing blanks from just one character argument instead of both (or all) character arguments, use the TRIM function instead of the KFINDC function with the T modifier.

u or U
adds uppercase letters to the list of characters.

v or V
searches for any character that does not appear in the list of characters. If you do not specify this modifier, then KFINDC searches for any character that appears in the list of characters. This modifier has the same functionality as the k or K modifier.

w or W
adds printable characters to the list of characters.

x or X
adds hexadecimal characters to the list of characters.

Tip If *modifier* is a constant, then enclose it in quotation marks. Specify multiple constants in a single set of quotation marks. *Modifier* can also be expressed as a variable or as an expression.

Optional Argument

start-position
is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

Details

The Basics

The KFINDC function searches *string* for the first occurrence of the specified characters and returns the position of the first character found. If no characters are found in *string*, then the KFINDC function returns a value of 0.

The KFINDC function allows character arguments to be null. Null arguments are treated as character strings that have a length of zero. Numeric arguments cannot be null.

If *start-position* is not specified, the KFINDC function begins the search at the end of the string if you use the B modifier or at the beginning of the string if you do not use the B modifier.

If *start-position* is specified, the absolute value of *start-position* specifies the position at which to begin the search. If you use the B modifier, the search always proceeds from right to left. If you do not use the B modifier, the sign of *start-position* specifies the direction in which to search. The following table summarizes the search directions:

Value of *startpos*	Action
Greater than 0	Search begins at position *start-position* and proceeds to the right. If *start-position* is greater than the length of the string, the KFINDC function returns a value of 0.
Less than 0	Search begins at position *start-position* and proceeds to the left. If *start-position* is less than the negative of the length of the string, the search begins at the end of the string.
Equal to 0	Returns a value of 0.

Example

Here is an example of the KFINDC function:

```
data _null_;
xyz='SAS是全球最大的软件公司之一，是全球商业智能和分析软件与服务领袖。';
pos_of=kfindc(xyz, '软件的');  /* Find minimum position of '软', '件' or'的' */
put pos_of = ;
run;
```

Here is the result from the KFINDC function example:

```
pos_of=9
```

KFINDW Function

Returns the character position of a word in a string or returns the number of the word in a string.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and is designed for DBCS data. However, if the first argument, *string*, has single-byte characters, then the KFINDW function processes the SBCS data. For more information, see Internationalization Compatibility on page 279.

Syntax

KFINDW(*string*,*word* <,*character-list*)>

KFINDW(*string*,*word*,*character-list*,*modifier(s)* <,*start-position*>)

KFINDW(*string*,*word*,*character-list*,*start-position* <,*modifier(s)*>)

KFINDW(*string*,*word*,*start-position* <,*character-list* <,*modifier(s)*>>)

Required Arguments

string
is a character constant, variable, or expression that specifies which character string to search.

word
is a character constant, variable, or expression that specifies the word to search for in *string*.

character-list
is an optional character constant, variable, or expression that initializes a list of characters.

The characters in this list are the delimiters that separate words, provided that you do not specify the K modifier in the *modifier* argument. If you specify the K modifier, then all characters that are not in this list are delimiters. You can add more characters to this list by using other modifiers.

start-position
is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should begin and the direction in which to search.

modifier
specifies a character constant, variable, or expression in which each non-blank character modifies the actions of the KFINDW function.

TIP If you use the *modifier* argument, then it must be positioned after the *character-list* argument.

You can use the following characters as modifiers:

blank
is ignored.

a or A
adds alphabetic characters to the list of characters.

b or B
scans from right to left instead of from left to right, regardless of the sign of the *start-position* argument.

c or C
adds control characters to the list of characters.

d or D
adds digits to the list of characters.

e or E
counts the words that are scanned until the specified word is found instead of determining the character position of the specified word in the string. Fragments of a word are not counted.

f or F
adds an underscore and English letters (that is, the characters that can begin a SAS variable name using VALIDVARNAME=V7) to the list of characters.

g or G
adds graphic characters to the list of characters.

h or H
adds a horizontal tab to the list of characters.

i or I
ignores the case of the characters.

k or K
treats all characters that are not in the list of characters as delimiters. If K is not specified, then all characters that are in the list of characters are treated as delimiters.

l or L
adds lowercase letters to the list of characters.

m or M
specifies that multiple consecutive delimiters and delimiters at the beginning or end of the *string* argument refer to words that have a length of zero.

n or N
adds digits, an underscore, and English letters (that is, the characters that can appear after the first character in a SAS variable name using VALIDVARNAME=V7) to the list of characters.

o or O
processes the *character-list* and *modifier* arguments only once rather than every time the KFINDW function is called. Using the O modifier in the DATA step (excluding WHERE clauses) or in the SQL procedure can make the KFINDW function run faster when you call it in a loop where the *character-list* and *modifier* arguments do not change.

p or P
adds punctuation marks to the list of characters.

q or Q
ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of the *string* argument contains unmatched quotation marks, then

scanning from left to right produces different words than scanning from right to left.

r or R
removes leading and trailing delimiters from the *word* argument.

s or S
adds space characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed) to the list of characters.

t or T
trims trailing blanks from the *string*, *word*, and *character-list* arguments.

u or U
adds uppercase letters to the list of characters.

w or W
adds printable characters to the list of characters.

x or X
adds hexadecimal characters to the list of characters.

Details

Definition of Delimiter

Delimiter refers to any of several characters that are used to separate words. You can specify the delimiters by using the *character* argument, the *modifier* argument, or both. If you specify the Q modifier, then the characters inside substrings that are enclosed in quotation marks are not treated as delimiters.

Definition of Word

Word refers to a substring that has both of the following characteristics:

- bounded on the left by a delimiter or the beginning of the string
- bounded on the right by a delimiter or the end of the string

Note: A word can contain delimiters. In this case, the KFINDW function differs from the SCAN function, in which words are defined as containing no delimiters.

Searching for a String

If the KFINDW function fails to find a substring that both matches the specified word and satisfies the definition of a word, then the KFINDW function returns a value of 0.

If the KFINDW function finds a substring that both matches the specified word and satisfies the definition of a word, the value that is returned by the KFINDW function depends on whether the E modifier is specified:

- If you specify the E modifier, then the KFINDW function returns the number of complete words that were scanned while searching for the specified word. If *start-position* specifies a position in the middle of a word, then that word is not counted.
- If you do not specify the E modifier, then the KFINDW function returns the character position of the substring that is found.

If you specify the *start-position* argument, then the absolute value of *start-position* specifies the position at which to begin the search. The sign of *start-position* specifies the direction in which to search:

Value of *startpos*	Action
Greater than 0	Search begins at position *start-position* and proceeds to the right. If *start-position* is greater than the length of the string, then the KFINDW function returns a value of 0.
Less than 0	Search begins at position *start-position* and proceeds to the left. If *start-position* is less than the negative of the length of the string, then the search begins at the end of the string.
Equal to 0	The KFINDW function returns a value of 0.

If you do not specify the *start-position* argument or the B modifier, then the KFINDW function searches from left to right starting at the beginning of the string. If you specify the B modifier, but do not use the *start-position* argument, then the KFINDW function searches from right to left starting at the end of the string.

Using the KFINDW Function in ASCII and EBCDIC Environments

If you use the KFINDW function with only two arguments, the default delimiters depend on whether your computer uses ASCII or EBCDIC characters.

- If your computer uses ASCII characters, then the default delimiters are as follows:

  blank ! $ % & ( ) * + , - . / ; < ^ |

  In ASCII environments that do not contain the ^ character, the KFINDW function uses the ~ character instead.

- If your computer uses EBCDIC characters, then the default delimiters are as follows:

  blank ! $ % & ( ) * + , - . / ; < ¬ | ¢

Using Null Arguments

The KFINDW function allows character arguments to be null. Null arguments are treated as character strings with a length of zero. Numeric arguments cannot be null.

Processing SBCS and DBCS Data

The KFINDW function is designed to process SBCS data, but it can process DBCS data with certain conditions. Here are the criteria for SBCS and DBCS processing:

- If *string* is declared as varchar and you are processing multi-byte data, then the KFINDW function processes DBCS.
- If *string* is not declared as varchar or you are processing single-byte data, then the KFINDW function processes SBCS.

Example

Here is an example of the KFINDW function:

```
data _null_;
xyz='SAS是全球最大的 软件 公司之一，SAS是全球商业智能和分析软件与服务领袖。';
pos_of=kfindw(xyz, '软件'); /* The default delimiter is SBCS blank. */
put pos_of = ;
run;
```

Here is the result from the KFINDW function example:

```
pos_of=11
```

Here is another example of the KFINDW function:

```
data _null_;
xyz='SAS是全球最大的软件公司之一，sas  是全球商业智能和分析软件与服务领袖。';
pos_of=kfindw(xyz, 'sas', '', 'sp'); /* Use spaces and punctuations as delimiter. */
put pos_of = ;
run;
```

Here is the result from the KFINDW function example:

```
pos_of=17
```

KINDEX Function

Searches a character expression for a string of characters.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is INDEX in *SAS Viya Functions and CALL Routines: Reference*

Syntax

KINDEX(*source*,*excerpt*)

Required Arguments

source
specifies the character expression to search.

excerpt
specifies the string of characters to search for in the character expression.

Tip Enclose a literal string of characters in quotation marks.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KINDEX function searches *source*, from left to right, for the first occurrence of the string that is specified in *excerpt*, and returns the position in *source* of the string's first

character. If the string is not found in *source*, the KINDEX function returns a value of 0. If there are multiple occurrences of the string, the KINDEX function returns only the position of the first occurrence.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  text='漢字文字列の検索';`
`  result=kindex(text,'検索');`
`  put result=;`
`  text2='探す';`
`  result=kindex(text,text2);`
`  put result=;`
`run;`
	`result=7`
	`result=0`

See Also

Functions:

- "KINDEXC Function" on page 348

KINDEXB Function

Searches a character expression for a string of characters.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

KINDEXB(*source*, *excerpt*)

Required Arguments

source
specifies the character expression to search for.

excerpt
specifies the string of characters to search for in the character expression.

TIP Enclose a literal string of characters in quotation marks.

Details

The KINDEXB function searches *source*, from left to right, for the first occurrence of the string that is specified in *excerpt* and returns the string's first character position in *source*. If the string is not found in *source*, KINDEXB returns a value of 0. If there are multiple occurrences of the string, KINDEXB returns only the position of the first occurrence.

Comparisons

KINDEXB returns byte-based values. KINDEX returns character-based values. Most Latin characters are 2 bytes wide and CJK characters are 3 bytes wide, that makes these functions return different results.

The KINDEXCB function searches for the first occurrence of any individual character that is present within the character string. The KINDEXB function searches for the first occurrence of the character string as a pattern.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  text='漢字文字列の検索';`
`  result=kindexb(text, '検索');`
`  put result=;`
`  text2='探す';`
`  result=kindexb(text,text2);`
`  put result=;`
`run;`
	`Result=19`
	`Result=0`

KINDEXC Function

Searches a character expression for specified characters.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: The Non-DBCS equivalent function is "INDEXC Function" in *SAS Viya Functions and CALL Routines: Reference*

Syntax

KINDEXC(*source*, *excerpt-1, ...excerpt-n*)

Required Arguments

source
specifies the character expression to search for.

excerpt-1, ...excerpt-n
specifies the characters to search for in the character expression.

Tips If you specify more than one excerpt, separate them with a comma.

Enclose a literal string of characters in quotation marks.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KINDEXC function searches *source*, from left to right, for the first occurrence of any character present in the excerpts and returns the position in *source* of that character. If none of the characters in *excerpt-1* through *excerpt-n* in *source* are found, KINDEXC returns a value of 0.

Comparisons

The KINDEXC function searches for the first occurrence of any individual character that is present within the character string, whereas the KINDEX function searches for the first occurrence of the character string as a pattern.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  a='ＡＢＣ.def (漢字123456)';`
`  result=kindexc(a,'1234567890','感漢');`
`  put result=;`
`run;`
	`result=9`

See Also

Function:

- "KINDEX Function" on page 346

KINDEXCB Function

Searches a character expression for specified characters.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

KINDEXCB(*source*, *excerpt-1*, …*excerpt-n*)

Required Arguments

source
specifies the character expression to search for.

excerpt-1, ...excerpt-n
specifies the characters to search for in the character expression.

Tips If you specify more than one excerpt, separate them with a comma.

Enclose a literal string of characters in quotation marks.

Details

The KINDEXCB function searches the data in *source*, from left to right, for the first occurrence of any character in the excerpts' arguments and returns the position in *source* of that character. If none of the characters in *excerpt-1* through *excerpt-n* in *source* are found, KINDEXCB returns a value of 0.

Comparisons

KINDEXCB returns byte-based values. KINDEXC returns character-based values. Most Latin characters are 2 bytes wide and CJK characters are 3 bytes wide, that makes these functions return different results.

The KINDEXCB function searches for the first occurrence of any individual character that is present within the character string, whereas the KINDEXB function searches for the first occurrence of the character string as a pattern.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  a='ＡＢＣ.def (漢字123456)';`
`  result=kindexcb(a, '1234567890','感漢');`
`  put result=;`
`run;`
	`Result=19`

KLEFT Function

Left-aligns a character expression by removing unnecessary leading DBCS blanks and SO/SI.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is LEFT in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KLEFT(*argument*)

Required Argument

argument
specifies any SAS character expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

KLEFT returns an argument and removes the leading blanks.

The KLEFT, KRIGHT, and KTRIM functions remove the following Unicode space characters.

Table 12.6 *Unicode Spaces That are Removed by KLEFT, KRIGHT, and KTRIM*

Unicode spaces definition in ANYSPACE	Unicode spaces that are removed by KLEFT, KRIGHT and KTRIM	Comment
	U+0009	CHARACTER TABULATION
	U+000A	LINE FEED
	U+000B	LINE TABULATION
	U+000C	FORM FEED
	U+000D	CARRIAGE RETURN
U+0020	U+0020	SINGLE-BYTE SPACE, DEPEND ON COMPILING
	U+0085	NEXT LINE
U+00A0	U+00A0	NO-BREAK SPACE
U+1680	U+1680	OGHAM SPACE MARK

Unicode spaces definition in ANYSPACE	Unicode spaces that are removed by KLEFT, KRIGHT and KTRIM	Comment
U+2000	U+2000	EN QUAD
U+2001	U+2001	EM QUAD
U+2002	U+2002	EN SPACE
U+2003	U+2003	EM SPACE
U+2004	U+2004	THREE-PER-EM SPACE
U+2005	U+2005	FOUR-PER-EM SPACE
U+2006	U+2006	SIX-PER-EM SPACE
U+2007	U+2007	FIGURE SPACE
U+2008	U+2008	PUNCTUATION SPACE
U+2009	U+2009	THIN SPACE
U+200A	U+200A	HAIR SPACE
U+2028	U+2028	U+2028 LINE SEPARATOR
U+2029	U+2029	U+2029 PARAGRAPH SEPARATOR
U+202f,	U+202F	NARROW NO-BREAK SPACE
U+205f,	U+205F	MEDIUM MATHEMATICAL SPACE
U+3000	U+3000	IDEOGRAPHIC SPACE

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  a=' 漢字 の テスト';`
`  b=kleft(a);`
`  put '----+----1----+----2----+';`
`  put a $25.;`
`  put b $25.;`
`run;`
	`----+----1----+----2----+`
	` 漢字 の テスト`
	`漢字 の テスト`

See Also

Functions:

- "KCOMPRESS Function" on page 327
- "KRIGHT Function" on page 362
- "KTRIM Function" on page 377

KLENGTH Function

Returns the length of an argument.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is LENGTH in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KLENGTH(*argument*)

Required Argument

argument
specifies any SAS expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KLENGTH function returns an integer that represents the position of the rightmost non-blank character in the argument. If the value of the argument is missing, KLENGTH returns a value of 0. If the argument is an uninitialized numeric variable, KLENGTH

returns a value of 12 and prints a note in the SAS log that the numeric values have been converted to character values.

If you are using SBCS EnglishVARCHAR data, performance issues might occur. To improve the performance, use the following code with non-K functions.

This code improves performance and the results reflect the KLENGTH function's behavior.

```
len = LENGTHC(TRIMN(x));
```

This code improves performance and the results reflect the LENGTH function's behavior.

```
len = LENGTHC(TRIMN(x));
   if NOT len then len = 1;
```

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  text='abcあいう';`
`  result=klength(text);`
`  put result=;`
`run;`
	`result=6`

KLOWCASE Function

Converts all single-width English alphabet letters in an argument to lowercase.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is LOWCASE in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KLOWCASE(*argument*)

Required Argument

argument
specifies any SAS character expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KLOWCASE function copies a character argument, converts all uppercase single-width English alphabet letters to lowercase letters, and returns the altered value as a result.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  result=klowcase('ABCあいうえお');`
`  put result=;`
`run;`
	`result=abcあいうえお`

KPROPCASE Function

Converts Chinese, Japanese, Korean, Taiwanese (CJKT) characters.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

str=**KPROPCASE**(<*instr*> , (<options>))

Required Arguments

str
data string that has been converted and is in the current SAS session encoding.

instr
input data string.

options
converts Japanese, Chinese, Korean, and Taiwanese characters based on specified options.

HALF-KATAKANA, FULL-KATAKANA
This option converts half-width katakana to full-width katakana and is used only with Japanese encoding.

Restriction This option cannot be used at the same time with the full-Katakana, half-Katakana option.

FULL-KATAKANA, HALF-KATAKANA

This option converts full-width katakana to half-width katakana and is used only with Japanese encoding.

Restriction This option cannot be used at the same time with the half-Katakana, full-Katakana option.

KATAKANA, ROMAJI

This option converts the katakana character string to a romaji character string and is used only with Japanese encoding.

Restriction This option cannot be used at the same time with the romaji, katakana option.

ROMAJI, KATAKANA

This option converts the romaji character string to a katakana character string and is used only with Japanese encoding.

Restriction This option cannot be used at the same time with the katakana, romaji option.

FULL-ALPHABET, HALF-ALPHABET

This option converts the Full-Alphabet characters to Half-Alphabet characters and is used only with Japanese, Chinese, Korean, and Taiwanese encoding.

Restriction This option cannot be used at the same time with the Half-Alphabet, Full-Alphabet option.

HALF-ALPHABET, FULL-ALPHABET

This option converts the Half-Alphabet characters to Full-Alphabet characters and is used only with Japanese, Chinese, Korean, and Taiwanese encoding.

Restriction This option cannot be used at the same time with the Full-Alphabet, Half-Alphabet option.

LOWERCASE, UPPERCASE

This option converts lowercase alphabet characters to uppercase alphabet characters.

Restriction This option cannot be used at the same time with the Uppercase, Lowercase option.

UPPERCASE, LOWERCASE

This option converts uppercase alphabet characters to lowercase alphabet characters.

Restriction This option cannot be used at the same time with the Lowercase, Uppercase option.

PROPER

This option specifies the following default options based on the encoding:

- Japanese encoding
- Half-Katakana,Full-Katakana
- Full-alphabet, Half-alphabet
- Lowercase, Uppercase

- Korean encoding:
- Full-alphabet, Half-alphabet
- Chinese encoding:
- Full-alphabet, Half-alphabet
- Taiwanese encoding:
- Full-alphabet, Half-alphabet

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

This function converts the input string based on the specified options and default options. The KPROPCASE function supports the Chinese, Japanese, Korean, Taiwanese (CJKT) environment.

Example

The following example demonstrates the functionality of the KPROPCASE function:

```
length fullkana halfkana upper lower fullalpha $ 200;
length str1 str2 str3 str4 str5 str7 str8 $ 30 str6 $44;
lower = 'do-naxtutsu';  /* Doughnuts in Japanese Roman word. */
upper = 'DO-NAXTUTSU';  /* Doughnuts in Japanese Roman word. */
fullkana = unicode('\u30C9\u30FC\u30CA\u30C3\u30C4');
halfkana = unicode('\uFF84\uFF9E\uFF70\uFF85\uFF6F\uFF82');
fullalpha = unicode('\uFF24\uFF2F\uFF0D\uFF2E\uFF21\uFF38\uFF34\uFF35\uFF34\
uFF33\uFF35');
str1 = kpropcase(fullkana, 'full-katakana,half-katakana');
if (halfkana EQ trim(str1)) then
  put str1= $hex14.;
str2 = kpropcase(halfkana, 'half-katakana, full-katakana');
if (fullkana EQ trim(str2)) then
  put str2= $hex22.;
str3 = kpropcase(fullkana, 'katakana,romaji');
if (trim(str3) EQ upper) then
  put str3=  ;
str4 = kpropcase(upper, 'romaji,katakana');
if (trim(str4) EQ fullkana) then
  put str4= $hex22.;
str5 = kpropcase(fullalpha, 'full-alphabet, half-alphabet');
if (trim(upper) EQ str5) then
  put str5=;
str6 = kpropcase(upper, 'half-alphabet, full-alphabet');
if (trim(str6) EQ fullalpha) then
  put str6= $hex46.;
str7 = kpropcase(lower, 'lowercase, uppercase');
if (trim(str7) EQ upper) then
  put str7=;
str8 = kpropcase(upper, 'uppercase, lowercase');
if (trim(str8) EQ lower) then
 put str8=;
RESULTS:
str1=C4DEB0C5AFC220
```

```
str2=8368815B83698362836320
str3=DO-NAXTUTSU
str4=8368815B83698362836320
str5=DO-NAXTUTSU
str6=8263826E817C826D82608277827382748273827282742 0
str7=DO-NAXTUTSU
str8=do-naxtutsu
```

KPROPCHAR Function

Converts special characters to normal characters.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

str=**KPROPCHAR**(<*instr*>)

Required Arguments

str
result string. Special characters are converted to normal characters.

instr
input data string.

Details

This function converts special characters to normal characters. The KPROPCHAR function converts the characters from the following ranges:

- Enclosed alphanumeric values: \u2460 to \u24FF. See **`http://www.unicode.org/charts/PDF/U2460.pdf`**.
- Dingbats: \u2776 to \u2793. See **`http://www.unicode.org/charts/PDF/U2700.pdf`**.
- Enclosed CJK letters and months: \u3200 to \u32FF. See **`http://www.unicode.org/charts/PDF/U3200.pdf`**.

Example

The following example demonstrates the functionality of the KPROPCHAR function. These examples are processed in a UTF8 SAS session:

```
length in1 out1 $30 ;
  in1=unicode('\u2460\u2473\u277F\u325F');
  out1=KPROPCHAR(in1);
  put out1;
RESULTS:
(1)(20)(-10)(35)
```

Statements	Results

`data _null_;`
`length in1 out1 $30 ;`
`in1=unicode('\u2460\u2473\u277F\u325F');`
`out1=KPROPCHAR(in1);`
`put out1;`
`run;`
	`(1)(20)(-10)(35)`

`data _null_;`
`length in1 out1 $30 ;`
`in1=unicode('\u2776\u2793');`
`out1=KPROPCHAR(in1);`
`put out1;`
`run;`
	`(-1)(-10)`

`data _null_;`
`length in1 out1 $30 ;`
`in1=unicode('\u3200\u32FE');`
`out1=KPROPCHAR(in1);`
`put out1;`
`run;`
	(㈠)(㋾)

KPROPDATA Function

Removes or converts unprintable characters.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

str=**KPROPDATA**(*<instr>* (*<options>*, *<input encode name>*, *<output encode name>*))

Required Arguments

str
data string that has been converted and is in session encoding.

instr
input data string.

options
specifies instructions on processing unprintable characters:

UESC
Converts unprintable characters using a Unicode escaped string (for example, \u0000\u1234).

TRIM
Removes unprintable characters. No replacement character is used.

BLANK or ' '
Replaces each unprintable character with a single-byte blank.

QUESTION or '?'
Replaces unprintable characters with a single-byte '?'.

HEX
Replaces unprintable characters with a hexadecimal representation (for example, 0x810x82).

TRUNCATE or TRUNC
Truncates the data string when the first unprintable character is encountered.

REMOVE
Removes the data string if any unprintable characters are found.

NCR
Encodes the unprintable characters using NCR representation if the code is available in Unicode.

input encode name
specifies the input data's encoding name if necessary. If the input encode name is not specified, then the KPROPDATA function processes the data as the current SAS session encoded string. For information about SAS encoding names, see "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

output encode name
specifies the output data's encoding name. If the encoding name is not specified, the KPROPDATA function recognizes the output as the current SAS session encoding. For information about SAS encoding names, see "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

Details

This function converts the input data string to the current SAS session encoding and removes or replaces unprintable characters based on the options.

Example

The following example demonstrates the functionality of the KPROPDATA function:

```
length instr $12;
  length str1 str2 str3 str4 str5 str6 str7 str8 str9 str10$ 50;
  instr = "534153"x||"ae"x || " System";
  put instr;
  str1 = kpropdata(instr);
  put str1= +2 str1= $hex26.;
  str2 = kpropdata(instr,'UESC');
  put str2= +2 str2= $hex26.;;
  str3 = kpropdata(instr, 'UESC','wlatin1');
  put str3= +2 str3= $hex34.;
  str4 = kpropdata(instr,'TRIM','wlatin1');
  put str4= +2 str4= $hex26.;
  str5 = kpropdata(instr,'BLANK', 'wlatin1');
  put str5= +2 str5= $hex26.;
  str6 = kpropdata(instr,'?', 'wlatin1');
  put str6= +2 str6= $hex26.;
  str7 = kpropdata(instr,'hex', 'wlatin1');
  put str7= +2 str7= $hex26.;
  str8 = kpropdata(instr,'TRUNC', 'wlatin1');
```

```
  put str8= +2 str8= $hex26.;
  str9 = kpropdata(instr,'REMOVE', 'wlatin1');
  put str9= +2 str9= $hex26.;
  str10 = kpropdata(instr,'NCR', 'wlatin1');
  put str10= +2 str10= $hex26.;
RESULTS:
SAS? System
str1=SAS? System   str1=534153AE2053797374656D2020
str2=SAS? System   str2=534153AE2053797374656D2020
str3=SAS\uff6e System   str3=5341535C756666366652053797374656D20
str4=SAS System   str4=5341532053797374656D202020
str5=SAS  System   str5=5341532020537973746 56D2020
str6=SAS? System   str6=5341533F2053797374656D2020
str7=SAS\xAE System   str7=5341535C784145205379737465
str8=SAS   str8=5341532020202020202020202020
str9=    str9=2020202020202020202020202020
str10=SAS® System   str10=53415326233137343B20537973
```

KREVERSE Function

Reverses a character expression.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is REVERSE in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KREVERSE(*argument*)

Required Argument

argument
specifies any SAS character expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  result=kreverse('漢字のテスト');`
`  put result=;`
`run;`
	`result=トステの字漢`

KRIGHT Function

Right-aligns a character expression by trimming trailing DBCS blanks and SO/SI.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: See "RIGHT Function" in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KRIGHT(*argument*)

Required Argument

argument
specifies any SAS character expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KRIGHT function returns an argument with trailing blanks moved to the start of the value. The argument's length does not change.

The KRIGHT, KLEFT and KTRIM functions remove Unicode space characters. See Table 12.6 on page 351 for a list of the Unicode characters.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  a='漢字 の テスト  ';`
`  b=kright(a);`
`  put '----+----1----+----2----+';`
`  put a $25.;`
`  put b $25.;`
`run;`
	`----+----1----+----2----+`
	`漢字 の テスト`
	`  漢字 の テスト`

See Also

Functions:

- "KCOMPRESS Function" on page 327
- "KLEFT Function" on page 350
- "KTRIM Function" on page 377

KSCAN Function

Selects a specified word from a character expression.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is SCAN in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KSCAN(*argument,n*<, *delimiters*>)

Required Arguments

argument
specifies any character expression.

n
specifies a numeric expression that produces the number of the word in the character expression that you want KSCAN to select.

Tip If *n* is negative, KSCAN selects the word in the character expression starting from the end of the string. If |*n*| is greater than the number of words in the character expression, KSCAN returns a blank value.

delimiters

specifies a character variable that produces characters that you want KSCAN to use as word separators in the character expression.

Defaults If you omit *delimiters* in an ASCII environment, SAS uses *blank* . < (+ & ! $ *) ; ^ – / , % |. In ASCII environments without the ^ character, KSCAN uses the ~ character instead.

If you omit *delimiters* on an EBCDIC environment, SAS uses *blank* . < (+ | & ! $ *) ; ¬ – / , % | ¢

Tip If you represent *delimiters* as a constant, enclose *delimiters* in quotation marks.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

Leading delimiters before the first word in the character string do not effect KSCAN. If there are two or more contiguous delimiters, KSCAN treats them as one.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  length x $20 y $20;`
`  text1='これは漢字関数のテストです。';`
`  x='nonblank';`
`  i=1;`
`  do until(x=' ');`
`  x=kscan(text1,i,'はの');`
`  put x=;`
`  i=i+1;`
`  end;`
`  y='nonblank';`
`  i=-1;`
`  do until(y=' ');`
`  y=kscan(text1,i,'はの');`
`  put y=;`
`  i=i-1;`
`  end;`
`run;`
	x=これ
	x=漢字関数
	x=テストです。
	x=
	y=テストです。
	y=漢字関数
	y=これ
	y=

KSCANX Function

Selects a specified word from a character expression.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status, and is designed for DBCS data. However, if the first argument, *string* has single-byte characters, then the KSCANX function processes the SBCS data. For more information, see Internationalization Compatibility on page 279.

Syntax

KSCANX(*string*, *count* <, *character-list* <, *modifier*>>)

Required Arguments

string
specifies a character constant, variable, or expression.

count
is a nonzero numeric constant, variable, or expression that has an integer value. The integer value specifies the number of the word in the character string that you want KSCANX to select. For example, a value of 1 indicates the first word, a value of 2 indicates the second word, and so on. The following rules apply:

- If *count* is positive, KSCANX counts words from left to right in the character string.
- If *count* is negative, KSCANX counts words from right to left in the character string.

Optional Arguments

character-list
specifies an optional character expression that initializes a list of characters. This list determines which characters are used as the delimiters that separate words. The following rules apply:

- By default, all characters in *character-list* are used as delimiters.
- Specifying a modifier can also change the character-list used as delimiters. For example, if you specify the K modifier in the *modifier* argument, then all characters that are *not* in *character-list* are used as delimiters.

  Note: For more information see: "Using Default Delimiters in ASCII and EBCDIC Environments" in *SAS Viya Functions and CALL Routines: Reference*.

Tip You can add more characters to *character-list* by using other modifiers.

modifier
specifies a character constant, variable, or expression in which each non-blank character modifies the action of the KSCANX function. Blanks are ignored. Use the following characters as modifiers:

a or A adds alphabetic characters to the list of characters.

b or B scans backward from right to left instead of from left to right, regardless of the sign of the *count* argument.

c or C adds control characters to the list of characters.

d or D adds digits to the list of characters.

f or F adds an underscore and English letters (that is, valid first characters in a SAS variable name by using VALIDVARNAME=V7) to the list of characters.

g or G adds graphic characters to the list of characters. Graphic characters are characters that, when printed, produce an image on paper.

h or H adds a horizontal tab to the list of characters.

i or I ignores the case of the characters.

k or K causes all characters that are not in the list of characters to be treated as delimiters. That is, if K is specified, then characters that are in the list of characters are kept in the returned value rather than being omitted because they are delimiters. If K is not specified, then all characters that are in the list of characters are treated as delimiters.

l or L adds lowercase letters to the list of characters.

m or M specifies that multiple consecutive delimiters, and delimiters at the beginning or end of the *string* argument, refer to words that have a length of zero. If the M modifier is not specified, then multiple consecutive delimiters are treated as one delimiter, and delimiters at the beginning or end of the *string* argument are ignored.

n or N adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name by using VALIDVARNAME=V7) to the list of characters.

o or O processes the *charlist* and *modifier* arguments only once, rather than every time the KSCANX function is called. Using the O modifier in the DATA step (excluding WHERE clauses), or in the SQL procedure can make KSCANX run faster when you call it in a loop where the *character-list* and *modifier* arguments do not change. The O modifier applies separately to each instance of the KSCANX function in your SAS code, and does *not* cause all instances of the KSCANX function to use the same delimiters and modifiers.

p or P adds punctuation marks to the list of characters.

q or Q ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of the *string* argument contains unmatched quotation marks, then scanning from left to right produces different words than scanning from right to left.

r or R removes leading and trailing blanks from the word that KSCANX returns. If you specify the Q and R modifiers, the KSCANX function first removes leading and trailing blanks from the word. Then, if the word begins with a quotation mark, KSCANX also removes one layer of quotation marks from the word.

s or S adds space characters to the list of characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed).

t or T — trims trailing blanks from the *string* and *charlist* arguments. If you want to remove trailing blanks from only one character argument instead of both character arguments, use the TRIM function instead of the KSCANX function with the T modifier.

u or U — adds uppercase letters to the list of characters.

w or W — adds printable (writable) characters to the list of characters.

x or X — adds hexadecimal characters to the list of characters.

Tip If the *modifier* argument is a character constant, enclose the argument in quotation marks. Specify multiple modifiers in a single set of quotation marks. A *modifier* argument can also be expressed as a character variable or expression.

Details

Definition of Delimiter and Word

A *delimiter* is any of several characters that are used to separate words. You can specify the delimiters in the *charlist* and *modifier* arguments.

If you specify the Q modifier, delimiters inside substrings that are enclosed in quotation marks are ignored.

In the KSCANX function, *word* refers to a substring that has all of these characteristics:

- is bounded on the left by a delimiter or the beginning of the string
- is bounded on the right by a delimiter or the end of the string
- contains no delimiters

A *word* can have a length of zero if there are delimiters at the beginning or end of the string, or if the string contains two or more consecutive delimiters. However, the KSCANX function ignores words that have a length of zero unless you specify the M modifier.

Using Default Delimiters in ASCII and EBCDIC Environments

If you use the KSCANX function with only two arguments, then the default delimiters depend on whether your computer uses ASCII or EBCDIC characters.

- If your computer uses ASCII characters, the default delimiters are as follows:

  blank ! $ % & ( ) * + , - . / ; < ^ |

  In ASCII environments that do not contain the ^ character, the KSCANX function uses the ~ character instead.

- If your computer uses EBCDIC characters, then the default delimiters are as follows:

  blank ! $ % & ( ) * + , - . / ; < ¬ | ¢

If you use the *modifier* argument without specifying any characters as delimiters, then the only delimiters that are used are delimiters that are defined by the *modifier* argument. In this case, the lists of default delimiters for ASCII and EBCDIC environments are not used. In other words, modifiers add to the list of delimiters that are explicitly specified by the *charlist* argument. Modifiers do not add to the list of default modifiers.

The Length of the Result

In a DATA step, most variables have a fixed length. If the word returned by the KSCANX function is assigned to a variable that has a fixed length greater than the length of the returned word, then the value of that variable is padded with blanks. Macro variables have varying lengths and are not padded with blanks.

The maximum length of the word that is returned by the KSCANX function depends on the environment from which it is called:

- In a DATA step, if the KSCANX function returns a value to a variable that has not yet been given a length, that variable is given the length of the first argument. **This behavior is different from the behavior in previous releases of SAS. In previous releases code that created a variable with a length of 200 might have produced a variable with a length that was greater than expected.** If you need the KSCANX function to assign to a variable a value that is different from the length of the first argument, use a LENGTH statement for that variable before the statement that uses the KSCANX function.

  If you use the KSCANX function in an expression that contains operators or other functions, a word that is returned by the KSCANX function can have a length of up to 32,767 characters, except in a WHERE clause. In that case, the maximum length is 200 characters.

- In the SQL procedure, or in a WHERE clause in any procedure, the maximum length of a word that is returned by the KSCANX function is 200 characters.
- In the macro processor, the maximum length of a word that is returned by the KSCANX function is 65,534 characters.

The minimum length of the word that is returned by the KSCANX function depends on whether the M modifier is specified. See "Using the SCAN Function with the M Modifier" in *SAS Viya Functions and CALL Routines: Reference*. See also "Using the SCAN Function without the M Modifier" in *SAS Viya Functions and CALL Routines: Reference*.

Using the KSCANX Function with the M Modifier

If you specify the M modifier, the number of words in a string is defined as one plus the number of delimiters in the string. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you specify the M modifier, the KSCANX function returns a word with a length of zero if one of these conditions is true:

- The string begins with a delimiter and you request the first word.
- The string ends with a delimiter and you request the last word.
- The string contains two consecutive delimiters and you request the word that is between the two delimiters.

Using the KSCANX Function without the M Modifier

If you do not specify the M modifier, the number of words in a string is defined as the number of maximal substrings of consecutive non-delimiters. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you do not specify the M modifier, the KSCANX function acts in these ways:

- ignores delimiters at the beginning or end of the string
- treats two or more consecutive delimiters as if they were a single delimiter

If the string contains no characters other than delimiters, or if you specify a count that is greater in absolute value than the number of words in the string, then the KSCANX function returns one of the following items:

- a single blank when you call the KSCANX function from a DATA step
- a string with a length of zero when you call the KSCANX function from the macro processor

Using Null Arguments

This example scans a string for the first and last words:

- A negative count instructs the KSCANX function to scan from right to left.
- Leading and trailing delimiters are ignored because the M modifier is not used.
- In the last observation, all characters in the string are delimiters.

Example

This example uses Chinese characters:

```
data _null_;
xyz='SAS是全球最大的软件公司之一，sas  是全球商业智能和分析软件与服务领袖。';
substr=kscanx(xyz, 2, '', 'sp'); /* Use spaces and punctuations as delimiter. */
put substr= ;
run;
```

Here is the result from the KSCANX function example:

```
substr=sas
```

KSTRCAT Function

Concatenates two or more character expressions.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is CAT in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KSTRCAT(*argument-1*, *argument-2*<, … *argument-n*>)

Required Argument

argument
specifies any single-byte or double-byte character expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

If the KSTRCAT function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  text1='漢字';`
`  text2='文字列';`
`  text3='連結';`
`  result=kstrcat(text1,text2,'の',text3);`
`  put result=;`
`run;`
	result=漢字文字列の連結

KSTRIP Function

Removes leading and trailing blanks from a character string.

Categories: Character
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279 .

Note: This function supports the varchar type.

Syntax

KSTRIP(*string*)

Required Argument

string
specifies a character constant, variable, or expression.

Details

The KSTRIP function returns the argument with all leading and trailing SBCS blanks removed. If the argument is blank, KSTRIP returns a string with a length of zero.

Assigning the results of KSTRIP to a variable does not affect the length of the receiving variable. If the value that is trimmed is shorter than the length of the receiving variable, SAS pads the value with new trailing blanks.

In a DATA step, if the KSTRIP function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the argument.

Comparisons

The KSTRIP and STRIP functions return a string with a length of zero, for strings that are blank, and the TRIM function returns a single blank.

The KSTRIP and STRIP functions truncate the output if the receiving variable length is less than the input string. The STRIP function processes only SBCS string truncation. The KSTRIP function processes SBCS and DBCS data.

The KSTRIP and STRIP functions normalize data by removing unnecessary blanks. The KSTRIP function should be used in a DBCS environment. The KSTRIP and STRIP functions behave similarly. The KTRIM and KLEFT functions can also be used for data normalization. However, the KTRIM and KLEFT functions remove all types of DBCS spaces and can cause performance problems, as in this code example:

```
long_char_variable = KTRIM(KLEFT(long_char_variable));
```

.

The KSTRIP function resolves these issues. You can modify the code using the KSTRIP function to successfully perform DBCS data normalizations in a DBCS SAS session using this code:

```
long_char_variable = KSTRIP(long_char_variable);
```

.

Example

The following example shows how the KSTRIP function deletes leading and trailing blanks, and how the DBCS character is truncated. This example uses the Japanese Shift_JIS encoding,

Statements	Results

`data _null_;`
`  length a $5;`
`  v = unicode('\u0020\u0020\u8404\u8404\u8404\u0020\u0020');`
`  put v = hex.;`
`  a = kstrip(v);`
`  put a = hex.;`
`run;`
	`93B893B820`

See Also

Function

- "STRIP Function" in *SAS Viya Functions and CALL Routines: Reference*

KSUBSTR Function

Extracts a substring from an argument.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: See "SUBSTR (right of =) Function" in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KSUBSTR(*argument, position*<,*n*>)

Required Arguments

argument
specifies any SAS character expression.

position
specifies a numeric expression that is the beginning character position.

n
specifies a numeric expression that is the length of the substring to extract.

Interaction	If *n* is larger than the length of the expression that remains in *argument* after *position*, SAS extracts the remainder of the expression.
Tip	If you omit *n*, SAS extracts the remainder of the expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KSUBSTR function returns a portion of an expression that you specify in *argument*. The portion begins with the character specified by *position* and is the number of characters specified by *n*.

A variable that is created by KSUBSTR obtains its length from the length of *argument*.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  text='漢字文字列の抽出';`
`  result=ksubstr(text,1,2);`
`  put result=;`
`  result=ksubstr(text,3,4);`
`  put result=;`
`  kstart=7;`
`  klen=2;`
`  result=ksubstr(text,kstart,klen);`
`  put result=;`
`run;`
	Result=漢字文
	Result=字文

See Also

Function:

- "KSUBSTRB Function" on page 375
- "KUPDATE Function" on page 381
- "KUPDATEB Function" on page 383
- "KUPDATES Function" on page 384

KSUBSTRN Function

Returns a substring, allowing a result with a length of zero.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8).

Note: This function supports the varchar type.

See: "SUBSTRN Function" in *SAS Viya Functions and CALL Routines: Reference* and "Internationalization Compatibility for SAS String Functions" on page 279 for more information.

Syntax

KSUBSTRN(*string, position*<,*length*>)

Required Arguments

string
specifies a character or numeric constant, variable, or expression.

If *string* is numeric, then it is converted to a character value that uses the BEST32. format. Leading and trailing blanks are removed, and no message is sent to the SAS log.

position
is an integer that specifies the position of the first character in the substring.

Optional Argument

length
is an integer that specifies the length of the substring. If you do not specify *length*, the KSUBSTRN function returns the substring that extends from the position that you specify to the end of the string.

Details

The functionality of KSUBTRN is similar to SUBSTRN. The difference is that if the first parameter is numeric, SUBSTRN's output length is 8, and KSUBSTRN's output length is 200.

Example

This example shows the different functionality between the functions KSUBSTR and KSUBSTRN.

Statements	Results

`data _null_;`
`  ksubstr_result = "*" || ksubstr(1234.5678,2,6) || "*";`
`  put ksubstr_result=;`
`  ksubstrn_result = "*" || ksubstrn(1234.5678,2,6) || "*";`
`  put ksubstrn_result=;`
`run;`
	`ksubstr_result=*  1234*`
	`ksubstrn_result=*234.56*`

KSUBSTRB Function

Extracts a substring from an argument according to the byte position of the substring in the argument.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

See: "SUBSTR (right of =) Function" in *SAS Viya Functions and CALL Routines: Reference*

Syntax

KSUBSTRB(*argument, position*<,*n*>)

Required Arguments

argument
specifies any SAS character expression.

position
specifies the beginning character position in byte units.

n
specifies the length of the substring to extract in byte units.

Interaction If *n* is larger than the length (in byte units) of the expression that remains in *argument* after *position*, SAS extracts the remainder of the expression.

Tip If you omit *n*, SAS extracts the remainder of the expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KSUBSTRB function returns a portion of an expression that you specify in *argument*. The portion begins with the byte unit specified by *position* and is the number of byte units specified by *n*.

A variable that is created by KSUBSTRB obtains its length from the length of *argument*.

Example

The following example uses Japanese characters.

Statements	Results

data _null_;
text='漢字文字列の抽出';
result=ksubstrb(text,1,9);
put result=;
result=ksubstrb(text,3,7);
put result=;
run;
	result=漢字文
	result=字文

See Also

Function:

- "KSUBSTR Function" on page 372

KTRANSLATE Function

Replaces specific characters in a character expression.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8)..For more information, see Internationalization Compatibility on page 279.

z/OS specifics: This function runs only on z/OS.
Uses the EBCIDIC code sequence.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is TRANSLATE in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KTRANSLATE(*source,to-1,from-1*<,…*to-n,from-n*>)

Required Arguments

source
specifies the SAS expression that contains the original character value.

to
specifies the characters that you want KTRANSLATE to use as substitutes.

from
specifies the characters that you want KTRANSLATE to replace.

