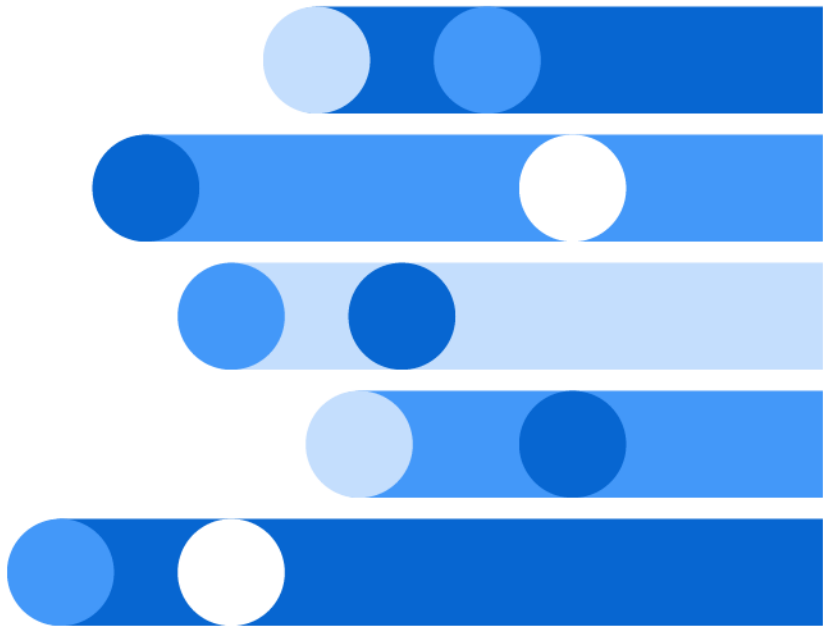
SAS® National Language Support (NLS): Reference Guide

2020.1 - 2026.09*



* This document might apply to additional versions of the software. Open this document in SAS Help Center and click on the version in the banner to see all available versions.

SAS® Documentation
September 15, 2026

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2020. *SAS® National Language Support (NLS): Reference Guide*. Cary, NC: SAS Institute Inc.

SAS® National Language Support (NLS): Reference Guide

Copyright © 2020, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

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

September 2026

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

v_001-P1:nlsref

Contents

Syntax Conventions for the SAS Language . . . . . *vii*

PART 1 NLS Concepts 1

Chapter 1 / National Language Support (NLS) **3**
Overview to National Language Support 3
Definition of Localization and Internationalization 4

Chapter 2 / Locale for NLS **5**
Overview of Locale Concepts for NLS 5
Specifying a Locale 6

Chapter 3 / Encoding for NLS **9**
Overview: Encoding for NLS 9
Difference between Encoding and Transcoding 10
Common Encoding Methods 11
Standards Organizations for NLS Encodings 14
Collating Sequence 14
Determining the Encoding of a SAS Data Set 19
Default SAS Session Encoding 20
Setting the Encoding of a SAS Session 20
Encoding Behavior in a SAS Session 22
Migrating Data to UTF-8 24
Migrating Data from WLATIN1 to UTF-8 24

Chapter 4 / Transcoding for NLS **31**
Overview of Transcoding 31
Common Reasons for Transcoding 32
Transcoding and Translation Tables 32
SAS Options That Transcode SAS Data 34
Transcoding Considerations 34
Compatible and Incompatible Encodings 35
Troubleshooting Truncation and Data Loss Issues during Transcoding 40
Avoiding Character Data Truncation By Using the CVP Engine 41
Avoiding Character Truncation Using the %COPY_TO_NEW_ENCODING Macro 46
Avoiding Character Data Loss during CEDA Transcoding by Using the KPROPDATA Function 47

Chapter 5 / Double-Byte Character Sets (DBCS) **51**
Overview to Double-Byte Character Sets (DBCS) 51

East Asian Languages . . . 52
Specifying DBCS . . . 52
Requirements for Displaying DBCS Character Sets . . . 53
When You Can Use DBCS Features . . . 53
DBCS and SAS on a Mainframe . . . 54
SAS Data Conversion between DBCS Encodings . . . 54
Avoiding Problems with Split DBCS Character Strings . . . 55

Chapter 6 / Time Zones . . . **57**
Overview of SAS Time Zones . . . 57
Specifying Time Zones in SAS . . . 57

PART 2 Autocall Macros for NLS 67

Chapter 7 / Dictionary of Autocall Macros for NLS . . . **69**
Autocall Macro Entries by Category . . . 69
Dictionary . . . 70

PART 3 Data Set Options for NLS 79

Chapter 8 / Dictionary of Data Set Options for NLS . . . **81**
Data Set Options by Category . . . 81
Dictionary . . . 82