Interaction Values of *to* and *from* correspond on a character-by-character basis; KTRANSLATE changes character one of *from* to character one of *to*, and so on. If *to* has fewer characters than *from*, KTRANSLATE changes the extra *from* characters to blanks. If *to* has more characters than *from*, KTRANSLATE ignores the extra *to* characters.

Note You must have pairs of *to* and *from* arguments on some operating environments. On other operating environments, a segment of the collating sequence replaces null *from* arguments.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

You can use KTRANSLATE to translate a single-byte character expression to a double-byte character expression, or translate a double-byte character expression to a single-byte character expression.

The maximum number of pairs of *to* and *from* arguments that KTRANSLATE accepts depends on which operating environment you use to run SAS. There is no functional difference between using several pairs of short arguments, or fewer pairs of longer arguments.

If the KTRANSLATE function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  result=ktranslate('正課','回答','二課');`
`  put result=;`
`  result=ktranslate('abc','アイウ','abc');`
`  put result=;`
`run;`
	`result=正答`
	`result=アイウ`

KTRIM Function

Removes trailing DBCS blanks and SO/SI from character expressions.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: Non-DBCS equivalent function is "TRIM Function" in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KTRIM(*argument*)

Required Argument

argument
specifies any SAS character expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

KTRIM copies a character argument, removes all trailing blanks, and returns the trimmed argument as a result. KTRIM is useful for concatenating because concatenation does not remove trailing blanks.

Note: If the argument is blank, KTRIM returns one blank for the char type. If the argument is blank, KTRIM returns an empty blank or 0 value for the varchar type. The varchar type uses character semantics, and the char type uses byte semantics.

Assigning the results of KTRIM to a variable does not affect the length of the receiving variable. If the trimmed value is shorter than the length of the receiving variable, SAS pads the value with new blanks as it assigns it to the variable.

The KRIGHT, KLEFT and KTRIM functions remove Unicode space characters. See Table 12.6 on page 351 for a list of the Unicode characters.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  part1='漢字の　';`
`  part2='テスト';`
`  hasblank=part1||part2;`
`  noblank=ktrim(part1)||part2;`
`  put hasblank;`
`  put noblank;`
`run;`
	漢字の　テスト
	漢字のテスト

See Also

Functions:

- "KCOMPRESS Function" on page 327

- "KLEFT Function" on page 350
- "KRIGHT Function" on page 362

KTRUNCATE Function

Truncates a string to a specified length in byte unit without breaking multibyte characters.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

KTRUNCATE(*argument*, *number*, *length*)

Required Arguments

argument
specifies any SAS character expression.

number
is numeric.

length
is an integer.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KTRUNCATE function truncates a full-length *number* (stored as a double) to a smaller number of bytes, as specified in *length* and pads the truncated bytes with 0s. The truncation and subsequent expansion duplicate the effect of storing numbers in less than full length and then reading them.

Example

The following example uses Japanese characters.

Statements	Results
data _null_; x1=ktruncate('漢字のテスト',6); x2=ktruncate('漢字のテスト',7); x3=ktruncate('漢字のテスト',9); y1=ktruncate('漢字のテスト',5,9); y2=ktruncate('漢字のテスト',6,6); y3=ktruncate('漢字のテスト',7,6); put x1= / x2= / x3= / y1= / y2= / y3=; run;	x1=漢字 x2=漢字 x3=漢字の y1=字のテ y2=のテ y3=のテス

KUPCASE Function

Converts all single-width English alphabet letters in an argument to uppercase.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Tip: See "UPCASE Function" in *SAS Viya Functions and CALL Routines: Reference*.

Syntax

KUPCASE(*argument*)

Required Argument

argument
specifies any SAS character expression.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KUPCASE function copies a character argument, converts all lowercase single-width English alphabet letters to uppercase letters, and returns the altered value as a result.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  result=kupcase('abcあいうえお');`
`  put result=;`
`run;`
	`result=ABCあいうえお`

KUPDATE Function

Inserts, deletes, and replaces character value contents.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Syntax

KUPDATE(*argument,position,n*<, *characters-to-replace*>)

KUPDATE(*argument, position*<,*n*> , *characters-to-replace*)

Required Arguments

argument
specifies a character variable.

position
specifies a numeric expression that is the beginning character position.

n
specifies a numeric expression that is the length of the substring to be replaced.

Restrictions *n* cannot be larger than the length of the expression that remains in *argument* after *position*.

n is optional, but you cannot omit both *n* and *characters-to-replace* from the function.

Tip If you omit *n*, SAS uses all of the characters in *characters-to-replace* to replace the values of *argument*.

characters-to-replace
specifies a character expression that replaces the contents of *argument*.

Restriction *characters-to-replace* is optional, but you cannot omit both *characters-to-replace* and *n* from the function.

Tip Enclose a literal string of characters in quotation marks.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KUPDATE function replaces the value of *argument* with the expression in *characters-to-replace* starting at the character that you specify in *position*.

If the KUPDATE function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

Examples

Example 1

The following example shows the difference between KUPDATE and KUPDATES.

Statements	Results

`data _null_;`
`  str='123456';`
`  x1=str; substr(x1,2,3)="";`
`  y1=kupdate(str,2,3);`
`  z1=kupdates(str,2,3);`
`  put x1= / y1= / z1=;`
`  x2=str; substr(x2,2,3)="abcd";`
`  y2=kupdate(str,2,3,'abcd');`
`  z2=kupdates(str,2,3,'abcd');`
`  put x2= / y2= / z2=;`
`  x3=str; substr(x3,2,3)="ab";`
`  y3=kupdate(str,2,3,'ab');`
`  z3=kupdates(str,2,3,'ab');`
`  put x3= / y3= / z3=;`
`run;`
	`x1=1  56`
	`y1=156`
	`z1=1  56`
	`x2=1abc56`
	`y2=1abcd56`
	`z2=1abc56`
	`x3=1ab 56`
	`y3=1ab56`
	`z3=1ab 56`

Example 2

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  x1=kupdate('感じのテスト',1,2,'漢字');`
`  x2=kupdate(x1,1,2,'kanji');`
`  x3=kupdate(x1,1,3);`
`  x4=kupdate(x1,3,'かんじ');`
`  put x1= / x2= / x3= / x4=;`
`run;`
	x1=漢字のテスト
	x2=kaのテスト
	x3=テスト
	x4=漢字かんじ

See Also

Functions:

- "KUPDATES Function" on page 384
- "KUPDATEB Function" on page 383

KUPDATEB Function

Inserts, deletes, and replaces the contents of the character value according to the byte position of the character value in the argument.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

KUPDATEB(*argument, position, n*<,*characters-to-replace*>)

KUPDATEB(*argument, position* <, *n*> , *characters-to-replace*)

Required Arguments

argument
specifies a character variable.

position
specifies the beginning character position in byte units.

n
specifies the length of the substring to be replaced in byte units.

Restrictions *n* cannot be larger than the length (in bytes) of the expression that remains in *argument* after *position*.

n is optional, but you cannot omit both *n* and *characters-to-replace* from the function.

Tip If you omit *n*, SAS uses all of the characters in *characters-to-replace* to replace the values of *argument*.

characters-to-replace
specifies a character expression to replace the contents of *argument*.

Restriction *characters-to-replace* is optional, but you cannot omit both *characters-to-replace* and *n* from the function.

Tip Enclose a literal string of characters in quotation marks.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KUPDATEB function replaces the value of *argument* with the expression in *characters-to-replace*. KUPDATEB replaces *n* byte units starting at the byte unit that you specify in *position*.

If the KUPDATEB function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  x1=kupdateb('漢字のテスト',1,9,'漢字');`
`  x2=kupdateb(x1,1,9,'kanji');`
`  x3=kupdateb(x1,1,9);`
`  x4=kupdateb(x1,7,'かんじ');`
`  put x1= / x2= / x3= / x4=;`
`run;`
	x1=漢字　テスト
	x2=kanji　テスト
	x3=テスト
	x4=漢字　かんじ

See Also

Function:

- "KUPDATE Function" on page 381

KUPDATES Function

Inserts, deletes, and replaces character value contents.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Note: This function supports the varchar type.

Syntax

KUPDATES(*argument,position,n*<, *characters-to-replace*>)

KUPDATES(*argument,position*<,*n*> , *characters-to-replace*)

Required Arguments

argument
specifies a character variable.

position
specifies a numeric expression that is the beginning character position.

Optional Arguments

n
specifies a numeric expression that is the length of the substring to be replaced.

Restrictions *n* cannot be larger than the length of the expression that remains in *argument* after *position*.

n is optional, but you cannot omit both *n* and *characters-to-replace* from the function.

Tip If you omit *n*, SAS uses all of the characters in *characters-to-replace* to replace the values of *argument*.

characters-to-replace
specifies a character expression that replaces the contents of *argument*.

Restriction *characters-to-replace* is optional, but you cannot omit both *characters-to-replace* and *n* from the function.

Tip Enclose a literal string of characters in quotation marks.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KUPDATES function replaces the value of *argument* with the expression in *characters-to-replace*. KUPDATES replaces *n* characters starting at the character that you specify in *position*.

If the KUPDATES function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

Examples

Example 1

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  x1=kupdates('感じのテスト',1,2,'漢字');`
`  x2=kupdates(x1,1,2,'kanji');`
`  x3=kupdates(x1,1,3);`
`  x4=kupdates(x1,3,'かんじ');`
`  put x1= / x2= / x3= / x4=;`
`run;`
	x1=漢字のテスト
	x2=kaのテスト
	x3=テスト
	x4=漢字かんじ

Example 2

The following example shows the difference between KUPDATE and KUPDATES.

Statements	Results

`data _null_;`
`  str='123456';`
`  x1=str; substr(x1,2,3)="";`
`  y1=kupdate(str,2,3);`
`  z1=kupdates(str,2,3);`
`  put x1= / y1= / z1=;`
`  x2=str; substr(x2,2,3)="abcd";`
`  y2=kupdate(str,2,3,'abcd');`
`  z2=kupdates(str,2,3,'abcd');`
`  put x2= / y2= / z2=;`
`  x3=str; substr(x3,2,3)="ab";`
`  y3=kupdate(str,2,3,'ab');`
`  z3=kupdates(str,2,3,'ab');`
`  put x3= / y3= / z3=;`
`run;`
	`x1=1  56`
	`y1=156`
	`z1=1  56`
	`x2=1abc56`
	`y2=1abcd56`
	`z2=1abc56`
	`x3=1ab 56`
	`y3=1ab56`
	`z3=1ab 56`

See Also

Functions:

- "KUPDATE Function" on page 381

KVERIFY Function

Returns the position of the first character that is unique to an expression.

Categories: DBCS
CAS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Tip: See "VERIFY Function" in *SAS Viya Functions and CALL Routines: Reference*

Syntax

KVERIFY(*source, excerpt-1*, …*excerpt-n*)

Required Arguments

source
specifies any SAS character expression.

excerpt-1, ...excerpt-n
specifies any SAS character expression.

Tips Enclose a literal string of characters in quotation marks.

If you specify more than one excerpt, separate them with a comma.

Details

See "Internationalization Compatibility for SAS String Functions" on page 279 for restrictions and more information.

The KVERIFY function returns the position of the first character in *source* that is not present in any *excerpt*. If KVERIFY finds every character in *source* in at least one *excerpt*, it returns a value of 0.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  input grade :$2. @@;`
`  check='良否';`
`  x=kverify(grade,check);`
`  if x gt 0 then put 'Invalid grade value: ' grade;`
`cards;`
`良 否 良 否 良 否 可`
`;`
	`invalid grade value: 可`

KVERIFYB Function

Returns the position of the first character that is unique to an expression.

Category: DBCS

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

KVERIFYB(*source*, *excerpt-1*,<*excerpt-n*>)

Required Arguments

source
specifies any SAS character expression.

excerpt
specifies any SAS character expression.

TIP If you specify more than one excerpt, separate them with a comma.

Details

The KVERIFYB function returns the position of the first character in source that is not present in any excerpt. If KVERIFYB finds every character in source in at least one excerpt, it returns a 0.

Comparisons

KVERIFYB returns byte-based value. KVERIFY returns character-based value. When process SBCS (For example, wlati1 encoding) string, they return identical result. But, in a DBCS session (For example, EUC-JP or SHIFT-JIS encoding), most CJK characters are 2-byte width, that makes these functions return different result.

Example

The following example uses Japanese characters.

Statements	Results

`data _null_;`
`  check='漢字';`
`  text='漢字の検索';`
`  x1=kverify(text,check);`
`  put x1=`
`  x2=kverifyb(text,check);`
`  put x2=;`
`run;`
	`x1=3`
	`x2=7`

NLDATE Function

Converts the SAS date value to the date value of the specified locale by using the date format descriptors.

Category: Date and Time

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

NLDATE(*date,descriptor*)

Required Arguments

date
specifies a SAS date value.

descriptor
is a variable or expression that specifies how dates and times are formatted in output. The following descriptors are case sensitive:

#
removes the leading zero from the result.

%%
specifies the % character.

%a
specifies the short-weekday descriptor. The range for the day descriptor is Mon–Sun.

%A
specifies the long-weekday descriptor. The range for the long-weekday descriptor is Monday–Sunday.

%b
specifies the short-month descriptor. The range for the short-month descriptor is Jan–Dec.

%B
specifies the long-month descriptor. The range for the long-month descriptor is January–December.

%C
specifies the long-month descriptor and uses blank padding. The range for the long-month descriptor is January–December.

%d
specifies the day descriptor and uses 0 padding. The range for the day modifier is 01–31.

%e
specifies the day descriptor and uses blank padding. The range for the day descriptor is 01–31.

%F
specifies the long-weekday descriptor and uses blank padding. The range for the day descriptor is Monday–Sunday.

%j
specifies the day-of-year descriptor as a decimal number and uses a leading zero. The range for the day-of-year descriptor is 1–366.

%m
specifies the month descriptor and uses 0 padding. The range for the month descriptor is 01–12.

%o
specifies the month descriptor. The range for the month descriptor is 1–12 with blank padding.

%u
specifies the weekday descriptor as a number in the range 1–7 that represents Monday–Sunday.

%U
specifies the week-number-of-year descriptor by calculating the descriptor value as the SAS date value using the number of week within the year (Sunday is considered the first day of the week). The number-of-the-week value is represented as a decimal number in the range 0–53 and uses a leading zero and a maximum value of 53.

%V
specifies the week-number-of-year descriptor by calculating the descriptor value as the SAS date value. The number-of-week value is represented as a decimal number in the range 01–53 and uses a leading zero and a maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year.

%w
specifies the weekday descriptor as a number in the range 0–6 that represents Sunday–Saturday.

%W
specifies the week-number-of-year descriptor by calculating the descriptor value as SAS date value by using the number of week within the year (Monday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0–53 and uses a leading zero and a maximum value of 53.

%y
specifies the year (2-digit) modifier. The range for the year descriptor is 00–99.

%Y
specifies the year (4-digit) descriptor. The range for the year descriptor is 1970–2069.

Details

The NLDATE function converts the SAS date value to the date value of the specified locale by using the date descriptors.

Example

The following example shows a log filename that is created from a SAS date value.

Statements	Results

`options locale=English_Unitedstates;`
`logfile=nldate('24Feb2003'd,'%B-%d.log');`
`put logfile;`
	`February-24.log`

Statements	Results

`options locale=German_Germany;`
`logfile=nldate('24Feb2003'd,'%B-%d.log');`
`put logfile;`
	`Februar-24.log`

The following example shows a weekday name that is created from a SAS date value.

Statements	Results
	`----+----1----+`

`options locale=English_unitedstates;`
`weekname=nldate('24Feb2003'd,'%A');`
`put weekname;`
	`Monday`

`options locale=German_Germany;`
`weekname=nldate('24Feb2003'd,'%A');`
`put weekname;`
	`Montag`

See Also

Format:

- "NLDATE Format" on page 106

NLDATM Function

Converts the SAS datetime value to the time value of the specified locale by using the datetime-format descriptors.

Category: Date and Time

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

NLDATM(*datetime,descriptor*)

Required Arguments

datetime
specifies a SAS datetime value.

descriptor
is a variable or expression that specifies how dates and times are formatted in output. The following descriptors are case sensitive:

#
removes the leading zero from the result.

%%
specifies the % character.

%a
specifies the short-weekday descriptor. The range for the day descriptor is Mon–Sun.

%A
specifies the long-weekday descriptor. The range for the long-weekday descriptor is Monday–Sunday.

%b
specifies the short-month descriptor. The range for the short-month descriptor is Jan–Dec.

%B
specifies the long-month descriptor. The range for the long-month descriptor is January–December.

%c
specifies the long-month descriptor and uses blank padding. The range for the long-month descriptor is January–December.

%d
specifies the day descriptor and uses 0 padding. The range for the day descriptor is 01–31.

%e
specifies the day descriptor and uses blank padding. The range for the day descriptor is 01–31.

%F
specifies the long-weekday descriptor and uses blank padding. The range for the day descriptor is Monday–Sunday.

%H
specifies the hour descriptor that is based on a 24-hour clock. The range for the hour descriptor is 00–23.

%I
specifies the hour descriptor that is based on a 12-hour clock. The range for the hour descriptor is 01–12.

%j
specifies the day-of-year descriptor as a decimal number and uses a leading zero. The range for the day-of-year descriptor is 1–366.

%m
specifies the month descriptor and uses 0 padding. The range for the month descriptor is 01–12.

%M
specifies the minute descriptor. The range for the minute descriptor is 00–59.

%o
specifies the month descriptor and uses blank padding. The range for the month descriptor is 1–12.

%p
specifies a.m. or p.m. descriptor.

%S
specifies the second descriptor. The range for the second descriptor is 00–59.

%u
specifies the weekday descriptor as a number in the range of 1–7 that represents Monday–Sunday.

%U
specifies the week-number-of-year descriptor by calculating the descriptor value as the SAS date value and uses the number-of-week value within the year (Sunday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0–53. A leading zero and a maximum value of 53 is used.

%V
specifies the week-number-of-year descriptor by calculating the descriptor value as the SAS date value. The number-of-week value is represented as a decimal number in the range 01–53. A leading zero and a maximum value of 53 is used. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year.

%w
specifies the weekday descriptor as a number in the range of 0–6 that represents Sunday–Saturday.

%W
specifies the week-number-of-year descriptor by calculating the descriptor value as SAS date value using the number of week within the year (Monday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range of 0–53. A leading zero and a maximum value of 53 are used.

%y
specifies the year (2-digit) descriptor. The range for the year descriptor is 00–99.

%Y
specifies the year (4-digit) descriptor. The range for the year descriptor is 1970–2069.

Details

The NLDATM function converts the SAS datetime value to the datetime value of the specified locale by using the datetime descriptors.

Example

The following example shows a time (a.m or p.m.) that is created from a SAS datetime value.

Statements	Results
	`----+----1----+`

Statements	Results

`options locale=English;`
`time_ampm=nldatm('24Feb2003:12:39:43'dt,'%I%p');`
`put time_ampm;`
	`12PM`

`options locale=German;`
`time_ampm=nldatm('24Feb2003:12:39:43'dt,'%I%p');`
`put time_ampm;`
	`12nachm`

See Also

Format:

- "NLDATM Format" on page 127

NLTIME Function

Converts the SAS time or the datetime value to the time value of the specified locale by using the NLTIME descriptors.

Category: Date and Time

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

NLTIME(*time*|*datetime*,*descriptor*,*startpos*)

Required Arguments

time
specifies a SAS time value.

datetime
specifies a SAS datetime value.

descriptor
is a variable, or expression, that specifies the value of a descriptor. You can enter the following descriptors in uppercase or lowercase:

#
removes the leading zero from the result.

%%
specifies the % character.

%H
specifies the hour descriptor that is based on a 24-hour clock. The range for the hour descriptor is 00–23.

%I
specifies the hour descriptor that is based on a 12-hour clock. The range for the hour descriptor is 01–12.

%M
specifies the minute modifier. The range for the minute descriptor is 00–59.

%P
specifies the a.m. or p.m. descriptor.

%S
specifies the second descriptor. The range for the second descriptor is 00–59.

startpos
is an integer that specifies the position at which the search should start and that specifies the direction of the search.

Details

The NLTIME function converts a SAS time or datetime value to the time value of the specified locale by using the time descriptors.

Example

The following example shows an a.m. or p.m. time that is created from a SAS time.

Statements	Results
	`----+----1----+`

`options locale=English;`
`time_ampm=nltime('12:39:43't,'%i%p');`
`put time_ampm;`
	`12 PM`

`options locale=German;`
`time_ampm=nltime('12:39:43't,'%i%p');`
`put time_ampm;`
	`12 nachm`

See Also

Format:

- "NLTIME Format" on page 242

SASMSG Function

Specifies a message from a data set. The returned message is based on the current locale and a specified key.

Category: Locale

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

SASMSG (*BASENAME", "KEY", <<"QUOTE"|"DQUOTE"|"NOQUOTE"> <, "substitution 1", ..., "substitution 7">>*)

Required Arguments

BASENAME
the name of the data set where the message is located.

KEY
the message key.

Note: If you specify an invalid key name, then the key name is returned.

QUOTE|DQUOTE|NOQUOTE
specifies the type of quotation marks that are added to the message text and substitution strings.

Default DQUOTE

substitution
string substitutions. The maximum string substitutions is 7.

Details

The SAS message data set must be a 7-bit ASCII data set. Any character that cannot be represented in the 7-bit ASCII encoding is represented in the Unicode escape format of '\uxxxx', where 'xxxx' is the base 10 numeric representation of the Unicode value of the character.

The data set used by the SASMSG function must have been created specifically for use with this function. The data set must contain the following variables:

#	Variable Name	Type	Length	Description
1	locale	char	5	language of the message
2	key	char	60	key to identify the message
3	lineno	num	5	line # of the message in reverse order
4	text	text	1,200	text of the message

The data set must be sorted on the following variables: *locale*, *key*, and *lineno*. The variable *lineno* must be in descending order. A composite index on locale and key must be defined. Here is a sample program to sort and create an indexed data set:

```
%let basename=MyProduct;
```

```
proc sort data=t.&basename;
by locale key descending lineno;
run;

proc datasets lib=t
memtype=data;
modify &basename;
index create indx=(LOCALE KEY);
run;
quit;
```

The returned message is based on the LOCALE system option. The LOCALE option is represented by *ll_RR* where ll represents the two-letter language code and RR represents the two-letter region code. If a match is not found, then the function searches for a match with the language only. If the pair locale and key are still not found, then the function defaults to the English language (en). If the key does not exist for English (en), then the key name is returned.

You can alter formatting. You can use string substitution by using the format code %s. You can change the order of substitution. In some cases, translation of a message to a language other than English might require changing the order of substitutions. You can change the order by placing an argument number specification, *#nn*, within a format string, where *nn* is the number of the argument in the substitution list. The following example demonstrates the order:

Statement	Result
`msg = sasmsg ("nls.mymsg","IN_CD_LOG", "noquote","cat","dog"); IN_CD_LOGINFO = My %#1s. Your %#2s`	`msg= My cat. Your dog.`
`IN_CD_LOGINFO = My %#2s. Your %#1s`	`msg= My dog. Your cat.`

The SASMSG function can be used in the open code macro with the %SYSFUNC macro function.

Arguments that are passed to a function called by the %SYSFUNC macro must not be in quotation marks. Arguments passed to the SASMSG function outside of %SYSFUNC must be quoted.

When the SASMSG function is used with the %SYSFUNC macro function, the returned string is wrapped with the %NRBQUOTE function.

Examples

Example 1

The following example demonstrates the formatting feature of SASMSG:

```
%macro demo_sasmsg;
  data _null_;
    msg = sasmsg("nls.mymsg","IN_APW_SAVE_OK","noquote");
    put msg=;
  run;
%mend demo_sasmsg;
```

SAS Statements	Results
`options locale = en_US;` `%demo_sasmsg ;`	`msg=The Access Control key was successfully saved.`
`options locale = es_ES;` `%demo_sasmsg;`	`msg=La clave de control de acceso se ha guardado.`
`options locale = french_France;` `%demo_sasmsg;`	`msg=La clé de contrôle d'accès a bien été enregistrée.`

Example 2

The following example demonstrates the open macro feature:

```
%MACRO PRT(loc,tb,key);
    option locale=&loc;
    %PUT %SYSFUNC(SASMSG(&tb,&key) );
%MEND PRT;
```

SAS Statements	Results
`%PRT(en_US,&TABLEID,IN_EDIT)`	`"Edit"`
`%PRT(es_ES,&TABLEID,IN_EDIT)`	`"Editar"`
`%PRT(fr_FR,&TABLEID,IN_EDIT)`	`"Modifier"`

SASMSGL Function

Specifies a message from a data set. The message is based on a specified locale value and a specified key value.

Category: Locale

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

SASMSGL(*"BASENAME"*, *"KEY"*, *"LOCALE"*, *(, , , <<"Q"|"D"|"N"> <, "substitution 1", ..., "substitution 6">>)*

Required Arguments

BASENAME

the name of the data set where the message is located.

KEY

the message key.

Note: If you specify an invalid key name, then the key name is returned.

LOCALE
the posix locale value (ll_RR).

QUOTE|DQUOTE|NOQUOTE
specifies the type of quotation marks that are added to the message text and substitution strings.

Default DQUOTE

substitution
string substitutions. The maximum string substitutions is 6.

Details

The SAS message data set must be a 7-bit ASCII data set. Any character that cannot be represented in the 7-bit ASCII encoding is represented in the Unicode escape format of '*uxxxx*, where the *xxxx* is the base 10 numeric representation of the Unicode value of the character.

The data set used by SASMSGL function must have been created specifically for use with this function. The data set must contain the following variables:

#	Variable Name	Type	Length	Description
1	locale	char	5	language of the message
2	key	char	60	key to identify the message
3	lineno	num	5	line number of the message in reverse order
4	text	text	1200	text of the message

The data set must be sorted on the following variables: *locale*, *key*, and *lineno*. The variable *lineno* must be in descending order. A composite index on locale and key must be defined. Here is a sample program to sort and create an indexed data set:

```
%let basename=MyProduct;

proc sort data=t.&basename;
by locale key descending lineno;
run;

proc datasets lib=t
memtype=data;
modify &basename;
index create indx=(LOCALE KEY);
run;
quit;
```

The returned message is based on the LOCALE system option. The LOCALE option is represented by *ll_RR* where *ll* represents the two- letter language code and RR represents the two-letter region code. If a match is not found, then the function searches for a match with the language only. If the pair locale and key are still not found, then the function

defaults to the English language (en). If the key does not exist for English (en), then the key name is returned.

You can alter formatting. You can use string substitution by using the format code %s. You can change the order of substitution. In some cases, translation of a message to a language other than English might require changing the order of substitutions. You can change the order by placing an argument number specification, *#nn*, within a format string, where *nn* is the number of the argument in the substitution list. The following example demonstrates changing the order:

Statement	Result
`msg = sasmsgl ("nls.mymsg","IN_CD_LOG","en_US","N", "cat","dog"); IN_CD_LOGINFO = My %#1s. Your %#2s`	`msg= My cat. Your dog.`
`IN_CD_LOGINFO = My %#2s. Your %#1s`	`msg= My dog. Your cat.`

The SASMSGL function can be used in the open code macro with the %SYSFUNC macro function.

Arguments that are passed to a function called by the %SYSFUNC macro must not be in quotation marks. Arguments passed to the SASMSGL function outside of %SYSFUNC must be quoted.

When the SASMSGL function is used with the %SYSFUNC macro function, the returned string is wrapped with the %NRBQUOTE function.

Examples

Example 1

The following example demonstrates the formatting feature of SASMSGL:

Statements	Results
`sasmsgl("nls.mymsg", "IN_APW_SAVE_OK", "en_US")`	`The Access Control key was successfully saved.`
`sasmsgl("nls.mymsg", "IN_APW_SAVE_OK", "es_ES")`	`La llave de control de acceso se ha guardado.`
`sasmsgl("nls.mymsg", "IN_APW_SAVE_OK", "fr_FR")`	`La clé de contrôle d'accès a bien été enregistrée.`

Example 2

The following example demonstrates the open macro feature:

SAS Statements	Results
`%PUT %SYSFUNC(SASMSGL(NLS.MYDS, IN_ASD_LABEL, "en_US"));`	`%PUT Edit`
`%PUT %SYSFUNC(SASMSGL(NLS.MYDS, IN_ASD_LABEL, "es_ES"));`	`%PUT Editar`

SAS Statements	Results
`%PUT %SYSFUNC(SASMSGL(NLS.MYDS, IN_ASD_LABEL, Modifier));`	

SORTKEY Function

Creates a linguistic sort key.

Category: Locale

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

sortKey(*string*, <*locale*, *strength*, *case_order*, *numeric_order*,>)

Required Arguments

string
character expression

locale
specifies the locale name in the form of a POSIX name (ja_JP). See Table 17.1 on page 629 for a list of locale names and POSIX values.

strength
The value of strength is related to the collation level. There are five collation-level values. The following table provides information about the five levels. The default value for strength is related to the locale.

Value	Type of Collation	Description
PRIMARY or P	PRIMARY specifies differences between base characters (for example, "a" < "b").	It is the strongest difference. For example, dictionaries are divided into different sections by base character.
SECONDARY or S	Accents in the characters are considered secondary differences (for example, "as" < "às" < "at").	Other differences between letters can also be considered secondary differences, depending on the language. A secondary difference is ignored when there is a primary difference anywhere in the strings.

TERTIARY or T	Upper and lowercase differences in characters are distinguished at the tertiary level (for example, "ao" < "Ao" < "aò").	An example is the difference between large and small Kana. A tertiary difference is ignored when there is a primary or secondary difference anywhere in the strings.
QUATERNARY or Q	When punctuation is ignored at level 1-3, an additional level can be used to distinguish words with and without punctuation (for example, "ab" < "a-b" < "aB").	This difference is ignored when there is a primary, secondary, or tertiary difference. The quaternary level should be used if ignoring punctuation is required or when processing Japanese text.
IDENTICAL or I	When all other levels are equal, the identical level is used as a tiebreaker. The Unicode code point values of the NFD form of each string are compared at this level, just in case there is no difference at levels 1-4.	For example, only Hebrew cantillation marks are distinguished at this level. This level should be used sparingly, as only code point values differences between two strings is an extremely rare occurrence.

case order

sorts uppercase and lowercase letters. This argument is valid for only TERTIARY, QUATERNARY, or IDENTICAL. The following table provides the values and information for the case order argument.

Value	Description
UPPER or U	Sorts uppercase letters first, then the lowercase letters.
LOWER or L	Sorts lowercase letters first, then the uppercase letters.

numeric order

orders numbers by the numeric value instead of the number's characters.

Value	Description
NUMERIC or N	Order numbers (integers) by the numeric value. For example, "8 Main St." would sort before "45 Main St.".

collation order

There are two types of collation values: Phonebook and Traditional. If you do not select a collation value, then the user's locale-default collation is selected. The following table provides more information.

Value	Description
PHONEBOOK or P	specifies a phonebook style ordering of characters. Select **PHONEBOOK** only with the German language.

TRADITIONAL or T	specifies a traditional style ordering of characters. Select **TRADITIONAL** only with the Spanish language.

Details

The SORTKEY function creates a linguistic sort key for data. You must enter at least one argument. If the length of the variable that receives the key is not large enough, the data truncates, and a warning is displayed.

locale
: Locale values use the POSIX name (ll_RR). LL represents the two-letter language code, and RR represents the two-letter region code. For example, en_US is the POSIX name for English, United States. en represents the English language, and US represents the United States. If a locale value is not specified, then the session locale is used.

strength
: The strength argument determines whether accents or case affect collating or matching text. If no value is specified for strength, then the locale determines the value. The following values can be specified for strength.

  PRIMARY
  : This value includes base letters. An example is the letters, A, a, and Å are all processed the same.

  SECONDARY
  : This value processes data the same as PRIMARY, and accents are processed. The letters A and a are processed equally, and Å is processed as an accented character.

  TERTIARY
  : This value processes data the same as SECONDARY, and the character's case is processed. For example, A, a, and Å are all processed differently.

  QUATERNARY
  : This value processes data the same as TERTIARY, and punctuation is processed.

  IDENTICAL
  : This value process data the same as QUATERNARY, and code point is processed.

case order
: specifies to sort data using uppercase or lowercase letter. The following table shows examples of specifying the UPPER value or the LOWER value.

UPPER	LOWER
Aztec	aztec
aztec	Aztec
Mars	mars
mars	Mars

collation order
The collation order value PHONEBOOK is ignored unless the locale is a German language.

The collation order value TRADITIONAL is ignored unless the locale is a Spanish language.

A warning message is displayed for other locales.

SETLOCALE Function

Specifies the locale keys for the current SAS locale.

Category: Locale

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

Setting SAS Locale

SETLOCALE (*sas_locale*)

Customize single locale elements

SETLOCALE (*key*,*value*)

Customize single locale elements

SETLOCALE (*category_name*, *sas_locale*)

Required Arguments

sas_locale
specifies a SAS locale name by using the SAS name or the posix name. You can also specify the locale alias.

key
specifies a SAS locale element key. See the list of element keys in the Details section.

value
specifies a value for the locale element.

category_name
specifies the category name:

- LC_TIME
- LC_MONETARY
- LC_NUMERIC
- LC_ALL

Details

You can modify the following locale elements. The value of *key* must be less than the value of *max length*. You can specify the following values for *type*:

0 String.

1 Unsigned integer. You must use double quotation marks.

Locale Element Key	Max Length	Type	Category
DATESTYLE	3	0	
PAPERSIZE	8	0	
FTITLE	512	0	
FTEXT	512	0	
SIMFONT	512	0	
SORTSEQ	8	0	
MESSAGES	8	0	
FORMATNAME_DATE	512	0	
FORMATNAME_DATETIME	512	0	
FORMATNAME_TIME	512	0	
FORMATNAME_NUMERIC	512	0	
FORMATNAME_PERCENT	512	0	
FONT_SERIF	32	0	
FONT_SANSSERIF	32	0	
FONT_CURSIVE	32	0	
FONT_FANTASY	32	0	
FONT_MONOSPACE	32	0	
BRUSH	32	0	
SIMPLEX	32	0	
COMPLEX	32	0	

Locale Element Key	Max Length	Type	Category
SWISS	32	0	
ITALIC	32	0	
DATE_FORMAT	512	0	LC_TIME
DATE_SHORT_FORMAT	512	0	LC_TIME
DATETIME_AMPM_FORMAT	512	0	LC_TIME
DATETIME_FORMAT	512	0	LC_TIME
DATETIME_SHORT_FORMAT	512	0	LC_TIME
DATETIME_WEEK_FORMAT	512	0	LC_TIME
DATETIME_WEEK_SHORT_FORMAT	512	0	LC_TIME
TIME_AMPM_FORMAT	512	0	LC_TIME
TIME_FORMAT	512	0	LC_TIME
DATE_WEEK_FORMAT	512	0	LC_TIME
DATE_WEEK_SHORT_FORMAT	512	0	LC_TIME
DATE_YYMM_FORMAT	512	0	LC_TIME
DATE_YYMM_SHORT_FORMAT	512	0	LC_TIME
DATE_MMDD_FORMAT	512	0	LC_TIME
DATE_MMDD_SHORT_FORMAT	512	0	LC_TIME
DATE_YEAR_FORMAT	512	0	LC_TIME

Locale Element Key	Max Length	Type	Category
DATE_YEAR_SHORT_FORMAT	512	0	LC_TIME
DATE_YYQQ_FORMAT	512	0	LC_TIME
DATE_YYQQ_SHORT_FORMAT	512	0	LC_TIME
DATE_YYWW_FORMAT	512	0	LC_TIME
DATE_YYWW_SHORT_FORMAT	512	0	LC_TIME
DATE_SEP	8	0	LC_TIME
ABMON01	512	0	LC_TIME
ABMON02	512	0	LC_TIME
ABMON03	512	0	LC_TIME
ABMON04	512	0	LC_TIME
ABMON05	512	0	LC_TIME
ABMON06	512	0	LC_TIME
ABMON07	512	0	LC_TIME
ABMON08	512	0	LC_TIME
ABMON09	512	0	LC_TIME
ABMON10	512	0	LC_TIME
ABMON11	512	0	LC_TIME
ABMON12	512	0	LC_TIME
MON01	512	0	LC_TIME
MON02	512	0	LC_TIME
MON03	512	0	LC_TIME
MON04	512	0	LC_TIME
MON05	512	0	LC_TIME

Locale Element Key	Max Length	Type	Category
MON06	512	0	LC_TIME
MON07	512	0	LC_TIME
MON08	512	0	LC_TIME
MON09	512	0	LC_TIME
MON10	512	0	LC_TIME
MON11	512	0	LC_TIME
MON12	512	0	LC_TIME
ABDAY1	512	0	LC_TIME
ABDAY2	512	0	LC_TIME
ABDAY3	512	0	LC_TIME
ABDAY4	512	0	LC_TIME
ABDAY5	512	0	LC_TIME
ABDAY6	512	0	LC_TIME
ABDAY7	512	0	LC_TIME
DAY1	512	0	LC_TIME
DAY2	512	0	LC_TIME
DAY3	512	0	LC_TIME
DAY4	512	0	LC_TIME
DAY5	512	0	LC_TIME
DAY6	512	0	LC_TIME
DAY7	512	0	LC_TIME
AM	512	0	LC_TIME
PM	512	0	LC_TIME
ABQTR1	512	0	LC_TIME
ABQTR2	512	0	LC_TIME

Locale Element Key	Max Length	Type	Category
ABQTR3	512	0	LC_TIME
ABQTR4	512	0	LC_TIME
QTR1	512	0	LC_TIME
QTR2	512	0	LC_TIME
QTR3	512	0	LC_TIME
QTR4	512	0	LC_TIME
INT_CURRENCY_SYMBOL	3	0	LC_MONETARY
CURRENCY_SYMBOL	32	0	LC_MONETARY
MON_DECIMAL_POINT	8	0	LC_MONETARY
MON_THOUSANDS_SEP	8	0	LC_MONETARY
MON_GROUPING	3	1	LC_MONETARY
MON_POSITIVE_SIGN	8	0	LC_MONETARY
MON_NEGATIVE_SIGN	8	0	LC_MONETARY
MON_INT_FRAC_DIGITS	3	1	LC_MONETARY
MON_FRAC_DIGITS	3	1	LC_MONETARY
MON_P_CS_PRECEDES	3	1	LC_MONETARY
MON_P_SEP_BY_SPACE	3	1	LC_MONETARY
MON_P_SIGN_POSN	3	1	LC_MONETARY
MON_N_SIGN_POSN	3	1	LC_MONETARY
DECIMAL_POINT	1	0	LC_NUMERIC

Locale Element Key	Max Length	Type	Category
THOUSANDS_SEP	1	0	LC_NUMERIC
GROUPING	3	1	LC_NUMERIC
POSITIVE_SIGN	8	0	LC_NUMERIC
NEGATIVE_SIGN	8	0	LC_NUMERIC
P_CS_PRECEDES	3	1	LC_NUMERIC
P_SEP_BY_SPACE	3	1	LC_NUMERIC
N_CS_PRECEDES	3	1	LC_NUMERIC
P_SEP_BY_SPACE	3	1	LC_NUMERIC
N_CS_PRECEDES	3	1	LC_NUMERIC
N_SEP_BY_SPACE	3	1	LC_NUMERIC
P_SIGN_POSN	3	1	LC_NUMERIC
N_SIGN_POSN	3	1	LC_NUMERIC
HEIGHT	3	1	
WIDTH	3	1	

Examples

Example 1

In the following locale example, the SETLOCALE function specifies the locale Japanese (jp_JP). The SETLOCALE function returns the previous locale. In this example, the previous locale was English_United States.

Statements	Results

`data _null_;`
`x=setlocale('ja_JP');`
`put x=;`
`run;`
	`x=English_UnitedStates`

Example 2

In the following example, the SETLOCALE function returns the locale name where the element values are being changed:

Statements	Results

`data _null_;`
`x = setlocale("LC_MONETARY", 'zh_CN');`
`put x=; run;`
	`x=Japanese_Japan`

Example 3

In the following example, the SETLOCALE function changes the value of the specified key, DATE_YEAR_FORMAT:

Statements	Results

`data _null;`
`x=setlocale('DATE_YEAR_FORMAT', '¥%Y');`
`put x=;`
`run;`
	x=%Y¥

TZONEID Function

Returns the current time zone ID.

Category: Date and Time

Alias: TZID

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONEID<*time-zone-id*>

Optional Argument

time-zone-id

specifies a region or area value that is defined by SAS. When you specify a zone ID, the time zone that SAS uses is determined by the time zone name and daylight savings time rules.

Details

The TZONEID function returns a blank value if the TIMEZONE= option is blank or a user-defined time zone is specified.

The TZONEID function validates the timezone ID. If you specify the timezone ID, the function returns the timezone ID if it is valid or returns a blank value if the ID is invalid.

Example

In the first example, the TIMEZONE option is set to JST. In the second example, TIMEZONE is set to a blank value. In the third example TIMEZONE is set to user-specified time zone. In the fourth example a valid timezoneid and an invalid timezoneid is displayed.

Statements	Results

`options timezone=jst;`
`data _null_ ;`
`  tzid=tzoneid() ;`
`  put tzid=;`
`run;`
	`tzid=ASIA/TOKYO`

`options timezone='';`
`data _null_ ;`
`  tzid=tzoneid() ;`
`  put tzid=;`
`run;`
	`tzid=`

`options timezone='xxx-12';`
`/* user defined timezone */`
`data _null_ ;`
`  tzid=tzoneid() ;`
`  put tzid=;`
`run;`
	`tzid=`

`data null;`
`  name_valid=tzoneid('asia/tokyo');`
`  name_invalid=tzoneid('Milky Way');`
`  put name_valid =;`
`  put name_invalid=;`
`run;`
	`name_valid=ASIA/TOKYO`
	`name_invalid=`

TZONENAME Function

Returns the current standard or daylight savings time, time zone name.

Category: Date and Time

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONENAME()

TZONENAME<time-zone-id,datetime>

Optional Arguments

time-zone-id
specifies a *region*/*area* value that is defined by SAS. When you specify a zone ID, the time zone that SAS uses is determined by time zone name and daylight savings time rules.

See For a list of the time zone IDs, see Appendix 2, "Time Zone IDs and Time Zone Names," on page 663.

datetime
specifies a SAS datetime value.

Details

The TZONENAME function returns a blank value if the TIMEZONE= option is blank.

The TZONENAME function returns the timezone name based on the specified timezone and datetime. If the SAS datetime is not specified, then the current date is used. If TZID is not specified, then the timezone ID that is specified with the TIMEZONE= option is used.

Example

In the first example, the TIMEZONE option is set to a blank value. In the second example, TIMEZONE is set to timezone name, JST. In the third example, TIMEZONE is set to a user-specified time zone. In the fourth example, TIMEZONE is set to a time zone ID.

Statements	Results

`options tz='';`
`data _null_;`
`  tzname=tzonename() ;`
`  put tzname =;`
`run;`
	`tzname=`

`options tz='jst';`
`data _null_;`
`  tzname=tzonename() ;`
`  put tzname =;`
`run;`
	`tzname=JST`

`options tz='xxx-12';`
`data _null_;`
`  tzname=tzonename() ;`
`  put tzname =;`
`run;`
	`tzname=XXX`

`options tz='American/Chicago';`
`data_null_;`
`  tzname=tzonename('01SEP2014:01:01:01'dt);`
`  put tzname =;`
`run;`
	`tzname=CDT`

TZONEOFF Function

Returns the user time zone offset.

Category: Date and Time

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONEOFF()

TZONEOFF(<time-zone-id, datetime>)

Optional Arguments

time-zone-id
specifies a *region*/*area* value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by time zone name and daylight savings time rules.

See For a list of time zone IDs, see Appendix 2, "Time Zone IDs and Time Zone Names," on page 663.

datetime
specifies a SAS datetime value.

Details

If no arguments are specified, the TZONEOFF function returns the time zone offset for the specified TIMEZONE option. The TZONEOFF (time-zone-id) function with the time zone ID argument returns the time zone offset for the specified time zone ID. The TZONEOFF function with the time zone ID argument returns the time zone offset for the specified time zone name. If the time zone name is not valid for the current locale, you receive an error. We recommend that you use the time zone ID, since it is not locale dependent.

If SASDTM is not provided, TZONEOFF returns the current timezone offset. If SASDTM is provided, it returns the offset to get the local time for specified time value.

Example

The first example has no argument, so the TZONEOFF function returns an offset for the current SAS session. The second example returns an offset based on a specific time zone ID. The third example returns an offset based on a specific time zone ID and a specific date and time. The fourth example returns an offset based on the Time Zone option and a specific date.

If the SAS datetime is not specified, then the TZONEOFF function returns the current timezone offset. If the SAS datetime is specified, then the function returns the offset to provide the local time for the specified time value.

Statements	Results
`option TIMEZONE='AUSTRALIA/MELBOURNE';` `%PUT %SYSFUNC(TZONEOFF());`	`39600`
`option TIMEZONE='AUSTRALIA/MELBOURNE';` `%PUT %SYSFUNC(TZONEOFF(EUROPE/ROME));`	`3600`

`data _null_ ;`
`  dt1='05DEC2012:08:17:52'dt  ;`
`  dt2='05JUN2012:08:17:52'dt  ;`
`  offset1= TZONEOFF('EUROPE/MOSCOW', dt1) ;`
`  offset2= TZONEOFF('EUROPE/MOSCOW', dt2) ;`
`put offset1= / offset2= ;`
`run ;`
	`offset1=10800`
	`offset2=14400`

`option TIMEZONE='EUROPE/MOSCOW' ;`
`data _null_ ;`
`  dt1='05DEC2012:08:17:52'dt  ;`
`  dt2='05JUN2012:08:17:52'dt  ;`
`  offset1= TZONEOFF(dt1) ;`
`  offset2= TZONEOFF(dt2) ;`
`put offset1= / offset2= ;`
`run ;`
	`offset1=10800`
	`offset2=14400`

TZONES2U Function

Converts a SAS date time value to a UTC date time value.

Category: Date and Time

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONES2U <datetime, time-zone-id>

Optional Arguments

datetime
specifies a SAS datetime value.

time-zone-id
specifies a region or area value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by time zone name and daylight savings time rules.

See For a list of time zone IDs, see Appendix 2, “Time Zone IDs and Time Zone Names,” on page 663

Details

The TZONES2U() function returns UTC-based time for the specified TIMEZONE. The TZONES2U(time-zone-id) function with the time zone ID argument returns UTC-based time for the specified time zone ID. If the time zone name is not valid for the current locale, you receive an error.

Example

The following example converts a SAS date time into UTC time.

Statements	Results

`option locale=ja_JP TZ='JST' ;`
`data _null_ ;`
`  dt='05Nov2012:08:17:52'dt  ;`
`  utc1 = tzones2u(dt) ;`
`  utc2 = tzones2u(dt,'ASIA/TOKYO') ;`
`  utc3 = tzones2u(dt,'JST') ;`
`  put dt= /utc1= is8601dz. //;`
`  put dt= /utc2= is8601dz. // ;`
`  put dt= /utc3= is8601dz. // ;`
`run ;`
	`dt=1667722672`
	`utc1=2012-11-04T23:17:52+00:00`
	
	`dt=1667722672`
	`utc2=2012-11-04T23:17:52+00:00`
	
	`dt=1667722672`
	`utc3=2012-11-04T23:17:52+00:00`

TZONEDSTNAME Function

Returns a daylight savings time name.

Category: Date and Time

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONEDSTNAME()

TZONEDSTNAME<*time-zone-id*>

Optional Argument

time-zone-id
specifies a region or area value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by the time zone name and daylight savings time rules.

Details

If the TZID is not specified, then the TIMEZONE ID that is specified by the TIMEZONE= option is used. If the daylight savings time is not specified, then the function returns a blank name.

Example

The following example specifies the America/Chicago time zone.

Statements	Results

`options timezone='America/Chicago';`
`data null;`
`  dstname=tzonedstname();`
`  put dstname;`
`run;`
	`dstname=CDT`

TZONEDSTOFF Function

Returns the time zone offset value for the specified daylight savings time.

Category: Date and Time

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONEDSTOFF()

TZONEDSTOFF<*time-zone id*>

Optional Argument

time zone id
specifies a region or area value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by the time zone name and daylight savings time rules.

Details

If the TZID is not specified, then the TIMEZONE ID that is specified by the TIMEZONE= option is used. If the daylight savings time is not specified, then the function returns a blank name.

Example

The following example specifies the America/Chicago time zone.

Statements	Results

`options timezone='America/Chicago';`
`data null;`
`  dstoff=tzonedstoff();`
`  put dstoff;`
`run;`
	`dstoff=18000`

TZONESTTNAME Function

Returns a standard time zone name.

Category: Date and Time

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONESTTNAME()

TZONESTTNAME<*time-zone id*>

Optional Argument

time-zone-id
specifies a region or area value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by the time zone name and daylight savings time rules.

Details

If the TZID is not specified, then the TIMEZONE ID that is specified by the TIMEZONE= option is used.

Example

The following example specifies the Asia/Osaka time zone.

Statements	Results

`options timezone='Asia/Osaka';`
`data null;`
`  name=tzonesttname();`
`  put name;`
`run;`
	JST

TZONESTTOFF Function

Returns the time zone offset value for the specified standard time.

Category: Date and Time

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONESTTOFF()

TZONESTTOFF<*time-zone id*>

Optional Argument

time-zone-id
specifies a region or area value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by the time zone name and daylight savings time rules.

Details

If the TZID is not specified, then the TIMEZONE ID that is specified by the TIMEZONE= option is used. If standard time is not specified, then the function returns a blank name.

Example

The following example specifies the Asia/Osaka time zone.

Statements	Results

`options timezone='Asia/Osaka';`
`data null;`
`  name=tzonesttoff();`
`  put name;`
`run;`
	`32400`

TZONEU2S Function

Converts a UTC date time value to a SAS date time value.

Category: Date and Time

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

This function is not valid in the CAS server.

Syntax

TZONEU2S <*UTC date time value,time-zone id*>

Optional Arguments

UTC date time value
specifies a Coordinated Universal Time (UTC) datetime value

time-zone id
specifies a region or area value that is defined by SAS. When you specify a zone ID, the time zone that SAS uses is determined by the time zone name and daylight savings time rules.

Details

The TZONEU2S(datetime) function returns the SAS datetime for a UTC time for the specified TIMEZONE option. The TZONEU2S(datetime, time-zone-id) function with the time zone ID argument, returns the SAS datetime for the UTC time for the specified time zone ID. If the time zone name is not valid for the current locale, you receive an error.

Example

The following example converts a UTC date time to three specific SAS date time values.

Statements	Results

`option locale=fr_FR TZ='AMERICA/DENVER';`
`data _null_;`
`  utc_date = '2012-09-02T02:34:56+00:00';`
`  udt = input(utc_date,is8601dz.);`
`  sdt1 = tzoneu2s(udt);`
`  sdt2 = tzoneu2s(udt,'EUROPE/AMSTERDAM');`
`  sdt3 = tzoneu2s(udt,'CET');`
`put sdt1= datetime. / sdt2= datetime. /`
`sdt3= datetime.;`
`run;`
	`sdt1=01SEP12:20:34:56`
	`sdt2=02SEP12:03:34:56`
	`sdt3=02SEP12:03:34:56`

TRANTAB Function

Transcodes data by using the specified translation table.

Category: Character

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

TRANTAB(*string, trantab_name*)

Required Arguments

string
input data that is transcoded.

trantab_name
translation table.

Details

The TRANTAB function transcodes a data string by using a translation table to remap the characters from one internal representation to another. The encoding of the data in the input string must match the encoding of table 1 in the translation table. The TRANTAB function remaps the data from the encoding using table 1.

Translation tables were introduced in SAS 6 to support the requirements of national languages. SAS 8.2 introduced the LOCALE= and ENCODING= system options as an improvement on direct use of translation tables. In SAS 9 translation tables are used only for transcoding external files. There is direct transcoding in SAS files, between the session encodings. SAS 9.2 supports the TRANTAB function for backward compatibility.

The LOCALE= system option is preferred in later SAS releases.

CAUTION:
Only experienced SAS users should use the TRANTAB function.

Example

The following example uses a translation table that transcodes data that is encoded in Latin2 to an uppercase Latin2 encoding:

Statements	Result
`teststrg=trantab('testing','lat2_ucs');` `put teststrg;`	TESTING

UNICODE Function

Converts Unicode characters to the current SAS session encoding.

Category: Character

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

STR=**UNICODE**(<*instr*> (,<Unicode type>))

Required Arguments

str
Data string that has been converted to the current SAS session encoding.

instr
input data string.

Unicode type
Unicode character formats

ESC	Unicode Escape (for example, \u0042). ESC is the default format.
NCR	Numeric Character Representation (for example, &#22823 or &#177 ;)
PAREN	Unicode Parenthesis Escape (for example, <u0061>)
UTF8	UTF8 encoding.
UTF16	UTF16 encoding with big endian. UCS2 is an alias.
UTF16B	UTF16 encoding with big endian. UCS2B is an alias.
UTF16L	UTF16 encoding with little endian. UCS2L is an alias.
UTF32	UTF32 encoding with big endian. UCS4 is an alias.
UTF32B	UTF32 encoding with big endian. UCS4B is an alias.
UTF32L	UTF32 encoding with little endian. UCS4L is an alias.

Details

This function reads Unicode characters and converts them to the current SAS session encoding.

The Unicode function supports the Unicode supplementary characters.

Example

The following example demonstrates the functionality of the UNICODE function:

```
/* Run this program using any CJK LOCALE */
data _null_;
str1=unicode("\u0041\u0042\u0043");put str1=;          /* default */
str2=unicode("\u0041\u0042\u0043", "esc");put str2=;   /* ESC - Unicode Escape */
str3=unicode("&#177;", "ncr");put str3=;               /* NCR - Numeric Character Representation */
str4=unicode("&#22823;", "ncr");put str4=;             /* NCR - Numeric Character Representation */
str5=unicode('<u0061><u0062>', 'paren');put str5=;     /* PAREN - Unicode Parenthesis Escape */
str6=unicode('2759'x, 'ucs2');put str6=;               /* UCS2 - UCS2 encoding */
str7=unicode('5927'x, 'ucs2b');put str7=;              /* UCS2B - UCS2 encoding with big endian */
str8=unicode('2759'x, 'ucs2l');put str8=;              /* UCS2L - UCS2 encoding with little endian */
str9=unicode('27590000'x,'ucs4');put str9=;            /* UCS4 - UCS4 encoding */
str10=unicode('00005927'x,'ucs4b');put str10=;         /* UCS4B - UCS4 encoding with big endian */
str11=unicode('27590000'x,'ucs4l');put str11=;         /* UCS4L - UCS4 encoding with little endian */
str12=unicode('E5A4A7'x, 'utf8');put str12=;           /* UTF8 - UTF8 encoding */
str13=unicode('2759'x, 'utf16');put str13=;            /* UTF16 - UTF16 encoding */
```

```
str14=unicode('5927'x,'utf16b');put str14=;          /* UTF16B - UTF16 encoding with big endian */
str15=unicode('2759'x,'utf16l');put str15=;          /* UTF16L - UTF16 encoding with little endian */
str16=unicode("\u00020bb7", "esc");put str16=;       /* ESC - Unicode Escape for Supplementary Character */
str17=unicode("&#134071;", "ncr");put str17=;        /* NCR - Numeric Character Representation for
                                                        Supplementary Character */
str18=unicode('<u00020BB7>','paren');put str18=;     /* PAREN -  Unicode Parenthesis Escape for
                                                        Supplementary Character*/
run;
```

Here are the results from the UNICODE function example:

```
str1=ABC
str2=ABC
str3=±
str4=大
str5=ab
str6=大
str7=大
str8=大
str9=大
str10=大
str11=大
str12=大
str13=大
str14=大
str15=大
```

UNICODEC Function

Converts characters in the current SAS session encoding to Unicode characters.

Category: Character

Restrictions: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

STR=**UNICODEC**(<*instr*> (,<Unicode type>))

Required Arguments

str
data string that has been converted to Unicode encoding.

instr
input data string.

Unicode type
Unicode character formats

ESC	Unicode Escape (for example, \u0042) ESC is the default format.
NCR	Numeric Character Representation (for example, &#22823 or &#177 ;)

PAREN	Unicode Parenthesis Escape (for example, <u0061>)
UTF8	UTF8 encoding.
UTF16	UTF16 encoding with big endian. UCS2 is an alias.
UTF16B	UTF16 encoding with big endian. UCS2B is an alias.
UTF16L	UTF16 encoding with little endian. UCS2L is an alias.
UTF32	UTF32 encoding with big endian. UCS4 is an alias.
UTF32B	UTF32 encoding with big endian. UCS4B is an alias.
UTF32L	UTF32L encoding with big endian. UCS4L is an alias.

Details

This function reads characters that are in the current SAS session encoding and converts them to Unicode encoding.

Example

The following example demonstrates the functionality of the UNICODEC function:

Statements	Results

`data _null_;`
 	
`  str1=unicodec("ABC", 'utf8');`
`  put str1= $hex12.;`
`  str2=unicodec("ABCé", 'utf8');`
`  put str2= $hex12.;`
`run;`
	`str1=414243202020`
	`str2=414243C3A920`

UNICODELEN Function

Specifies the length of the character unit for the Unicode data.

Category: Character

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

UNICODELEN()

Details

The UNICODELEN function specifies the length of the character unit for the UNICODE data.

Example

This example uses UTF-8 encoding.

Statements	Results
`len1=unicodelen("abc");`	len1=4
`len2=unicodelen("\u0041\u0042\u0043\u5927",'esc');`	len2=4
`len3=unicodelen("大' ,'ncr');`	len3=1
`len4=unicodelen("<u0061><u0062>",'paren');`	len4=2

See Also

Functions:

- "UNICODEWIDTH Function" on page 425

UNICODEWIDTH Function

Specifies the length of a display unit for the Unicode data.

Category: Character

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility on page 279.

Syntax

UNICODEWIDTH()

Details

The UNICODEWIDTH function specifies the length of a display unit for the Unicode data. The display unit displays the width of a character when the character is displayed with fixed width font.

The characters in CJK Symbols, CJK Unified Ideographs, Full width Alphabets and Punctuation, Full width currency symbols, CJK Unified Ideograph Extension B-D and CJK Compatibility Ideographs Supplement have the value of a display unit 2. Other characters are display unit 1.

Example

This example uses UTF-8 encoding.

Statements	Results

`len1=unicodewidth("abc大");`	len1=5
`len2=unicodewidth("\u0041\u0042\u0043\u5927",'esc');`	len2=5
`len3=unicodewidth("大 ",'ncr');`	len3=2
`len4=unicodewidth("<u0061><u0062>",'paren');`	len4=2

See Also

Functions:

- "UNICODELEN Function" on page 424

VARTRANSCODE Function

Returns the transcode attribute of a SAS data set variable.

Category: Variable Information

Syntax

VARTRANSCODE(*data-set-id*, *var-num*)

Required Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
specifies the position of the variable in the SAS data set.

Tip The VARNUM function returns this value.

Details

Transcoding is the process of converting data from one encoding to another. The VARTRANSCODE function returns 0 if the *var-num* variable does not transcode its value, or 1 if the *var-num* variable transcodes its value.

For more information about transcoding variables, see Transcoding on page 19 in *SAS Viya National Language Support: Reference Guide*. For information about encoding values and transcoding data, see SBCS, DBCS, and Unicode Encoding Values When Transcoding SAS Data on page 643 in *SAS Viya National Language Support: Reference Guide*.

Example

The following example shows how to determine whether a character variable is transcoded:

```
data a;
```

```
   attrib x length=$3. transcode=no;
   attrib y length=$3. transcode=yes;
   x='abc';
   y='xyz';
run;
data _null_;
   dsid=open('work.a','i');
   nobs=attrn(dsid,"nobs");
   nvars=attrn(dsid,"nvars");
   do i=1 to nobs;
      xrc=fetch(dsid,1);
      do j=1 to nvars;
         transcode = vartranscode(dsid,j);
         put transcode=;
      end;
   end;
run;
```

SAS writes the following output to the log:

```
transcode=0
transcode=1
```

See Also

Functions:

- "ATTRN Function" in *SAS Viya Functions and CALL Routines: Reference*
- "OPEN Function" in *SAS Viya Functions and CALL Routines: Reference*
- "VARNUM Function" in *SAS Viya Functions and CALL Routines: Reference*
- "VTRANSCODE Function" on page 427
- "VTRANSCODEX Function" on page 428

VTRANSCODE Function

Returns a value that indicates whether transcoding is enabled for the specified character variable.

Categories: Variable Information
CAS

Note: This function supports the varchar type.

Syntax

VTRANSCODE (*var*)

Required Argument

var
specifies a character variable that is expressed as a scalar or as an array reference.

Restriction You cannot use an expression as an argument.

Details

The VTRANSCODE function returns 0 if transcoding is off, and 1 if transcoding is on.

By default, all character variables in the DATA step are transcoded. You can use the TRANSCODE= attribute of the ATTRIB statement to turn transcoding off.

Comparisons

- The VTRANSCODE function returns a value that indicates whether transcoding is enabled for the specified variable. The VTRANSCODEX function, however, evaluates the argument to determine the variable name. The function then returns the transcoding status (on or off) that is associated with that variable name.
- The VTRANSCODE function does not accept an expression as an argument. The VTRANSCODEX function accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, format, and length. For a list of the variable attributes, see the "Variable Information" functions in *SAS Viya Functions and CALL Routines: Reference*.

Example

Statements	Result
	`----+----1----+`

`attrib x transcode = yes;`
`attrib y transcode = no;`
`rc1 = vtranscode(y);`
`put rc1=;`
	`rc1=0`

See Also

Functions:

- "VTRANSCODEX Function" on page 428

Statements:

- ATTRIB in

VTRANSCODEX Function

Returns a value that indicates whether transcoding is enabled for the specified argument.

Categories: Variable Information
CAS

Note: This function supports the varchar type.

Syntax

VTRANSCODEX (*var*)

Required Argument

var

specifies any SAS character expression that evaluates to a character variable name.

Restriction The value of the specified expression cannot denote an array reference.

Details

The VTRANSCODEX function returns 0 if transcoding is off, and 1 if transcoding is on.

By default, all character variables in the DATA step are transcoded. You can use the TRANSCODE= attribute of the ATTRIB statement to turn transcoding off.

Comparisons

- The VTRANSCODE function returns a value that indicates whether transcoding is enabled for the specified variable. The VTRANSCODEX function, however, evaluates the argument to determine the variable name. The function then returns the transcoding status (on or off) that is associated with that variable name.
- The VTRANSCODE function does not accept an expression as an argument. The VTRANSCODEX function accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, format, and length. For a list of the variable attributes, see the “Variable Information” functions in *SAS Viya Functions and CALL Routines: Reference*.

Example

Statements	Result
	`----+----1----+`

`attrib x transcode = yes;`
`attrib y transcode = no;`
`rc1 = vtranscodex('y');`
`put rc1=;`
	`rc1=0`

See Also

Functions:

- “VTRANSCODE Function” on page 427

Statements:

- ATTRIB

Part 6

Informats for NLS

Chapter 13

Dictionary of Informats for NLS

Informats by Category

There are six categories of SAS informats that support NLS:

Category	Description
BIDI Text Handling	Instructs SAS to read bidirectional data values from data variables.
Character	Instructs SAS to read character data values into character variables.
DBCS	Instructs SAS to manage various Asian languages.
Date and Time	Instructs SAS to read data values into variables that represent dates, times, and datetimes.
Hebrew Text Handling	Instructs SAS to read Hebrew data from data variables.
Numeric	Instructs SAS to read numeric data values into numeric variables.

The following table provides brief descriptions of the SAS informats. For more detailed descriptions, see the NLS entry for each informat.

Category	Language Elements	Description
Character	$UCS2B Informat (p. 538)	Reads a character string that is encoded in big-endian, 16-bit, UCS2, Unicode encoding, and then converts the character string to the encoding of the current SAS session.

Category	Language Elements	Description
	$UCS2BE Informat (p. 539)	Reads a character string that is in the encoding of the current SAS session and then converts the character string to big-endian, 16-bit, UCS2, Unicode encoding.
	$UCS2L Informat (p. 540)	Reads a character string that is encoded in little-endian, 16-bit, UCS2, Unicode encoding, and then converts the character string to the encoding of the current SAS session.
	$UCS2LE Informat (p. 542)	Reads a character string that is in the encoding of the current SAS session and then converts the character string to little-endian, 16-bit, UCS2, Unicode encoding.
	$UCS2X Informat (p. 543)	Reads a character string that is encoded in 16-bit, UCS2, Unicode encoding, and then converts the character string to the encoding of the current SAS session.
	$UCS2XE Informat (p. 544)	Reads a character string that is in the encoding of the current SAS session and then converts the character string to 16-bit, UCS2, Unicode encoding.
	$UCS4B Informat (p. 545)	Reads a character string that is encoded in big-endian, 32-bit, UCS4, Unicode encoding, and then converts the character string to the encoding of the current SAS session.
	$UCS4L Informat (p. 546)	Reads a character string that is encoded in little-endian, 32-bit, UCS4, Unicode encoding, and then converts the character string to the encoding of the current SAS session.
	$UCS4X Informat (p. 547)	Reads a character string that is encoded in 32-bit, UCS4, Unicode encoding, and then converts the character string to the encoding of the current SAS session.
	$UCS4XE Informat (p. 548)	Reads a character string that is in the encoding of the current SAS session, and then converts the character string to 32-bit, UCS4, Unicode encoding.
	$UESC Informat (p. 549)	Reads a character string that is encoded in UESC representation, and then converts the character string to the encoding of the current SAS session.
	$UESCE Informat (p. 550)	Reads a character string that is in the encoding of the current SAS session, and then converts the character string to UESC representation.
	$UNCR Informat (p. 551)	Reads an NCR character string, and then converts the character string to the encoding of the current SAS session.
	$UNCRE Informat (p. 552)	Reads a character string in the encoding of the current SAS session, and then converts the character string to NCR.
	$UPAREN Informat (p. 553)	Reads a character string that is encoded in UPAREN representation, and then converts the character string to the encoding of the current SAS session.

Category	Language Elements	Description
	$UPARENE Informat (p. 555)	Reads a character string that is in the encoding of the current SAS session, and then converts the character string to UPAREN representation.
	$UPARENP Informat (p. 556)	Reads a character string that is encoded in UPAREN representation, and then converts the character string to the encoding of the current SAS session, with national characters that remain in the encoding of the UPAREN representation.
	$UTF8X Informat (p. 557)	Reads a character string that is encoded in UTF-8, and then converts the character string to the encoding of the current SAS session.
Date and Time	MINGUO Informat (p. 447)	Reads dates in Taiwanese format.
	NENGO Informat (p. 449)	Reads Japanese date values in the form eyymmdd.
	NLDATE Informat (p. 450)	Reads the date value in the specified locale, and then converts the date value to the local SAS date value.
	NLDATEW Informat (p. 451)	Reads the date value in the specified locale and then converts the date value to the local SAS date and the day of the week.
	NLDATM Informat (p. 452)	Reads the datetime value of the specified locale, and then converts the datetime value to the local SAS datetime value.
	NLDATMAP Informat (p. 453)	Reads the date value in the specified locale, and then converts the date value to the local SAS datetime with either a.m. or p.m.
	NLDATMW Informat (p. 454)	Reads the date value in the specified locale and then converts the date value to the local SAS day of the week and the datetime.
	NLTIMAP Informat (p. 537)	Reads the time value and uses a.m. and p.m. in the specified locale, and then converts the time value to the local SAS time value.
	NLTIME Informat (p. 537)	Reads the time value in the specified locale, and then converts the time value to the local SAS time value.
Hebrew Text Handling	$CPTDW Informat (p. 442)	Reads a character string that is in Hebrew DOS (cp862) encoding, and then converts the character string to Windows (cp1255) encoding.
	$CPTWD Informat (p. 443)	Reads a character string that is in Windows (cp1255) encoding, and then converts the character string to Hebrew DOS (cp862) encoding.
Numeric	EURO Informat (p. 444)	Reads numeric values, removes embedded characters in European currency, and reverses the comma and decimal point.
	EUROX Informat (p. 445)	Reads numeric values and removes embedded characters in European currency.

Category	Language Elements	Description
	NLMNIAED Informat (p. 455)	Reads the monetary format of the international expression for the United Arab Emirates.
	NLMNIAUD Informat (p. 456)	Reads the monetary format of the international expression for Australia.
	NLMNIBGN Informat (p. 457)	Reads the monetary format of the international expression for Bulgaria.
	NLMNIBRL Informat (p. 458)	Reads the monetary format of the international expression for Brazil.
	NLMNICAD Informat (p. 459)	Reads the monetary format of the international expression for Canada.
	NLMNICHF Informat (p. 460)	Reads the monetary format of the international expression for Liechtenstein and Switzerland.
	NLMNICNY Informat (p. 461)	Reads the monetary format of the international expression for China.
	NLMNICZK Informat (p. 462)	Reads the monetary format of the international expression for the Czech Republic.
	NLMNIDKK Informat (p. 463)	Reads the monetary format of the international expression for Denmark, Faroe Island, and Greenland.
	NLMNIEEK Informat (p. 464)	Reads the monetary format of the international expression for Estonia.
	NLMNIEGP Informat (p. 465)	Reads the monetary format of the international expression for Egypt.
	NLMNIEUR Informat (p. 466)	Reads the monetary format of the international expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNIGBP Informat (p. 467)	Reads the monetary format of the international expression for the United Kingdom.
	NLMNIHKD Informat (p. 468)	Reads the monetary format of the international expression for Hong Kong.
	NLMNIHRK Informat (p. 469)	Reads the monetary format of the international expression for Croatia.
	NLMNIHUF Informat (p. 470)	Reads the monetary format of the international expression for Hungary.
	NLMNIIDR Informat (p. 471)	Reads the monetary format of the international expression for Indonesia.

Category	Language Elements	Description
	NLMNIILS Informat (p. 472)	Reads the monetary format of the international expression for Israel.
	NLMNIINR Informat (p. 473)	Reads the monetary format of the international expression for India.
	NLMNIJPY Informat (p. 474)	Reads the monetary format of the international expression for Japan.
	NLMNIKRW Informat (p. 475)	Reads the monetary format of the international expression for South Korea.
	NLMNILTL Informat (p. 476)	Reads the monetary format of the international expression for Lithuania.
	NLMNILVL Informat (p. 477)	Reads the monetary format of the international expression for Latvia.
	NLMNIMOP Informat (p. 478)	Reads the monetary format of the international expression for Macau.
	NLMNIMXN Informat (p. 479)	Reads the monetary format of the international expression for Mexico.
	NLMNIMYR Informat (p. 480)	Reads the monetary format of the international expression for Malaysia.
	NLMNINOK Informat (p. 481)	Reads the monetary format of the international expression for Norway.
	NLMNINZD Informat (p. 482)	Reads the monetary format of the international expression for New Zealand.
	NLMNIPLN Informat (p. 483)	Reads the monetary format of the international expression for Poland.
	NLMNIRUB Informat (p. 484)	Reads the monetary format of the international expression for Russia.
	NLMNISEK Informat (p. 485)	Reads the monetary format of the international expression for Sweden.
	NLMNISGD Informat (p. 486)	Reads the monetary format of the international expression for Singapore.
	NLMNITHB Informat (p. 487)	Reads the monetary format of the international expression for Thailand.
	NLMNITRY Informat (p. 488)	Reads the monetary format of the international expression for Turkey.
	NLMNITWD Informat (p. 489)	Reads the monetary format of the international expression for Taiwan.

Category	Language Elements	Description
	NLMNIUSD Informat (p. 490)	Reads the monetary format of the international expression for Puerto Rico and the United States.
	NLMNIZAR Informat (p. 491)	Reads the monetary format of the international expression for South Africa.
	NLMNLAED Informat (p. 492)	Reads the monetary format of the local expression for the United Arab Emirates.
	NLMNLAUD Informat (p. 493)	Reads the monetary format of the local expression for Australia.
	NLMNLBGN Informat (p. 494)	Reads the monetary format of the local expression for Bulgaria.
	NLMNLBRL Informat (p. 495)	Reads the monetary format of the local expression for Brazil.
	NLMNLCAD Informat (p. 496)	Reads the monetary format of the local expression for Canada.
	NLMNLCHF Informat (p. 497)	Reads the monetary format of the local expression for Liechtenstein and Switzerland.
	NLMNLCNY Informat (p. 498)	Reads the monetary format of the local expression for China.
	NLMNLCZK Informat (p. 499)	Reads the monetary format of the local expression for the Czech Republic.
	NLMNLDKK Informat (p. 500)	Reads the monetary format of the local expression for Denmark, the Faroe Island, and Greenland.
	NLMNLEEK Informat (p. 501)	Reads the monetary format of the local expression for Estonia.
	NLMNLEGP Informat (p. 502)	Reads the monetary format of the local expression for Egypt.
	NLMNLEUR Informat (p. 503)	Reads the monetary format of the local expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNLGBP Informat (p. 504)	Reads the monetary format of the local expression for the United Kingdom.
	NLMNLHKD Informat (p. 505)	Reads the monetary format of the local expression for Hong Kong.
	NLMNLHRK Informat (p. 506)	Reads the monetary format of the local expression for Croatia.
	NLMNLHUF Informat (p. 507)	Reads the monetary format of the local expression for Hungary.
	NLMNLIDR Informat (p. 508)	Reads the monetary format of the local expression for Indonesia.
	NLMNLILS Informat (p. 509)	Reads the monetary format of the local expression for Israel.
	NLMNLINR Informat (p. 510)	Reads the monetary format of the local expression for India.
	NLMNLJPY Informat (p. 511)	Reads the monetary format of the local expression for Japan.

Category	Language Elements	Description
	NLMNLKRW Informat (p. 512)	Reads the monetary format of the local expression for South Korea.
	NLMNLLTL Informat (p. 513)	Reads the monetary format of the local expression for Lithuania.
	NLMNLLVL Informat (p. 514)	Reads the monetary format of the local expression for Latvia.
	NLMNLMOP Informat (p. 515)	Reads the monetary format of the local expression for Macau.
	NLMNLMXN Informat (p. 516)	Reads the monetary format of the local expression for Mexico.
	NLMNLMYR Informat (p. 517)	Reads the monetary format of the local expression for Malaysia.
	NLMNLNOK Informat (p. 518)	Reads the monetary format of the local expression for Norway.
	NLMNLNZD Informat (p. 519)	Reads the monetary format of the local expression for New Zealand.
	NLMNLPLN Informat (p. 520)	Reads the monetary format of the local expression for Poland.
	NLMNLRUB Informat (p. 521)	Reads the monetary format of the local expression for Russia.
	NLMNLSEK Informat (p. 522)	Reads the monetary format of the local expression for Sweden.
	NLMNLSGD Informat (p. 523)	Reads the monetary format of the local expression for Singapore.
	NLMNLTHB Informat (p. 524)	Reads the monetary format of the local expression for Thailand.
	NLMNLTRY Informat (p. 525)	Reads the monetary format of the local expression for Turkey.
	NLMNLTWD Informat (p. 526)	Reads the monetary format of the local expression for Taiwan.
	NLMNLUSD Informat (p. 527)	Reads the monetary format of the local expression for Puerto Rico, and the United States.
	NLMNLZAR Informat (p. 528)	Reads the monetary format of the local expression for South Africa.
	NLMNY Informat (p. 529)	Reads monetary data in the specified locale for the local expression, and then converts the data to a numeric value.
	NLMNYI Informat (p. 530)	Reads monetary data in the specified locale for the international expression, and then converts the data to a numeric value.
	NLNUM Informat (p. 531)	Reads numeric data in the specified locale for local expressions, and then converts the data to a numeric value.
	NLNUMI Informat (p. 533)	Reads numeric data in the specified locale for international expressions, and then converts the data to a numeric value.
	NLPCT Informat (p. 534)	Reads percentage data in the specified locale for local expressions, and then converts the data to a numeric value.

Category	Language Elements	Description
	NLPCTI Informat (p. 535)	Reads percentage data in the specified locale for international expressions, and then converts the data to a numeric value.

Dictionary

$CPTDW Informat

Reads a character string that is in Hebrew DOS (cp862) encoding, and then converts the character string to Windows (cp1255) encoding.

Category: Hebrew Text Handling

Syntax

$CPTDW*w*.

Syntax Description

w

specifies the width of the input field.

Default 200

Range 1–32767

Comparisons

The $CPTDW*w*. informat performs processing that is opposite of the $CPTWD*w*. informat.

Example

The following example uses the input value of 808182.

Statements	Result
	`----+----1----+`
`x=input('808182',$cptdw6.);` `put x;`	אבג

See Also

Formats:

- "$CPTDW Format" on page 88
- "$CPTWD Format" on page 89

Informat:

- "$CPTWD Informat" on page 443

$CPTWD Informat

Reads a character string that is in Windows (cp1255) encoding, and then converts the character string to Hebrew DOS (cp862) encoding.

Category: Hebrew Text Handling

Syntax

$CPTWD*w*.

Syntax Description

w

specifies the width of the input field.

Default	200
Range	1–32767

Comparisons

The $CPTWD*w*. informat performs processing that is opposite of the $CPTDW*w*. informat.

Example

The following example uses the input value of אבג.

Statements	Result
	`----+----1----+`
`x=input ('אבג',$cptwd6.);` `put x;`	€□,

See Also

Formats:

- "$CPTWD Format" on page 89
- "$CPTDW Format" on page 88

Informat:

- "$CPTDW Informat" on page 442

EURO Informat

Reads numeric values, removes embedded characters in European currency, and reverses the comma and decimal point.

Category: Numeric

Syntax

EURO*w.d*

Syntax Description

w

specifies the width of the input field.

Default 6

Range 1–32

d

specifies the power of 10 by which to divide the value. If the data contains decimal points, the *d* value is ignored.

Default 0

Range 0–31

Details

The EURO*w.d* informat reads numeric values and removes embedded euro symbols (E), commas, blanks, percent signs, hyphens, and close parentheses from the input data. A decimal point is assumed to be a separator between the whole number and the decimal portion. The EURO*w.d* informat converts an open parenthesis at the beginning of a field to a minus sign.

Comparisons

- The EURO*w.d* informat is similar to the EUROX*w.d* informat, but EUROX*w.d* reverses the roles of the decimal point and the comma. This convention is common in European countries.
- If no commas or periods appear in the input, then the EURO*w.d* and the EUROX*w.d* informats are interchangeable.

Example

The following table shows input values for currency in euros, the SAS statements that are applied, and the results.

```
data _null_;
```

```
   input x euro10.;
   put x=;
   datalines;
E1
E1.23
1.23
1,234.56
;
run;
SAS Log:
x=1
x=1.23
x=1.23
x=1234.56
```

Values	Statements	Results
		----+----1----2
E1	input x euro10.; put x;	1
E1.23	input x euro10.; put x;	1.23
1.23	input x euro10.; put x;	1.23
1,234.56	input x euro10.; put x;	1234.56

See Also

Formats:

- “EURO Format” on page 96
- “EUROX Format” on page 99

Informat:

- “EUROX Informat” on page 445

EUROX Informat

Reads numeric values and removes embedded characters in European currency.

Category: Numeric

Syntax

EUROX*w.d*

Syntax Description

w

specifies the width of the input field.

Default 6

Range 1–32

d

specifies the power of 10 by which to divide the value. If the data contains a comma, which represents a decimal point, the *d* value is ignored.

Default 0

Range 0–31

Details

The EUROX*w.d* informat reads numeric values and removes embedded euro symbols (E), periods, blanks, percent signs, hyphens, and close parentheses from the input data. A comma is assumed to be a separator between the whole number and the decimal portion. The EUROX*w.d* informat converts an open parenthesis at the beginning of a field to a minus sign.

Comparisons

- The EUROX*w.d* informat is similar to the EURO*w.d* informat, but EURO*w.d* reverses the roles of the comma and the decimal point. This convention is common in English-speaking countries.
- If no commas or periods appear in the input, the EUROX*w.d* and the EURO*w.d* informats are interchangeable.

Example

The following table shows input values for currency in euros, the SAS statements that are applied, and the results.

```
data _null_;
   input x eurox10.;
   put x=;
   datalines;
E1
E1.23
1.23
1,234.56
; run;
SAS Log:
7       input x eurox10.;
8       put x=;
9       datalines;
x=1
```

```
x=123
x=123
x=1.23456
```

Values	Statements	Results
		----+----1----2
E1	input x eurox10.; put x;	1
E1.23	input x eurox10.; put x;	123
1.23	input x eurox10.; put x;	123
1,234.56	input x eurox10.; put x;	1.23456

See Also

Formats:

- “EURO Format” on page 96
- “EUROX Format” on page 99

Informat:

- “EURO Informat” on page 444

MINGUO Informat

Reads dates in Taiwanese format.

Category: Date and Time

Syntax

MINGUO*w.*

Syntax Description

w

specifies the width of the input field.

Default 6

Range 6–10

Details

The general form of a Taiwanese date is *yyyymmdd*:

yyyy
is an integer that represents the year.

mm
is an integer from 01 through 12 that represents the month.

dd
is an integer from 01 through 31 that represents the day of the month.

The Taiwanese calendar uses 1912 as the base year (01/01/01 is January 1, 1912). Dates before 1912 are not valid. Year values do not roll over after 100 years. Instead, they continue to increase.

You can separate the year, month, and day values with any delimiters, such as blanks, slashes, or hyphens, that are permitted by the YYMMDD*w*. informat. If delimiters are used, place them between all the values. If you omit delimiters, be sure to use a leading zero for days or months that have a value less than 10.

Example

The following examples use different dates for input values.

```
input date minguo10.;
put date date9.;
```

```
data _null_;
  input date minguo10.;
  put date date9.;
  datalines;
49/01/01
891215
03-01-01
;
```

Values	Results
	----+----1----+
49/01/01	01JAN1960
891215	15DEC2000
103-01-01	01JAN2014

See Also

Format:

- "MINGUO Format" on page 101

Informat:

- "YYMMDD Informat" in *SAS Viya Formats and Informats: Reference*

NENGO Informat

Reads Japanese date values in the form *eyymmdd*.

Category: Date and Time

Syntax

NENGO*w*.

Syntax Description

w
specifies the width of the input field.

Default 10

Range 7–32

Details

The general form of a Japanese date is *eyymmdd*:

e
is the first letter of the name of the imperial era(Meiji, Taisho, Showa, or Heisei).

yy
is an integer that represents the year.

mm
is an integer from 01 through 12 that represents the month.

dd
is an integer from 01 through 31 that represents the day of the month.

The *e* value can be separated from the integers by a period. If you omit *e*, SAS uses the current imperial era. You can separate the year, month, and day values by blanks or any nonnumeric character. However, if delimiters are used, place them between all the values. If you omit delimiters, be sure to use a leading zero for days or months that are values less than 10.

Example

The following examples use different input values.

```
input nengo_date nengo8.;
put nengo_date date9.;
```

```
data _null_;
   input nengo_date nengo8.;
   put nengo_date date9.;
   put nengo_date= ;
   datalines;
h11108
h.11108
11/10/08
```

```
;
```

Values	Results
	`----+----1----+`
`h11108`	`08OCT1999`
`h.11108`	`08OCT1999`
`11/10/08`	`08OCT1999`

See Also

Formats:

- "NENGO Format" on page 102

NLDATE Informat

Reads the date value in the specified locale, and then converts the date value to the local SAS date value.

Category: Date and Time

Alias: NLDATEW

Syntax

NLDATE*w*.

Syntax Description

w

specifies the width of the input field.

Default 10

Range 10–200

Example

The following examples use the input February 24, 2003.

Statements	Results
	`----+----1----+`

Statements	Results

options locale=English_UnitedStates;
dy='February 24, 2003';
y=input('dy,nldate200.);
put y=;
	15760

options locale=German_Germany;
dy='24. Februar 2003';
y=input(dy,nldate16.);
put y=;
	15760

See Also

Format:

- "NLDATE Format" on page 106

NLDATEW Informat

Reads the date value in the specified locale and then converts the date value to the local SAS date and the day of the week.

Category: Date and Time

Alignment: Left

Syntax

NLDATEW*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 20

Range 16–200

Example

The following examples use the input February 24, 2014.

Statements	Results
	----+----1----+

Statements	Results

`options locale=English_UnitedStates;`
`data;`
`  dy='Monday, February 24, 2014';`
`  y=input(dy,nldatew200.);`
`  put y=;`
`run;`
	19778

`options locale=German_Germany;`
`  dy='Mo. 24.Februar 2014';`
`  y=input(dy,nldatew16.);`
`  put y=;`
`run;`
	19778

NLDATM Informat

Reads the datetime value of the specified locale, and then converts the datetime value to the local SAS datetime value.

Category: Date and Time

Alias: NLDATMW, NLDATMAP

Syntax

NLDATM*w*.

Syntax Description

w

specifies the width of the input field.

Default 19

Range 19–200

Example

The following examples use the input value of February 24, 2003 12:39:43.

Statements	Results
	`----+----1----+`

`options locale=English_UnitedStates;`
`y=input('24.Feb03:12:39:43', nldatm.);`
`put y=;`
	`1361709583`

Statements	Results

`options locale=German_Germany;`
`y=input('24.Februar 2003 12.39 Uhr', nldatm.);`
`put y=;`
	`1330171200`

See Also

Format:

- "NLDATM Format" on page 127

NLDATMAP Informat

Reads the date value in the specified locale, and then converts the date value to the local SAS datetime with either a.m. or p.m.

Category: Date and Time

Alignment: Left

Syntax

NLDATMAP*w*.

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the datetime value to fit the format width.

Default 32

Range 17–200

Example

These examples use the input value of 12:39:43 p.m. on February 24, 2014.

Statements	Results
	`----+----1----+----2----+----3`

`options locale=English_UnitedStates;`
`data;`
`  dy='February 24, 2014 12:39:43 PM';`
`  y=input(dy,nldatmap200.);`
`  put y=;`
`run;`
	1708864783

Statements	Results

`options locale=Spanish_Mexico;`
`data;`
`  dy='24/02/2003 12:39:43 PM';`
`  y=input(dy,nldatmap200.);`
`  put y=;`
`run;`
	1708864783

NLDATMW Informat

Reads the date value in the specified locale and then converts the date value to the local SAS day of the week and the datetime.

Category: Date and Time

Alignment: Left

Syntax

NLDATMW*w*

Syntax Description

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

Default 40

Range 34–200

Example

The following examples use the input Mon, Feb 24, 2014 12:39:43 PM.

Statements	Results
	`----+----1----+`

`options locale=English_UnitedStates;`
`data;`
`  dy='Mon, Feb 24, 2014 12:39:43 PM';`
`  y=input(dy,nldatmw200.);`
`  put y=;`
`run;`
	1708864783

`options locale=German_Germany;`
`  dy='Mo, 24. Feb 2014 12.39 Uhr';`
`  y=input(dy,nldatmw16.);`
`  put y=;`
`run;`
	1708864783

NLMNIAED Informat

Reads the monetary format of the international expression for the United Arab Emirates.

Category: Numeric

Alignment: Left

Syntax

NLMNIAED*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 3

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmniaed32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLAED Informat" on page 492

NLMNIAUD Informat

Reads the monetary format of the international expression for Australia.

Category: Numeric

Alignment: Left

Syntax

NLMNIAUD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmniaud32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLAUD Informat" on page 493

NLMNIBGN Informat

Reads the monetary format of the international expression for Bulgaria.