PART 4 Formats for NLS 87

Chapter 9 / Overview to NLS Formats . . . **89**
International Date and Datetime Formats . . . 89
Currency Representation . . . 90
Exceptions for Date and Time Default Widths . . . 98
NL Formats that Use Uniform Patterns . . . 100

Chapter 10 / Dictionary of Formats for NLS . . . **107**
Categories of NLS Formats . . . 112
Dictionary . . . 144

PART 5 Functions for NLS 553

Chapter 11 / Internationalization Compatibility for SAS String Functions . . . **555**
Internationalization Compatibility for SAS String Functions . . . 555

Chapter 12 / VARCHAR Support in Functions **583**
VARCHAR Data Type in String Functions 583

Chapter 13 / Dictionary of Functions for NLS **587**
Functions by Category 588
Dictionary 595

PART 6 Informats for NLS 763

Chapter 14 / Dictionary of Informats for NLS **765**
Informats by Category 767
Dictionary 781

PART 7 Macro Functions for NLS 945

Chapter 15 / Dictionary of Macro Functions for NLS **947**
Macro Functions by Category 947
Dictionary 948

PART 8 System Options for NLS 969

Chapter 16 / Dictionary of System Options for NLS **971**
System Option Entries by Category 972
Dictionary 973

PART 9 Options for Commands, Statements, and Procedures for NLS 1011

Chapter 17 / Dictionary of Command, Statement, and Procedure Option for NLS **1013**
Commands, Statements, and Procedures for NLS by Category 1013
Dictionary 1015

PART 10 Procedures for NLS 1053

Chapter 18 / DBCSTAB Procedure **1055**
Overview: DBCSTAB Procedure 1055
Syntax: DBCSTAB Procedure 1056
Examples: DBCSTAB Procedure 1057

Chapter 19 / LOCALEDATA Procedure **1063**
Overview: LOCALEDATA Procedure 1063
Syntax: LOCALEDATA Procedure 1063
Examples: LOCALEDATA Procedure 1072

Chapter 20 / TRANTAB Procedure **1079**
Overview: TRANTAB Procedure 1080
Concepts: TRANTAB Procedure 1081
Syntax: TRANTAB Procedure 1084
Examples: TRANTAB Procedure 1091

PART 11 Values for Locale, Encoding, and Transcoding 1107

Chapter 21 / Values for the LOCALE= System Option **1109**
Locale-to-Encoding Mapping 1109

Chapter 22 / SAS System Options for Processing DBCS Data **1125**
Overview to System Options Used in a SAS Session for DBCS 1125
DBCS Values for a SAS Session 1125

Chapter 23 / Encoding Values in SAS Language Elements **1127**
Overview to SAS Language Elements That Use Encoding Values 1127
SBCS, DBCS, and Unicode Encoding Values for Transcoding Data 1127

Chapter 24 / Encoding Values for a SAS Session **1139**
LINUX Encoding Values 1139

PART 12 Appendixes 1141

Appendix 1 / Encoding and Alias Names **1143**
Encodings and Their Aliases and Encoding Character Set Compatibility 1143

Appendix 2 / Time Zone IDs and Time Zone Names **1153**
Area: Africa 1153
Area: America–North, Central, and South 1156
Area: Antarctica 1166
Area: Arctic 1167
Area: Asia 1167
Area: Atlantic 1172
Area: Australia 1173
Area: Miscellaneous 1176
Area: Europe 1182
Area: Pacific 1189

Appendix 3 / TK_NLS_REPLACE_ICVEXT **1193**

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

SAS uses standard conventions in the documentation of syntax for SAS language elements. These conventions enable you to easily identify the components of SAS syntax. The conventions can be divided into these parts:

- syntax components
- style conventions
- special characters
- references to SAS libraries and external files

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=). The syntax for arguments has multiple forms in order to demonstrate the syntax of multiple arguments, with and without punctuation.

keyword
specifies the name of the SAS language element that you use when you write your program. Keyword is a literal that is usually the first word in the syntax. In a CALL routine, the first two words are keywords.

In these examples of SAS syntax, the keywords are bold:

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

In this example, the first two words of the CALL routine are the keywords:

CALL RANBIN(*seed*, *n*, *p*, *x*)

The syntax of some SAS statements consists of a single keyword without arguments:

DO;
... *SAS code* ...
END;

Some system options require that one of two keyword values be specified:

DUPLEX | **NODUPLEX**

Some procedure statements have multiple keywords throughout the statement syntax:

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <, *column-2*, ...>)

argument
specifies a numeric or character constant, variable, or expression. Arguments follow the keyword or an equal sign after the keyword. The arguments are used by SAS to process the language element. Arguments can be required or optional. In the syntax, optional arguments are enclosed in angle brackets (< >).