Category: Numeric

Alignment: Left

Syntax

NLMNIBGN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put('(-1234.56789)',nlmnibgn32.2);
y=put('(-1234.56789)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLBGN Informat" on page 494

NLMNIBRL Informat

Reads the monetary format of the international expression for Brazil.

Category: Numeric

Alignment: Left

Syntax

NLMNIBRL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnibrl32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLBRL Informat" on page 495

NLMNICAD Informat

Reads the monetary format of the international expression for Canada.

Category: Numeric

Alignment: Left

Syntax

NLMNICAD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnicad32.2);
y=input'($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNICAD Format" on page 156

NLMNICHF Informat

Reads the monetary format of the international expression for Liechtenstein and Switzerland.

Category: Numeric

Alignment: Left

Syntax

NLMNICHF*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnichf32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNICHF Format" on page 157

NLMNICNY Informat

Reads the monetary format of the international expression for China.

Category: Numeric

Alignment: Left

Syntax

NLMNICNY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnicny32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNICNY Format" on page 158

NLMNICZK Informat

Reads the monetary format of the international expression for the Czech Republic.

Category: Numeric

Alignment: Left

Syntax

NLMNICZK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmniczk32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLCZK Informat" on page 499

NLMNIDKK Informat

Reads the monetary format of the international expression for Denmark, Faroe Island, and Greenland.

Category: Numeric

Alignment: Left

Syntax

NLMNIDKK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 912

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmndkk32.2);
y=input'($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIDKK Format" on page 160

NLMNIEEK Informat

Reads the monetary format of the international expression for Estonia.

Category: Numeric

Alignment: Left

Syntax

NLMNIEEK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnieek32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLEEK Informat" on page 501

NLMNIEGP Informat

Reads the monetary format of the international expression for Egypt.

Category: Numeric

Alignment: Left

Syntax

NLMNIEGP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 3

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmniegp32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLEGP Informat" on page 502

NLMNIEUR Informat

Reads the monetary format of the international expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.

Category: Numeric

Alignment: Left

Syntax

NLMNIEUR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnieur32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIEUR Format" on page 163

NLMNIGBP Informat

Reads the monetary format of the international expression for the United Kingdom.

Category: Numeric

Alignment: Left

Syntax

NLMNIGBP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnigbp32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIGBP Format" on page 164

NLMNIHKD Informat

Reads the monetary format of the international expression for Hong Kong.

Category: Numeric

Alignment: Left

Syntax

NLMNIHKD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnihkd32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIHKD Format" on page 165

NLMNIHRK Informat

Reads the monetary format of the international expression for Croatia.

Category: Numeric

Alignment: Left

Syntax

NLMNIHRK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnihrk32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLHRK Informat" on page 506

NLMNIHUF Informat

Reads the monetary format of the international expression for Hungary.

Category: Numeric

Alignment: Left

Syntax

NLMNIHUF*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnihuf32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLHUF Informat" on page 507

NLMNIIDR Informat

Reads the monetary format of the international expression for Indonesia.

Category: Numeric

Alignment: Left

Syntax

NLMNIIDR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmniidr32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informats:

- "NLMNLIDR Informat" on page 508

NLMNIILS Informat

Reads the monetary format of the international expression for Israel.

Category: Numeric

Alignment: Left

Syntax

NLMNIILS*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmniils32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIILS Format" on page 169

NLMNIINR Informat

Reads the monetary format of the international expression for India.

Category: Numeric

Alignment: Left

Syntax

NLMNIINR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmniinr32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLINR Informat" on page 510

NLMNIJPY Informat

Reads the monetary format of the international expression for Japan.

Category: Numeric

Alignment: Left

Syntax

NLMNIJPY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 0

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnijpy32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIJPY Format" on page 171

NLMNIKRW Informat

Reads the monetary format of the international expression for South Korea.

Category: Numeric

Alignment: Left

Syntax

NLMNIKRW*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 0

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnikrw32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLKRW Informat" on page 512

NLMNILTL Informat

Reads the monetary format of the international expression for Lithuania.

Category: Numeric

Alignment: Left

Syntax

NLMNILTL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmniltl32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLLTL Informat" on page 513

NLMNILVL Informat

Reads the monetary format of the international expression for Latvia.

Category: Numeric

Alignment: Left

Syntax

NLMNILVL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnilvl32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLLVL Informat" on page 514

NLMNIMOP Informat

Reads the monetary format of the international expression for Macau.

Category: Numeric

Alignment: Left

Syntax

NLMNIMOP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnimop32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLMOP Informat" on page 515

NLMNIMXN Informat

Reads the monetary format of the international expression for Mexico.

Category: Numeric

Alignment: Left

Syntax

NLMNIMXN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnimxn32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLMXN Informat" on page 516

NLMNIMYR Informat

Reads the monetary format of the international expression for Malaysia.

Category: Numeric

Alignment: Left

Syntax

NLMNIMYR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input{'($12,345.67)',nlmnimyr32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIMYR Format" on page 177

NLMNINOK Informat

Reads the monetary format of the international expression for Norway.

Category: Numeric

Alignment: Left

Syntax

NLMNINOK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmninok32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNINOK Format" on page 178

NLMNINZD Informat

Reads the monetary format of the international expression for New Zealand.

Category: Numeric

Alignment: Left

Syntax

NLMNINZD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmninzd32.2);
y=input'($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNINZD Format" on page 179

NLMNIPLN Informat

Reads the monetary format of the international expression for Poland.

Category: Numeric

Alignment: Left

Syntax

NLMNIPLN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnipln32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIPLN Format" on page 180

NLMNIRUB Informat

Reads the monetary format of the international expression for Russia.

Category: Numeric

Alignment: Left

Syntax

NLMNIRUB*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnirub32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIRUB Format" on page 181

NLMNISEK Informat

Reads the monetary format of the international expression for Sweden.

Category: Numeric

Alignment: Left

Syntax

NLMNISEK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnisek32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNISEK Format" on page 182

NLMNISGD Informat

Reads the monetary format of the international expression for Singapore.

Category: Numeric

Alignment: Left

Syntax

NLMNISGD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnisgd32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNISGD Format" on page 183

NLMNITHB Informat

Reads the monetary format of the international expression for Thailand.

Category: Numeric

Alignment: Left

Syntax

NLMNITHB*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnithb32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLTHB Informat" on page 524

NLMNITRY Informat

Reads the monetary format of the international expression for Turkey.

Category: Numeric

Alignment: Left

Syntax

NLMNITRY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnitry32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNLTRY Informat" on page 525

NLMNITWD Informat

Reads the monetary format of the international expression for Taiwan.

Category: Numeric

Alignment: Left

Syntax

NLMNITWD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnitwd32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNITWD Format" on page 186

NLMNIUSD Informat

Reads the monetary format of the international expression for Puerto Rico and the United States.

Category: Numeric

Alignment: Left

Syntax

NLMNIUSD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmniusd32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIUSD Format" on page 187

NLMNIZAR Informat

Reads the monetary format of the international expression for South Africa.

Category: Numeric

Alignment: Left

Syntax

NLMNIZAR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input'($12,345.67)',nlmnizar32.2);
y=input'($12,345.67)'dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNIZAR Format" on page 188

NLMNLAED Informat

Reads the monetary format of the local expression for the United Arab Emirates.

Category: Numeric

Alignment: Left

Syntax

NLMNLAED*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 3

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlaed32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIAED Informat" on page 455

NLMNLAUD Informat

Reads the monetary format of the local expression for Australia.

Category: Numeric

Alignment: Left

Syntax

NLMNLAUD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlaud32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLAUD Format" on page 190

NLMNLBGN Informat

Reads the monetary format of the local expression for Bulgaria.

Category: Numeric

Alignment: Left

Syntax

NLMNLBGN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input(-12345.67,nlmnlbgn32.2);
y=input(-12345.67,dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIBGN Informat" on page 457

NLMNLBRL Informat

Reads the monetary format of the local expression for Brazil.

Category: Numeric

Alignment: Left

Syntax

NLMNLBRL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlbrl32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIBRL Informat" on page 458

NLMNLCAD Informat

Reads the monetary format of the local expression for Canada.

Category: Numeric

Alignment: Left

Syntax

NLMNLCAD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlcad32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLCAD Format" on page 193

NLMNLCHF Informat

Reads the monetary format of the local expression for Liechtenstein and Switzerland.

Category: Numeric

Alignment: Left

Syntax

NLMNLCHF*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlchf32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLCHF Format" on page 194

NLMNLCNY Informat

Reads the monetary format of the local expression for China.

Category: Numeric

Alignment: Left

Syntax

NLMNLCNY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlcny32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLCNY Format" on page 195

NLMNLCZK Informat

Reads the monetary format of the local expression for the Czech Republic.

Category: Numeric

Alignment: Left

Syntax

NLMNLCZK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlczk32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNICZK Informat" on page 462

NLMNLDKK Informat

Reads the monetary format of the local expression for Denmark, the Faroe Island, and Greenland.

Category: Numeric

Alignment: Left

Syntax

NLMNLDKK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnldkk32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLDKK Format" on page 197

NLMNLEEK Informat

Reads the monetary format of the local expression for Estonia.

Category: Numeric

Alignment: Left

Syntax

NLMNLEEK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnleek32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIEEK Informat" on page 464

NLMNLEGP Informat

Reads the monetary format of the local expression for Egypt.

Category: Numeric

Alignment: Left

Syntax

NLMNLEGP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 3

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlegp32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIEGP Informat" on page 465

NLMNLEUR Informat

Reads the monetary format of the local expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.

Category: Numeric

Alignment: Left

Syntax

NLMNLEUR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnleur32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLEUR Format" on page 200

NLMNLGBP Informat

Reads the monetary format of the local expression for the United Kingdom.

Category: Numeric

Alignment: Left

Syntax

NLMNLGBP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlgbp32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLGBP Format" on page 201

NLMNLHKD Informat

Reads the monetary format of the local expression for Hong Kong.

Category: Numeric

Alignment: Left

Syntax

NLMNLHKD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlhkd32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLHKD Format" on page 202

NLMNLHRK Informat

Reads the monetary format of the local expression for Croatia.

Category: Numeric

Alignment: Left

Syntax

NLMNLHRK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlhrk32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-1,234.57`

See Also

Informat:

- "NLMNIHRK Informat" on page 469

NLMNLHUF Informat

Reads the monetary format of the local expression for Hungary.

Category: Numeric

Alignment: Left

Syntax

NLMNLHUF*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlhuf32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIHUF Informat" on page 470

NLMNLIDR Informat

Reads the monetary format of the local expression for Indonesia.

Category: Numeric

Alignment: Left

Syntax

NLMNLIDR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlidr32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIIDR Informat" on page 471

NLMNLILS Informat

Reads the monetary format of the local expression for Israel.

Category: Numeric

Alignment: Left

Syntax

NLMNLILS*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlils32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLILS Format" on page 206

NLMNLINR Informat

Reads the monetary format of the local expression for India.

Category: Numeric

Alignment: Left

Syntax

NLMNLINR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlinr32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIINR Informat" on page 473

NLMNLJPY Informat

Reads the monetary format of the local expression for Japan.

Category: Numeric

Alignment: Left

Syntax

NLMNLJPY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 0

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnljpy32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLJPY Format" on page 208

NLMNLKRW Informat

Reads the monetary format of the local expression for South Korea.

Category: Numeric

Alignment: Left

Syntax

NLMNLKRW*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 0

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlkrw32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIKRW Informat" on page 475

NLMNLLTL Informat

Reads the monetary format of the local expression for Lithuania.

Category: Numeric

Alignment: Left

Syntax

NLMNLLTL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlltl32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNILTL Informat" on page 476

NLMNLLVL Informat

Reads the monetary format of the local expression for Latvia.

Category: Numeric

Alignment: Left

Syntax

NLMNLLVL*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnllvl32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNILVL Informat" on page 477

NLMNLMOP Informat

Reads the monetary format of the local expression for Macau.

Category: Numeric

Alignment: Left

Syntax

NLMNLMOP*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlmop32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIMOP Informat" on page 478

NLMNLMXN Informat

Reads the monetary format of the local expression for Mexico.

Category: Numeric

Alignment: Left

Syntax

NLMNLMXN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlmxn32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNIMXN Informat" on page 479

NLMNLMYR Informat

Reads the monetary format of the local expression for Malaysia.

Category: Numeric

Alignment: Left

Syntax

NLMNLMYR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlmyr32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLMYR Format" on page 214

NLMNLNOK Informat

Reads the monetary format of the local expression for Norway.

Category: Numeric

Alignment: Left

Syntax

NLMNLNOK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlnok32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLNOK Format" on page 215

NLMNLNZD Informat

Reads the monetary format of the local expression for New Zealand.

Category: Numeric

Alignment: Left

Syntax

NLMNLNZD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlnzd32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLNZD Format" on page 216

NLMNLPLN Informat

Reads the monetary format of the local expression for Poland.

Category: Numeric

Alignment: Left

Syntax

NLMNLPLN*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlpln32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLPLN Format" on page 217

NLMNLRUB Informat

Reads the monetary format of the local expression for Russia.

Category: Numeric

Alignment: Left

Syntax

NLMNLRUB*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlrub32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLRUB Format" on page 218

NLMNLSEK Informat

Reads the monetary format of the local expression for Sweden.

Category: Numeric

Alignment: Left

Syntax

NLMNLSEK*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlsek32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLSEK Format" on page 219

NLMNLSGD Informat

Reads the monetary format of the local expression for Singapore.

Category: Numeric

Alignment: Left

Syntax

NLMNLSGD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlsgd32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLSGD Format" on page 220

NLMNLTHB Informat

Reads the monetary format of the local expression for Thailand.

Category: Numeric

Alignment: Left

Syntax

NLMNLTHB*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlthb32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNITHB Informat" on page 487

NLMNLTRY Informat

Reads the monetary format of the local expression for Turkey.

Category: Numeric

Alignment: Left

Syntax

NLMNLTRY*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 4

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnltry32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Informat:

- "NLMNITRY Informat" on page 488

NLMNLTWD Informat

Reads the monetary format of the local expression for Taiwan.

Category: Numeric

Alignment: Left

Syntax

NLMNLTWD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnltwd32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLTWD Format" on page 223

NLMNLUSD Informat

Reads the monetary format of the local expression for Puerto Rico, and the United States.

Category: Numeric

Alignment: Left

Syntax

NLMNLUSD*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlusd32.2);
y=input'($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLUSD Format" on page 224

NLMNLZAR Informat

Reads the monetary format of the local expression for South Africa.

Category: Numeric

Alignment: Left

Syntax

NLMNLZAR*w.d*

Syntax Description

w

specifies the width of the output field.

Default 12

Range 8–32

d

specifies to divide the number by 10^d. If the data contains decimal points, the *d* value is ignored.

Default 2

Range 0–28

Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=input('($12,345.67)',nlmnlzar32.2);
y=input'($12,345.67)',dollar32.2);
```

Statements	Results
	`----+----1----+`
`put x=;`	`-12345.67`
`put y=;`	`-12345.67`

See Also

Format:

- "NLMNLZAR Format" on page 225

NLMNY Informat

Reads monetary data in the specified locale for the local expression, and then converts the data to a numeric value.

Category: Numeric

Syntax

NLMNY*w.d*

Syntax Description

w

specifies the width of the input field.

Default 9

Range 1–32

d

specifies whether to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Details

The NLMNY*w.d* informat reads monetary data in the specified locale for the local expression, and then converts the data to a numeric value. It removes any thousands separators, decimal separators, blanks, the currency symbol, and the close parenthesis from the input data.

Comparisons

The NLMNY*w.d* informat performs processing that is the opposite of the NLMNYI*w.d* informat.

The NLMNY*w.d* informat is similar to the DOLLAR*w.d* informat except that the NLMNY*w.d* informat is locale-specific.

NLMNY returns an error message if you enclose numerical data with apostrophes. The apostrophes specify that the data is character. The following example produces an error message because the numerical value 1 is enclosed in apostrophes.

```
data;
x=input("'1'",NLMNY32.);
put x=;
run;
```

Example

The following examples use the input value of $12,345.67.

Statements	Results
	----+----1----+

options LOCALE=English_UnitedStates;
x=input('($12,345.67)',nlmny32.2);
y=input('($12,345.67)',dollar32.2);
put x=;
put y=;
	-12345.67
	-12345.67

See Also

Formats:

- "NLMNY Format" on page 226
- "NLMNYI Format" on page 227

Informat:

- "NLMNYI Informat" on page 530

NLMNYI Informat

Reads monetary data in the specified locale for the international expression, and then converts the data to a numeric value.

Category: Numeric

Syntax

NLMNYI*w.d*

Syntax Description

w

specifies the width of the input field.

Default 9

Range 1–32

d

specifies whether to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Details

The NLMNYI*w.d* informat reads monetary data in the specified locale for the international expression, and then converts the data to a numeric value. It removes any thousands separators, decimal separators, blanks, the currency symbol, and the close parenthesis from the input data.

Comparisons

The NLMNYI*w.d* informat performs processing that is the opposite of the NLMNY*w.d* informat.

Example

The following examples use the input value of 12,345.67.

Statements	Results
	`----+----1----+`
`options LOCALE=English_UnitedStates;`	`-12345.67`
`x=input('(USD12,345.67)',nlmnyi32.2);`	`-12345.67`
`y=input('$-12,345.67)',dollar32.2);`	
`put x=;`	
`put y=;`	

See Also

Formats:

- "NLMNY Format" on page 226
- "NLMNYI Format" on page 227

Informat:

- "NLMNY Informat" on page 529

NLNUM Informat

Reads numeric data in the specified locale for local expressions, and then converts the data to a numeric value.

Category: Numeric

Syntax

NLNUM*w.d*

Syntax Description

w

specifies the width of the input field.

Default 6

Range 1–32

d

specifies whether to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Details

The NLNUM*w*.*d*) informat reads numeric data in the specified locale for local expressions, and then converts the data to a numeric value. It removes any thousands separators, decimal separators, blanks, the currency symbol, and the close parenthesis from the input data.

NLNUM returns an error message if you enclose numerical data with apostrophes. The apostrophes specify that the data is character. The following example produces an error message because the numeric value 1 is enclosed in apostrophes:

```
data;
x=input("'1'",NLNUM32.);
put x=;
run;
```

Comparisons

The NLNUM*w*.*d* informat performs processing that is opposite to the NLNUMI*w*.*d* informat.

Example

The following example uses –1234356.78 as the input value.

Statements	Results
	`----+----1----+`

`options locale=English_UnitedStates;`
`x=input('-1,234,356.78',nlnum32.2);`
`put x=;`
	`-1234356.78`

See Also

Formats:

- "NLNUM Format" on page 228

- "NLMNYI Format" on page 227

Informat:

- "NLNUMI Informat" on page 533

NLNUMI Informat

Reads numeric data in the specified locale for international expressions, and then converts the data to a numeric value.

Category: Numeric

Syntax

NLNUMI*w.d*

Syntax Description

w

specifies the width of the input field.

Default 6

Range 1–32

d

specifies to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Details

The NLNUMI*w.d* informat reads numeric data in the specified locale for international expressions, and then converts the data to a numeric value. It removes any thousands separators, decimal separators, blanks, the currency symbol, and the close parenthesis from the input data.

Comparisons

The NLNUMI*w.d* informat performs processing that is opposite of the NLNUM*w.d* informat.

Example

The following example uses –1,234,356.78 as the input value.

Statements	Results
	`----+----1----+`

Statements	Results

`options locale=English_UnitedStates;`
`x=input('-1,234,356.78', nlnumi32.2);`
`put x=;`
	`-1234356.78`

See Also

Formats:

- "NLNUM Format" on page 228
- "NLNUMI Format" on page 230
- "NLNUM Informat" on page 531

NLPCT Informat

Reads percentage data in the specified locale for local expressions, and then converts the data to a numeric value.

Category: Numeric

Syntax

NLPCT*w.d*

Syntax Description

w

specifies the width of the input field.

Default 6

Range 1–32

d

specifies whether to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Details

The NLPCT*w.d* informat reads percentage data in the specified locale for local expressions, and then converts the data to a numeric value. It divides the value by 100 and removes any thousands separators, decimal separators, blanks, the percent sign, and the close parenthesis from the input data.

Comparisons

The NLPCT*w.d* informat performs processing that is opposite of the NLPCTI*w.d* informat. The NLPCT*w.d* informat is similar to the PERCENT*w.d* informat except that the NLPCT*w.d* informat is locale-specific.

NLPCT returns an error message if you enclose numerical data with apostrophes. The apostrophes specify that the data is character. The following example produces an error message because the numerical value 1 is enclosed in apostrophes.

```
data;
x=input("'1'",NLPCT32.);
put x=;
run;
```

Example

The following example uses –12,345.67% as the input value.

Statements	Result
	`----+----1----+`
`options LOCALE=English_UnitedStates;`	`-123.4567`
`x=input('-12,345.67%',nlpct32.2);`	`-123.4567`
`y=input('(12,345.67%)',percent32.2);`	
`put x=;`	
`put y=;`	

See Also

Formats:

- "NLPCT Format" on page 231
- "NLPCTI Format" on page 232
- "NLPCTI Informat" on page 535

NLPCTI Informat

Reads percentage data in the specified locale for international expressions, and then converts the data to a numeric value.

Category: Numeric

Syntax

NLPCTI*w.d*

Syntax Description

w

specifies the width of the input field.

Default 6

Range 1–32

d

specifies whether to divide the number by 10^d. If the data contains decimal separators, the *d* value is ignored.

Default 0

Range 0–31

Details

The NLPCTI*w.d* informat reads percentage data in the specified locale for international expressions, and then converts the data to a numeric value. It divides the value by 100 and removes any thousands separators, decimal separators, blanks, the percent sign, and the close parentheses from the input data.

Comparisons

The NLPCTI*w.d* informat performs processing that is opposite of the NLPCT*w.d* informat.

Example

The following example uses -12,345.67% as the input value.

Statements	Results
	`----+----1----+`

`options LOCALE=English_UnitedStates;`
`x=input('-12,345.67%',nlpct32.2);`
`y=input('(12,345.67%)',percent32.2);`
`put x=;`
`put y=;`
	`-123.4567`
	`-123.4567`

See Also

Formats:

- "NLPCT Format" on page 231
- "NLPCTI Format" on page 232

Informat:

- "NLPCT Informat" on page 534

NLTIMAP Informat

Reads the time value and uses a.m. and p.m. in the specified locale, and then converts the time value to the local SAS time value.

Category: Date and Time

Syntax

NLTIMAP*w*.

Syntax Description

w

specifies the width of the input field.

Default 10

Range 4–200

Example

The following example uses 04:24:43 p.m. as the input value.

Statements	Results
	`----+----1----+`

`options locale=English_UnitedStates;`
`y=input('04:24:43 PM',nltimap11.);`
`put y time.;`
	`16:24:43`

`options locale=German_Germany;`
`y=input('16.24 Uhr',nltimap11.);`
`put y time.;`
	`16:24:00`

See Also

Format:

- "NLTIMAP Format" on page 241

NLTIME Informat

Reads the time value in the specified locale, and then converts the time value to the local SAS time value.

Category: Date and Time

Alias: NLTIMAP

Syntax

NLTIME*w*.

Syntax Description

w

specifies the width of the input field.

Default 20

Range 10–200

Example

The following example uses 16:24:43 as the input value.

Statements	Results
	`----+----1----+`

`options locale=English_UnitedStates;`
`y=input('16:24:43',nltime.);`
`put y time.;`
	`16:24:43`

`options locale=German_Germany;`
`y=input('16.24 Uhr',nltime.);`
`put y time.;`
	`16:24:00`

See Also

Format:

- "NLTIME Format" on page 242

$UCS2B Informat

Reads a character string that is encoded in big-endian, 16-bit, UCS2, Unicode encoding, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS2B*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Comparisons

The $UCS2B*w*. informat performs processing that is opposite of the $UCS2BE*w*. informat. If you are processing data within the same operating environment, then use the $UCS2X*w*. informat. If you are processing data from different operating environments, then use the $UCS2B*w*. and $UCS2L*w*. informats.

Example

This example uses the UTF-8 encoding.

Statements	Result
	`----+----1----+`
`x=input('5927'x,$ucs2b.);` `put x=$hex6.;`	`x=E5A4A7`

See Also

Formats:

- "$UCS2B Format" on page 243
- "$UCS2L Format" on page 245
- "$UCS2X Format" on page 248
- "$UTF8X Format" on page 264

Informats:

- "$UCS2L Informat" on page 540
- "$UCS2X Informat" on page 543
- "$UTF8X Informat" on page 557

$UCS2BE Informat

Reads a character string that is in the encoding of the current SAS session and then converts the character string to big-endian, 16-bit, UCS2, Unicode encoding.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS2BE*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Comparisons

The $UCS2BE*w*. informat performs processing that is opposite of the $UCS2B*w*. informat.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1----+`
`ucs2str=input (' 大',` **`$ucs2be4.);`** `put ucs2str=$hex8.;`	`ucs2str=00205927`

See Also

Formats:

- "$UCS2B Format" on page 243
- "$UCS2BE Format" on page 244

Informat:

- "$UCS2B Informat" on page 538

$UCS2L Informat

Reads a character string that is encoded in little-endian, 16-bit, UCS2, Unicode encoding, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS2L*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Comparisons

The $UCS2L*w*. informat performs processing that is opposite of the $UCS2LE*w*. informat. If you are processing data within the same operating environment, then use the $UCS2X*w*.informat. If you are processing data from different operating environments, then use the $UCS2B*w*. and $UCS2L*w*. informats.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1----+`
`x=input('2759'x,$ucs2l.);` `put x=$hex6.;`	`x=E5A4A7`

See Also

Formats:

- "$UCS2B Format" on page 243
- "$UCS2L Format" on page 245
- "$UCS2X Format" on page 248
- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2X Informat" on page 543
- "$UTF8X Informat" on page 557

$UCS2LE Informat

Reads a character string that is in the encoding of the current SAS session and then converts the character string to little-endian, 16-bit, UCS2, Unicode encoding.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS2LE*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Comparisons

The $UCS2LE*w*. informat performs processing that is opposite of the $UCS2L*w*. informat.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1----+`
`ucs2str=input ('大', $ucs2le4.);` `put ucs2str=$hex4;`	`ucs2str=2759`

See Also

Formats:

- "$UCS2L Format" on page 245
- "$UCS2LE Format" on page 247

Informat:

- "$UCS2L Informat" on page 540

$UCS2X Informat

Reads a character string that is encoded in 16-bit, UCS2, Unicode encoding, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS2X*w*.

Syntax Description

w

specifies the width of the output field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Comparisons

The $UCS2X*w*. informat performs processing that is the opposite of the $UCS2XE*w*. informat. If you are processing data within the same operating environment, then use the $UCS2X*w*. informat. If you are processing data from different operating environments, then use the $UCS2B*w*. and $UCS2L*w*. informats.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1----+`
`x=input('5927'x,$ucs2x.);` `put x=$hex6.;`	`x=E29D99`

See Also

Formats:

- "$UCS2B Format" on page 243
- "$UCS2L Format" on page 245
- "$UCS2X Format" on page 248
- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UTF8X Informat" on page 557

$UCS2XE Informat

Reads a character string that is in the encoding of the current SAS session and then converts the character string to 16-bit, UCS2, Unicode encoding.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS2XE*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 16-bit size of the Unicode characters.

Default 8

Range 2–32767

Comparisons

The $UCS2XE*w*. informat performs processing that is opposite of the $UCS2X*w*. informat.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1----+`
`ucs2str=input (' 大', $ucs2xe4.);` `put ucs2str=$hex8;`	`ucs2str=20002759`

See Also

Formats:

- "$UCS2X Format" on page 248

- "$UCS2XE Format" on page 249

Informat:

- "$UCS2X Informat" on page 543

$UCS4B Informat

Reads a character string that is encoded in big-endian, 32-bit, UCS4, Unicode encoding, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS4B*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default 8

Range 4–32767

Comparisons

If you are processing data within the same operating environment, then use the $UCS4X*w*. informat. If you are processing data from different operating environments, then use the $UCS4B*w*. and $UCS4L*w*. informats.

Example

These examples use UTF-8 encoding.

Statements	Result
	`----+----1----+`
`ucs4str=input('大', $ucs4be8.);` `put ucs4str= $hex16.;`	`ucs4str=0000002000005927`

See Also

Format:

- "$UCS4B Format" on page 250

Informats:

- "$UCS4L Informat" on page 546
- "$UCS4X Informat" on page 547

$UCS4L Informat

Reads a character string that is encoded in little-endian, 32-bit, UCS4, Unicode encoding, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS4L*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default 8

Range 4–32767

Comparisons

If you are processing data within the same operating environment, then use the $UCS4X*w*. informat. If you are processing data from different operating environments, then use the $UCS4B*w*. and $UCS4L*w*. informats.

Example

These examples use UTF-8 encoding.

Statements	Result
	`----+----1----+----2----+----3----+`
`z=put('.com',$UCS4L16.);` `put z $hex32.;`	`2E000000630000006F0000006D000000`

See Also

Format:

- "$UCS4L Format" on page 252

Informats:

- "$UCS4B Informat" on page 545
- "$UCS4X Informat" on page 547

$UCS4X Informat

Reads a character string that is encoded in 32-bit, UCS4, Unicode encoding, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS4X*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default 8

Range 4–32767

Comparisons

The $UCS4X*w*. informat performs processing that is the opposite of the $UCS4XE*w*. informat. Use the $UCS4X*w*. informat when you are processing data within the same operating environment. Use the $UCS4B*w*. and $UCS4L*w*. informats when you are processing data from different operating environments.

Example

These examples use UTF-8 encoding and little-endian formatting.

Statements	Results
	`----+----1----+`

`ucs4=put('91e5'x,$ucs4x.);`
`utf8=input(ucs4,$ucs4x.);`
`put ucs4=$hex8. utf8=$hex8.;`
`run;`
	`ucs4=FDFF0000`
	`utf8=EFBFBD20`

See Also

Formats:

- "$UCS2X Format" on page 248
- "$UCS2B Format" on page 243
- "$UCS2L Format" on page 245
- "$UCS4X Format" on page 255
- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UTF8X Informat" on page 557

$UCS4XE Informat

Reads a character string that is in the encoding of the current SAS session, and then converts the character string to 32-bit, UCS4, Unicode encoding.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UCS4XE*w*.

Syntax Description

w

specifies the width of the input field. Specify enough width to accommodate the 32-bit size of the Unicode characters.

Default	8
Range	4–32767

Comparisons

The $UCS4XE*w*. informat performs processing that is the opposite of the $UCS4X*w*. informat.

Example

This example uses UTF-8 encoding.

Statements	Result
	`----+----1----+`
`$ucs4xe4(' 大', $ucs4xe8.);` `put ucs4str=$hex16;`	`ucs4str=2000000027590000`

See Also

Formats:

- "$UCS4X Format" on page 255
- "$UCS4XE Format" on page 256

Informat:

- "$UCS4X Informat" on page 547

$UESC Informat

Reads a character string that is encoded in UESC representation, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UESC*w*.

Syntax Description

w

specifies the width of the output field.

Default 8

Range 1–32767

Details

If the characters are not available on all operating environments (for example, 0–9, a–z, A–Z) they must be represented in UESC representation. The $UESC*w*. informat can be nested.

Comparisons

The $UESC*w*. informat performs processing that is the opposite of the $UESCE*w*. informat.

Example

These examples use the UTF-8 encoding, which is supported under the Linux operating system.

Statements	Results
	`----+----1----+`

`x=input('/u5927', $uesc10.);`
`y=input('\uu5927', $uesc10.);`
`z=input('\uuu5927', $uesc10.);`
`put x;`
`put y;`
`put z;`
	大
	`\u5927`
	`\uu5927`

See Also

Formats:

- "$UESC Format" on page 257
- "$UESCE Format" on page 259

Informat:

- "$UESCE Informat" on page 550

$UESCE Informat

Reads a character string that is in the encoding of the current SAS session, and then converts the character string to UESC representation.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UESCE*w*.

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Details

The $UESCE*w*. informat can be nested.

Comparisons

The $UESCE*w*. informat performs processing that is opposite of the $UESC*w*. informat.

Example

These examples use the Japanese Shift_JIS encoding, which is supported under the Linux operating system.

Statements	Results
	`----+----1----+`
`x=input('大', $uesce10.);` `y=input('¥u5927',$uesce10.);`	`\u5927`

See Also

Formats:

- "$UESC Format" on page 257
- "$UESCE Format" on page 259

Informat:

- "$UESC Informat" on page 549

$UNCR Informat

Reads an NCR character string, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UNCR*w*.

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Details

The input string must contain only characters and NCR. Any national characters must be represented in NCR.

Comparisons

The $UNCR*w*. informat performs processing that is opposite of the $UNCRE*w*. informat.

Example

These examples use UTF-8 encoding.

Statements	Result
	`----+----1----+`

`x=input ('大', $uncr10.);`
`y=input('abc', $uncr10.);`
`put X;`
`put Y;`
	大
	abc

See Also

Formats:

- "$UNCR Format" on page 260
- "$UNCRE Format" on page 261

Informat:

- "$UNCRE Informat" on page 552

$UNCRE Informat

Reads a character string in the encoding of the current SAS session, and then converts the character string to NCR.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UNCRE*w*.

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Details

The output string converts to plain characters and NCR. Any national characters convert to NCR.

Comparisons

The $UNCRE*w*. informat performs processing that is the opposite of the $UNCR*w*. informat.

Example

These examples use UTF-8 encoding.

Statements	Result
	`----+----1----+`
`x=input ('大abc', $uncre12.);` `put x;`	`大abc`

See Also

Formats:

- "$UNCR Format" on page 260
- "$UNCRE Format" on page 261

Informat:

- "$UNCR Informat" on page 551

$UPAREN Informat

Reads a character string that is encoded in UPAREN representation, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UPAREN*w*.

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Details

If the SAS session encoding does not have a corresponding Unicode expression, the expression remains in encoding of the current SAS session.

Comparisons

The $UPAREN*w*. informat performs processing that is opposite of the $UPARENE*w*. informat.

Example

These examples use UTF-8 encoding.

Statements	Results

`v=input('<u0061>',$uparen10.);`
`w=input('<u0062>',$uparen10.);`
`x=input('<u0063>',$uparen10.);`
`y=input('<u0033>',$uparen10.);`
`z=input('<u5927>',$uparen10.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	a
	b
	c
	3
	大

See Also

Formats:

- "$UPAREN Format" on page 262
- "$UPARENE Format" on page 263

Informats:

- "$UPARENE Informat" on page 555
- "$UPARENP Informat" on page 556

$UPARENE Informat

Reads a character string that is in the encoding of the current SAS session, and then converts the character string to UPAREN representation.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UPARENE*w*.

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Comparisons

The $UPARENE*w*. informat performs processing that is opposite of the $UPAREN*w*. informat.

Example

These examples use UTF-8 encoding.

Statements	Results
	`----+----1----+`

`v=input('a',$uparene10.);`
`w=input('b',$uparene10.);`
`x=input('c',$uparene10.);`
`y=input('3',$uparene10.);`
`z=input('大',$uparene10.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	`<u0061>`
	`<u0062>`
	`<u0063>`
	`<u0033>`
	`<u5927>`

See Also

Formats:

- "$UPAREN Format" on page 262

- "$UPARENE Format" on page 263

Informats:

- "$UPAREN Informat" on page 553
- "$UPARENP Informat" on page 556

$UPARENP Informat

Reads a character string that is encoded in UPAREN representation, and then converts the character string to the encoding of the current SAS session, with national characters that remain in the encoding of the UPAREN representation.

Category: Character

Restriction: UTF-8 is the only SAS session encoding supported by SAS Viya.

Syntax

$UPARENP*w*.

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Details

If the UPAREN expression contains a national character, whose value is greater than Unicode 0x00ff, the expression remains as a UPAREN expression.

Example

These examples use UTF-8 encoding.

Statements	Results
	`----+----1----+`

`v=input('<u0061>',$uparenp10.);`
`w=input('<u0062>',$uparenp10.);`
`x=input('<u0063>',$uparenp10.);`
`y=input('<u0033>',$uparenp10.);`
`z=input('<u5927>',$uparepnp10.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	`a`
	`b`
	`c`
	`3`
	`<u5927>`

See Also

Formats:

- "$UPAREN Format" on page 262
- "$UPARENE Format" on page 263

Informats:

- "$UPAREN Informat" on page 553
- "$UPARENE Informat" on page 555

$UTF8X Informat

Reads a character string that is encoded in UTF-8, and then converts the character string to the encoding of the current SAS session.

Category: Character

Syntax

$UTF8X*w*.

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Comparisons

This example uses the Japanese UTF-8 encoding, which is supported under the Linux operating environment.

Statements	Result
	`----+----1----+`
`x=input ('e5a4a7'x, $utf8x3.);` `put x;`	大

See Also

Formats:

- "$UCS2B Format" on page 243

- "$UCS2L Format" on page 245
- "$UCS2X Format" on page 248
- "$UTF8X Format" on page 264

Informats:

- "$UCS2B Informat" on page 538
- "$UCS2L Informat" on page 540
- "$UCS2X Informat" on page 543

Part 7

Macro Functions for NLS

Chapter 14

Dictionary of Macro Functions for NLS

Macro Functions by Category

The following table provides brief descriptions of the SAS NLS macro functions. For more information, see the NLS entry for each macro function.

Category	Language Elements	Description
DBCS	%KCMPRES and %QKCMPRES Macro Function (p. 562)	Compresses multiple blanks and removes leading and trailing blanks.
	%KINDEX Macro Function (p. 562)	Returns the position of the first character of a string.
	%KLEFT and %QKLEFT Macro Functions (p. 563)	Left-aligns an argument by removing leading blanks.
	%KLENGTH Macro Function (p. 563)	Returns the length of a string.
	%KSCAN and %QKSCAN Functions (p. 564)	Search for a word that is specified by its position in a string.
	%KSUBSTR and %QKSUBSTR Macro Functions (p. 566)	Produce a substring of a character string.

Category	Language Elements	Description
	%KUPCASE and %QKUPCASE Macro Functions (p. 568)	Convert values to uppercase.

Dictionary

%KCMPRES and %QKCMPRES Macro Function

Compresses multiple blanks and removes leading and trailing blanks.

Category: DBCS

Type: NLS macro function

Syntax

%KCMPRES (*text* | *text expression*)

%QKCMPRES (*text* | *text expression*)

Details

The %KCMPRES and %QKCMPRES macro functions compress multiple blanks and removes leading and trailing blanks. %KCMPRES returns an unquoted result, even if the argument is quoted. %QKCMPRES returns a quoted result.

%QKCMPRES produces a result with the following special characters and mnemonic operators masked, so the macro processor interprets them as text instead of as elements of the macro language:

```
& % ' " ( ) + - * / < > = ^ ~ ; , # blank AND OR NOT EQ NE LE LT GE GT IN
```

%KINDEX Macro Function

Returns the position of the first character of a string.

Category: DBCS

Type: NLS macro function

Syntax

%KINDEX (*source*, *string*)

Required Arguments

source
is a character string or text expression.

string
is a character string or text expression.

Details

The %KINDEX function searches *source* for the first occurrence of *string* and returns the position of its first character. If *string* is not found, the function returns 0.

Example: Locating a Character

The following statements find the first character **v** in a string:

```
%let a=a very long value;
%let b=%kindex(&a,v);
%put V appears at position &b..;
```

When these statements execute, the following line is written to the SAS log:

```
V appears at position 3.
```

%KLEFT and %QKLEFT Macro Functions

Left-aligns an argument by removing leading blanks.

Category: DBCS

Requirement: MAUTOSOURCE system option

Syntax

%KLEFT (*text* | *text expression*)

%QKLEFT (*text* | *text expression*)

Details

The %KLEFT and %QKLEFT macro functions left-align arguments by removing leading blanks. If the argument contains a special character or mnemonic operator, listed here, use QKLEFT.

%KLEFT returns an unquoted result, even if the argument is quoted. %QKLEFT produces a result with the following special characters and mnemonic operators masked so that the macro processor interprets them as text instead of as elements of the macro language:

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

%KLENGTH Macro Function

Returns the length of a string.

Category: DBCS

Type: NLS macro function

Syntax

%KLENGTH (*character string* | *text expression*)

Details

If the argument is a character string, %KLENGTH returns the length of the string. If the argument is a text expression, %KLENGTH returns the length of the resolved value. If the argument has a null value, %KLENGTH returns 0.

Example: Returning String Lengths

The following statements find the lengths of character strings and text expressions:

```
%let a=Happy;
%let b=Birthday;
%let c=René;
%put The length of &a is %klength(&a).;
%put The length of &b is %klength(&b).;
%put The length of &c is %klength(&c).;
%put The length of &a &b To &c is %klength(&a &b to &c).;

%put with LENGTH macro %length(&c);
```

When these statements execute, the following is written to the SAS log:

```
The length of Happy is 5.
 The length of Birthday is 8.
 The length of René is 4.
 The length of Happy Birthday To René is 22.

 with LENGTH macro 5
```

%KSCAN and %QKSCAN Functions

Search for a word that is specified by its position in a string.

Category: DBCS

Type: NLS macro function

Syntax

%KSCAN (*argument*, *n*<,*delimiters* >)

%QKSCAN (*argument*, *n*<,*delimiters* >)

Required Arguments

argument
is a character string or a text expression. If *argument* contains a special character or mnemonic operator, listed here, use %QKSCAN.

n
is an integer or a text expression that yields an integer, which specifies the position of the word to return. If *n* is greater than the number of words in *argument*, the functions return a null string. If *n* is negative, %KSCAN examines the character string and selects the word that starts at the end of the string and searches backward.

delimiter
specifies a character variable that produces characters that you want %QKSCAN to use as word separators in the character expression.

Details

The %KSCAN and %QKSCAN functions search *argument* and return the *n*th word. A word is one or more characters separated by one or more delimiters.

%KSCAN does not mask special characters or mnemonic operators in its results, even when the argument was previously masked by a macro quoting function. %QKSCAN masks the following special characters and mnemonic operators in its results:

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

In the %KSCAN function, *word* refers to a substring that has all of the following characteristics:

- is bounded on the left by a delimiter or the beginning of the string
- is bounded on the right by a delimiter or the end of the string
- contains no delimiters

A word can have a length of zero if there are delimiters at the beginning or end of the string or if the string contains two or more consecutive delimiters.

If you use the %KSCAN function with only two arguments, then the default delimiters depend on whether your computer uses ASCII or EBCDIC characters:

- If your computer uses ASCII characters, then the default delimiters are as follows:

  blank ! $ % & ( ) * + , - . / ; < ^¦

  In ASCII environments that do not contain the ^ character, the %KSCAN function uses the ~ character instead.

- If your computer uses EBCDIC characters, then the default delimiters are as follows:

  blank ! $ % & ( ) * + , - . / ; < ¬ | ¢¦

The %KSCAN function allows character arguments to be null. Null arguments are treated as character strings with a length of zero. Numeric arguments cannot be null.

Example: Comparing the Actions of %KSCAN and %QKSCAN

This example illustrates the actions of %KSCAN and %QKSCAN:

```
%macro a;
   aaaaaa
```

```
%mend a;
%macro b;
   bbbbbb
%mend b;
%macro c;
   cccccc
%mend c;
%let x=%nrstr(%a*%b*%c);
%put X: &x;
%put The third word in X, with KSCAN: %kscan(&x,3,*);
%put The third word in X, with QKSCAN: %qkscan(&x,3,*);
```

The %PUT statement writes these lines to the log:

```
X: %a*%b*%c
The third word in X, with KSCAN: cccccc
The third word in X, with QKSCAN: %c
```

%KSUBSTR and %QKSUBSTR Macro Functions

Produce a substring of a character string.

Category: DBCS

Type: NLS macro function

Syntax

%KSUBSTR (*argument*, *position*<, *length*>)

%QKSUBSTR (*argument*, *position*<, *length*>)

Required Arguments

argument
is a character string or a text expression. If *argument* contains a special character or mnemonic operator, listed here, use %QKSUBSTR.

position
is an integer or an expression (text, logical, or arithmetic) that yields an integer that specifies the position of the first character in the substring. If *position* is greater than the number of characters in the string, %KSUBSTR and %QKSUBSTR issue a warning message and return a null value.

length
is an optional integer or an expression (text, logical, or arithmetic) that yields an integer that specifies the number of characters in the substring. If *length* is greater than the number of characters following *position* in *argument*, %KSUBSTR and %QKSUBSTR issue a warning message and return a substring containing the characters from *position* to the end of the string. By default, %KSUBSTR and %QKSUBSTR produce a string containing the characters from *position* to the end of the character string.

Details

The %KSUBSTR and %QKSUBSTR functions produce a substring of *argument*, which begins at *position* and continues for the number of characters in *length*.

%KSUBSTR does not mask special characters or mnemonic operators in its result. %QKSUBSTR masks the following special characters and mnemonic operators:

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , #  blank
AND OR NOT EQ NE LE LT GE GT IN
```

Examples

Example 1: Limiting a Fileref to Eight Characters

The macro MAKEFREF uses %KSUBSTR to assign the first eight characters of a parameter as a fileref, in case a user assigns one that is longer:

```
%macro makefref(fileref,file);
   %if %klength(&fileref) gt 8 %then
       %let fileref = %ksubstr(&fileref,1,8);
   filename &fileref "&file";
%mend makefref;
%makefref(humanresource,/dept/humanresource/report96)
```

SAS reads the following statement:

```
FILENAME HUMANRES "/dept/humanresource/report96";
```

Example 2: Storing a Long Macro Variable Value in Segments

The macro SEPMSG separates the value of the macro variable MSG into 40-character units and stores each unit in a separate variable:

```
%macro sepmsg(msg);
   %let i=1;
   %let start=1;
   %if %length(&msg)>40 %then
      %do;
          %do %until(%klength(&&msg&i)<40);
             %let msg&i=%qksubstr(&msg,&start,40);
             %put Message &i is: &&msg&i;
             %let i=%eval(&i+1);
             %let start=%eval(&start+40);
             %let msg&i=%qksubstr(&msg,&start);
          %end;
          %put Message &i is: &&msg&i;
      %end;
   %else %put No subdivision was needed.;
%mend sepmsg;
%sepmsg(%nrstr(A character operand was found in the %EVAL function
or %IF condition where a numeric operand is required.  A character
operand was found in the %EVAL function or %IF condition where a
numeric operand is required.));
```

When this program executes, these lines are written to the SAS log:

```
Message 1 is: A character operand was found in the %EV
Message 2 is: AL function or  %IF condition where a nu
Message 3 is: meric operand is required.  A character
Message 4 is: operand was  found in the %EVAL function
Message 5 is: or %IF condition where a numeric operan
Message 6 is: d is required.
```

Example 3: Comparing the Actions of %KSUBSTR and %QKSUBSTR

%KSUBSTR produces a resolved result because it does not mask special characters and mnemonic operators in the C language before processing it:

```
%let a=one;
%let b=two;
%let c=%nrstr(&a &b);
%put C: &c;
%put With KSUBSTR: %ksubstr(&c,1,2);
%put With QKSUBSTR: %qKsubstr(&c,1,2);
```

When these statements execute, these lines are written to the SAS log:

```
C: &a &b
With KSUBSTR: one
With QKSUBSTR: &a
```

%KUPCASE and %QKUPCASE Macro Functions

Convert values to uppercase.