In this example, *string* and *position* follow the keyword CHAR. These arguments are required arguments for the CHAR function:

CHAR (*string, position*)

Each argument has a value. In this example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:

```
x=char('summer', 4);
```

In this example, *string* and *substring* are required arguments, whereas *modifiers* and *startpos* are optional.

FIND(*string, substring* <, *modifiers*> <, *startpos*>

argument(s)
specifies that one argument is required and that multiple arguments are allowed. Separate arguments with a space. Punctuation, such as a comma (,) is not required between arguments.

The MISSING statement is an example of this form of multiple arguments:

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2* ... >
specifies that one argument is required and that a literal argument can be associated with the argument. You can specify multiple literals and argument

pairs. No punctuation is required between the literal and argument pairs. The ellipsis (...) indicates that additional literals and arguments are allowed.

The BY statement is an example of this argument:

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* ...>;

argument-1 <*options*> <*argument-2* <*options*> ...>
specifies that one argument is required and that one or more options can be associated with the argument. You can specify multiple arguments and associated options. No punctuation is required between the argument and the option. The ellipsis (...) indicates that additional arguments with an associated option are allowed.

The FORMAT procedure PICTURE statement is an example of this form of multiple arguments:

PICTURE name <(*format-options*)>
<*value-range-set-1* <(*picture-1-options*)>
<*value-range-set-2* <(*picture-2-options*)> ...>>;

argument-1=value-1 <*argument-2=value-2* ...>
specifies that the argument must be assigned a value and that you can specify multiple arguments. The ellipsis (...) indicates that additional arguments are allowed. No punctuation is required between arguments.

The LABEL statement is an example of this form of multiple arguments:

LABEL *variable-1=label-1* <*variable-2=label-2* ...>;

argument-1 <, *argument-2*, ...>
specifies that one argument is required and that you can specify multiple arguments that are separated by a comma or other punctuation. The ellipsis (...) indicates a continuation of the arguments, separated by a comma. Both forms are used in the SAS documentation.

Here are examples of this form of multiple arguments:

AUTHPROVIDERDOMAIN (*provider-1:domain-1* <, *provider-2:domain-2*, ...>

INTO :*macro-variable-specification-1* <, :*macro-variable-specification-2*, ...>

Note: In most cases, example code in SAS documentation is written in lowercase with a monospace font. You can use uppercase, lowercase, or mixed case in the code that you write.

Style Conventions

The style conventions that are used in documenting SAS syntax include uppercase bold, uppercase, and italic:

UPPERCASE BOLD
identifies SAS keywords such as the names of functions or statements. In this example, the keyword ERROR is written in uppercase bold:

ERROR <*message*>;

UPPERCASE
identifies arguments that are literals.

In this example of the CMPMODEL= system option, the literals include BOTH, CATALOG, and XML:

CMPMODEL=BOTH | CATALOG | XML |

italic
identifies arguments or values that you supply. Items in italic represent user-supplied values that are either one of the following:

- nonliteral arguments. In this example of the LINK statement, the argument *label* is a user-supplied value and therefore appears in italic:

  LINK *label*;

- nonliteral values that are assigned to an argument.

  In this example of the FORMAT statement, the argument DEFAULT is assigned the variable *default-format*:

  FORMAT *variable(s)* <*format* > <DEFAULT = *default-format*>;

Special Characters

The syntax of SAS language elements can contain the following special characters:

=
an equal sign identifies a value for a literal in some language elements such as system options.

In this example of the MAPS system option, the equal sign sets the value of MAPS:

MAPS=*location-of-maps*

< >
angle brackets identify optional arguments. A required argument is not enclosed in angle brackets.

In this example of the CAT function, at least one item is required:

CAT (*item-1* <, *item-2*, ...>)

|
a vertical bar indicates that you can choose one value from a group of values. Values that are separated by the vertical bar are mutually exclusive.

In this example of the CMPMODEL= system option, you can choose only one of the arguments:

CMPMODEL=BOTH | CATALOG | XML

...
an ellipsis indicates that the argument can be repeated. If an argument and the ellipsis are enclosed in angle brackets, then the argument is optional. The repeated argument must contain punctuation if it appears before or after the argument.

In this example of the CAT function, multiple *item* arguments are allowed, and they must be separated by a comma:

CAT (*item-1* <, *item-2*, ...>)

'*value*' or "*value*"
indicates that an argument that is enclosed in single or double quotation marks must have a value that is also enclosed in single or double quotation marks.