Category: DBCS

Type: NLS macro function

Syntax

%KUPCASE (*character string* | *text expression*)

%QKUPCASE (*character string* | *text expression*)

Details

The %KUPCASE and %QKUPCASE functions convert lowercase characters in the argument to uppercase. %KUPCASE does not mask special characters or mnemonic operators in its results.

If the argument contains a special character or mnemonic operator, listed here, use %QKUPCASE. %QKUPCASE masks the following special characters and mnemonic operators in its results:

```
& % ' " ( ) + - * / < > = ¬ ^ ~ ; , # blank
AND OR NOT EQ NE LE LT GE GT IN
```

%KUPCASE and %QKUPCASE are useful in comparing values because the macro facility does not automatically convert lowercase characters to uppercase before comparing them.

Examples

Example 1: Capitalizing a Value to Be Compared

In this example, the macro RUNREPT compares a value input for the macro variable MONTH to the string DEC. If the uppercase value of the response is DEC, then PROC FSVIEW runs on the data set REPORTS.ENDYEAR. Otherwise, PROC FSVIEW runs on the data set with the name of the month in the REPORTS data library.

```
%macro runrept(month);
   %if %kupcase(&month)=DEC %then
       %str(proc fsview data=reports.endyear; run;);
   %else %str(proc fsview data=reports.&month; run;);
%mend runrept;
```

You can invoke the macro in any of these ways to satisfy the %IF condition:

```
%runrept(DEC)
%runrept(Dec)
%runrept(dec)
```

Example 2: Comparing %KUPCASE and %QKUPCASE

These statements show the results produced by %KUPCASE and %QKUPCASE:

```
%let a=begin;
%let b=%nrstr(&a);
%put KUPCASE produces: %kupcase(&b);
%put QKUPCASE produces: %qkupcase(&b);
```

When these statements execute, the following is written to the SAS log:

```
KUPCASE produces: BEGIN
QKUPCASE produces: &A
```

Part 8

System Options for NLS

Chapter 15

Dictionary of System Options for NLS

System Option Entries by Category

The language control category of SAS system options is affected by NLS. The following table provides brief descriptions of the SAS system options. For more detailed descriptions, see the dictionary entry for each SAS system option.

Category	Language Elements	Description
Environment Control: Language Control	DFLANG= System Option (p. 575)	Specifies the language for international date informats and formats.
	ENCODING System Option (p. 577)	Specifies the default character-set encoding for the SAS session.
	LOCALE System Option (p. 578)	Specifies a set of attributes in a SAS session that reflect the language, local conventions, and culture for a geographical region.

Category	Language Elements	Description
	LOCALEDATA System Option (p. 579)	Specifies the source database for the locale information.
	LOGLANGENG System Option (p. 581)	Specifies using the English language for SAS log message text when the LOCALE option is set after start-up.
	MAPEBCDIC2ASCII= System Option (p. 583)	Specifies a translation table that SAS uses to transcode from EBCDIC to ASCII and from ASCII to EBCDIC.
	ODSLANGCHG System Option (p. 585)	Determines whether the language of the text of the ODS output can be changed
	TIMEZONE= System Option (p. 588)	Specifies the user local time zone.
Files: External Files	BOMFILE System Option (p. 574)	Specifies whether to write the byte-order mark (BOM) prefix on Unicode-encoded external files.
Files: SAS Files	RSASIOTRANSERROR System Option (p. 586)	Displays a transcoding error when invalid data is read from a remote application.
	VALIDMEMNAME= System Option (p. 589)	Specifies the rules for naming SAS data sets, SAS data views, and item stores.
	VALIDVARNAME= System Option (p. 592)	Specifies the rules for valid SAS variable names that can be created and processed during a SAS session.
Language Control	LSWLANG System Option (p. 582)	Specifies the language for the language switching feature when the LOGLANGCHG or ODSLANGCHG system option is set at SAS invocation.
Sort: Procedure Options	SORTSEQ= System Option (p. 587)	Specifies a language-specific collating sequence for the SORT and SQL procedures to use in the current SAS session.

Dictionary

BOMFILE System Option

Specifies whether to write the byte-order mark (BOM) prefix on Unicode-encoded external files.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category: Files: External Files

PROC OPTIONS GROUP= EXTFILES

Syntax

BOMFILE | NOBOMFILE

Syntax Description

BOMFILE
Specifies to write a byte-order mark (BOM) prefix when a Unicode-encoded file is written to an external file.

NOBOMFILE
Specifies not to write a BOM prefix when a Unicode-encoded file is written to an external file.

Details

The BOMFILE system option does not apply when a Unicode-encoded external file is read.

A BOM is a signature at the beginning of a Unicode data stream. The size of the BOM varies depending on the encoding.

DFLANG= System Option

Specifies the language for international date informats and formats.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Environment Control: Language Control
PROC OPTIONS GROUP=	LANGUAGECONTROL
Default:	English

Syntax

DFLANG='*language*', **locale**

Syntax Description

'language'
specifies the language that is used for international date informats and formats.

These languages are valid values for *language*:

- Afrikaans
- Catalan
- Croatian
- Czech
- Danish
- Dutch
- English

- Finnish
- French
- German
- Hungarian
- Italian
- Japanese
- Macedonian
- Norwegian
- Polish
- Portuguese
- Russian
- Slovenian
- Spanish
- Swedish
- Swiss_French
- Swiss_German

locale
the locale that is specified with the locale system option becomes the active locale.

Details

You can change the value of the DFLANG system option during a SAS session, but you can use only one language at a time. The values for *language* are not case-sensitive.

When you specify **DFLANG=locale**, the locale that is specified in the system option of the LOCALE statement becomes the active locale. The locale or language must be supported by the DFLANG system option.

In the following example, the international date informats and formats would be German. The posix name for the German locale is de_DE. The German locale is supported by the DFLANG system option.

```
option locale=de_DE; /* German locale */
  option  DFLANG=locale;
```

In the following example, the international date informats and formats would be English. Maltese is not supported by DFLANG, so the default locale is English.

```
option locale=mt_MT;  /* Maltese locale */
  option  DFLANG=locale;
```

When you specify **DFLANG=locale**, the output of the date format is displayed in the locale that is specified with the LOCALE system option. To control the date format in the output, the DFLANG locale uses the value based on the LOCALE system option that has been set at start-up. If DFLANG is set to a valid language, then the date format in the output is English by default. In the following example, the locale is set to French.

```
Sas.exe -locale French
  Proc print data=sashelp.class ; run ;
```

```
mercredi 09 mars 2011 14 h 25
```

If you set **DFLANG=JAPANESE**, then DFLANG behaves the same way as **DFLANG=LOCALE**.

The value JAPANESE is supported to allow customers running the Japanese image to see the datetime stamp in the SAS Log and the Listing formatted for Japanese. The value LOCALE was added later to give all customers the same feature.

- If DFLANG is not set at start up, the option value is set by the LOCALE option.
- The default format of the datetime stamp of the SAS log is English. If DFLANG is set to LOCALE or JAPANESE, the datetime stamp in the SAS Log is formatted according to the current SAS LOCALE.
- The default format of the datetime stamp of the SAS listing is English. If DFLANG is set to LOCALE or JAPANESE, the datetime stamp in each page of the output sent to the SAS Listing or ODS destinations is formatted according to the SAS LOCALE that was in effect immediately after DFLANG=LOCALE is set.
- Changes to the LOCALE option do not impact how the datetime stamp is formatted unless the DTRESET option is enabled. Then the datetime stamp is formatted according to the current SAS LOCALE.

You can use the value **JAPANESE** to process the Japanese image to see the datetime stamp in the SAS log and the listing formatted for Japanese.

See Also

DTRESET System Option

ENCODING System Option

Specifies the default character-set encoding for the SAS session.

Valid in: configuration file, SAS invocation

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Syntax

-ENCODING= *encoding-value*

Required Argument

encoding-value
UTF-8

Note: UTF-8, Unicode encoding is the only supported encoding.

Details

A character-set encoding is a set of characters that have been mapped to numeric values called code points.

UTF-8 is the only SAS session encoding supported by SAS Viya. The encoding for a SAS session is determined by the values of the ENCODING= option. If ENCODING= is not specified, UTF-8 is set as default.

See Also

Conceptual Information:

- "Overview of Locale Concepts for NLS" on page 5
- "Overview: Encoding for NLS" on page 9
- "Overview to Transcoding" on page 19
- Chapter 18, "Encoding Values in SAS Language Elements," on page 643

LOCALE System Option

Specifies a set of attributes in a SAS session that reflect the language, local conventions, and culture for a geographical region.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: English_UnitedStates

Operating environment: Also valid in SASV9_OPTIONS environment variable

Syntax

-LOCALE *locale-name*

LOCALE=*locale-name*

Required Argument

locale-name

For a complete list of locale values (SAS names and POSIX names), see "LOCALE= Values for PAPERSIZE and DFLANG Options" on page 629.

Details

The LOCALE= system option is used to specify the locale, which reflects the local conventions, language, and culture a geographical region.

If the value of the LOCALE= system option is not compatible with the value of the ENCODING= system option, the character-set encoding is determined by the value of the ENCODING= system option.

When you set a value for LOCALE=, the value of the following system options is modified unless explicit values have been specified:

DATESTYLE=

When LOCALE= is set, the DATESTYLE= system option uses the value that corresponds to the chosen locale.

DFLANG=
When LOCALE= is set, the DFLANG= system option is set to a value that corresponds to the chosen locale.

PAPERSIZE=
When LOCALE= is set, the PAPERSIZE= system option is set to a value that corresponds to the chosen locale and the ODS printer is set to the preferred unit of measurement, inches or centimeters, for that locale.

See Also

Conceptual Information:

- Chapter 2, "Locale for NLS," on page 5
- "LOCALE= Values for PAPERSIZE and DFLANG Options" on page 629

System Options:

- "ENCODING System Option" on page 577
- "DFLANG= System Option" on page 575

LOCALEDATA System Option

Specifies the source database for the locale information.

Valid in: configuration file, SAS invocation

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: SASLOCALE

Syntax

LOCALEDATA=SASLOCALE | REGISTRY

Syntax Description

SASLOCALE
specifies the locale definition from an internal SAS database.

REGISTRY
specifies the locale definition from the SAS registry.

Details

The SASLOCALE value system option specifies an internal database as the source of the locale's definition. This database does not allow customization.

The REGISTRY value specifies an external database where locale definitions are customized by using PROC LOCALEDATA.

LOGLANGCHG System Option

Specifies whether the language of the text output to the SAS log can be changed.

Valid in: configuration file, SAS invocation

PROC OPTIONS GROUP= LOGCONTROL

Default: NOLOGLANGCHG

Syntax

LOGLANGCHG | NOLOGLANGCHG

Syntax Description

LOGLANGCHG
specifies that the language of the SAS log messages can be changed after start-up.

NOLOGLANGCHG
specifies that the language of the SAS log message cannot be changed after start-up.

Details

The following conditions result in these actions:

ODSLANGCHG	LOGLANGCHG	Actions
ON	ON	Language for ODS output and the SAS log can be changed based on LSWLANG= option setting.
OFF	OFF	Language for ODS output and the SAS log cannot be changed.
OFF	ON	Language for the SAS log can be changed based on LSWLANG= option setting.
ON	OFF	Language for ODS output can be changed based on LSWLANG= option.

See Also

- "ODSLANGCHG System Option" on page 585
- "LSWLANG System Option" on page 582
- "LOGLANGENG System Option" on page 581

LOGLANGENG System Option

Specifies using the English language for SAS log message text when the LOCALE option is set after start-up.

Valid in: configuration file, SAS invocation

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: NOLOGLANGENG

Syntax

LOGLANGENG | **NOLOGLANGENG**

Syntax Description

LOGLANGENG
SAS log messages are displayed in English using the LSW feature.

NOLOGLANGENG
The LSW feature is not used to enable English for SAS log messages.

Details

The LOGLANGENG option interacts with the LOGLANGCHG, ODSLANGCHG, and LSWLANG options.

- LOGLANGENG has the same functionality as LSWLANG=EN and NOLOGLANGCHG.
- ODSLANGCHG is valid with LOGLANGENG.
- NOLOGLANGENG has no impact on the SAS session.
- During start-up, the configuration file (!sasroot\nls) and the LOCALE option determine the language for SAS messages. After start-up, if the LOCALE option is set and the LOGLANGENG option is on, the language for the SAS log output is English.
- If LOGLANGENG is enabled at start-up and the value of LOCALE is changed during the session, the SAS log output remains in English.
- If LOGLANGCHG is on, the SAS log is controlled by the LSWLANG= option.
- If ODSLANGCHG is on, the ODS text is controlled by the LSWLANG= option.
- If LOGLANGENG is not enabled at start-up and LOCALE is changed during the session, the SAS log output is displayed in the language that was set at start-up, unless both LOGLANGCHG and LSWLANG= are specified and LSWLANG has a value other than LOCALE.

Example

This example is a French SAS session with LOGLANGENG set to ON.

If LOGLANGENG is set to ON, then LSWLANG=EN and LOGLANGCHG is ON automatically. The SAS session always uses English for the SAS log, but the language of the ODS output is determined by the current SAS configuration.

See Also

- "LSWLANG System Option" on page 582
- "LOGLANGCHG System Option" on page 580
- "ODSLANGCHG System Option" on page 585

LSWLANG System Option

Specifies the language for the language switching feature when the LOGLANGCHG or ODSLANGCHG system option is set at SAS invocation.

Valid in: configuration file, SAS invocation

Category: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: LOCALE

Syntax

LSWLANG=LOCALE | *language*

Required Arguments

LOCALE
specifying the LOCALE argument preserves the behavior prior to SAS 9.4, where the SAS message text matches the value of the LOCALE= option.

language
Specifies the language for ODS and the SAS log output. The following values can be specified:

Code	Language
EN	English
FR	French
IT	Italian
DE	German
ES	Spanish
ZH	Chinese S
ZT	Chinese T

Code	Language
JA	Japanese
KO	Korean
PL	Polish
RU	Russian

See Also

- "ODSLANGCHG System Option" on page 585
- "LOGLANGCHG System Option" on page 580
- "LOGLANGENG System Option" on page 581

MAPEBCDIC2ASCII= System Option

Specifies a translation table that SAS uses to transcode from EBCDIC to ASCII and from ASCII to EBCDIC.

Valid in: configuration file, SAS invocation

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Alias: MAPE2A

Interaction: The MAPEBCDIC2ASCII= system option specifies a translation table to use for the SAS session. The table specified with MAPEBCDIC2ASCII is used by components such as $EBCDIC and $ASCII formats.

Syntax

MAPEBCDIC2ASCII= TRANTAB *catalog-entry*

Required Argument

catalog-entry
specifies a SAS catalog TRANTAB entry that contains translation tables. If you specify *entry-name.type*, SAS first searches SASUSER.PROFILE, then SASHELP.LOCALE, and then SASHELP.HOST for the name specified.

Details

MAPEBCDIC2ASCII= supports the requirements of national languages.

CAUTION:
Do not change a translation table unless you are familiar with its purpose.
Translation tables are used internally by SAS to implement NLS. If you are

unfamiliar with translation tables, do not change the specifications without proper technical advice.

NLDECSEPARATOR System Option

Specifies whether SAS produces locale sensitive numeric output for the decimal separator or continues to format numbers with US English preferences.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

PROC OPTIONS GROUP= LANGUAGECONTROL

Alias: NLD

Default: NONLSDECSEPARATOR

Syntax

NLDECSEPARATOR | NONLDECSEPARATOR

Required Arguments

NLDECSEPARATOR
Enables locale-sensitive numeric output for the decimal separator.

NONLDECSEPARATOR
Disables locale-sensitive numeric output for the decimal separator.

Details

Use the BESTDOTX format with the NLDECSEPARATOR option to produce a numeric value with a dot as a decimal separator. If you do not use the BESTDOTX format, the result could contain a comma (1,2). SAS does not interpret this output as a numeric value. It would result in an error if the value is used in a later assignment statement.

Some SAS functions, such as CALL SYMPUT, convert numeric values to a character string using the BEST format by default. When NLDECSEPARATOR is enabled, the BEST format uses the separator character that matches the preferences for the current locale. If the locale uses a dot as a separator character, the resulting string can be used later in a SAS statement that evaluates the value as a number. However, if the separator character is a comma or other character, the result causes a syntax error if the SAS code attempts to use it as a number.

To avoid this type of error, use BESTDOTX to format the value before assigning it within the CALL SYMPUT statement. BESTDOTX always uses a dot as a decimal separator in the results that it produces. For example, in the SAS code here, BESTDOTX is used to format the value of the variable X before it is assigned to the macro variable &MACX. The value of &MACX can then be assigned to the SAS variable Y without causing a syntax error.

```
data a;
   x=1.2;
   CALL SYMPUT('macx', put(x, bestdotx.));
run;
```

```
data b;
   y=&macx
run;
```

See Also

"BESTDOTX Format" on page 87

ODSLANGCHG System Option

Determines whether the language of the text of the ODS output can be changed

Valid in: configuration file, SAS invocation

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: ODSLANGCHG is set to off in all servers except for the UNICODE server

Tip: The language used for the language switching feature is controlled by LSWLANG= option. By default, LSWLANG is set to LOCALE, which specifies that the language of LSW is controlled by the language of LOCALE.

Syntax

ODSLANGCHG | NOODSLANGCHG

Syntax Description

ODSLANGCHG
Specifies that the language of ODS output can change after start-up.

NOODSLANGCHG
Specifies that the language of ODS output cannot change after start-up.

Details

The Language Switching feature (LSW) enables you to change the language of SAS messages and ODS templates after start-up. In order to use the LSW for ODS output, you must enable ODSLANGCHG.

During start-up, the configuration file determines the language for SAS messages and ODS templates. If the ODSLANGCHG option is set, the language of ODS output can change to reflect the LSWLANG= setting when the localizations are available.

You can enable ODSLANGCHG but not translate into the language of the locale. For example, if you enable ODSLANGCHG, then start a SAS session in French and set the locale to Greek, NLDATE is displayed in Greek. The output is displayed in French. The output is displayed in French because SAS does not translate into Greek.

Comparisons

If ODSLANGCHG is enabled and LSWLANG=LOCALE is set, the ODS PATH is updated based on the LOCALE setting to include the localized templates and corresponding localized messages are used to generate ODS output.

If ODSLANGCHG is not enabled at start-up, ODS output appears in the language that was specified in the configuration file.

Example

Example 1 is a French server with ODSLANGCHG not enabled (NOODSLANGCHG).

If a French-client application connects to the server, the output appears in French and dates, formatted by using the NL format, appear in French. If an English-client application connects to the French server, and the locale is changed to English on the server, then output messages appears in French, and dates formatted with NL formats appear in English.

Example 2 is a French server with ODSLANGCHG enabled (ODSLANGCHG) and LSWLANG=LOCALE.

If a French-client application connects to the server, the output appears in French and dates formatted by using the NL format, appear in French. If an English-client application connects to the French server, and the locale is changed to English on the server, then output messages appears in English, and dates formatted with NL format appears in English.

Example 3 is a French server with ODSLANGCHG enabled (ODSLANGCHG) and LSWLANG=English.

If a French-client application connects to the server, the output appears in English and dates formatted by using the NL format, appear in French. If an English-client application connects to the French server, and the locale is changed to English on the server, then output messages appear in English, and dates formatted with NL format appears in English.

See Also

- "LOGLANGENG System Option" on page 581
- "LOGLANGCHG System Option" on page 580
- "LSWLANG System Option" on page 582

RSASIOTRANSERROR System Option

Displays a transcoding error when invalid data is read from a remote application.

Valid in:	configuration file, SAS invocation, OPTIONS statement, SAS System Options window
Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Default:	RSASIOTRANSERROR

Syntax

RSASIOTRANSERROR | NORSASIOTRANSERROR

Syntax Description

RSASIOTRANSERROR
specifies to display a transcoding error when invalid values are read from a remote application.

NORSASIOTRANSERROR
specifies not to display a transcoding error when invalid values are read from a remote application.

Details

The RSASIOTRANSERROR system option enables remote users of SASIO, for example, SAS Enterprise Guide and SAS Enterprise Miner, to ignore invalid data values. An invalid data value typically causes a transcoding error when the data is read by a remote application.

SORTSEQ= System Option

Specifies a language-specific collating sequence for the SORT and SQL procedures to use in the current SAS session.

Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category: Sort: Procedure Options

PROC OPTIONS GROUP= SORT

Syntax

SORTSEQ=| <LINGUISTIC>

Syntax Description

LINGUISTIC
Sorts characters according to the rules of a specified language. The rules and default collating sequence are based on the language specified in the current locale setting. The implementation is provided by the International Components for Unicode (ICU) library. Refer to " Linguistic Sorting of Data Sets and ICU" in *SAS Viya Visual Data Management and Utility Procedures Guide* for more information.

Example

This example demonstrates the functionality of SORTSEQ with PROC SORT and PROC SQL:

```
options sortseq=linguistic;
proc sort data=sashelp.class out=foo1;
by name;
run;
proc sql;
create table foo2 as select * from sashelp.class order by name;
quit;
run;
```

SAS provides ICU collation when the linguistic option (SORTSEQ=LINGUISTIC) is specified on the Base SAS procedure, PROC SORT. Starting in the third maintenance release of SAS 9.4, linguistic collation can also be specified using the SORTSEQ= option in the SQL Procedure and by specifying system option SORTSEQ=LINGUISTIC.

Note: Only PROC SORT and PROC SQL are affected when the SORTSEQ=LINGUISTIC system option is specified.

When the linguistic option is specified, SAS relies on the ICU libraries as the reference implementation of the Unicode Collation Algorithm (UCA) and as a de facto standard. For in-depth information about the UCA algorithm or the International Components for Unicode (ICU) library implementation, see Download the ICU 4.8 Release and CLDR 2.0 Release Note.

See Also

"Collating Sequence" on page 14

TIMEZONE= System Option

Specifies the user local time zone.

Valid in: Configuration file, SAS command, OPTIONS statement, SASV9_OPTIONS environment variable

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Alias: TZ=, except in the restricted options configuration file where TIMEZONE= must be used

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya System Options: Reference*.

Syntax

TIMEZONE='*time-zone-name*' | '*time-zone-ID*'

Syntax Description

time-zone-name

specifies a three- or four-character time zone name. For example, EST is a time zone name for Eastern Time.

Default BLANK, indicating that the SAS server time zone and the client time zone are the same

See For a list of time zone names, see Appendix 2, "Time Zone IDs and Time Zone Names," on page 663.

time-zone-ID

specifies a *region*/*area* value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by time zone name and daylight saving time rules.

Note Time zone IDs are compatible with Java time zone names.

See For a list of time zone IDs, see Appendix 2, "Time Zone IDs and Time Zone Names," on page 663.

Details

You set the TIMEZONE= option to a time zone ID or a time zone name in order for SAS to use a particular time zone. The time zone setting affects the following SAS components:

- times that are recorded by events and logs
- time of data set creation or modification
- DATE( ) function
- DATETIME( ) function
- TIME( ) function
- TODAY( ) function
- time zone functions TZONEOFF( ), TZONEID( ), TZONENAME( ), TZONES2U( ), and TZONEU2S( ),
- time zone formats B8601DX*w*., E8601DX*w*., B8601LX*w*., E8601LX*w*., B8601TX*w*., E8601TX*w*., NLDATMZ*w*., NLDATMTZ*w*., and NLDATMWZ*w*.

You set a time zone by specifying a time zone ID or a time zone name. A time zone ID is a region and an area separated by a forward slash (/). For example, America/New_York and Asia/Osaka are time zone IDs.

A time zone name is a three- or four-character name for a time zone. For example, EST is Eastern Time and JST is Japan Time. SAS determines the time by using time zone rules, including daylight saving time rules, before using a time value.

Some time zones names are valid for different locales. For example, CST is Central Daylight Time, Cuba Daylight Time, and China Daylight Time. SAS uses the value of the LOCALE= system option to determine the region and area to use. If TIMEZONE='CST' and LOCALE='zh_CN', SAS uses the Asia/Beijing time zone. If the time zone name does not exist for the locale, SAS searches all time zones and sets the time zone to the first match that it finds.

When this option is restricted and the value of TIMEZONE= is the default value of BLANK, time zone behavior does not use time zone information.

See Also

"Specifying Time Zones in SAS" on page 25

VALIDMEMNAME= System Option

Specifies the rules for naming SAS data sets, SAS data views, and item stores.

Valid in: Configuration file, SAS command, OPTIONS statement, SASV9_OPTIONS environment variable

Category: Files: SAS Files

PROC OPTIONS GROUP= SASFILES

Default: The shipped default is COMPATIBLE.

Applies to: BASE engine

Restriction: Regardless of the value of VALIDMEMNAME, a member name cannot end in the special character # followed by three digits. This is because it would conflict with the naming conventions for generation data sets. Using such a member name results in an error.

Interaction: SAS Studio sets VALIDMEMNAME=COMPAT before and after each code submission. For more information, see "System Options in SAS Studio" in *SAS Viya System Options: Reference*.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya System Options: Reference*.

Syntax

VALIDMEMNAME=COMPATIBLE | EXTEND

Syntax Description

COMPATIBLE
specifies that a SAS data set name, a SAS data view name, or an item store name must follow these rules:

- The length of the names can be up to 32 characters.
- Names must begin with a letter of the Latin alphabet (A–Z, a–z) or an underscore. Subsequent characters can be letters of the Latin alphabet, numerals, or underscores.
- Names cannot contain blanks or special characters except for the underscore.
- Names can contain mixed-case letters. SAS internally converts the member name to uppercase. Therefore, you cannot use the same member name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`customer`**, **`Customer`**, and **`CUSTOMER`** all represent the same member name. How the name is saved on disk is determined by the operating environment.

Alias COMPAT

EXTEND
specifies that a SAS data set name, a SAS data view name, or an item store name must follow these rules:

- Names can include national characters.
- The name can include special characters, except for the / \ * ? " < > |: -. characters.
- The name must contain at least one character (letters, numbers, valid special characters, and national characters).
- The length of the name can be up to 32 bytes.
- Null bytes are not allowed.
- Names cannot begin with a blank or a '.' (the period).
- Leading and trailing blanks are deleted when the member is created.

- Names can contain mixed-case letters. SAS internally converts the member name to uppercase. Therefore, you cannot use the same member name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`customer`**, **`Customer`**, and **`CUSTOMER`** all represent the same member name. How the name appears is determined by the operating environment.

Requirement	When VALIDMEMNAME=EXTEND, SAS data set names, SAS data view names, and item store names must be written as a SAS name literal if the name includes blank spaces, special characters, or national characters. If you use either the percent sign (%) or the ampersand (&), then you must use single quotation marks in the name literal in order to avoid interaction with the SAS Macro Facility.
Operating environment	When you reference a SAS file directly by its physical name, the final embedded period is an extension delimiter. If a physical file reference includes a SAS member name that contains a period, you must add the file extension. For example, if you reference the data set name my.member as a physical file, you would add the file extension sas7bdat to the reference, as shown in this SET statement: set './saslib/my.member.sas7bdat'.
Tip	The name is displayed in uppercase letters.
Examples	data “August Purchases”n;
	data ‘Años de empleo’n.;
CAUTION	**Throughout SAS, using the name literal syntax with SAS member names that exceed the 32-byte limit or that have excessive embedded quotation marks might cause unexpected results.** The intent of the VALIDMEMNAME=EXTEND system option is to enable compatibility with other DBMS member naming conventions, such as allowing embedded blanks and national characters.

Details

When VALIDMEMNAME= EXTEND, valid characters that are allowed in a SAS data set name, SAS data view name, and an item store name are extended to these characters:

- international characters
- characters supported by third-party databases
- characters that are commonly used in a filename

Only the DATA, VIEW, and ITEMSTOR SAS member types support the extension of characters. The other member types, such as CATALOG and PROGRAM, do not support the extended characters. INDEX and AUDIT types that exist only with the associated DATA member support extended characters.

See Also

System Options:

- “VALIDVARNAME= System Option” on page 592

VALIDVARNAME= System Option

Specifies the rules for valid SAS variable names that can be created and processed during a SAS session.

Valid in: Configuration file, SAS command, OPTIONS statement, SASV9_OPTIONS environment variable

Category: Files: SAS Files

PROC OPTIONS GROUP= SASFILES

Default: The shipped default is V7.

Interaction: SAS Studio sets VALIDVARNAME= to the Preferences setting before each code submission. For more information, see "System Options in SAS Studio" in *SAS Viya System Options: Reference*.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya System Options: Reference*.

Syntax

VALIDVARNAME=V7 | UPCASE | ANY

Syntax Description

V7

specifies that variable names must follow these rules:

- The length of a SAS variable names can be up to 32 characters.
- The first character must begin with a letter of the Latin alphabet (A–Z, a–z) or the underscore. Subsequent characters can be letters of the Latin alphabet, numerals, or underscores.
- Trailing blanks are ignored. The variable name alignment is left-justified.
- A variable name cannot contain blanks or special characters except for the underscore.
- A variable name can contain mixed-case letters. SAS stores and writes the variable name in the same case that is used in the first reference to the variable. However, when SAS processes a variable name, SAS internally converts it to uppercase. Therefore, you cannot use the same variable name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`cat`**, **`Cat`**, and **`CAT`** all represent the same variable.
- Do not assign variables the names of special SAS automatic variables (such as _N_ and _ERROR_) or variable list names (such as _NUMERIC_, _CHARACTER_, and _ALL_) to variables.

Examples

```
season='summer';
```

```
percent_of_profit=percent;
```

UPCASE

specifies that the variable name follows the same rules as V7, except that the variable name is uppercase, as in earlier versions of SAS.

ANY

specifies that SAS variable names must follow these rules:

- The name can begin with or contain any characters, including blanks, national characters, special characters, and multi-byte characters.
- The name can be up to 32 bytes in length
- The name cannot contain any null bytes
- Leading blanks are preserved, but trailing blanks are ignored
- The name must contain at least one character. A name with all blanks is not permitted.
- The name contains mixed-case letters. SAS stores and writes the variable name in the same case that is used in the first reference to the variable. However, when SAS processes a variable name, SAS internally converts it to uppercase. Therefore, you cannot use the same variable name with a different combination of uppercase and lowercase letters to represent different variables. For example, **`cat`**, **`Cat`**, and **`CAT`** all represent the same variable.

See

Examples ‘% of profit’n=percent;

‘items@warehouse’n=itemnum;

CAUTION **Throughout SAS, using the name literal syntax with SAS member names that exceed the 32-byte limit or have excessive embedded quotation marks might cause unexpected results.** The intent of the VALIDVARNAME=ANY system option is to enable compatibility with other DBMS variable (column) naming conventions, such as allowing embedded blanks and national characters.

See Also

System Options:

- “VALIDMEMNAME= System Option” on page 589

Part 9

Options for Commands, Statements, and Procedures for NLS

Chapter 16

Dictionary of Command, Statement, and Procedure Option for NLS

Commands, Statements, and Procedures for NLS by Category

The data set control and data access categories of options for selected SAS statements are affected by NLS. The following table provides brief descriptions of the statement options. For more detailed descriptions, see the dictionary entry for each statement option.

Category	Language Elements	Description
Data Access	CVPBYTES=, CVPENGINE=, and CVPMULTIPLIER= Options (p. 605)	Specifies attributes for character variables that are needed to transcode a SAS file.
	ENCODING= Option (p. 612)	Overrides and transcodes the encoding for input or output processing of external files.
	INENCODING= and OUTENCODING= Options (p. 615)	Overrides and changes the encoding when reading or writing SAS data sets in the SAS library.

Category	Language Elements	Description
	ODSCHARSET= Option (p. 617)	Specifies the character set to be generated in the META declaration for the output.
	ODSTRANTAB= Option (p. 618)	Specifies the translation table to use when transcoding an XML document for an output file.
	XMLENCODING= Option (p. 622)	Overrides the encoding of an XML document to import or export an external document.
Information	TRANSCODE= Option (p. 619)	Specifies an attribute in the ATTRIB statement (which associates a format, informat, label, and length with one or more variables) that indicates whether character variables are to be transcoded.
ODS: Third-Party Formatted	CHARSET= Option (p. 598)	Specifies the character set to be generated in the META declaration for the output.
	TRANTAB= Option (p. 621)	Specifies the translation table to use when you are transcoding character data.

Dictionary

CHARSET= Option

Specifies the character set to be generated in the META declaration for the output.

Valid in: LIBNAME statement for the ODS MARKUP and ODS HTML statements

Category: ODS: Third-Party Formatted

Syntax

CHARSET=*character-set* ;

Required Argument

character-set

Specifies the character set to use in the META tag for HTML output.

An example of an encoding is UTF-8. Official character sets for use on the Internet are registered by IANA (Internet Assigned Numbers Authority). IANA is the central registry for various Internet protocol parameters, such as port, protocol and enterprise numbers, and options, codes and types. For a complete list of character-set values, see **`www.unicode.org/reports/tr22/index.html`** and **`www.iana.org/assignments/character-sets`**.

A *character set* is like an *encoding-value* in this context. However, *character set* is the term that is used to identify an encoding that is suitable for use on the Internet.

Example: Generated Output in a META Declaration for an ODS MARKUP Statement

```
<META http-equiv="Content-Type" content="text/html; charset=utf-8">
```

See Also

Conceptual Information:

- Chapter 3, “Encoding for NLS,” on page 9

Collating Sequence Option

Specifies the collating sequence for PROC SORT.

Valid in: PROC SORT statement

Note: The PROC SORT statement sorts observations in a SAS data set by one or more characters or numeric variables.

Syntax

PROC SORT *collating-sequence-option <other option(s)>* ;

Options

Options can include one *collating-sequence-option* and multiple *other options*. The order of the two types of options does not matter and both types are not necessary in the same PROC SORT step. Only the explanations for the PROC SORT collating-sequence-options follow.

DANISH NORWEGIAN
sorts characters according to the Danish and Norwegian

The Danish and Norwegian collating sequence is shown in Figure 16.1 on page 600.

POLISH
sorts characters according to the Polish convention.

FINNISH SWEDISH
sorts characters according to the Finnish and Swedish convention. The Finnish and Swedish collating sequence is shown in Figure 16.1 on page 600.

NORWEGIAN
See DANISH

SWEDISH
See FINNISH

SORTSEQ=*collating-sequence*
specifies the collating sequence. The *collating-sequence* can be a collating-sequence-option, a translation table, an encoding, or the keyword LINGUISTIC. Only one collating sequence can be specified. For more information, see“Collating Sequence” on page 14.

Here are descriptions of the collating sequences:

collating—sequence—option | translation_table

specifies either a translation table, which can be one that SAS provides or any user-defined translation table, or one of the PROC SORT statement Collating-Sequence-Options..

The available translation tables are

- DANISH
- EBCDIC
- FINNISH
- ITALIAN
- NORWEGIAN
- POLISH
- REVERSE
- SPANISH
- SWEDISH

The following figure shows how the alphanumeric characters in each language sorts:

Figure 16.1 *Alphanumeric Characters Sorted for Each Language*

```
Danish:     0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Finnish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
Italian:    0123456789AÀBCÇDEÉÈFGHIÌJKLMNOÒPQRSTUÙVWXYZaàbcçdeéèfghiìjklmnoòpqrstuùvwxyz
Norwegian:  0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Spanish:    0123456789AÁaáBbCcDdEÉeéFfGgHhIÍiíJjKkLlMmNnÑñOÓoóPpQqRrSsTtUÚuúÜüVvWwXxYyZz
Swedish:    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖabcdefghijklmnopqrstuvwxyzåäö
```

Restriction You can specify only one collating-sequence-option in a PROC SORT step.

Tip The SORTSEQ= collating sequence options are specified without parenthesis and have no arguments that are associated with them. An example of how to specify a collating sequence follows: **`proc sort data=mydata SORTSEQ=ASCII;`**

encoding-value

specifies an encoding value. The result is the same as a binary collation of the character data represented in the specified encoding. See the supported encoding values in "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

Restriction PROC SORT is the only procedure or part of the SAS system that recognizes an encoding specified for the SORTSEQ= option.

Tip When the encoding value contains a character other than an alphanumeric character or underscore, the value needs to be enclosed in quotation marks.

See The list of the encodings that can be specified in "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

LINGUISTIC<*(collating—rules)*>

specifies linguistic collation, which sorts characters according to rules of the specified language. The rules and default collating sequence options are based on the language specified in the current locale setting. The implementation is provided by the International Components for Unicode (ICU) library and produces results that are largely compatible with the Unicode Collation Algorithms (UCA).

Alias UCA

Restriction The SORTSEQ=LINGUISTIC option is available on the PROC SORT SORTSEQ= option and PROC SQL.

Notes Linguistic collation can also be specified using the SORTSEQ= option in the SQL Procedure and by specifying system option SORTSEQ=LINGUISTIC. For more information, see "SORTSEQ= System Option" on page 587 and "SORT Procedure" in *SAS Viya Visual Data Management and Utility Procedures Guide*.

Collating rules are supported only for PROC SORT, not for the system option or SORTSEQ= on PROC SQL.

Tips The collating-rules must be enclosed in parentheses. More than one collating rule can be specified.

When BY processing is performed on data sets that are sorted with linguistic collation, the NOBYSORTED system option might need to be specified in order for the data set to be treated properly. BY processing is performed differently than collating sequence processing.

See The "ICU License" in *SAS Viya Visual Data Management and Utility Procedures Guide*

The "Collating Sequence" on page 14 for detailed information about linguistic collation.

The website for the Unicode Collation Algorithm (UCA) specification, *http://www.unicode.org*.

Here are the collation-rules that can be specified for the LINGUISTIC option. These rules modify the linguistic collating sequence:

ALTERNATE_HANDLING=SHIFTED

controls the handling of variable characters like spaces, punctuation, and symbols. When this option is not specified (using the default value Non-Ignorable), differences among these variable characters are of the same importance as differences among letters. If the ALTERNATE_HANDLING option is specified, these variable characters are of minor importance.

Default NON_IGNORABLE

Tip The SHIFTED value is often used in combination with STRENGTH= set to Quaternary. In such a case, whitespace characters, punctuation, and symbols are considered when comparing strings, but only if all other aspects of the strings (base letters, accents, and case) are identical.

CASE_FIRST=

specify order of uppercase and lowercase letters. This argument is valid for only TERTIARY, QUATERNARY, or IDENTICAL levels. The following table provides the values and information for the CASE_FIRST argument:

Value	Description
UPPER	Sorts uppercase letters first, then the lowercase letters.
LOWER	Sorts lowercase letters first, then the uppercase letters.

COLLATION=

The following table lists the available COLLATION= values: If you do not select a collation value, then the user's locale-default collation is selected.

Value	Description
BIG5HAN	specifies pinyin ordering for Latin and specifies big5 charset ordering for Chinese, Japanese, and Korean characters.
DIRECT	specifies a Hindi variant.
GB2312HAN	specifies pinyin ordering for Latin and specifies gb2312han charset ordering for Chinese, Japanese, and Korean characters.
PHONEBOOK	specifies a telephone-book style for ordering of characters. Select **PHONEBOOK** only with the German language.
PINYIN	specifies an ordering for Chinese, Japanese, and Korean characters based on character-by-character transliteration into pinyin. This ordering is typically used with simplified Chinese.
POSIX	is the Portable Operating System Interface. This option specifies a "C" locale ordering of characters.
STROKE	specifies a nonalphabetic writing style ordering of characters. Select **STROKE** with Chinese, Japanese, Korean, or Vietnamese languages. This ordering is typically used with Traditional Chinese.
TRADITIONAL	specifies a traditional style for ordering of characters. For example, select **TRADITIONAL** with the Spanish language.

LOCALE=locale_name

specifies the locale name in the form of a POSIX name, for example, ja_JP. See the "LOCALE= Values for PAPERSIZE and DFLANG Options" on page 629 for a list of locale and POSIX values supported by PROC SORT.

Restriction The following locales are not supported by PROC SORT:

- Afrikaans_SouthAfrica, af_ZA
- Cornish_UnitedKingdom, kw_GB

NUMERIC_COLLATION=

orders integer values within the text by the numeric value instead of characters used to represent the numbers.

Value	Description
ON	Order numbers by the numeric value. For example, "8 Main St." would sort before "45 Main St.".
OFF	Order numbers by the character value. For example, "45 Main St." would sort before "8 Main St.".

Default OFF

STRENGTH=

The value of strength is related to the collation level. There are five collation-level values. The following table provides information about the five levels. The default value for strength is related to the locale.

Value	Type of Collation	Description
PRIMARY or 1	PRIMARY specifies differences between base characters (for example, "a" < "b").	It is the strongest difference. For example, dictionaries are divided into different sections by base character.
SECONDARY or 2	Accents in the characters are considered secondary differences (for example, "as" < "às" < "at").	A secondary difference is ignored when there is a primary difference anywhere in the strings. Other differences between letters can also be considered secondary differences, depending on the language.
TERTIARY or 3	Upper and lowercase differences in characters are distinguished at the tertiary level (for example, "ao" < "Ao" < "aò").	A tertiary difference is ignored when there is a primary or secondary difference anywhere in the strings. Another example is the difference between large and small Kana.

Value	Type of Collation	Description
QUATERNARY or 4	When punctuation is ignored at level 1-3, an additional level can be used to distinguish words with and without punctuation (for example, "ab" < "a-b" < "aB").	The quaternary level should be used if ignoring punctuation is required or when processing Japanese text. This difference is ignored when there is a primary, secondary, or tertiary difference.
IDENTICAL or 5	When all other levels are equal, the identical level is used as a tiebreaker. The Unicode code point values of the Normalization Form D (NFD) form of each string are compared at this level, just in case there is no difference at levels 1-4.	This level should be used sparingly, as only code point values differences between two strings is an extremely rare occurrence. For example, only Hebrew cantillation marks are distinguished at this level.

Alias Level=

CAUTION:
If you use a host sort utility to sort your data, then specifying a translation table based collating sequence with the SORTSEQ= option might corrupt the character BY variables. For more information, see the PROC SORT documentation for your operating environment.

Details

The collating sequence option in the PROC SORT statement sorts observations in a SAS data set by one or more characters or numeric variables.

Table 16.1 *Options*

Task	Option
Specify the collating sequence	
Specify Danish	DANISH on page 599
Specify Finnish	FINNISH on page 599
Specify Norwegian	NORWEGIAN on page 599
Specify Polish	POLISH on page 599
Specify Swedish	SWEDISH on page 599
Specify a customized sequence	NATIONAL on page 599

Task	Option
Specify any of the collating sequences listed above (ASCII, EBCDIC, DANISH, FINNISH, ITALIAN, NORWEGIAN, POLISH, SPANISH, SWEDISH, or NATIONAL), the name of any other system provided translation table (POLISH, SPANISH), and the name of a user-created translation table. You can specify an encoding. You can also specify either the keyword LINGUISTIC or UCA to achieve a locale-appropriate collating sequence.	SORTSEQ= on page 599

See Also

- "Collating Sequence" on page 14
- "SORT Procedure" in *SAS Viya Visual Data Management and Utility Procedures Guide*

System Options:

- "SORTSEQ= System Option" on page 587

CVPBYTES=, CVPENGINE=, and CVPMULTIPLIER= Options

Specifies attributes for character variables that are needed to transcode a SAS file.

Valid in: LIBNAME statement

Category: Data Access

See: LIBNAME, SAS/ACCESS

Syntax

LIBNAME *libref* '*SAS data-library*'; <CVPBYTES=*bytes*> <CVPENGINE=*engine*> <CVPMULTIPLIER=*multiplier*> <CVPVARCHAR=YES | NO>

Optional Arguments

CVPBYTES=*bytes*

specifies the number of bytes by which to expand character variable lengths when processing a SAS data file that requires transcoding. The CVP engine expands the lengths so that character data truncation does not occur. The lengths for character variables are increased by adding the specified value to the current length. You can specify a value from 0 to 32766.

For example, the following LIBNAME statement implicitly assigns the CVP engine by specifying the CVPBYTES= option.

```
libname expand 'SAS data-library' cvpbytes=5;
```

Character variable lengths are increased by adding 5 bytes. A character variable with a length of 10 is increased to 15, and a character variable with a length of 100 is increased to 105.

Default If you specify CVPBYTES=, SAS automatically uses the CVP engine in order to expand the character variable lengths according to your specification. If you explicitly assign the CVP engine but do not specify either CVPBYTES= or CVPMULTIPLIER=, then SAS uses CVPMULTIPLIER=1.5 to increase the lengths of the character variables.

Restrictions The CVP engine supports SAS data files, no SAS views, catalogs, item stores, and so on.

The CVP engine is available for input (read) processing only.

For library concatenation with mixed engines that include the CVP engine, only SAS data files are processed. For example, if you execute the COPY procedure, only SAS data files are copied.

Requirement The number of bytes that you specify must be large enough to accommodate any expansion. Otherwise, truncation occurs, which results in an error message in the SAS log.

Interaction You cannot specify both CVPBYTES= and CVPMULTIPLIER=. Specify one of these options.

See "Avoiding Character Data Truncation By Using the CVP Engine" on page 23

CVPENGINE=*engine*

specifies the engine to use in order to process a SAS data file that requires transcoding. The CVP engine expands the character variable lengths to transcoding so that character data truncation does not occur. Then the specified engine does the actual file processing.

Alias CVPENG

Default SAS uses the default SAS engine.

See "Avoiding Character Data Truncation By Using the CVP Engine" on page 23

CVPMULTIPLIER=*multiplier*

specifies a multiplier value in order to expand character variable lengths when you are processing a SAS data file that requires transcoding. The CVP engine expands the lengths so that character data truncation does not occur. The lengths for character variables are increased by multiplying the current length by the specified value. You can specify a multiplier value from 1 to 5.

For example, the following LIBNAME statement implicitly assigns the CVP engine by specifying the CVPMULTIPLIER= option.

```
libname expand 'SAS data-library' cvpmultiplier=2.5;
```

Character variable lengths are increased by multiplying the lengths by 2.5. A character variable with a length of 10 is increased to 25, and a character variable with a length of 100 is increased to 250.

Alias CVPMULT

Default If you specify CVPMULTIPLIER=, SAS automatically uses the CVP engine in order to expand the character variable lengths according to

your specification. If you explicitly specify the CVP engine but do not specify either CVPMULTIPLIER= or CVPBYTES=, then SAS uses CVPMULTIPLIER=1.5 to increase the lengths.

Restrictions The CVP engine supports SAS data files, no SAS views, catalogs, item stores, and so on.

The CVP engine is available for input (read) processing only.

For library concatenation with mixed engines that include the CVP engine, only SAS data files are processed. For example, if you execute the COPY procedure, only SAS data files are copied.

Requirement The number of bytes that you specify must be large enough to accommodate any expansion. Otherwise, truncation occurs, which results in an error in the SAS log.

Interaction You cannot specify both CVPMULTIPLIER= and CVPBYTES=. Specify one of these options.

See "Avoiding Character Data Truncation By Using the CVP Engine" on page 23

CVPVARCHAR=YES | NO

specifies whether to convert fixed-width character variables to variable-width characters during input file processing. The byte length of the new-width character variable is the maximum number of bytes per character from the SAS session encoding multiplied by the specified fixed-width character length.

Default No

Interaction If you specify CVPVARCHAR=YES, the CVPMULTIPLIER= and CVPBYTES= options are ignored.

Notes Trailing blanks are removed from string data that is under CHAR columns.

Fixed-width character variables with a format of TRANSCODE=NO are excluded during conversion.

Details

The character variable padding (CVP) engine converts variables defined with the CHAR to a VARCHAR data type when **`CVPVARCHAR=YES`**. Because CVP is available only for input processing, the VARCHAR data type is not automatically saved.

Without saving the data, the conversion to VARCHAR is lost when a SAS session ends. To save the changes, use the SET statement or PROC COPY and an engine that supports VARCHAR. If the data is saved using an engine that does not support VARCHAR, such as the BASE engine, the character columns in the new data set revert to the CHAR data type. The length of the CHAR variables in the new data set is the number of bytes that is needed to store the VARCHAR.

If the data is read by a procedure that does not support VARCHAR, **`CVPVARCHAR=YES`** is ignored. The CVP engine uses CVPBYTES, CVPMULTIPLIER, or the default multiplier to expand the length of the character columns in the data.

Example: Using the CVP Engine

The following example illustrates how to avoid character data truncation by using the CVP engine. The example uses a SAS data set named MYFILES.WLATIN2, which contains some national characters in the Wlatin2 encoding. The data set is created in SAS 9 in an SBCS session. This code sample is run in SAS9.

```
libname myfiles 'SAS-Data Library';
data myfiles.wlatin2 (encoding=wlatin2);
   var1='A';
   var2='Š';
   var3='ś';
   var4='ł';
;
proc print data=myfiles.wlatin2;
run;
```

Obs	var1	var2	var3	var4
1	A	Š	ś	ł

Here is the PROC CONTENTS output for MYFILES.WLATIN2 that was run in SAS 9, which shows that the encoding is Wlatin2 and the length for each character variable is 1 byte:

Output 16.1 *PROC CONTENTS Output for MYFILES.WLATIN2*

The SAS System

The CONTENTS Procedure

Data Set Name	MYFILES.WLATIN2	**Observations**	1
Member Type	DATA	**Variables**	4
Engine	V9	**Indexes**	0
Created	08/17/2016 09:00:44	**Observation Length**	4
Last Modified	08/17/2016 09:00:44	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin2 Central Europe (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	15881
Obs in First Data Page	1
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	U:\wlatin2.sas7bdat
Release Created	9.0401M3
Host Created	W32_7PRO

Alphabetic List of Variables and Attributes			
#	**Variable**	**Type**	**Len**
1	var1	Char	1
2	var2	Char	1
3	var3	Char	1
4	var4	Char	1

The following code is executed in SAS Viya. The default session encoding in UTF-8.

```
options msglevel=i;
```

```
libname myfiles 'SAS data-library';
data myfiles.utf8;
   set myfiles.wlatin2;
run;
```

The DATA step requests a new data set named MYFILES.UTF8. It also requests that the data be read into the new data set in the UTF-8 encoding, which means that the data must be transcoded from Wlatin2 to UTF-8. The request results in errors when the transcoding truncates the character data. The new data set MYFILES.UTF8 is created but does not contain any data.

Log 16.1 *SAS Log with Transcoding Error*

```
INFO: Data file MYFILES.UTF8.DATA is in a format native to another
host or the file encoding does not match the session encoding.
Cross Environment Data Access will be used, which may require additional
CPU resources and reduce performance.
   ERROR: Some character data was lost during transcoding in the data set
MYFILES.UTF8.
   NOTE: The data step has been abnormally terminated.
   NOTE: The SAS System stopped processing this step because of errors.
   NOTE: There were 1 observations read from the data set MYFILES.WLATIN2.
   WARNING: The data set MYFILES.UTF8 may be incomplete.  When this step was
stopped there were 0
            observations and 4 variables.
```

The following code is executed in SAS Viya and expands the variable lengths:

```
options msglevel=i;
libname myfiles 'SAS data-library';
libname expand cvp 'SAS data-library' cvpbytes=2;
data myfiles.utf8;
   set expand.wlatin2;
run;
```

In this example, the CVP engine expands character variable lengths by adding 2 bytes to each length. The data is read into the new file in UTF-8 encoding by transcoding from Wlatin2 to UTF-8. There is no data truncation due to the expanded character variable lengths, and the new data set is successfully created:

Log 16.2 *SAS Log Output for MYFILES.UTF8*

```
57     options msglevel=i;
58          libname myfiles 'directory path';
NOTE: Libref MYFILES was successfully assigned as follows:
       Engine:        V9
       Physical Name: /directory path
59          libname expand cvp '/directory path' cvpbytes=2;
 WARNING: Libref EXPAND refers to the same physical library as MYFILES.
 NOTE: Libref EXPAND was successfully assigned as follows:
       Engine:        CVP
       Physical Name: /directory path
60         data myfiles.utf8;
61         set expand.wlatin2;
 INFO: Data file EXPAND.WLATIN2.DATA is in a format that is native to another
host, or the file encoding does not match the session
 encoding. Cross Environment Data Access will be used, which might require
additional CPU resources and might reduce performance.
62         run;

NOTE: There were 1 observations read from the data set EXPAND.WLATIN2.
NOTE: The data set MYFILES.UTF8 has 1 observations and 4 variables.
NOTE: DATA statement used (Total process time):
       real time           0.02 seconds
       cpu time            0.01 seconds
```

Finally, here is PROC CONTENTS output for MYFILES.UTF8 showing that it is in the UTF-8 encoding and that the length of each character variable is 3:

Output 16.2 *PROC CONTENTS Output for MYFILES.UTF8*

The CONTENTS Procedure

Data Set Name	MYFILES.UTF8	Observations	1
Member Type	DATA	Variables	4
Engine	V9	Indexes	0
Created	08/17/2016 09:32:34	Observation Length	12
Last Modified	08/17/2016 09:32:34	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	5401
Obs in First Data Page	1
Number of Data Set Repairs	0
Filename	/u/daharr/utf8.sas7bdat
Release Created	V.0301M0
Host Created	Linux
Inode Number	83031400
Access Permission	rw-r--r--
Owner Name	daharr
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	var1	Char	3
2	var2	Char	3
3	var3	Char	3
4	var4	Char	3

ENCODING= Option

Overrides and transcodes the encoding for input or output processing of external files.

Valid in: %INCLUDE statement; FILE statement; FILENAME statement; FILENAME statement, EMAIL (SMTP) Access Method; INFILE statement; ODS statements; FILE command; INCLUDE command

Category: Data Access

Syntax

ENCODING= '*encoding-value*'

Optional Argument

ENCODING= '*encoding-value*'

specifies the encoding to use for reading, writing, copying, or saving an external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

When you read, write, copy, or save data using an external file, SAS transcodes the data from the session encoding to the specified encoding.

For details, see "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

Default SAS uses the current session encoding.

Details

The following table provides information about how the ENCODING option is used with the corresponding statement:

%INCLUDE statement:	reads SAS statements and data lines from the specified source file.
FILE statement:	writes to an external file.
FILENAME statement:	reads from or writes to an external file.
FILENAME statement, EMAIL (SMTP) Access Method:	sends electronic mail programmatically.
INFILE statement:	reads from an external file.
ODS statements:	controls features of the Output Delivery System that are used to generate, store, or reproduce SAS procedure and DATA step output.
FILE command:	saves the contents of a window to an external file.
INCLUDE command:	Copies an external file into the current window.

Examples

Example 1: Using the FILE Statement to Specify an Encoding for Writing to an External File

This example creates an external file from a SAS data set. The current session encoding is UTF-8, but the external file's encoding needs to be LATIN1. By default, SAS writes the external file using the current session encoding.

To specify what encoding to use for writing data to the external file, specify the ENCODING= option:

```
libname myfiles 'SAS data-library';
```

```
filename outfile 'external-file';
data _null_;
   set myfiles.cars;
   file outfile encoding="latin1";
   put Make Model Year;
run;
```

Example 2: Using the FILENAME Statement to Specify an Encoding for Reading an External File

This example creates a SAS data set from an external file. The external file is in WLATIN1 character-set encoding, and the current SAS session is in the UTF-8 encoding. By default, SAS assumes that an external file is in the same encoding as the session encoding, which causes the character data to be written to the new SAS data set incorrectly.

To specify which encoding to use when reading the external file, specify the ENCODING= option:

```
libname myfiles  'SAS data-library';

filename extfile 'external-file' encoding="utf-8";
data myfiles.unicode;
   infile extfile;
   input Make $ Model $ Year;
run;
```

When you specify that the external file is in WLATIN1, SAS then transcodes the external file from WLATIN1 to the current session encoding when writing to the new SAS data set. Therefore, the data is written to the new data set correctly in UTF-8.

Example 3: Using the FILENAME Statement to Specify an Encoding for Writing to an External File

This example creates an external file from a SAS data set. By default, SAS writes the external file using the current session encoding. The current session encoding is UTF-8, but the external file's encoding needs to be WLATIN1.

To specify which encoding to use when writing data to the external file, specify the ENCODING= option:

```
libname myfiles  'SAS data-library';
filename outfile 'external-file' encoding="WLATIN1";
data _null_;
   set myfiles.cars;
   file outfile;
   put Make Model Year;
run;
```

When you specify that the external file is to be in WLATIN1 encoding, SAS then transcodes the data from UTF-8 to the specified encoding, WLATIN1,when writing to the external file.

Example 4: Changing Encoding for Message Body and Attachment

This example illustrates how to change text encoding for the message body as well as for the attachment.

```
filename mymail email 'Joe.Developer@sas.com';
data _null_;
   file mymail
```

```
      subject='Text Encoding'
      encoding=greek
      attach=('C:\My Files\Test.out'
         content_type='text/plain'
         encoding='ebcdic1047'
         outencoding='latin1');
run;
```

In the program, the following occurs:

- The ENCODING= email option specifies that the message body is encoded to Greek (ISO) before being sent.
- For the ATTACH= email option, the attachment option ENCODING= specifies the encoding of the attachment that is read into SAS, which is Western (EBCDIC).
- Because SMTP and other email interfaces do not support EBCDIC, the attachment option OUTENCODING= converts the attachment to Western (ISO) before sending it.

Example 5: Using the INFILE= Statement to Specify an Encoding for Reading from an External File

This example creates a SAS data set from an external file. The external file's encoding is in WLATIN2, and the current SAS session encoding is UTF-8. By default, SAS assumes that the external file is in the same encoding as the session encoding, which causes the character data to be written to the new SAS data set incorrectly.

To specify which encoding to use when reading the external file, specify the ENCODING= option:

```
libname myfiles 'SAS data-library';
filename extfile 'external-file';
data myfiles.unicode;
   infile extfile encoding="utf-8";
   input Make $ Model $ Year;
run;
```

When you specify that the external file is in WLATIN2, SAS then transcodes the external file from WLATIN2 to the current session encoding when writing to the new SAS data set. Therefore, the data is written to the new data set correctly in UTF-8.

See Also

Statements:

- "FILE Statement" in *SAS Viya Statements: Reference*
- "FILENAME Statement" in *SAS Viya Statements: Reference*
- "INFILE Statement" in *SAS Viya Statements: Reference*

INENCODING= and OUTENCODING= Options

Overrides and changes the encoding when reading or writing SAS data sets in the SAS library.

Valid in: LIBNAME statement

Category: Data Access

Syntax

INENCODING= ANY | ASCIIANY | EBCDICANY | *encoding-value*

OUTENCODING= ANY | ASCIIANY | EBCDICANY | *encoding-value*

Syntax Description

ANY
specifies no transcoding between ASCII and EBCDIC encodings.

NOTE: ANY is a synonym for binary. Because the data is binary, the actual encoding is irrelevant.

ASCIIANY
specifies that no transcoding occurs, assuming that the mixed encodings are ASCII encodings.

EBCDICANY
specifies that no transcoding occurs, assuming that the mixed encodings are EBCDIC encodings.

encoding-value
specifies an encoding value. For a list of encoding values, see Chapter 18, "Encoding Values in SAS Language Elements," on page 643.

Details

The INENCODING= option is used to read SAS data sets in the SAS library. The OUTENCODING= option is used to write SAS data sets in the SAS library.

The INENCODING= value or the OUTENCODING= value is written to the SAS log when you use the LIST argument.

INENCODING= and OUTENCODING= are most appropriate when using an existing library that contains mixed encodings. To read a library that contains mixed encodings, you can set INENCODING= to ASCIIANY or EBCDICANY. To write a separate data set, you can use OUTENCODING= to specify a specific encoding, which is applied to the data set when it is created.

Comparisons

- Session encoding is specified using the ENCODING= system option.
- You can specify the encoding for reading data sets in a SAS library by using the LIBNAME statement INENCODING= option for input files. If both the LIBNAME statement option and the ENCODING= data set option are specified, SAS uses the data set option.
- You can specify the encoding for writing data sets to a SAS library by using the LIBNAME statement OUTENCODING= option for output files. If both the LIBNAME statement option and the ENCODING= data set option are specified, SAS uses the data set option.
- For the COPY procedure, the default CLONE option uses the encoding attribute of the input data set instead of the encoding value specified on the OUTENCODING= option.

See Also

- "Overview: Encoding for NLS" on page 9

Statements:

- "LIBNAME Statement" in *SAS Viya Statements: Reference*

System Options:

- "ENCODING System Option" on page 577
- "LOCALE System Option" on page 578

Data Set Options:

- "ENCODING= Data Set Option" on page 41

ODSCHARSET= Option

Specifies the character set to be generated in the META declaration for the output.

Valid in: LIBNAME statement for the XMLV2 engine

Category: Data Access

Syntax

ODSCHARSET=*character-set*;

Required Argument

character-set

For the LIBNAME statement for the XML engine, specifies the character set to use in the ENCODING= attribute.

An example of an encoding is ISO-8859-1. Official character sets for use on the Internet are registered by IANA (Internet Assigned Numbers Authority). IANA is the central registry for various Internet protocol parameters, such as port, protocol and enterprise numbers, options, codes, and types. For a complete list of character-set values, see **`www.unicode.org/reports/tr22/index.html`** and **`www.iana.org/assignments/character-sets`**.

A *character set* is like an *encoding-value* in this context. However, *character set* is the term that is used to identify an encoding that is suitable for use on the Internet.

Details

An XML declaration is not required in all XML documents. Such a declaration is required only when the character encoding of the document is other than the default UTF-8 or UTF-16 and no encoding was determined by a higher-level protocol.

The ODSCHARSET option, in the LIBNAME statement for the XML engine, specifies the character set to use for generating an output XML document.

See Also

Conceptual Information:

- Chapter 3, "Encoding for NLS," on page 9

Statements:

- *SAS Viya XML LIBNAME Engine: User's Guide*

ODSTRANTAB= Option

Specifies the translation table to use when transcoding an XML document for an output file.

Valid in: the LIBNAME statement for the XMLV2 engine

Category: Data Access

Syntax

TRANTAB ='*translation-table*'

Optional Argument

translation-table
specifies the translation table to use for the output file. The translation table is an encoding method that maps characters (letters, logograms, digits, punctuation, symbols, control characters, and so on) in the character set to numeric values. An example of a translation table is one that converts characters from EBCDIC to ASCII-ISO. The *table-name* can be any translation table that SAS provides, or any user-defined translation table. The value must be the name of a SAS catalog entry in either the SASUSER.PROFILE catalog or the SASHELP.HOST catalog.

Details

The preferred method for specifying an encoding is to use the LOCALE= system option.

See Also

Conceptual Information:

- Conceptual discussion of Chapter 2, "Locale for NLS," on page 5

System Options:

- "LOCALE System Option" on page 578

Statements:

- *SAS Viya XML LIBNAME Engine: User's Guide*

TRANSCODE= Column Modifier on PROC SQL

Specifies whether values can be transcoded for character columns.

Valid in: Column modifier component in the SQL Procedure

Syntax

TRANSCODE=YES|NO

Required Argument

TRANSCODE=YES|NO
for character columns, specifies whether values can be transcoded. Use TRANSCODE=NO to suppress transcoding. Note that when you create a table using the CREATE TABLE AS statement, the transcoding attribute for a particular character column in the created table is the same as it is in the source table unless you change it with the TRANSCODE= column modifier.

Default YES

Restriction Suppression of transcoding is not supported for the V6TAPE engine.

See Also

- Chapter 4, "Transcoding for NLS," on page 19
- *SAS Viya Visual Data Management and Utility Procedures Guide*

TRANSCODE= Option

Specifies an attribute in the ATTRIB statement (which associates a format, informat, label, and length with one or more variables) that indicates whether character variables are to be transcoded.

Valid in: the ATTRIB statement in a DATA step

Category: Information

Type: Declarative

See: ATTRIB Statement "ATTRIB Statement" in *SAS Viya Statements: Reference*.

Syntax

ATTRIB *variable-list(s) attribute-list(s)* ;

Required Arguments

variable-list
names the variables that you want to associate with the attributes.

Tip List the variables in any form that SAS allows.

attribute-list
specifies one or more attributes to assign to *variable-list*. Multiple attributes can be specified in the ATTRIB statement. For a complete list of attributes, see the "ATTRIB Statement" in *SAS Viya Statements: Reference* .

TRANSCODE= YES | NO
Specifies whether to transcode character variables. Use TRANSCODE=NO to suppress transcoding. For more information, see "Overview to Transcoding" on page 19 .

Default YES

Restriction The TRANSCODE=NO attribute is not supported by some SAS Workspace Server clients. Variables with TRANSCODE=NO are not returned in SAS Viya. Prior to SAS 9.4, variables with TRANSCODE=NO are transcoded. Prior releases of SAS cannot access a SAS 9.4 data set that contains a variable with a TRANSCODE=NO attribute.

Interactions You can use the VTRANSCODE and VTRANSCODEX functions to return whether transcoding is on or off for a character variable.

If the TRANSCODE= attribute is set to NO for any character variable in a data set, PROC CONTENTS prints a transcode column that contains the TRANSCODE= value for each variable in the data set. If all variables in the data set are set to the default TRANSCODE= value (YES), no transcode column is printed.

Examples

Example 1: Using the TRANSCODE= Option with the SET Statement

When you use the SET statement to create a data set from several data sets, SAS makes the TRANSCODE= attribute of the variable in the output data set equal to the TRANSCODE= value of the variable in the first data set. In this example, the variable Z's TRANSCODE= attribute in data set A is NO because B is the first data set and Z's TRANSCODE= attribute in data set B is NO.

```
data b;
   length z $4;
   z = 'ice';
   attrib z transcode = NO;
data c;
   length z $4;
   z = 'snow';
   attrib z transcode = YES;
data a;
   set b;
   set c;
   /* Check transcode setting for variable Z */
   rc1 = vtranscode(z);
   put rc1=;
run;
```

Example 2: Using the TRANSCODE= Option with the MERGE Statement

When you use the MERGE statement to create a data set from several data sets, SAS makes the TRANSCODE= attribute of the variable in the output data set equal to the TRANSCODE= value of the variable in the first data set. In this example, the variable Z's TRANSCODE= attribute in data set A is YES because C is the first data set and Z's TRANSCODE= attribute in data set C is YES.

```
data b;
   length z $4;
   z = 'ice';
```

```
   attrib z transcode = NO;
data c;
   length z $4;
   z = 'snow';
   attrib z transcode = YES;
data a;
   merge c b;
   /* Check transcode setting for variable Z */
   rc1 = vtranscode(z);
   put rc1=;
run;
```

Note: The TRANSCODE= attribute is set when the variable is first seen on an input data set or in an ATTRIB TRANSCODE= statement. If a SET or MERGE statement comes before an ATTRIB TRANSCODE= statement and the TRANSCODE= attribute contradicts the SET statement, an error message occurs.

See Also

Functions:

- "VTRANSCODE Function" on page 427
- "VTRANSCODEX Function" on page 428

TRANTAB= Option

Specifies the translation table to use when you are transcoding character data.

Valid in: ODS MARKUP statement and ODS RTF statement

Category: ODS: Third-Party Formatted

Syntax

TRANTAB = (*translation-table*)

Optional Argument

translation-table
specifies the translation table to use for the output file. The translation table is an encoding method that maps characters (letters, logograms, digits, punctuation, symbols, control characters, and so on) in the character set to numeric values. An example of a translation table is one that converts characters from EBCDIC to ASCII-ISO. The *table-name* can be any translation table that SAS provides, or any user-defined translation table. The value must be the name of a SAS catalog entry in either the SASUSER.PROFILE catalog or the SASHELP.HOST catalog.

Details

The ENCODING= option is preferred when specifying the encoding.

XMLENCODING= Option

Overrides the encoding of an XML document to import or export an external document.

Valid in: LIBNAME statement for the XML engine

Category: Data Access

Syntax

XMLENCODING= '*encoding-value*'

Details

The LIBNAME statement for the XML engine, associates a SAS libref with an XML document to import or export an external document.

Comparisons

Options

encoding-value

specifies the encoding to use when you read, write, copy, or save an external file. The value for XMLENCODING= indicates that the external file has a different encoding from the current session encoding.

For details, see "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 643.

The default for *encoding-value* is the current session encoding.

See Also

Statements:

- *SAS Viya XML LIBNAME Engine: User's Guide*

TRANTAB Statement

Specifies the translation table to use when you transcode character data in order to export or transfer a SAS file.

Valid in: CPORT Procedure, UPLOAD procedure, DOWNLOAD procedure

Restriction: You can specify only one translation table per TRANTAB statement. To specify additional translation tables, use additional TRANTAB statements.

Interaction: The TRANTAB statement specifies a customized translation table (for example, to map an EBCDIC character to an ASCII character) to apply to the character set in the SAS file that is being exported or transferred. The TRANTAB= system option specifies a translation table to use for the SAS session, including file transfers.

Syntax

TRANTAB NAME=*translation-table-name* <TYPE=(*etype-list)* <OPT=DISP | SRC | (DISP SRC)> > ;

Required Argument

NAME=*translation-table-name*

specifies the name of the translation table to apply to the SAS catalog that you want to export (PROC CPORT) or transfer (PROC UPLOAD or PROC DOWNLOAD). The *translation-table-name* that you specify as the name of a catalog entry in either your SASUSER.PROFILE catalog or the SASHELP.HOST catalog. The SASUSER.PROFILE catalog is searched first, and then the SASHELP.HOST catalog is searched.

In most cases, the default translation table is the correct one to use, but you might need to apply additional translation tables if, for example, your application requires different national language characters.

You can specify a translation table other than the default in two ways:

- To specify a translation table for an invocation of the procedure, use the TRANTAB statement in the procedure, as appropriate.
- To specify a translation table for your entire SAS session or job (including all file exports or transfers), use the TRANTAB= system option.

Optional Arguments

TYPE=(*etype-list*)

applies the translation table only to the entries with the type or types that you specify. The *etype-list* can be one or more entry types. Examples of catalog entry types include DATA and FORMAT. If *etype-list* is a simple entry type, omit the parentheses.

By default, the UPLOAD, DOWNLOAD, and CPORT procedures apply the translation table to all specified catalog entries.

OPT=DISP | SRC | (DISP SRC)

OPT=DISP	applies the translation table only to the specified catalog entries, which produce window displays.
OPT=SRC	applies the translation table only to the specified catalog entries that are of the type SOURCE.
OPT=(DISP SRC)	applies the translation table only to the specified catalog entries that either produce window displays or are of type SOURCE.

If you do not specify the OPT= option, the UPLOAD or DOWNLOAD procedure applies the translation table to all of the entries in the catalog that you specify.

Default PROC CPORT, PROC UPLOAD, and PROC DOWNLOAD apply the translation table to all entries and data sets in the specified catalog.

Details

Translation tables were introduced in SAS 6 to support the requirements of national languages. SAS 8.2 introduced the LOCALE= system option as an improvement on

direct use of translation tables. SAS 9.3 supports the TRANTAB statement for backward compatibility. However, using the LOCALE= system option is preferred in later SAS releases. For more information, see TS-639, Data Conversion Issues in V6–V8. This technical support note provides information for customers using non-English languages **http://support.sas.com/techsup/technote/ts639.pdf**

PROC CPORT is used when you transfer a SAS file across a network. PROC UPLOAD and PROC DOWNLOAD are used when you transfer a SAS file across a network.

You must specify the INCAT= and OUTCAT= options in the PROC UPLOAD or PROC DOWNLOAD statement when using the TRANTAB statement.

Examples

Example 1

The information that follows applies to procedure features:

- PROC CPORT statement option: FILE=
- TRANTAB statement option: TYPE=

This example shows how to apply a customized translation table to the transport file before PROC CPORT exports it. For this example, assume that you have already created a customized translation table called TTABLE1.

Example 2: Program

Assign library references. </userSuppliedValue>The LIBNAME and FILENAME statements assign a libref for the source library and a fileref for the transport file, respectively.

```
libname source
'\\sashq\root\pub\pubdoc\doc\901\authoring\proc\miscsrc\sasfiles\cport';
filename tranfile 'trans3';
proc trantab table=ascii;
save table=ttable1;
```

```
libname source 'SAS data-library';
filename tranfile 'transport-file'
                  host-option(s)-for-file-characteristics;
```

Apply the translation specifics.</userSuppliedValue> The TRANTAB statement applies the translation that you specify with the customized translation table TTABLE1. TYPE= limits the translation to FORMAT entries.

```
proc cport catalog=source.formats file=tranfile;
   trantab name=ttable1 type=(format);
run;
```

Example 3: SAS Log

```
NOTE: Proc CPORT begins to transport catalog SOURCE.FORMATS
NOTE: The catalog has 2 entries and its maximum logical record length is 104.
NOTE: Entry REVENUE.FORMAT has been transported.
NOTE: Entry DEPT.FORMATC has been transported.
```

See Also

Conceptual Information:

- Chapter 4, “Transcoding for NLS,” on page 19

Part 10

Values for Locale and Transcoding

Chapter 17

Values for the LOCALE= System Option

LOCALE= Values for PAPERSIZE and DFLANG Options

The following table lists the valid LOCALE= values, specified by using the SAS name or the POSIX name. The alias name is also listed. Some locales do not have an alias.

Table 17.1 *Values for the LOCALE= System Option*

SAS Name	POSIX Locale	Aliases
Afrikaans_SouthAfrica	af_ZA	Afrikaans af
Albanian_Albania	sq_AL	Albanian sq
Arabic_Algeria	ar_DZ	
Arabic_Bahrain	ar_BH	
Arabic_Egypt	ar_EG	
Arabic_India	ar_IN	
Arabic_Iraq	ar_IQ	
Arabic_Jordan	ar_JO	
Arabic_Kuwait	ar_KW	
Arabic_Lebanon	ar_LB	
Arabic_Libya	ar_LY	

SAS Name	POSIX Locale	Aliases
Arabic_Morocco	ar_MA	
Arabic_Oman	ar_OM	
Arabic_Qatar	ar_QA	
Arabic_SaudiArabia	ar_SA	
Arabic_Sudan	ar_SD	
Arabic_Syria	ar_SY	
Arabic_Tunisia	ar_TN	
Arabic_UnitedArabEmirates	ar_AE	Arabic ar
Arabic_Yemen	ar_YE	
Basque_Spain	eu_ES	Basque Euskara eu
Bengali_India	bn_IN	Bengali bn
Bosnian_BosniaHerzegovina	bs_BA	Bosnian bs
Bulgarian_Bulgaria	bg_BG	Bulgarian bg
Byelorussian_Belarus	be_BY	Byelorussian Belarusian Byelorussian_Belarus be
Catalan_Spain	ca_ES	Catalan ca
Chinese_China	zh_CN	Chinese zh
Chinese_HongKong	zh_HK	
Chinese_Macau	zh_MO	

SAS Name	POSIX Locale	Aliases
Chinese_Singapore	zh_SG	
Chinese_Taiwan	zh_TW	
Cornish_UnitedKingdom	kw_GB	Cornish kw
Croatian_BosniaHerzegovina	hr_BA	
Croatian_Croatia	hr_HR	Croatian hr
Czech_CzechRepublic	cs_CZ	Czech cs
Danish_Denmark	da_DK	Danish da
Dutch_Belgium	nl_BE	
Dutch_Netherlands	nl_NL	Dutch nl
English_Australia	en_AU	
English_Belgium	en_BE	
English_Botswana	en_BW	
English_Canada	en_CA	
English_Caribbean	en_CB	
English_HongKong	en_HK	
English_India	en_IN	
English_Ireland	en_IE	
English_Jamaica	en_JM	
English_Malta	en_MT	
English_NewZealand	en_NZ	
English_Philippines	en_PH	

SAS Name	POSIX Locale	Aliases
English_Singapore	en_SG	English en
English_SouthAfrica	en_ZA	
English_UnitedKingdom	en_GB	
English_UnitedStates	en_US	English en
English_Zimbabwe	en_ZW	
Estonian_Estonia	et_EE	Estonian et
Faroese_FaroeIslands	fo_FO	Faroese fo
Finnish_Finland	fi_FI	Finnish fi
French_Belgium	fr_BE	
French_Canada	fr_CA	
French_France	fr_FR	French fr
French_Luxembourg	fr_LU	
French_Switzerland	fr_CH	
German_Austria	de_AT	
German_Germany	de_DE	German de
German_Liechtenstein	de_LI	
German_Luxembourg	de_LU	
German_Switzerland	de_CH	
Greek_Cyprus	el_CY	
Greek_Greece	el_GR	Greek el

SAS Name	POSIX Locale	Aliases
Greenlandic_Greenland	kl_GL	Greenlandic kl
Hebrew_Israel	he_IL	Hebrew he
Hindi_India	hi_IN	Hindi hi
Hungarian_Hungary	hu_HU	Hungarian hu
Icelandic_Iceland	is_IS	Icelandic is
Indonesian_Indonesia	id_ID	Indonesian id
Irish_Ireland	ga_IE	Irish ga
Italian_Italy	it_IT	Italian it
Italian_Switzerland	it_CH	
Japanese_Japan	ja_JP	Japanese ja
Kazakh_Kazakhstan	kk_KZ	Kazakh kk
Korean_Korea	ko_KR	Korean ko Korean_SouthKorea
Latvian_Latvia	lv_LV	Latvian lv
Lithuanian_Lithuania	lt_LT	Lithuanian lt
Macedonian_Macedonia	mk_MK	Macedonian mk

SAS Name	POSIX Locale	Aliases
Malay_Malaysia	ms_MY	Malay ms
Maltese_Malta	mt_MT	Maltese mt
Marathi_India	mr_IN	Marathi mr
NorwegianBokmal_Norway	nb_NO	NorwegianBokmal nb
NorwegianNynorsk_Norway	nn_NO	NorwegianNynorsk nn
Norwegian_Norway	no_NO	Norwegian no
Persian_India	fa_IN	
Persian_Iran	fa_IR	Persian fa
Polish_Poland	pl_PL	Polish pl
Portuguese_Brazil	pt_BR	
Portuguese_Portugal	pt_PT	Portuguese pt
Romanian_Romania	ro_RO	Romanian ro
Russian_Russia	ru_RU	Russian ru
Russian_Ukraine	ru_UA	
Serbian_BosniaHerzegovina	sr_BA	
Serbian_Montenegro	sr_ME	
Serbian_Serbia	sr_RS	Serbian sr
SerbianLatin_BosniaHerzegovina	sh_BA	

SAS Name	POSIX Locale	Aliases
SerbianLatin_Montenegro	sh_ME	
SerbianLatin_Serbia	sh_RS	SerbianLatin sh
Slovak_Slovakia	sk_SK	Slovak Slovakian Slovakian_Slovakia sk
Slovenian_Slovenia	sl_SI	Slovenian sl
Spanish_Argentina	es_AR	
Spanish_Bolivia	es_BO	
Spanish_Chile	es_CL	
Spanish_Colombia	es_CO	
Spanish_CostaRica	es_CR	
Spanish_DominicanRepublic	es_DO	
Spanish_Ecuador	es_EC	
Spanish_ElSalvador	es_SV	
Spanish_Guatemala	es_GT	
Spanish_Honduras	es_HN	
Spanish_Mexico	es_MX	
Spanish_Nicaragua	es_NI	
Spanish_Panama	es_PA	
Spanish_Paraguay	es_PY	
Spanish_Peru	es_PE	
Spanish_PuertoRico	es_PR	
Spanish_Spain	es_ES	Spanish es

SAS Name	POSIX Locale	Aliases
Spanish_UnitedStates	es_US	
Spanish_Uruguay	es_UY	
Spanish_Venezuela	es_VE	
Swedish_Sweden	sv_SE	Swedish sv
Tamil_India	ta_IN	Tamil ta
Telugu_India	te_IN	Telugu te
Thai_Thailand	th_TH	Thai th
Turkish_Turkey	tr_TR	Turkish tr
Ukrainian_Ukraine	uk_UA	Ukrainian uk
Vietnamese_Vietnam	vi_VN	Vietnamese vi

The following table lists the valid POSIX values. The settings for DFLANG, DATESTYLE, and PAPERSIZE system options are set automatically.

Here is an example:

```
sas -locale arabic_algeria
```

When the Arabic_Algeria LOCALE= value is specified, corresponding default settings for the system options are as follows:

```
DFLANG=English
DATESTYLE=DMY
PAPERSIZE=A4
```

Table 17.2 *Default Values for the DFLANG, DATESTYLE, and PAPERSIZE System Options Based on the LOCALE= System Option*

POSIX Locale	DFLANG=	DATESTYLE=	PAPERSIZE=
af_ZA	English	YMD	A4
ar_AE	English	DMY	A4

POSIX Locale	DFLANG=	DATESTYLE=	PAPERSIZE=
ar_BH	English	DMY	A4
ar_DZ	English	DMY	A4
ar_EG	English	DMY	A4
ar_IN	English	DMY	A4
ar_IQ	English	DMY	A4
ar_JO	English	DMY	A4
ar_KW	English	DMY	A4
ar_LB	English	DMY	A4
ar_LY	English	DMY	A4
ar_MA	English	DMY	A4
ar_OM	English	DMY	A4
ar_QA	English	DMY	A4
ar_SA	English	DMY	A4
ar_SD	English	DMY	A4
ar_SY	English	DMY	A4
ar_TN	English	DMY	A4
ar_YE	English	DMY	A4
be_BY	English	DMY	A4
bg_BG	English	YMD	A4
bn_IN	English	DMY	A4
bs_BA	English	YMD	A4
ca_ES	English	DMY	A4
cs_CZ	Czech	DMY	A4
da_DK	Danish	DMY	A4
de_AT	German	DMY	A4

POSIX Locale	DFLANG=	DATESTYLE=	PAPERSIZE=
de_CH	Swiss_ German	DMY	A4
de_DE	German	DMY	A4
de_LI	German	DMY	A4
de_LU	German	DMY	A4
el_CY	English	DMY	A4
el_GR	English	DMY	A4
en_AU	English	DMY	A4
en_BE	English	DMY	A4
en_BW	English	DMY	A4
en_CA	English	DMY	letter
en_CB	English	MDY	A4
en_GB	English	DMY	A4
en_HK	English	DMY	A4
en_IE	English	DMY	A4
en_IN	English	DMY	A4
en_JM	English	DMY	A4
en_MT	English	DMY	A4
en_NZ	English	DMY	A4
en_PH	English	MDY	letter
en_SG	English	DMY	A4
en_US	English	MDY	letter
en_ZA	English	DMY	A4
en_ZW	English	DMY	A4
es_AR	Spanish	DMY	A4
es_BO	Spanish	DMY	A4

POSIX Locale	DFLANG=	DATESTYLE=	PAPERSIZE=
es_CL	Spanish	DMY	letter
es_CO	Spanish	DMY	letter
es_CR	Spanish	DMY	letter
es_DO	Spanish	DMY	A4
es_EC	Spanish	DMY	A4
es_ES	Spanish	DMY	A4
es_GT	Spanish	DMY	letter
es_HN	Spanish	MDY	A4
es_MX	Spanish	DMY	letter
es_NI	Spanish	MDY	letter
es_PA	Spanish	MDY	letter
es_PE	Spanish	DMY	A4
es_PR	Spanish	MDY	letter
es_PY	Spanish	DMY	A4
es_SV	Spanish	MDY	letter
es_US	Spanish	DMY	letter
es_UY	Spanish	DMY	A4
es_VE	Spanish	DMY	letter
et_EE	English	DMY	A4
eu_ES	English	DMY	A4
fa_IN	English	YMD	A4
fa_IR	English	YMD	A4
fi_FI	Finnish	DMY	A4
fo_FO	English	DMY	A4
fr_BE	French	DMY	A4

POSIX Locale	DFLANG=	DATESTYLE=	PAPERSIZE=
fr_CA	French	DMY	letter
fr_CH	Swiss_French	DMY	A4
fr_FR	French	DMY	A4
fr-LU	French	DMY	A4
ga_IE	English	DMY	A4
gv_GB	English	DMY	A4
he_IL	English	DMY	A4
hi_IN	English	DMY	A4
hr_BA	Croatian	YMD	A4
hr_HR	Croatian	YMD	A4
hu_HU	Hungarian	YMD	A4
id_ID	English	DMY	A4
is_IS	English	DMY	A4
it_CH	Italian	DMY	A4
it_IT	Italian	DMY	A4
ja_JP	Japanese	YMD	A4
kk_KZ	English	DMY	A4
kl_GL	English	DMY	A4
ko_KR	Locale	YMD	A4
kw_GB	English	DMY	A4
lt_LT	English	YMD	A4
lv_LV	English	YMD	A4
mk_MK	English	DMY	A4
mr_IN	English	DMY	A4
ms_MY	English	DMY	A4

POSIX Locale	DFLANG=	DATESTYLE=	PAPERSIZE=
mt_MT	English	DMY	A4
nb_NO	Norwegian	DMY	A4
nl_BE	Dutch	DMY	A4
nl_NL	Dutch	DMY	A4
nn_NO	Norwegian	DMY	A4
no_NO	Norwegian	DMY	A4
pl_PL	Polish	YMD	A4
pt_BR	Portuguese	DMY	A4
pt_PT	Portuguese	DMY	A4
ro_RO	English	DMY	A4
ru_RU	Russian	DMY	A4
ru_UA	Russian	DMY	A4
sh_BA	English	DMY	A4
sh_ME	English	DMY	A4
sh_RS	English	DMY	A4
sk_SK	English	DMY	A4
sl_SI	Slovenian	YMD	A4
sq_AL	English	YMD	A4
sr_BA	English	DMY	A4
sr_ME	English	DMY	A4
sr_RS	English	DMY	A4
sv_SE	Swedish	YMD	A4
ta_IN	English	DMY	A4
te_IN	English	DMY	A4
th_TH	English	DMY	A4

POSIX Locale	DFLANG=	DATESTYLE=	PAPERSIZE=
tr_TR	English	DMY	A4
uk_UA	English	DMY	A4
vi_VN	English	DMY	A4
zh_CN	Locale	YMD	A4
zh_HK	Locale	YMD	A4
zh_MO	Locale	YMD	A4
zh_SG	Locale	DMY	A4
zh_TW	Locale	YMD	A4

Chapter 18

Encoding Values in SAS Language Elements

Overview to SAS Language Elements That Use Encoding Values

When the encoding of the SAS session is different from the encoding of the SAS file or from the data that resides in the SAS file, transcoding must occur. Consider a SAS file that was created in the Western Latin1 encoding, then read into SAS Studio connecting to SAS Viya running the UTF-8 encoding. In order for SAS Viya to successfully access the file, the SAS data file must be transcoded from the Western Latin1 encoding to UTF-8 encoding. For information about transcoding concepts, including SAS language elements that contain options for transcoding, see Chapter 4, “Transcoding for NLS,” on page 19 .

SBCS, DBCS, and Unicode Encoding Values for Transcoding Data

The following table presents a list of SBCS, DBCS, and Unicode encoding values for transcoding data for all operating environments. The encoding values are valid for SAS language elements that contain options for transcoding.

Note: If an encoding value contains a hyphen (-), enclose the encoding value in quotation marks.

Table 18.1 *SBCS, DBCS, and Unicode Encoding Values Used to Transcode Data*

Encoding Name	Short Name	Description	Maximum bytes per Character
aarabic	aara	Arabic Macintosh	1
acroatian	acro	MacOS 36 Croatian	1

Encoding Name	Short Name	Description	Maximum bytes per Character
acyrillic	acyr	MacOS 7 Cyrillic	1
agreek	agrk	Greek Macintosh	1
ahebrew	aheb	Hebrew Macintosh	1
aiceland	aice	Icelandic Macintosh	1
any	anye	no transcoding is specified	1
arabic	arab	Arabic ISO	1
aroman	arom	Roman Macintosh	1
aromania	arma	MacOS 38 Romania	1
athai	atha	MacOS 21-Thai	1
aturkish	atur	Turkish Macintosh	1
aukrainian	aukr	Ukrainian Macintosh	1
big5	big5	Traditional Chinese Big5	2
cyrillic	cyrl	Cyrillic ISO	1
dec-cn	zvms	Simplified Chinese DEC	4
dec-jp	jvms	Japanese DEC	2
dec-tw	yvms	Traditional Chinese DEC	4
ebcdic037	e037	North American EBCDIC	1
ebcdic275	e275	Brazil EBCDIC	1
ebcdic424	e424	Hebrew EBCDIC	1
ebcdic425	e425	Arabic EBCDIC	1
ebcdic500	e500	International EBCDIC	1
ebcdic838	e838	Thai EBCDIC	1

Encoding Name	Short Name	Description	Maximum bytes per Character
ebcdic870	e870	Central European EBCDIC	1
ebcdic875	e875	Greek EBCDIC	1
ebcdic905	e905	Latin 3 EBCDIC	1
ebcdic924	e924	European EBCDIC	1
ebcdic1025	ecyr	Cyrillic EBCDIC	1
ebcdic1026	etur	Turkish EBCDIC	1
ebcdic1047	elat	Western EBCDIC	1
ebcdic1097	e097	Farsi Bilingual EBCDIC	1
ebcdic1112	ebal	Baltic EBCDIC	1
ebcdic1122	eest	Estonian EBCDIC	1
ebcdic1130	evie	Vietnamese EBCDIC	1
ebcdic1137	e137	Devanagari EBCDIC	1
ebcdic1140	e140	North American EBCDIC	1
ebcdic1141	e141	Austria/Germany EBCDIC	1
ebcdic1142	e142	Denmark/Norway EBCDIC	1
ebcdic1143	e143	Finland/Sweden EBCDIC	1
ebcdic1144	e144	Italy EBCDIC	1
ebcdic1145	e145	Spain EBCDIC	1
ebcdic1146	e146	United Kingdom EBCDIC	1
ebcdic1147	e147	France EBCDIC	1
ebcdic1148	e148	International EBCDIC	1

Encoding Name	Short Name	Description	Maximum bytes per Character
ebcdic1149	e149	Iceland EBCDIC	1
ebcdic1153	e153	Latin 2 Euro EBCDIC	1
ebcdic1154	e154	Cyrillic Euro EBCDIC	1
ebcdic1155	e155	Turkey Euro EBCDIC	1
ebcdic1156	e156	Baltic Euro EBCDIC	1
ebcdic1157	e157	Estonia Euro EBCDIC	1
ebcdic1158	e158	Cryillic Ukraine Euro EBCDIC	1
ebcdic1160	e160	cp1160 EBCDIC	1
ebcdic1164	e164	cp1164 EBCDIC	1
ebcdicany	eany	enables you to create a data set that is compatible with all EBCDIC encodings	1
euc-cn	zeuc	Simplified Chinese EUC	2
euc-jp	jeuc	Japanese EUC	4
euc-kr	keuc	Korean EUC	4
euc-tw	yeuc	Traditional Chinese EUC	4
fujitsu-cn	zfuj	Simplified Chinese FACOM	4
fujitsu-jp	jfug	Japanese FACOM	4
fujitsu-ko	kfuj	Korean FACOM	4
fujitsu-tw	yfuj	Traditional Chinese FACOM	4
gb18030	gbke	Simplified Chinese GB18030	4

Encoding Name	Short Name	Description	Maximum bytes per Character
greek	grek	Greek ISO	1
hebrew	hebr	Hebrew ISO	1
hitachi-cn	zhit	Simplified Chinese HITAC	6
hitachi-jp	jhit	Japanese HITAC	6
hitachi-ko	khit	Korean HITAC	6
hitachi-tw	yhit	Traditional Chinese HITAC	4
hitsas-jp	jhts	Japanese XHITAC	4
hitsas-ko	khts	Korean XHITAC	4
hitsas-tw	yhts	Traditional Chinese XHITAC	4
hp15-tw	yhpx	Traditional Chinese HP15	2
ibm-1381	zpce	Simplified Chinese PCIBM	2
ibm-930	j930	Japanese katakana	4
ibm-933	kibm	Korean IBM	4
ibm-935	zibm	Simplified Chinese IBM	4
ibm-937	yibm	Traditional Chinese IBM	4
ibm-939	jibm	Japanese IBM	4
ibm-942	j942	Japanese PCIBM	2
ibm-949	kpce	Korean PCIBM	2
iso2022cncns	zist	Traditional Chinese ISO-2022	4
iso2022cngb	ziso	Simplified Chinese ISO-2022	4
iso2022jp	jiso	Japanese ISO-2022	8

Encoding Name	Short Name	Description	Maximum bytes per Character
iso2022kr	kiso	Korean ISO-2022	4
kz1048	kz48	Kazakh cp1048 Cyrillic	1
latin1	lat1	Western ISO	1
latin2	lat2	Central European ISO	1
latin3	lat3	Latin 3 ISO 8859/3	1
latin4	lat4	Latin 4 ISO 8859/4	1
latin5	lat5	Turkish ISO	1
latin6	lat6	Baltic ISO	1
latin7	lat7	Baltic Rim	1
latin8	lat8	Latin 8 ISO 8859/14	1
latin9	lat9	European ISO	1
latin10	lat10	South-eastern Europe	1
macos-1	jmac	Japanese PCMAC	2
macos-2	ymac	Traditional Chinese PCMAC	2
macos-3	kmac	Korean PCMAC	2
macos-25	zmac	Simplified Chinese PCMAC	2
ms-932	j932	Japanese PCMS	2
ms-936	zwin	Simplified Chinese PCMS	2
ms-949	kwin	Korean PCMS	2
ms-950	ywin	Traditional Chinese PCMS	2
msdos720	p720	Arabic MS-DOS	1
msdos737	p737	Greek MS-DOS	1

Encoding Name	Short Name	Description	Maximum bytes per Character
msdos775	p775	Baltic MS-DOS	1
open_ed-037	eous	USA Open Edition	1
open_ed-275	eobr	Brazil OpenEdition	1
open_ed-424	eoiw	Hebrew OpenEdition	1
open_ed-425	eoa2	Arabic OpenEdition	1
open_ed-838	eoth	Thai OpenEdition	1
open_ed-870	eol2	Central European OpenEdition	1
open_ed-875	eoel	Greek OpenEdition	1
open_ed-905	eol3	Latin 3 Open Edition EBCDIC	1
open_ed-924	eolt	European OpenEdition	1
open_ed-930	oe30	Katakana OpenEdition	4
open_ed-933	oe33	Korean OpenEdition	4
open_ed-935	oe35	Simplified Chinese OpenEdition	4
open_ed-937	oe37	Traditional Chinese OpenEdition	4
open_ed-939	oe39	Japanese IBM	4
open_ed-939E	oe9e	Japanese IBM Extended’	4
open_ed-1025	eocy	Cyrillic OpenEdition	1
open_ed-1026	eotr	Turkish OpenEdition	1
open_ed-1047	eol1	Western OpenEdition	1
open_ed-1097	eofa	Farsi Bilingual OpenEdition EBCDIC	1
open_ed-1112	eobl	Baltic OpenEdition	1

Encoding Name	Short Name	Description	Maximum bytes per Character
open_ed-1122	eoet	Estonian OpenEdition	1
open_ed-1130	eovi	Vietnamese OpenEdition	1
open_ed-1140	eo40	North American OpenEdition	1
open_ed-1141	eo41	Austria/Germany OpenEdition	1
open_ed-1142	eo42	Denmark/Norway OpenEdition	1
open_ed-1143	eo43	Finland/Sweden OpenEdition	1
open_ed-1144	eo44	Italy OpenEdition	1
open_ed-1145	eo45	Spain OpenEdition	1
open_ed-1146	eo46	United Kingdom OpenEdition	1
open_ed-1147	eo47	France OpenEdition	1
open_ed-1148	eo48	International OpenEdition	1
open_ed-1149	eoIS	Iceland OpenEdition EBCDIC	1
open_ed-1153	eo53	Latin 2 Euro OpenEdition EBCDIC	1
open_ed-1154	eo54	Cyrillic Euro OpenEdition EBCDIC	1
open_ed-1155	eo55	Turkey Euro OpenEdition EBCDIC	1
open_ed-1156	eo56	Baltic Euro OpenEdition EBCDIC	1
open_ed-1157	eo57	Estonia Euro OpenEdition EBCDIC	1

Encoding Name	Short Name	Description	Maximum bytes per Character
open_ed-1158	eo58	Cryrillic Ukraine Euro 1158 OpenEdition EBCDIC	1
open_ed-1160	eo60	1160 OpenEdition EBCDIC	1
open_ed-1164	eo64	1164 OpenEdition EBCDIC	1
open_ed-1166	eo66	1166 OpenEdition kz-cyr	1
pciscii806	p806	Indian PC	1
pcoem437	p437	USA IBM-PC	1
pcoem850	p850	Western IBM-PC	1
pcoem852	p852	Central European IBM-PC	1
pcoem857	p857	Turkish IBM-PC	1
pcoem858	p858	European IBM-PC	1
pcoem860	p860	Portuguese MS-DOS	1
pcoem862	p862	Hebrew IBM-PC	1
pcoem863	p863	French Canadian IBM-PC	1
pcoem864	p864	Arabic IBM-PC	1
pcoem865	p865	Nordic IBM-PC	1
pcoem866	p866	Cyrillic IBM-PC	1
pcoem869	p869	Greek IBM-PC	1
pcoem874	p874	Thai IBM-PC	1
pcoem921	p921	Baltic IBM-PC	1
pcoem922	p922	Estonia IBM-PC	1
pcoem1129	pvie	Vietnamese IBM-PC	1

Encoding Name	Short Name	Description	Maximum bytes per Character
pc1098	po98	Farsi PC	1
roman8	rom8	HP Roman 8	1
shift-jis	sjis	Japanese SJIS	2
shift-jis04	sjs4	Shift_JIS-2004 surrogate pair support	2
thai	thai	Thai ISO	1
us-ascii	ansi	enables you to create a data set that is compatible with all ASCII encodings	1
utf-8	utf8	Unicode (UTF-8)	4
utf-16be	u16b	Unicode (UTF-16BE)	2 *
utf-16le	u16l	Unicode (UTF-16LE)	2 *
utf-32be	u32b	Unicode (UTF-32BE)	4 **
utf-32le	u32l	Unicode (UTF-32LE)	4 **
warabic	wara	Arabic Windows	1
wbaltic	wbal	Baltic Windows	1
wcyrillic	wcyr	Cyrillic Windows	1
wgreek	wgrk	Greek Windows	1
whebrew	wheb	Hebrew Windows	1
wlatin1	wlt1	Western Windows	1
wlatin2	wlt2	Central European Windows	1
wturkish	wtur	Turkish Windows	1
wvietnamese	wvie	Vietnamese Windows	1

* UTF-16BE and UTF-16LE have a fixed length of two bytes per character.

** UTF-32BE and UTF-32LE have a fixed length of four bytes per character.

Part 11

Appendixes

Appendix 1

Encodings and Their Aliases and Encoding Character Set Compatibility

This table lists common encoding methods and their corresponding encoding and alias names.

Table A1.1 *Encoding and Alias Names*

Encoding Methods and Character Sets	Encoding Names in SAS	Alias
Unicode	UTF-8	UTF-8
SBCS	WLATIN2	CP1250
	WCYRILLIC	CP1251
	WLATIN1	CP1252
	WGREEK	CP1253
	WTURKISH	CP1254
	WHEBREW	CP1255
	WARABIC	CP1256
	WBALTIC	CP1257
	LATIN1	ISO8859_1, ISO8859-1
	LATIN2	ISO8859_2, ISO8859-2
	LATIN3	ISO8859_3, ISO8859-3
	CYRILLIC	ISO8859_5, ISO8859-5
	ARABIC	ISO8859_6, ISO8859-6
	GREEK	ISO8859_7, ISO8859-7
	HEBREW	ISO8859_8, ISO8859-8

Encoding Methods and Character Sets	Encoding Names in SAS	Alias
	TURKISH	ISO8859_9, ISO8859-9
	THAI	ISO8859_11, ISO8859-11
	LATIN7	ISO8859_13, ISO8859-13
	LATIN9	ISO8859_15, ISO8859-15
	LATIN10	ISO8859_16, ISO8859-16
	OPEN_ED-838	EBCDIC838, CP838
	OPEN_ED-870	EBCDIC870, CP870
	OPEN_ED-875	EBCDIC875, CP875
	OPEN_ED-1047	EBCDIC1047, CP1047
	OPEN_ED-1122	EBCDIC1122, CP1122
	OPEN_ED-1130	EBCDIC1130, CP1130
	OPEN_ED-1141	EBCDIC1141, CP1141
	OPEN_ED-1142	EBCDIC1142, CP1142
	OPEN_ED-1143	EBCDIC1143, CP1143
	OPEN_ED-1144	EBCDIC1144, CP1144
	OPEN_ED-1145	EBCDIC1145, CP1145
	OPEN_ED-1146	EBCDIC1146, CP1146
	OPEN_ED-1147	EBCDIC1147, CP1147
	OPEN_ED-1148	EBCDIC1148, CP1148
DBCS	SHIFT-JIS	MS-932, IBM-942, MACOS-1
	MS-949	MACOS-3, EUC-KR
	EUC-CN	MS-936, MACOS-25, DEC-CN
	EUC-TW	DEC-TW
	MS-950	MACOS-2, BIG5

This table shows languages (locales) that support the same set of characters. For example, the Arabic locale in the first column has five compatible character sets: WARABIC, ARABIC, MSDOS720, PCOEM864, and OPEN_ED-425. Encoding names used by SAS for the character sets are listed in the fourth column.

Table A1.2 *Encoding Character Set Compatibility*

Language	Language Code (ISO 639)	Script Group	Encoding Names (in SAS)	Encoding Names (in Java)
Arabic	ar	Arabic	WARABIC, ARABIC, MSDOS720, PCOEM864, OPEN_ED-425	CP1256, ISO8859_6, CP864
Bengali	bn	Bengali		
Persian	fa	Arabic	WARABIC, ARABIC, MSDOS720, PCOEM864, OPEN_ED-425	CP1256, ISO8859_6, CP864
Chinese (Simplified)	zh	Chinese (Simplified)	MS-936, EUC-CN, IBM-935	GBK, GBK, CP935
Chinese (Traditional)	zh	Chinese (Traditional)	MS-950, EUC-TW, IBM-937	BIG5, BIG5, CP937
Belarusian	be	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED-1025	CP1251, ISO8859_5, CP866, CP1025
Bulgarian	bg	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED-1025	CP1251, ISO8859_5, CP866, CP1025
Macedonian	mk	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED-1025	CP1251, ISO8859_5, CP866, CP1025
Russian	ru	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED-1025	CP1251, ISO8859_5, CP866, CP1025

Language	Language Code (ISO 639)	Script Group	Encoding Names (in SAS)	Encoding Names (in Java)
Serbian	sr	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED-1025	CP1251, ISO8859_5, CP866, CP1025
Ukrainian	uk	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED-1025	CP1251, ISO8859_5, CP866, CP1025
Hindi	hi	Devanagari	PCISCII806, OPEN_ED-1137	
Marathi	mr	Devanagari	PCISCII806, OPEN_ED-1137	
Greek	el	Greek	WGREEK, GREEK, MSDOS737, PCOEM869, OPEN_ED-875	CP1253, ISO8859_7, CP869, CP875
Hebrew	he	Hebrew	WHEBREW, HEBREW, PCOEM862, OPEN_ED-424	CP1255, ISO8859_8, CP862, CP424
Japanese	ja	Japanese	SHIFT-JIS, EUC-JP, IBM-939	SJIS, EUC_JP, CP939
Korean	ko	Korean	MS-949, EUC-KR, IBM-933	MS649, EUC_KR, CP933
Afrikaans	af	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047
Catalan	ca	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED-1145	CP1252, ISO8859_1, ISO8859_15, CP850, CP1145
Cornish	kw	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1146	CP1252, ISO8859_1, ISO8859_15, CP850, CP1146

Language	Language Code (ISO 639)	Script Group	Encoding Names (in SAS)	Encoding Names (in Java)
Danish	da	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1142	CP1252, ISO8859_1, ISO8859_15, CP850, CP1142
Dutch	nl	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED-1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047
English	en	LATIN1	WLATIN1, LATIN1, PCOEM437, OPEN_ED-1047	CP1252, ISO8859_1, ISO8859_15, CP437, CP1047
Faroese	fo	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1142	CP1252, ISO8859_1, ISO8859_15, CP850, CP1142
Finnish	fi	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED-1143	CP1252, ISO8859_1, ISO8859_15, CP850, CP1143
French	fr	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED-1147	CP1252, ISO8859_1, ISO8859_15, CP850, CP1147
German	de	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED-1141	CP1252, ISO8859_1, ISO8859_15, CP850, CP1141
Greenlandic	kl	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1142	CP1252, ISO8859_1, ISO8859_15, CP850, CP1142
Icelandic	is	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047

Language	Language Code (ISO 639)	Script Group	Encoding Names (in SAS)	Encoding Names (in Java)
Indonesian	íd	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047
Italian	it	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED-1144	CP1252, ISO8859_1, ISO8859_15, CP850, CP1144
Malay	ms	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047
Manx	gv	LATIN1	WLATIN1, LATIN8, PCOEM850, OPEN_ED-1148	CP1252, ISO8859_14, CP850, CP1148
Norwegian	no	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED-1142	CP1252, ISO8859_1, ISO8859_15, CP850, CP1142
Portuguese	pt	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED-1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047
Spanish	es	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED-1145	CP1252, ISO8859_1, ISO8859_15, CP850, CP1145
Albanian	sq	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Bosnian	bs	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Croatian	hr	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870

Language	Language Code (ISO 639)	Script Group	Encoding Names (in SAS)	Encoding Names (in Java)
Czech	cs	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Hungarian	hu	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Polish	pl	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Romanian	ro	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Serbian (Latin)	sh	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Slovak	sk	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Slovenian	sl	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870
Maltese	mt	LATIN3	CP1254, ISO8859_9, CP857, CP1026	ISO8859_3
Turkish	tr	LATIN5	WTURKISH, TURKISH, PCOEM857, OPEN_ED-102 6	CP1254, ISO8859_9, CP857, CP1026
Estonian	et	LATIN6	WBALTIC, LATIN6, MSDOS775, PCOEM921, OPEN_ED-1112	CP1257, N/A, CP775, CP1112
Latvian	lv	LATIN6	WBALTIC, LATIN6, MSDOS775, PCOEM921, OPEN_ED-1112	CP1257, N/A, CP775, CP1112

Language	Language Code (ISO 639)	Script Group	Encoding Names (in SAS)	Encoding Names (in Java)
Lithuanian	lt	LATIN6	WBALTIC, LATIN6, MSDOS775, PCOEM921, OPEN_ED-1112	PCOEM874, THAI
Tamil	ta	Tamil		
Telugu	te	Telegu		
Thai	th	Thai	PCOEM874, THAI	CP874, TIS620
Vietnamese	vi	Vietnamese	WVIETNAMESE, OPEN_ED-1130	CP1258, CP1130

Appendix 2

Time Zone IDs and Time Zone Names

Area: Africa

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Africa/Abidjan	GMT	CI	Greenwich Mean Time	00:00
Africa/Accra	GMT	GH	Greenwich Mean Time	00:00
Africa/Bamako	GMT	ML	Greenwich Mean Time	00:00
Africa/Banjul	GMT	GM	Greenwich Mean Time	00:00
Africa/Bissau	GMT	GW	Greenwich Mean Time	00:00
Africa/Conakry	GMT	GN	Greenwich Mean Time	00:00
Africa/Dakar	GMT	SN	Greenwich Mean Time	00:00
Africa/Freetown	GMT	SL	Greenwich Mean Time	00:00
Africa/Lome	GMT	TG	Greenwich Mean Time	00:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Africa/Monrovia	GMT	LR	Greenwich Mean Time	00:00
Africa/Nouakchott	GMT	MR	Greenwich Mean Time	00:00
Africa/Ouagadougou	GMT	BF	Greenwich Mean Time	00:00
Africa/Sao_Tome	GMT	ST	Greenwich Mean Time	00:00
Africa/Timbuktu	GMT	ML	Greenwich Mean Time	0.00
Africa/Casablanca	WET WEST	MA MA	Western European Time Western European Summer Time	00:00 01.00
Africa/El_Aaiun	WET WEST	EH EH	Western European Time Western European Summer Time	00:00 01.00
Africa/Algiers	CET	DZ	Central European Time	01:00
Africa/Ceuta	CET CEST	ES	Central European Time Central European Summer Time	01:00 02:00
Africa/Tunis	CET	TN	Central European Time	01:00
Africa/Bangui	WAT	CF	West Africa Time	01:00
Africa/Brazzaville	WAT	CG	West Africa Time	01:00
Africa/Douala	WAT	CM	West Africa Time	01:00
Africa/Kinshasa	WAT	CD	West Africa Time	01:00
Africa/Lagos	WAT	NG	West Africa Time	01:00
Africa/Libreville	WAT	GA	West Africa Time	01:00
Africa/Luanda	WAT	AO	West Africa Time	01:00
Africa/Malabo	WAT	GQ	West Africa Time	01:00
Africa/Ndjamena	WAT	TD	West Africa Time	01:00
Africa/Niamey	WAT	NE	West Africa Time	01:00
Africa/Porto-Novo	WAT	BJ	West Africa Time	01:00
Africa/Windhoek	WAT WAST	NA	West Africa Time West Africa Summer Time	01:00 02:00
Africa/Blantyre	CAT	MW	Central Africa Time	02:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Africa/Bujumbura	CAT	BI	Central Africa Time	02:00
Africa/Gaborone	CAT	BW	Central Africa Time	02:00
Africa/Harare	CAT	ZW	Central Africa Time	02:00
Africa/Kigali	CAT	RW	Central Africa Time	02:00
Africa/Lubumbashi	CAT	CD	Central Africa Time	02:00
Africa/Lusaka	CAT	ZM	Central Africa Time	02:00
Africa/Maputo	CAT	MZ	Central Africa Time	02:00
Africa/Cairo	EET EET	EG EG	Eastern European Time Eastern European Summer Tim	02:00 2:00
Africa/Tripoli	EET	LY	Eastern European Time	02:00
Africa/Johannesburg	SAST	ZA	South Africa Standard Time	02:00
Africa/Maseru	SAST	LS	South Africa Standard Time	02:00
Africa/Mbabane	SAST	SZ	South Africa Standard Time	02:00
Africa/Addis_Ababa	EAT	ET	East Africa Time	03:00
Africa/Asmara	EAT	ER	East Africa Time	03.00
Africa/Asmera	EAT	ER	East Africa Time	03:00
Africa/Dar_es_Salaam	EAT	TZ	East Africa Time	03:00
Africa/Djibouti	EAT	DJ	East Africa Time	03:00
Africa/Juba	EAT	SS	East Africa Time	03:00
Africa/Kampala	EAT	UG	East Africa Time	03:00
Africa/Khartoum	EAT	SD	East Africa Time	03:00
Africa/Mogadishu	EAT	SO	East Africa Time	03:00
Africa/Nairobi	EAT	KE	East Africa Time	03:00

Area: America–North, Central, and South

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Adak	HAST	US	Hawaii-Aleutian Standard Time	-10:00
America/Adak	HADT	US	Hawaii-Aleutian Daylight Time	-9:00
America/Anchorage	AKST	US	Alaska Standard Time	-9:00
America/Anchorage	AKDT	US	Alaska Daylight Time	-8:00
America/Anguilla	AST	AI	Atlantic Standard Time	-4:00
America/Antigua	AST	AG	Atlantic Standard Time	-4:00
America/Araguaina	BRT	BR	Brasilia Standard Time	-3:00
America/Argentina/Buenos_Aires	ART	AR	Argentina Standard Time	-3:00
America/Argentina/Catamarca	ART	AR	Argentina Standard Time	-3:00
America/Argentina/ ComodRivadavia	ART	AR	Argentina Standard Time	-3:00
America/Argentina/Cordoba	ART	AR	Argentina Standard Time	-3:00
America/Argentina/Jujuy	ART	AR	Argentina Standard Time	-3:00
America/Argentina/La_Rioja	ART	AR	Argentina Standard Time	-3:00
America/Argentina/Mendoza	ART	AR	Argentina Standard Time	-3:00
America/Argentina/Rio_Gallegos	ART	AR	Argentina Standard Time	-3:00
America/Argentina/Salta	ART	AR	Argentina Standard Time	-3:00
America/Argentina/San_Juan	ART	AR	Argentina Standard Time	-3:00
America/Argentina/San_Luis	ART	AR	Western Argentina Standard Time	-3:00
America/Argentina/Tucuman	ART	AR	Argentina Standard Time	-3:00
America/Argentina/Ushuaia	ART	AR	Argentina Standard Time	-3:00
America/Aruba	AST	AW	Atlantic Standard Time	-4:00
America/Asuncion	PYT	PY	Paraguay Standard Time	-4:00
America/Asuncion	PYST	PY	Paraguay Summer Time	-3:00
America/Atikokan	EST	CA	Eastern Standard Time	-5:00
America/Atka	HAST	US	Hawaii-Aleutian Standard Time	-10:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Atka	HADT	US	Hawaii-Aleutian Daylight Time	-9:00
America/Bahia	BRT	BR	Brasilia Standard Time	-3:00
America/Bahia_Banderas	CST	MX	Central Standard Time	-6:00
America/Bahia_Banderas	CDT	MX	Central Daylight Time	-5:00
America/Barbados	AST	BB	Atlantic Standard Time	-4:00
America/Belem	BRT	BR	Brasilia Standard Time	-3:00
America/Belize	CST	BZ	Central Standard Time	-6:00
America/Blanc-Sablon	AST	CA	Atlantic Standard Time	-4:00
America/Boa_Vista	AMT	BR	Amazon Standard Time	-4:00
America/Bogota	COT	CO	Colombia Standard Time	-5:00
America/Boise	MST	US	Mountain Standard Time	-7:00
America/Boise	MDT	US	Mountain Daylight Time	-6:00
America/Buenos_Aires	ART	AR	Argentina Standard Time	-3:00
America/Cambridge_Bay	MST	CA	Mountain Standard Time	-7:00
America/Cambridge_Bay	MDT	CA	Mountain Daylight Time	-6:00
America/Campo_Grande	AMT	BR	Amazon Standard Time	-4:00
America/Campo_Grande	AMST	BR	Amazon Summer Time	-3:00
America/Cancun	CST	MX	Central Standard Time	-6:00
America/Cancun	CDT	MX	Central Daylight Time	-5:00
America/Caracas	VET	VE	Venezuela Time	-1:27
America/Catamarca	ART	AR	Argentina Standard Time	-3:00
America/Cayenne	GFT	GF	French Guiana Time	-3:00
America/Cayman	EST	KY	Eastern Standard Time	-5:00
America/Chicago	CST	US	Central Standard Time	-6:00
America/Chicago	CDT	US	Central Daylight Time	-5:00
America/Chihuahua	MST	MX	Mountain Standard Time	-7:00
America/Chihuahua	MDT	MX	Mountain Daylight Time	-6:00
America/Coral_Harbour	EST	CA	Eastern Standard Time	-5:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Cordoba	ART	AR	Argentina Standard Time	-3:00
America/Costa_Rica	CST	CR	Central Standard Time	-6:00
America/Creston	MST	CA	Mountain Standard Time	-7:00
America/Cuiaba	AMT	BR	Amazon Standard Time	-4:00
America/Cuiaba	AMST	BR	Amazon Summer Time	-3:00
America/Curacao	AST	CW	Atlantic Standard Time	-4:00
America/Danmarkshavn	GMT	GL	Greenwich Mean Time	0:00
America/Dawson	PST	CA	Pacific Standard Time	-8:00
America/Dawson	PDT	CA	Pacific Daylight Time	-7:00
America/Dawson_Creek	MST	CA	Mountain Standard Time	-7:00
America/Denver	MST	US	Mountain Standard Time	-7:00
America/Denver	MDT	US	Mountain Daylight Time	-6:00
America/Detroit	EST	US	Eastern Standard Time	-5:00
America/Detroit	EDT	US	Eastern Daylight Time	-4:00
America/Dominica	AST	DM	Atlantic Standard Time	-4:00
America/Edmonton	MST	CA	Mountain Standard Time	-7:00
America/Edmonton	MDT	CA	Mountain Daylight Time	-6:00
America/Eirunepe	ACT	BR	Amazon Standard Time	-5:00
America/El_Salvador	CST	SV	Central Standard Time	-6:00
America/Ensenada	PST	MX	Pacific Standard Time	-8:00
America/Ensenada	PDT	MX	Pacific Daylight Time	-7:00
America/Fort_Wayne	EST	US	Eastern Standard Time	-5:00
America/Fort_Wayne	EDT	US	Eastern Daylight Time	-4:00
America/Fortaleza	BRT	BR	Brasilia Standard Time	-3:00
America/Glace_Bay	AST	CA	Atlantic Standard Time	-4:00
America/Glace_Bay	ADT	CA	Atlantic Daylight Time	-3:00
America/Godthab	WGT	GL	West Greenland Standard Time	-3:00
America/Godthab	WGST	GL	West Greenland Summer Time	-2:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Goose_Bay	AST	CA	Atlantic Standard Time	-4:00
America/Goose_Bay	ADT	CA	Atlantic Daylight Time	-3:00
America/Grand_Turk	EST	TC	Eastern Standard Time	-5:00
America/Grand_Turk	EDT	TC	Eastern Daylight Time	-4:00
America/Grenada	AST	GD	Atlantic Standard Time	-4:00
America/Guadeloupe	AST	GP	Atlantic Standard Time	-4:00
America/Guatemala	CST	GT	Central Standard Time	-6:00
America/Guayaquil	ECT	EC	Ecuador Time	-5:00
America/Guyana	GYT	GY	Guyana Time	-4:00
America/Halifax	AST	CA	Atlantic Standard Time	-4:00
America/Halifax	ADT	CA	Atlantic Daylight Time	-3:00
America/Havana	CST	CU	Cuba Standard Time	-5:00
America/Havana	CDT	CU	Cuba Daylight Time	-4:00
America/Hermosillo	MST	MX	Mountain Standard Time	-7:00
America/Indiana/Indianapolis	EST	US	Eastern Standard Time	-5:00
America/Indiana/Indianapolis	EDT	US	Eastern Daylight Time	-4:00
America/Indiana/Knox	CST	US	Central Standard Time	-6:00
America/Indiana/Knox	CDT	US	Central Daylight Time	-5:00
America/Indiana/Marengo	EST	US	Eastern Standard Time	-5:00
America/Indiana/Marengo	EDT	US	Eastern Daylight Time	-4:00
America/Indiana/Petersburg	EST	US	Eastern Standard Time	-5:00
America/Indiana/Petersburg	EDT	US	Eastern Daylight Time	-4:00
America/Indiana/Tell_City	CST	US	Central Standard Time	-6:00
America/Indiana/Tell_City	CDT	US	Central Daylight Time	-5:00
America/Indiana/Vevay	EST	US	Eastern Standard Time	-5:00
America/Indiana/Vevay	EDT	US	Eastern Daylight Time	-4:00
America/Indiana/Vincennes	EST	US	Eastern Standard Time	-5:00
America/Indiana/Vincennes	EDT	US	Eastern Daylight Time	-4:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Indiana/Winamac	EST	US	Eastern Standard Time	-5:00
America/Indiana/Winamac	EDT	US	Eastern Daylight Time	-4:00
America/Indianapolis	EST	US	Eastern Standard Time	-5:00
America/Indianapolis	EDT	US	Eastern Daylight Time	-4:00
America/Inuvik	MST	CA	Mountain Standard Time	-7:00
America/Inuvik	MDT	CA	Mountain Daylight Time	-6:00
America/Iqaluit	EST	CA	Eastern Standard Time	-5:00
America/Iqaluit	EDT	CA	Eastern Daylight Time	-4:00
America/Jamaica	EST	JM	Eastern Standard Time	-5:00
America/Jujuy	ART	AR	Argentina Standard Time	-3:00
America/Juneau	AKST	US	Alaska Standard Time	-9:00
America/Juneau	AKDT	US	Alaska Daylight Time	-8:00
America/Kentucky/Louisville	EST	US	Eastern Standard Time	-5:00
America/Kentucky/Louisville	EDT	US	Eastern Daylight Time	-4:00
America/Kentucky/Monticello	EST	US	Eastern Standard Time	-5:00
America/Kentucky/Monticello	EDT	US	Eastern Daylight Time	-4:00
America/Knox_IN	CST	US	Central Standard Time	-6:00
America/Knox_IN	CDT	US	Central Daylight Time	-5:00
America/Kralendijk	AST	BQ	Atlantic Standard Time	-4:00
America/La_Paz	BOT	BO	Bolivia Time	-4:00
America/Lima	PET	PE	Peru Standard Time	-5:00
America/Los_Angeles	PST	US	Pacific Standard Time	-8:00
America/Los_Angeles	PDT	US	Pacific Daylight Time	-7:00
America/Louisville	EST	US	Eastern Standard Time	-5:00
America/Louisville	EDT	US	Eastern Daylight Time	-4:00
America/Lower_Princes	AST	SX	Atlantic Standard Time	-4:00
America/Maceio	BRT	BR	Brasilia Standard Time	-3:00
America/Managua	CST	NI	Central Standard Time	-6:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Manaus	AMT	BR	Amazon Standard Time	-4:00
America/Marigot	AST	MF	Atlantic Standard Time	-4:00
America/Martinique	AST	MQ	Atlantic Standard Time	-4:00
America/Matamoros	CST	MX	Central Standard Time	-6:00
America/Matamoros	CDT	MX	Central Daylight Time	-5:00
America/Mazatlan	MST	MX	Mountain Standard Time	-7:00
America/Mazatlan	MDT	MX	Mountain Daylight Time	-6:00
America/Mendoza	ART	AR	Argentina Standard Time	-3:00
America/Menominee	CST	US	Central Standard Time	-6:00
America/Menominee	CDT	US	Central Daylight Time	-5:00
America/Merida	CST	MX	Central Standard Time	-6:00
America/Merida	CDT	MX	Central Daylight Time	-5:00
America/Metlakatla	MEST	US	Pacific Standard Time	-8:00
America/Mexico_City	CST	MX	Central Standard Time	-6:00
America/Mexico_City	CDT	MX	Central Daylight Time	-5:00
America/Miquelon	PMST	PM	Saint Pierre and Miquelon Standard Time	-3:00
America/Miquelon	PMDT	PM	Saint Pierre and Miquelon Daylight Time	-2:00
America/Moncton	AST	CA	Atlantic Standard Time	-4:00
America/Moncton	ADT	CA	Atlantic Daylight Time	-3:00
America/Monterrey	CST	MX	Central Standard Time	-6:00
America/Monterrey	CDT	MX	Central Daylight Time	-5:00
America/Montevideo	UYT	UY	Uruguay Standard Time	-3:00
America/Montevideo	UYST	UY	Uruguay Summer Time	-2:00
America/Montserrat	AST	MS	Atlantic Standard Time	-4:00
America/Nassau	EST	BS	Eastern Standard Time	-5:00
America/Nassau	EDT	BS	Eastern Daylight Time	-4:00
America/New_York	EST	US	Eastern Standard Time	-5:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/New_York	EDT	US	Eastern Daylight Time	-4:00
America/Nipigon	EST	CA	Eastern Standard Time	-5:00
America/Nipigon	EDT	CA	Eastern Daylight Time	-4:00
America/Nome	AKST	US	Alaska Standard Time	-9:00
America/Nome	AKDT	US	Alaska Daylight Time	-8:00
America/Noronha	FNT	BR	Fernando de Noronha Standard Time	-2:00
America/North_Dakota/Beulah	CST	US	Central Standard Time	-6:00
America/North_Dakota/Beulah	CDT	US	Central Daylight Time	-5:00
America/North_Dakota/Center	CST	US	Central Standard Time	-6:00
America/North_Dakota/Center	CDT	US	Central Daylight Time	-5:00
America/North_Dakota/ New_Salem	CST	US	Central Standard Time	-6:00
America/North_Dakota/ New_Salem	CDT	US	Central Daylight Time	-5:00
America/Ojinaga	MST	MX	Mountain Standard Time	-7:00
America/Ojinaga	MDT	MX	Mountain Daylight Time	-6:00
America/Panama	EST	PA	Eastern Standard Time	-5:00
America/Pangnirtung	EST	CA	Eastern Standard Time	-5:00
America/Pangnirtung	EDT	CA	Eastern Daylight Time	-4:00
America/Paramaribo	SRT	SR	Suriname Time	-3:00
America/Phoenix	MST	US	Mountain Standard Time	-7:00
America/Port-au-Prince	EST	HT	Eastern Standard Time	-5:00
America/Port-au-Prince	EDT	HT	Eastern Daylight Time	-4:00
America/Port_of_Spain	AST	TT	Atlantic Standard Time	-4:00
America/Porto_Acre	ACT	BR	Amazon Standard Time	-5:00
America/Porto_Velho	AMT	BR	Amazon Standard Time	-4:00
America/Puerto_Rico	AST	PR	Atlantic Standard Time	-4:00
America/Rainy_River	CST	CA	Central Standard Time	-6:00
America/Rainy_River	CDT	CA	Central Daylight Time	-5:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Rankin_Inlet	CST	CA	Central Standard Time	-6:00
America/Rankin_Inlet	CDT	CA	Central Daylight Time	-5:00
America/Recife	BRT	BR	Brasilia Standard Time	-3:00
America/Regina	CST	CA	Central Standard Time	-6:00
America/Resolute	CST	CA	Central Standard Time	-6:00
America/Resolute	CDT	CA	Central Daylight Time	-5:00
America/Rio_Branco	ACT	BR	Amazon Standard Time	-5:00
America/Rosario	ART	AR	Argentina Standard Time	-3:00
America/Santa_Isabel	PST	MX	Pacific Standard Time	-8:00
America/Santa_Isabel	PDT	MX	Pacific Daylight Time	-7:00
America/Santarem	BRT	BR	Brasilia Standard Time	-3:00
America/Santiago	CLT	CL	Chile Standard Time	-4:00
America/Santiago	CLST	CL	Chile Summer Time	-3:00
America/Santo_Domingo	AST	DO	Atlantic Standard Time	-4:00
America/Sao_Paulo	BRT	BR	Brasilia Standard Time	-3:00
America/Sao_Paulo	BRST	BR	Brasilia Summer Time	-2:00
America/Scoresbysund	EGT	GL	East Greenland Standard Time	-1:00
America/Scoresbysund	EGST	GL	East Greenland Summer Time	0:00
America/Shiprock	MST	US	Mountain Standard Time	-7:00
America/Shiprock	MDT	US	Mountain Daylight Time	-6:00
America/Sitka	AKST	US	Alaska Standard Time	-9:00
America/Sitka	AKDT	US	Alaska Daylight Time	-8:00
America/St_Barthelemy	AST	BL	Atlantic Standard Time	-4:00
America/St_Johns	NST	CA	Newfoundland Standard Time	-1:27
America/St_Johns	NDT	CA	Newfoundland Daylight Time	-1:27
America/St_Kitts	AST	KN	Atlantic Standard Time	-4:00
America/St_Lucia	AST	LC	Atlantic Standard Time	-4:00
America/St_Thomas	AST	VI	Atlantic Standard Time	-4:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/St_Vincent	AST	VC	Atlantic Standard Time	-4:00
America/Swift_Current	CST	CA	Central Standard Time	-6:00
America/Tegucigalpa	CST	HN	Central Standard Time	-6:00
America/Thule	AST	GL	Atlantic Standard Time	-4:00
America/Thule	ADT	GL	Atlantic Daylight Time	-3:00
America/Thunder_Bay	EST	CA	Eastern Standard Time	-5:00
America/Thunder_Bay	EDT	CA	Eastern Daylight Time	-4:00
America/Tijuana	PST	MX	Pacific Standard Time	-8:00
America/Tijuana	PDT	MX	Pacific Daylight Time	-7:00
America/Toronto	EST	CA	Eastern Standard Time	-5:00
America/Toronto	EDT	CA	Eastern Daylight Time	-4:00
America/Tortola	AST	VG	Atlantic Standard Time	-4:00
America/Vancouver	PST	CA	Pacific Standard Time	-8:00
America/Vancouver	PDT	CA	Pacific Daylight Time	-7:00
America/Virgin	AST	TT	Atlantic Standard Time	-4:00
America/Whitehorse	PST	CA	Pacific Standard Time	-8:00
America/Whitehorse	PDT	CA	Pacific Daylight Time	-7:00
America/Winnipeg	CST	CA	Central Standard Time	-6:00
America/Winnipeg	CDT	CA	Central Daylight Time	-5:00
America/Yakutat	AKST	US	Alaska Standard Time	-9:00
America/Yakutat	AKDT	US	Alaska Daylight Time	-8:00
America/Yellowknife	MST	CA	Mountain Standard Time	-7:00
America/Yellowknife	MDT	CA	Mountain Daylight Time	-6:00

Area: Antarctica

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Antarctica/Casey	WST	AQ	Australian Western Standard Time	8:00
Antarctica/Davis	DAVT	AQ	Davis Time	7:00
Antarctica/DumontDUrville	DDUT	AQ	Dumont-d\u2019Urville Time	10:00
Antarctica/Macquarie	MIST	AU	Macquarie Island Time	11:00
Antarctica/Mawson	MAWT	AQ	Mawson Time	5:00
Antarctica/McMurdo	NZST	AQ	New Zealand Standard Time	12:00
Antarctica/McMurdo	NZDT	AQ	New Zealand Daylight Time	13:00
Antarctica/Palmer	CLT	AQ	Chile Standard Time	-4:00
Antarctica/Palmer	CLST	AQ	Chile Summer Time	-3:00
Antarctica/Rothera	ROTT	AQ	Rothera Time	-3:00
Antarctica/South_Pole	NZST	NZ	New Zealand Standard Time	12:00
Antarctica/South_Pole	NZDT	NZ	New Zealand Daylight Time	13:00
Antarctica/Syowa	SYOT	AQ	Syowa Time	3:00
Antarctica/Vostok	VOST	AQ	Vostok Time	6:00

Area: Asia

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Aden	AST	YE	Arabian Standard Time	3:00
Asia/Almaty	ALMT	KZ	East Kazakhstan Time	6:00
Asia/Amman	EET	JO	Eastern European Standard Time	2:00
Asia/Amman	EEST	JO	Eastern European Summer Time	3:00
Asia/Anadyr	ANAT	RU	Magadan Standard Time	12:00
Asia/Aqtau	AQTT	KZ	West Kazakhstan Time	5:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Aqtobe	AQTT	KZ	West Kazakhstan Time	5:00
Asia/Ashgabat	TMT	TM	Turkmenistan Standard Time	5:00
Asia/Ashkhabad	TMT	TM	Turkmenistan Standard Time	5:00
Asia/Baghdad	AST	IQ	Arabian Standard Time	3:00
Asia/Bahrain	AST	BH	Arabian Standard Time	3:00
Asia/Baku	AZT	AZ	Azerbaijan Standard Time	4:00
Asia/Baku	AZST	AZ	Azerbaijan Summer Time	5:00
Asia/Bangkok	ICT	TH	Indochina Time	7:00
Asia/Beijing	CST	CN	China Standard Time	8:00
Asia/Beirut	EET	LB	Eastern European Standard Time	2:00
Asia/Beirut	EEST	LB	Eastern European Summer Time	3:00
Asia/Bishkek	KGT	KG	Kyrgyzstan Time	6:00
Asia/Brunei	BNT	BN	Brunei Darussalam Time	8:00
Asia/Calcutta	IST	IN	India Standard Time	5:30
Asia/Choibalsan	CHOT	MN	Choibalsan Standard Time	8:00
Asia/Chongqing	CST	CN	China Standard Time	8:00
Asia/Chungking	CST	CN	China Standard Time	8:00
Asia/Colombo	IST	LK	India Standard Time	5:30
Asia/Dacca	BDT	BD	Bangladesh Standard Time	6:00
Asia/Damascus	EET	SY	Eastern European Standard Time	2:00
Asia/Damascus	EEST	SY	Eastern European Summer Time	3:00
Asia/Dhaka	BDT	BD	Bangladesh Standard Time	6:00
Asia/Dili	TLT	TL	East Timor Time	9:00
Asia/Dubai	GST	AE	Gulf Standard Time	4:00
Asia/Dushanbe	TJT	TJ	Tajikistan Time	5:00
Asia/Gaza	EET	PS	Eastern European Standard Time	2:00
Asia/Gaza	EEST	PS	Eastern European Summer Time	3:00
Asia/Harbin	CST	CN	China Standard Time	8:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Hebron	EET	PS	Eastern European Standard Time	2:00
Asia/Hebron	EEST	PS	Eastern European Summer Time	3:00
Asia/Ho_Chi_Minh	ICT	VN	Indochina Time	7:00
Asia/Hong_Kong	HKT	HK	Hong Kong Standard Time	8:00
Asia/Hovd	HOVT	MN	Hovd Standard Time	7:00
Asia/Irkutsk	IRKT	RU	Irkutsk Standard Time	9:00
Asia/Istanbul	EET	TR	Eastern European Standard Time	2:00
Asia/Istanbul	EEST	TR	Eastern European Summer Time	3:00
Asia/Jakarta	WIB	ID	Western Indonesia Time	7:00
Asia/Jayapura	WIT	ID	Eastern Indonesia Time	9:00
Asia/Jerusalem	IST	IL	Israel Standard Time	2:00
Asia/Jerusalem	IDT	IL	Israel Daylight Time	3:00
Asia/Kabul	AFT	AF	Afghanistan Time	4:30
Asia/Kamchatka	PETT	RU	Magadan Standard Time	12:00
Asia/Karachi	PKT	PK	Pakistan Standard Time	5:00
Asia/Kashgar	CST	CN	China Standard Time	8:00
Asia/Kathmandu	NPT	NP	Nepal Time	5:45
Asia/Katmandu	NPT	NP	Nepal Time	5:45
Asia/Khandyga	YAKT	RU	Yakutsk Standard Time	10:00
Asia/Kolkata	IST	IN	India Standard Time	5:30
Asia/Krasnoyarsk	KRAT	RU	Krasnoyarsk Standard Time	8:00
Asia/Kuala_Lumpur	MYT	MY	Malaysia Time	8:00
Asia/Kuching	MYT	MY	Malaysia Time	8:00
Asia/Kuwait	AST	KW	Arabian Standard Time	3:00
Asia/Macao	CST	MO	China Standard Time	8:00
Asia/Macau	CST	MO	China Standard Time	8:00
Asia/Magadan	MAGT	RU	Magadan Standard Time	12:00
Asia/Makassar	WITA	ID	Central Indonesia Time	8:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Manila	PHT	PH	Philippine Standard Time	8:00
Asia/Muscat	GST	OM	Gulf Standard Time	4:00
Asia/Nicosia	EET	CY	Eastern European Standard Time	2:00
Asia/Nicosia	EEST	CY	Eastern European Summer Time	3:00
Asia/Novokuznetsk	NOVT	RU	Novosibirsk Standard Time	7:00
Asia/Novosibirsk	NOVT	RU	Novosibirsk Standard Time	7:00
Asia/Omsk	OMST	RU	Omsk Standard Time	7:00
Asia/Oral	ORAT	KZ	West Kazakhstan Time	5:00
Asia/Osaka	JST	JP	Japan Standard Time	9:00
Asia/Phnom_Penh	ICT	KH	Indochina Time	7:00
Asia/Pontianak	WIB	ID	Western Indonesia Time	7:00
Asia/Pyongyang	KST	KP	Korean Standard Time	9:00
Asia/Qatar	AST	QA	Arabian Standard Time	3:00
Asia/Qyzylorda	QYZT	KZ	East Kazakhstan Time	6:00
Asia/Rangoon	MMT	MM	Myanmar Time	6:30
Asia/Riyadh	AST	SA	Arabian Standard Time	3:00
Asia/Saigon	ICT	VN	Indochina Time	7:00
Asia/Sakhalin	SAKT	RU	Sakhalin Standard Time	11:00
Asia/Samarkand	UZT	UZ	Uzbekistan Standard Time	5:00
Asia/Sapporo	JST	JP	Japan Standard Time	9:00
Asia/Seoul	KST	KR	Korean Standard Time	9:00
Asia/Shanghai	CST	CN	China Standard Time	8:00
Asia/Singapore	SGT	SG	Singapore Standard Time	8:00
Asia/Taipei	CST	TW	Taipei Standard Time	8:00
Asia/Tashkent	UZT	UZ	Uzbekistan Standard Time	5:00
Asia/Tbilisi	GET	GE	Georgia Standard Time	4:00
Asia/Tehran	IRST	IR	Iran Standard Time	3:30
Asia/Tel_Aviv	IST	IL	Israel Standard Time	2:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Tel_Aviv	IDT	IL	Israel Daylight Time	3:00
Asia/Thimbu	BTT	BT	Bhutan Time	6:00
Asia/Thimphu	BTT	BT	Bhutan Time	6:00
Asia/Tokyo	JST	JP	Japan Standard Time	9:00
Asia/Ujung_Pandang	WITA	ID	Central Indonesia Time	8:00
Asia/Ulaanbaatar	ULAT	MN	Ulan Bator Standard Time	8:00
Asia/Ulan_Bator	ULAT	MN	Ulan Bator Standard Time	8:00
Asia/Urumqi	CST	CN	China Standard Time	8:00
Asia/Ust-Nera	VLAT	RU	Vladivostok Standard Time	11:00
Asia/Vientiane	ICT	LA	Indochina Time	7:00
Asia/Vladivostok	VLAT	RU	Vladivostok Standard Time	11:00
Asia/Yakutsk	YAKT	RU	Yakutsk Standard Time	10:00
Asia/Yekaterinburg	YEKT	RU	Yekaterinburg Standard Time	6:00
Asia/Yerevan	AMT	AM	Armenia Standard Time	4:00

Area: Atlantic

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Atlantic/Azores	AZOT	PT	Azores Standard Time	-1:00
Atlantic/Azores	AZOST	PT	Azores Summer Time	0:00
Atlantic/Bermuda	AST	BM	Atlantic Standard Time	-4:00
Atlantic/Bermuda	ADT	BM	Atlantic Daylight Time	-3:00
Atlantic/Canary	WET	ES	Western European Standard Time	0:00
Atlantic/Canary	WEST	ES	Western European Summer Time	1:00
Atlantic/Cape_Verde	CVT	CV	Cape Verde Standard Time	-1:00
Atlantic/Faeroe	WET	FO	Western European Standard Time	0:00
Atlantic/Faeroe	WEST	FO	Western European Summer Time	1:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Atlantic/Faroe	WET	FO	Western European Standard Time	0:00
Atlantic/Faroe	WEST	FO	Western European Summer Time	1:00
Atlantic/Jan_Mayen	CET	NO	Central European Standard Time	1:00
Atlantic/Jan_Mayen	CEST	NO	Central European Summer Time	2:00
Atlantic/Madeira	WET	PT	Western European Standard Time	0:00
Atlantic/Madeira	WEST	PT	Western European Summer Time	1:00
Atlantic/Reykjavik	GMT	IS	Greenwich Mean Time	0:00
Atlantic/South_Georgia	GST	GS	South Georgia Time	-2:00
Atlantic/St_Helena	GMT	SH	Greenwich Mean Time	0:00
Atlantic/Stanley	FKST	FK	Falkland Islands Standard Time	-3:00

Area: Australia

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Australia/ACT	EST	AU	Australian Eastern Standard Time	10:00
Australia/ACT	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/Adelaide	CST	AU	Australian Central Standard Time	9:30
Australia/Adelaide	CDT	AU	Australian Central Daylight Time	10:30
Australia/Brisbane	EST	AU	Australian Eastern Standard Time	10:00
Australia/Broken_Hill	CST	AU	Australian Central Standard Time	9:30
Australia/Broken_Hill	CDT	AU	Australian Central Daylight Time	10:30
Australia/Canberra	EST	AU	Australian Eastern Standard Time	10:00
Australia/Canberra	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/Currie	EST	AU	Australian Eastern Standard Time	10:00
Australia/Currie	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/Darwin	CST	AU	Australian Central Standard Time	9:30

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Australia/Eucla	CWST	AU	Australian Central Western Standard Time	8:45
Australia/Hobart	EST	AU	Australian Eastern Standard Time	10:00
Australia/Hobart	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/LHI	LHST	AU	Lord Howe Standard Time	10:30
Australia/LHI	LHDT	AU	Lord Howe Daylight Time	11:00
Australia/Lindeman	EST	AU	Australian Eastern Standard Time	10:00
Australia/Lord_Howe	LHST	AU	Lord Howe Standard Time	10:30
Australia/Lord_Howe	LHDT	AU	Lord Howe Daylight Time	11:00
Australia/Melbourne	EST	AU	Australian Eastern Standard Time	10:00
Australia/Melbourne	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/NSW	EST	AU	Australian Eastern Standard Time	10:00
Australia/NSW	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/North	CST	AU	Australian Central Standard Time	9:30
Australia/Perth	WST	AU	Australian Western Standard Time	8:00
Australia/Queensland	EST	AU	Australian Eastern Standard Time	10:00
Australia/South	CST	AU	Australian Central Standard Time	9:30
Australia/South	CDT	AU	Australian Central Daylight Time	10:30
Australia/Sydney	EST	AU	Australian Eastern Standard Time	10:00
Australia/Sydney	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/Tasmania	EST	AU	Australian Eastern Standard Time	10:00
Australia/Tasmania	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/Victoria	EST	AU	Australian Eastern Standard Time	10:00
Australia/Victoria	EDT	AU	Australian Eastern Daylight Time	11:00
Australia/West	WST	AU	Australian Western Standard Time	8:00
Australia/Yancowinna	CST	AU	Australian Central Standard Time	9:30
Australia/Yancowinna	CDT	AU	Australian Central Daylight Time	10:30

Area: Miscellaneous

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Arctic/Longyearbyen	CET	SJ	Central European Standard Time	1:00
Arctic/Longyearbyen	CEST	SJ	Central European Summer Time	2:00
Brazil/Acre	ACT	BR	Amazon Standard Time	-5:00
Brazil/DeNoronha	FNT	BR	Fernando de Noronha Standard Time	-2:00
Brazil/East	BRT	BR	Brasilia Standard Time	-3:00
Brazil/East	BRST	BR	Brasilia Summer Time	-2:00
Brazil/West	AMT	BR	Amazon Standard Time	-4:00
CST6CDT	CST		Central Standard Time	-6:00
CST6CDT	CDT		Central Daylight Time	-5:00
Canada/Atlantic	AST	CA	Atlantic Standard Time	-4:00
Canada/Atlantic	ADT	CA	Atlantic Daylight Time	-3:00
Canada/Central	CST	CA	Central Standard Time	-6:00
Canada/Central	CDT	CA	Central Daylight Time	-5:00
Canada/East-Saskatchewan	CST	CA	Central Standard Time	-6:00
Canada/Eastern	EST	CA	Eastern Standard Time	-5:00
Canada/Eastern	EDT	CA	Eastern Daylight Time	-4:00
Canada/Mountain	MST	CA	Mountain Standard Time	-7:00
Canada/Mountain	MDT	CA	Mountain Daylight Time	-6:00
Canada/Newfoundland	NST	CA	Newfoundland Standard Time	-1:27
Canada/Newfoundland	NDT	CA	Newfoundland Daylight Time	-1:27
Canada/Pacific	PST	CA	Pacific Standard Time	-8:00
Canada/Pacific	PDT	CA	Pacific Daylight Time	-7:00
Canada/Saskatchewan	CST	CA	Central Standard Time	-6:00
Canada/Yukon	PST	CA	Pacific Standard Time	-8:00
Canada/Yukon	PDT	CA	Pacific Daylight Time	-7:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Chile/Continental	CLT	CL	Chile Standard Time	-4:00
Chile/Continental	CLST	CL	Chile Summer Time	-3:00
Chile/EasterIsland	EAST	CL	Easter Island Standard Time	-6:00
Chile/EasterIsland	EASST	CL	Easter Island Summer Time	-5:00
Cuba	CST	CU	Cuba Standard Time	-5:00
Cuba	CDT	CU	Cuba Daylight Time	-4:00
EST5EDT	EST		Eastern Standard Time	-5:00
EST5EDT	EDT		Eastern Daylight Time	-4:00
Egypt	EET EET	EG EG	Eastern European Standard Time Eastern European Summer Time	3:00 0.00
Eire	GMT	IE	Greenwich Mean Time	0:00
Eire	IST	IE		1:00
Etc/GMT	GMT		Greenwich Mean Time	0:00
Etc/GMT+0	GMT		Greenwich Mean Time	0:00
Etc/GMT+1	GMT-01		1 hours behind GMT	-1:00
Etc/GMT+10	GMT-10		10 hours behind GMT	-10:00
Etc/GMT+11	GMT-11		11 hours behind GMT	-11:00
Etc/GMT+12	GMT-12		12 hours behind GMT	-12:00
Etc/GMT+2	GMT-02		2 hours behind GMT	-2:00
Etc/GMT+3	GMT-03		3 hours behind GMT	-3:00
Etc/GMT+4	GMT-04		4 hours behind GMT	-4:00
Etc/GMT+5	GMT-05		5 hours behind GMT	-5:00
Etc/GMT+6	GMT-06		6 hours behind GMT	-6:00
Etc/GMT+7	GMT-07		7 hours behind GMT	-7:00
Etc/GMT+8	GMT-08		8 hours behind GMT	-8:00
Etc/GMT+9	GMT-09		9 hours behind GMT	-9:00
Etc/GMT-0	GMT		Greenwich Mean Time	0:00
Etc/GMT-1	GMT+01		1 hours ahead of GMT	1:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Etc/GMT-10	GMT+10		10 hours ahead of GMT	10:00
Etc/GMT-11	GMT+11		11 hours ahead of GMT	11:00
Etc/GMT-12	GMT+12		12 hours ahead of GMT	12:00
Etc/GMT-13	GMT+13		13 hours ahead of GMT	13:00
Etc/GMT-14	GMT+14		14 hours ahead of GMT	14:00
Etc/GMT-2	GMT+02		2 hours ahead of GMT	2:00
Etc/GMT-3	GMT+03		3 hours ahead of GMT	3:00
Etc/GMT-4	GMT+04		4 hours ahead of GMT	4:00
Etc/GMT-5	GMT+05		5 hours ahead of GMT	5:00
Etc/GMT-6	GMT+06		6 hours ahead of GMT	6:00
Etc/GMT-7	GMT+07		7 hours ahead of GMT	7:00
Etc/GMT-8	GMT+08		8 hours ahead of GMT	8:00
Etc/GMT-9	GMT+09		9 hours ahead of GMT	9:00
Etc/GMT0	GMT		Greenwich Mean Time	0:00
Etc/Greenwich	GMT		Greenwich Mean Time	0:00
Etc/UCT	UCT		Greenwich Mean Time	0:00
Etc/UTC	UTC		Greenwich Mean Time	0:00
Etc/Universal	UTC		Greenwich Mean Time	0:00
Etc/Zulu	UTC		Greenwich Mean Time	0:00
GB	GMT	GB	Greenwich Mean Time	0:00
GB	BST	GB		1:00
GB-Eire	GMT	GB	Greenwich Mean Time	0:00
GB-Eire	BST	GB		1:00
GMT	GMT		Greenwich Mean Time	0:00
GMT+0	GMT		Greenwich Mean Time	0:00
GMT-0	GMT		Greenwich Mean Time	0:00
GMT0	GMT		Greenwich Mean Time	0:00
Greenwich	GMT		Greenwich Mean Time	0:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Hongkong	HKT	HK	Hong Kong Standard Time	8:00
Iceland	GMT	IS	Greenwich Mean Time	0:00
Indian/Antananarivo	EAT	MG	East Africa Time	3:00
Indian/Chagos	IOT	IO	Indian Ocean Time	6:00
Indian/Christmas	CXT	CX	Christmas Island Time	7:00
Indian/Cocos	CCT	CC	Cocos Islands Time	6:30
Indian/Comoro	EAT	KM	East Africa Time	3:00
Indian/Kerguelen	TFT	TF	French Southern and Antarctic Time	5:00
Indian/Mahe	SCT	SC	Seychelles Time	4:00
Indian/Maldives	MVT	MV	Maldives Time	5:00
Indian/Mauritius	MUT	MU	Mauritius Standard Time	4:00
Indian/Mayotte	EAT	YT	East Africa Time	3:00
Indian/Reunion	RET	RE	Reunion Time	4:00
Iran	IRST	IR	Iran Standard Time	3:30
Israel	IST	IL	Israel Standard Time	2:00
Israel	IDT	IL	Israel Daylight Time	3:00
Jamaica	EST	JM	Eastern Standard Time	-5:00
Japan	JST	JP	Japan Standard Time	9:00
Kwajalein	MHT	MH	Marshall Islands Time	12:00
Libya	EET	LY	Central European Standard Time	2:00
MST7MDT	MST		Mountain Standard Time	-7:00
MST7MDT	MDT		Mountain Daylight Time	-6:00
Mexico/BajaNorte	PST	MX	Pacific Standard Time	-8:00
Mexico/BajaNorte	PDT	MX	Pacific Daylight Time	-7:00
Mexico/BajaSur	MST	MX	Mountain Standard Time	-7:00
Mexico/BajaSur	MDT	MX	Mountain Daylight Time	-6:00
Mexico/General	CST	MX	Central Standard Time	-6:00
Mexico/General	CDT	MX	Central Daylight Time	-5:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
NZ	NZST	NZ	New Zealand Standard Time	12:00
NZ	NZDT	NZ	New Zealand Daylight Time	13:00
NZ-CHAT	CHAST	NZ	Chatham Standard Time	12:45
NZ-CHAT	CHADT	NZ	Chatham Daylight Time	13:45
Navajo	MST	US	Mountain Standard Time	-7:00
Navajo	MDT	US	Mountain Daylight Time	-6:00
PRC	CST	CN	China Standard Time	8:00
PST8PDT	PST		Pacific Standard Time	-8:00
PST8PDT	PDT		Pacific Daylight Time	-7:00
Poland	CET	PL	Central European Standard Time	1:00
Poland	CEST	PL	Central European Summer Time	2:00
Portugal	WET	PT	Western European Standard Time	0:00
Portugal	WEST	PT	Western European Summer Time	1:00
ROC	CST	TW	Taipei Standard Time	8:00
ROK	KST	KR	Korean Standard Time	9:00
Singapore	SGT	SG	Singapore Standard Time	8:00
Turkey	EET	TR	Eastern European Standard Time	2:00
Turkey	EEST	TR	Eastern European Summer Time	3:00
UCT	UCT		Greenwich Mean Time	0:00
US/Alaska	AKST	US	Alaska Standard Time	-9:00
US/Alaska	AKDT	US	Alaska Daylight Time	-8:00
US/Aleutian	HAST	US	Hawaii-Aleutian Standard Time	-10:00
US/Aleutian	HADT	US	Hawaii-Aleutian Daylight Time	-9:00
US/Arizona	MST	US	Mountain Standard Time	-7:00
US/Central	CST	US	Central Standard Time	-6:00
US/Central	CDT	US	Central Daylight Time	-5:00
US/East-Indiana	EST	US	Eastern Standard Time	-5:00
US/East-Indiana	EDT	US	Eastern Daylight Time	-4:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
US/Eastern	EST	US	Eastern Standard Time	-5:00
US/Eastern	EDT	US	Eastern Daylight Time	-4:00
US/Hawaii	HST	US	Hawaii-Aleutian Standard Time	-10:00
US/Indiana-Starke	CST	US	Central Standard Time	-6:00
US/Indiana-Starke	CDT	US	Central Daylight Time	-5:00
US/Michigan	EST	US	Eastern Standard Time	-5:00
US/Michigan	EDT	US	Eastern Daylight Time	-4:00
US/Mountain	MST	US	Mountain Standard Time	-7:00
US/Mountain	MDT	US	Mountain Daylight Time	-6:00
US/Pacific	PST	US	Pacific Standard Time	-8:00
US/Pacific	PDT	US	Pacific Daylight Time	-7:00
US/Pacific-New	PST	US	Pacific Standard Time	-8:00
US/Pacific-New	PDT	US	Pacific Daylight Time	-7:00
US/Samoa	SST	AS	Samoa Standard Time	-11:00
UTC	UTC		Greenwich Mean Time	0:00
Universal	UTC		Greenwich Mean Time	0:00
W-SU	MSK	RU	Moscow Standard Time	4:00
Zulu	UTC		Greenwich Mean Time	0:00

Area: Europe

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Amsterdam	CET	NL	Central European Standard Time	1:00
Europe/Amsterdam	CEST	NL	Central European Summer Time	2:00
Europe/Andorra	CET	AD	Central European Standard Time	1:00
Europe/Andorra	CEST	AD	Central European Summer Time	2:00
Europe/Athens	EET	GR	Eastern European Standard Time	2:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Athens	EEST	GR	Eastern European Summer Time	3:00
Europe/Belfast	GMT	GB	Greenwich Mean Time	0:00
Europe/Belfast	BST	GB		1:00
Europe/Belgrade	CET	RS	Central European Standard Time	1:00
Europe/Belgrade	CEST	RS	Central European Summer Time	2:00
Europe/Berlin	CET	DE	Central European Standard Time	1:00
Europe/Berlin	CEST	DE	Central European Summer Time	2:00
Europe/Bratislava	CET	SK	Central European Standard Time	1:00
Europe/Bratislava	CEST	SK	Central European Summer Time	2:00
Europe/Brussels	CET	BE	Central European Standard Time	1:00
Europe/Brussels	CEST	BE	Central European Summer Time	2:00
Europe/Bucharest	EET	RO	Eastern European Standard Time	2:00
Europe/Bucharest	EEST	RO	Eastern European Summer Time	3:00
Europe/Budapest	CET	HU	Central European Standard Time	1:00
Europe/Budapest	CEST	HU	Central European Summer Time	2:00
Europe/Busingen	CET	DE	Central European Standard Time	1:00
Europe/Busingen	CEST	DE	Central European Summer Time	2:00
Europe/Chisinau	EET	MD	Eastern European Standard Time	2:00
Europe/Chisinau	EEST	MD	Eastern European Summer Time	3:00
Europe/Copenhagen	CET	DK	Central European Standard Time	1:00
Europe/Copenhagen	CEST	DK	Central European Summer Time	2:00
Europe/Dublin	GMT	IE	Greenwich Mean Time	0:00
Europe/Dublin	IST	IE		1:00
Europe/Gibraltar	CET	GI	Central European Standard Time	1:00
Europe/Gibraltar	CEST	GI	Central European Summer Time	2:00
Europe/Guernsey	GMT	GG	Greenwich Mean Time	0:00
Europe/Guernsey	BST	GG		1:00
Europe/Helsinki	EET	FI	Eastern European Standard Time	2:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Helsinki	EEST	FI	Eastern European Summer Time	3:00
Europe/Isle_of_Man	GMT	IM	Greenwich Mean Time	0:00
Europe/Isle_of_Man	BST	IM		1:00
Europe/Istanbul	EET	TR	Eastern European Standard Time	2:00
Europe/Istanbul	EEST	TR	Eastern European Summer Time	3:00
Europe/Jersey	GMT	JE	Greenwich Mean Time	0:00
Europe/Jersey	BST	JE		1:00
Europe/Kaliningrad	FET	RU	Eastern European Standard Time	3:00
Europe/Kiev	EET	UA	Eastern European Standard Time	2:00
Europe/Kiev	EEST	UA	Eastern European Summer Time	3:00
Europe/Lisbon	WET	PT	Western European Standard Time	0:00
Europe/Lisbon	WEST	PT	Western European Summer Time	1:00
Europe/Ljubljana	CET	SI	Central European Standard Time	1:00
Europe/Ljubljana	CEST	SI	Central European Summer Time	2:00
Europe/London	GMT	GB	Greenwich Mean Time	0:00
Europe/London	BST	GB		1:00
Europe/Luxembourg	CET	LU	Central European Standard Time	1:00
Europe/Luxembourg	CEST	LU	Central European Summer Time	2:00
Europe/Madrid	CET	ES	Central European Standard Time	1:00
Europe/Madrid	CEST	ES	Central European Summer Time	2:00
Europe/Malta	CET	MT	Central European Standard Time	1:00
Europe/Malta	CEST	MT	Central European Summer Time	2:00
Europe/Mariehamn	EET	AX	Eastern European Standard Time	2:00
Europe/Mariehamn	EEST	AX	Eastern European Summer Time	3:00
Europe/Minsk	FET	BY	Eastern European Standard Time	3:00
Europe/Monaco	CET	MC	Central European Standard Time	1:00
Europe/Monaco	CEST	MC	Central European Summer Time	2:00
Europe/Moscow	MSK	RU	Moscow Standard Time	4:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Nicosia	EET	CY	Eastern European Standard Time	2:00
Europe/Nicosia	EEST	CY	Eastern European Summer Time	3:00
Europe/Oslo	CET	NO	Central European Standard Time	1:00
Europe/Oslo	CEST	NO	Central European Summer Time	2:00
Europe/Paris	CET	FR	Central European Standard Time	1:00
Europe/Paris	CEST	FR	Central European Summer Time	2:00
Europe/Podgorica	CET	ME	Central European Standard Time	1:00
Europe/Podgorica	CEST	ME	Central European Summer Time	2:00
Europe/Prague	CET	CZ	Central European Standard Time	1:00
Europe/Prague	CEST	CZ	Central European Summer Time	2:00
Europe/Riga	EET	LV	Eastern European Standard Time	2:00
Europe/Riga	EEST	LV	Eastern European Summer Time	3:00
Europe/Rome	CET	IT	Central European Standard Time	1:00
Europe/Rome	CEST	IT	Central European Summer Time	2:00
Europe/Samara	SAMT	RU	Moscow Standard Time	4:00
Europe/San_Marino	CET	SM	Central European Standard Time	1:00
Europe/San_Marino	CEST	SM	Central European Summer Time	2:00
Europe/Sarajevo	CET	BA	Central European Standard Time	1:00
Europe/Sarajevo	CEST	BA	Central European Summer Time	2:00
Europe/Simferopol	MSK	UA	Eastern European Standard Time	4:00
Europe/Skopje	CET	MK	Central European Standard Time	1:00
Europe/Skopje	CEST	MK	Central European Summer Time	2:00
Europe/Sofia	EET	BG	Eastern European Standard Time	2:00
Europe/Sofia	EEST	BG	Eastern European Summer Time	3:00
Europe/Stockholm	CET	SE	Central European Standard Time	1:00
Europe/Stockholm	CEST	SE	Central European Summer Time	2:00
Europe/Tallinn	EET	EE	Eastern European Standard Time	2:00
Europe/Tallinn	EEST	EE	Eastern European Summer Time	3:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Tirane	CET	AL	Central European Standard Time	1:00
Europe/Tirane	CEST	AL	Central European Summer Time	2:00
Europe/Tiraspol	EET	MD	Eastern European Standard Time	2:00
Europe/Tiraspol	EEST	MD	Eastern European Summer Time	3:00
Europe/Uzhgorod	EET	UA	Eastern European Standard Time	2:00
Europe/Uzhgorod	EEST	UA	Eastern European Summer Time	3:00
Europe/Vaduz	CET	LI	Central European Standard Time	1:00
Europe/Vaduz	CEST	LI	Central European Summer Time	2:00
Europe/Vatican	CET	VA	Central European Standard Time	1:00
Europe/Vatican	CEST	VA	Central European Summer Time	2:00
Europe/Vienna	CET	AT	Central European Standard Time	1:00
Europe/Vienna	CEST	AT	Central European Summer Time	2:00
Europe/Vilnius	EET	LT	Eastern European Standard Time	2:00
Europe/Vilnius	EEST	LT	Eastern European Summer Time	3:00
Europe/Volgograd	VOLT	RU	Volgograd Standard Time	4:00
Europe/Warsaw	CET	PL	Central European Standard Time	1:00
Europe/Warsaw	CEST	PL	Central European Summer Time	2:00
Europe/Zagreb	CET	HR	Central European Standard Time	1:00
Europe/Zagreb	CEST	HR	Central European Summer Time	2:00
Europe/Zaporozhye	EET	UA	Eastern European Standard Time	2:00
Europe/Zaporozhye	EEST	UA	Eastern European Summer Time	3:00
Europe/Zurich	CET	CH	Central European Standard Time	1:00
Europe/Zurich	CEST	CH	Central European Summer Time	2:00

Area: Pacific

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Pacific/Apia	WST	WS	Samoa Standard Time	13:00
Pacific/Apia	WSDT	WS	Samoa Daylight Time	14:00
Pacific/Auckland	NZST	NZ	New Zealand Standard Time	12:00
Pacific/Auckland	NZDT	NZ	New Zealand Daylight Time	13:00
Pacific/Chatham	CHAST	NZ	Chatham Standard Time	12:45
Pacific/Chatham	CHADT	NZ	Chatham Daylight Time	13:45
Pacific/Chuuk	CHUT	FM	Chuuk Time	10:00
Pacific/Easter	EAST	CL	Easter Island Standard Time	-6:00
Pacific/Easter	EASST	CL	Easter Island Summer Time	-5:00
Pacific/Efate	VUT	VU	Vanuatu Standard Time	11:00
Pacific/Enderbury	PHOT	KI	Phoenix Islands Time	13:00
Pacific/Fakaofo	TKT	TK	Tokelau Time	13:00
Pacific/Fiji	FJT FJST	FJ FJ	Fiji Standard Time Fiji Summer Time	12:00 12:00
Pacific/Funafuti	TVT	TV	Tuvalu Time	12:00
Pacific/Galapagos	GALT	EC	Galapagos Time	-6:00
Pacific/Gambier	GAMT	PF	Gambier Time	-9:00
Pacific/Guadalcanal	SBT	SB	Solomon Islands Time	11:00
Pacific/Guam	CHST	GU	Chamorro Standard Time	10:00
Pacific/Honolulu	HST	US	Hawaii-Aleutian Standard Time	-10:00
Pacific/Johnston	HST	UM	Hawaii-Aleutian Standard Time	-10:00
Pacific/Kiritimati	LINT	KI	Line Islands Time	14:00
Pacific/Kosrae	KOST	FM	Kosrae Time	11:00
Pacific/Kwajalein	MHT	MH	Marshall Islands Time	12:00
Pacific/Majuro	MHT	MH	Marshall Islands Time	12:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Pacific/Marquesas	MART	PF	Marquesas Time	-6:27
Pacific/Midway	SST	UM	Samoa Standard Time	-11:00
Pacific/Nauru	NRT	NR	Nauru Time	12:00
Pacific/Niue	NUT	NU	Niue Time	-11:00
Pacific/Norfolk	NFT	NF	Norfolk Island Time	11:30
Pacific/Noumea	NCT	NC	New Caledonia Standard Time	11:00
Pacific/Pago_Pago	SST	AS	Samoa Standard Time	-11:00
Pacific/Palau	PWT	PW	Palau Time	9:00
Pacific/Pitcairn	PST	PN	Pitcairn Time	-8:00
Pacific/Pohnpei	PONT	FM	Ponape Time	11:00
Pacific/Ponape	PONT	FM	Ponape Time	11:00
Pacific/Port_Moresby	PGT	PG	Papua New Guinea Time	10:00
Pacific/Rarotonga	CKT	CK	Cook Islands Standard Time	-10:00
Pacific/Saipan	CHST	MP	Chamorro Standard Time	10:00
Pacific/Samoa	SST	AS	Samoa Standard Time	-11:00
Pacific/Tahiti	TAHT	PF	Tahiti Time	-10:00
Pacific/Tarawa	GILT	KI	Gilbert Islands Time	12:00
Pacific/Tongatapu	TOT	TO	Tonga Standard Time	13:00
Pacific/Truk	CHUT	FM	Chuuk Time	10:00
Pacific/Wake	WAKT	UM	Wake Island Time	12:00
Pacific/Wallis	WFT	WF	Wallis and Futuna Time	12:00
Pacific/Yap	CHUT	FM	Chuuk Time	10:00

Recommended Reading

Here is the recommended reading list for this title:

- *SAS Viya Visual Data Management and Utility Procedures Guide*
- *SAS Viya Data Set Options: Reference*
- *SAS Viya Formats and Informats: Reference*
- *SAS Viya Functions and CALL Routines: Reference*
- *SAS Viya System Options: Reference*
- *SAS Viya Statements: Reference*
- *SAS Encoding - Understanding the Details*

For a complete list of SAS publications, go to sas.com/store/books. If you have questions about which titles you need, please contact a SAS Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books

Glossary

accented character
a type of character that is modified by the addition of an accent mark that alters the pronunciation of the character. An example is "ñ", which results from combining the tilde (~) with the character "n".

American National Standards Institute (ANSI)
the organization that coordinates the development of voluntary consensus standards for products, services, processes, systems, and personnel in the United States. ANSI works with the International Organization for Standardization to establish global standards.

American Standard Code for Information Interchange
See ASCII.

ANSI
See American National Standards Institute.

ASCII (American Standard Code for Information Interchange)
a 7-bit encoding standard that provides a basic set of 128 characters, supporting a variety of computer systems. ASCII encodes the uppercase and lowercase letters of the English alphabet, punctuation marks, the digits 0-9, and control characters. This set of 128 characters is also included in most other encodings. *See also* EBCDIC, encoding.

bidirectional text
text in a writing system such as Arabic and Hebrew that generally runs from right to left, except for numbers and embedded text written in other languages that run from left to right.

BOM
See byte-order mark.

byte-order mark (BOM)
the Unicode character that indicates the byte order of the Unicode text that follows in the text file or stream. The BOM can also indicate which of several Unicode encodings the text stream is encoded as. In UTF-16, the code point of the byte-order mark is U+FEFF (hexadecimal).

CCS
See coded character set.

CEDA
See Cross-Environment Data Access.

character
the smallest component of a writing system that has a semantic value, such as the letters of an alphabet, digits, or ideographs.

character encoding
a mapping of an abstract character repertoire to a set of numeric values. Character encodings are used in computation, data storage, and transmission of textual data. A character encoding includes national characters, special characters, the digits 0-9, and control characters.

CJK
the languages of Chinese, Japanese, and Korean, referred to as a collective grouping because all use Chinese characters. The grouping is relevant to translation and information processing. Vietnamese is occasionally included in this grouping (CJKV) because of its use of Chinese characters historically.

code page
the representation of a character set that associates a hexadecimal value with each character. The term code page originated from IBMs EBCDIC-based mainframe systems, but many vendors use this term including Microsoft, SAP, and Oracle Corporation. Vendors often allocate their own code page number to a character encoding, even if it is better known by another name (for example, UTF-8 character encoding has code page numbers 1208 at IBM, 65001 at Microsoft, 4110 at SAP). *See also* encoding, code page.

coded character set (CCS)
a mapping of an abstract character repertoire to a set of numeric values. The ISO Latin-1 coded character set provides the Western European alphabet and symbols and their numeric representations. For example, the letter "Ä" is represented as C4 (hexadecimal).

collating sequence
a set of rules that determine how textual data is ordered and compared.

control character
a nonprinting character that is represented by a code point in a character set, and that does not itself represent a written symbol.

Cross-Environment Data Access (CEDA)
a feature of SAS software that enables a SAS data file that was created in a directory-based operating environment to be read by a SAS session in another directory-based environment. *See also* data representation.

data representation
the form in which data is stored in a particular operating environment. Different operating environments use different standards or conventions for storing floating-point numbers (for example, IEEE or IBM 390); for character encoding (ASCII or EBCDIC); for the ordering of bytes in memory (big Endian or little Endian); for word alignment (4-byte boundaries or 8-byte boundaries); and for data-type length (16-bit, 32-bit, or 64-bit).

DBCS
See double-byte character set.

double-byte character set (DBCS)
a character set that requires a variable-width encoding because many characters occupy two bytes of memory. The term DBCS, as traditionally applied to languages such as Japanese, Korean, and Chinese, is somewhat misleading because some DBCS characters actually require only one byte. *See also* single-byte character set, multi-byte character set.

EBCDIC (Extended Binary Coded Decimal Interchange Code)
a family of single-byte and multi-byte encodings for the representation of data on IBM mainframe and mid-range computers. *See also* ASCII, encoding.

encode
to represent data in a particular character encoding scheme. For example, in ASCII, the letter "A" is represented as 41 (hexadecimal).

encoding
a mapping of a coded character set to code values.

encoding method
the application of established industry rules to a coded character set to produce an encoded character scheme. Such rules prescribe the number of bits required for storing the numeric representation of a specific character and its code position in the encoding. ISO 2022 and UTF-8 are examples of encoding methods.

EUC
See Extended UNIX Code.

Extended Binary Coded Decimal Interchange Code
See EBCDIC.

Extended UNIX Code (EUC)
a multibyte encoding scheme used primarily to encode Chinese, Japanese, and Korean writing systems with a maximum of 94 characters in sequences of 7-bit codes.

graphic character
in ISO standards, an encoded character that is intended to be written, printed, or otherwise displayed in a form that can be read by humans. Graphic characters are associated with one or more glyphs.

Hangul
the name for the Korean alphabet, consisting of 24 consonant and vowel letters, arranged into blocks, one per syllable.

I18N
See internationalization.

ICU (International Components for Unicode)
the open-source project containing C/C++ and Java libraries that provide Unicode and globalization support for software applications.

input method
a type of character conversion that is designed for interactive data input, generally from a keyboard. The simplest input method works by mapping ASCII characters onto another alphabet. Another method is to use composition, converting sequences of characters into a single letter. Languages like Thai and Korean can use both.

International Components for Unicode
See ICU.

International Organization for Standardization (ISO)
an organization that promotes the development of standards, and sponsors related activities that foster the sharing of products, services, and information among nations.

internationalization (I18N)
the process of designing a software product without making assumptions that are based on a single language or locale, thereby facilitating localization. Internationalization ensures that international conventions (including rules for sorting strings and for formatting dates, times, numbers, and currencies) are supported. It also facilitates a consistent user experience across different language editions of a product. *See also* national language support.

ISO
See International Organization for Standardization.

ISO 646 family
a group of 7-bit encodings that are defined in the ISO 646 standard. The encoding contains both the 116 invariant ASCII code positions and the 12 variant code positions that can be replaced by national characters. For example, code position 23 (hexadecimal) is reserved for a variant character. This position is occupied by the number sign (#) in the US, and the symbol (£) in the UK. The national variants of ISO 646 are largely obsolete.

ISO 8859 family
a group of 16 8-bit encodings that are defined in the ISO 8859 standard. Each encoding contains both the 128 ASCII characters and the 128 extended characters, which are used in the language or languages that are supported by the encoding. For example, ISO 8859-1, also called Latin-1, is a commonly used encoding in the ISO 8859 family that contains the ASCII characters as well as characters used by Western European languages. *See also* International Organization for Standardization.

L10N
See localization.

locale
a setting that reflects the language, local conventions, and culture for a geographic region. Local conventions can include specific formatting rules for paper sizes, dates, times, and numbers, and a currency symbol for the country or region. Some examples of locale values are French_Canada, Portuguese_Brazil, and Chinese_Singapore.

localization (L10N)
the process of adapting software for a particular geocultural region (locale). Translation of the user interface, system messages, and documentation is a large part of the localization process. *See also* internationalization.

logogram
a visual symbol that represents a word or morpheme rather than a speech sound. An example of a logogram in the Chinese language is 山 for the word "mountain".

MBCS
See multi-byte character set.

multi-byte character set (MBCS)
a character set that requires a variable-width encoding because many characters occupy more than one byte of memory. DBCS and MBCS are sometimes used interchangeably, but MBCS is more accurate for describing the character sets of languages such as Japanese, Korean, and Chinese. *See also* single-byte character set, double-byte character set.

national character
a character that is specific to a language as it is written in a nation or group of nations. For example, the letter "n" with a tilde (ñ) is a Spanish national character.

national language support (NLS)
the set of features that enable a software product to function properly in every global market for which the product is targeted.

NLS
See national language support.

SBCS
See single-byte character set.

single-byte character set (SBCS)
a type of encoding for which each character is represented using one byte of computer memory. An example of a single-byte character set is Latin 1. *See also* double-byte character set, multi-byte character set.

special character
a type of character other than alphanumeric characters, the underscore (_), and the blank. An example is the asterisk (*).

transcoding
the process of converting the contents of a SAS file from one encoding to another encoding. Transcoding is necessary if the session encoding and the file encoding are different, such as when transferring data from a Latin 1 encoding under UNIX to a German EBCDIC encoding on an IBM mainframe. *See also* encoding, translation table.

translation table
a SAS catalog entry that is used to map data from one encoding to another encoding. SAS language elements that control locale values and encoding properties automatically invoke the appropriate translation table. Translation tables are specific to the operating environment. An example is a translation table that maps the Windows Latin 2 encoding to the ISO Latin 2 encoding. *See also* encoding, transcoding.

Unicode
a 16-bit encoding that is the industry standard for supporting the interchange, processing, and display of characters and symbols from most of the world's writing systems.

Index

C

D

E

F

L

M

N

O

P

R

S

T

U

V

W

X

Y