In this example of the FOOTNOTE statement, the argument *text* is enclosed in quotation marks:

FOOTNOTE <*n*> <*ods-format-options* '*text*' | "*text*">;

;
a semicolon indicates the end of a statement or CALL routine.

In this example, each statement ends with a semicolon:

```
data namegame;
   length color name $8;
   color = 'black';
   name = 'jack';
   game = trim(color) || name;
run;
```

References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks.

If you use a logical name, you typically have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the reference. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

In the examples that use external files, SAS documentation uses the italicized phrase *file-specification*. In the examples that use SAS libraries, SAS documentation uses the italicized phrase *SAS-library* enclosed in quotation marks:

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

PART 1

NLS Concepts

Chapter 1
National Language Support (NLS) 3

Chapter 2
Locale for NLS 5

Chapter 3
Encoding for NLS 9

Chapter 4
Transcoding for NLS 31

Chapter 5
Double-Byte Character Sets (DBCS) 51

Chapter 6
Time Zones 57

1

National Language Support (NLS)

Overview to National Language Support 3
Definition of Localization and Internationalization 4

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 uppercase and lowercase 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 are 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

- set of characters in an encoding
- 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 languages 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".

2

Locale for NLS

Overview of Locale Concepts for NLS 5

Specifying a Locale 6
How Locale Is Specified at SAS Invocation 6
How Locale Is Specified during a SAS Session 7
Language Switching 8

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 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 locale.

For example, “the third day of October in the year 2020” would be displayed in a different way for each of these locales:

English
October 03, 2020

German
03. Oktober 2020

Italian
03 ottobre 2020

Simplified Chinese
2020 年 10 月 03 日

Language is part of a locale, but 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 that 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, 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 North American countries; A4 is used by most other locales. Most locales use centimeters, some locales use inches.

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

Specifying a Locale

How 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 Table 21.1.

The English_UnitedStates value for LOCALE= causes the following options to be implicitly set to the specified Windows default values SAS invocation:

- DATESTYLE=MDY
- DFLANG=English
- ENCODING=WLATIN1
- 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 System Options: Reference* .

How Locale Is Specified during a SAS Session

You can specify the locale option with this code. In this example you are specifying Denmark, Faroe Island, and Greenland locale using the posix value da_DK.

```
option locale=da_DK;
data test;
   x=put(-1234.56789,nlmnldkk32.2);
   put x=;
run;
```

The code specifies the Denmark, Faroe Island, and Greenland locale and processes -1234.56789 with the local NL format for Denmark, Faroe Island, and Greenland, nlmnldkk.

```
x=1,234.57 .|.د-
```

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 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 start-up. 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 start-up. 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". 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".

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".

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 SAS configuration. If `ODSLANGCHG=ON`, all messages are in English because of the LSWLANG setting. For more information, see "LOGLANGENG System Option".

For more information about Language Switching, see Multilingual Computing with SAS® 9.4.

3

Encoding for NLS

Overview: Encoding for NLS ... 9
Difference between Encoding and Transcoding ... 10
Common Encoding Methods ... 11
Standards Organizations for NLS Encodings ... 14
Collating Sequence ... 14
Overview of Collating Sequence ... 14
Request Alternate Collating Sequence ... 16
Specifying a Translation Table ... 17
Specifying an Encoding Value ... 17
Specifying Linguistic Collation ... 18
Determining the Encoding of a SAS Data Set ... 19
Using PROC CONTENTS to Display the Encoding of a SAS Data Set ... 19
Default SAS Session Encoding ... 20
Setting the Encoding of a SAS Session ... 20
Encoding Behavior in a SAS Session ... 22
Encoding Support for Data Sets by SAS Release ... 22
Output Processing ... 23
Input Processing ... 23
Reading and Writing External Files ... 23
Migrating Data to UTF-8 ... 24
Migrating Data from WLATIN1 to UTF-8 ... 24

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.

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 (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.

Each SAS session is set to a default encoding, which can be specified by using various SAS language elements.

Difference between Encoding and Transcoding

Encoding establishes the default working environment for your SAS session. For example, the Windows LATIN1 encoding is an encoding for a SAS session under Windows in a Western European locale such as the de_DE locale for German in Germany. The Windows LATIN1 code point for the uppercase letter Ä is C4 hexadecimal.

However, if you are working in an international environment, and you access SAS data that is encoded in another encoding, Ä might not map to C4 hexadecimal in the other encoding. If characters do not match between encodings, the data must be transcoded.

Note: The default encoding varies according to the operating environment and the locale. UTF-8 is becoming the preferred encoding.

Transcoding is the process of converting data from one encoding to another. For conceptual information, see Chapter 4, “Transcoding for NLS”.

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".

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.

Common encoding methods are listed here:

Unicode
: Unicode is a common and popular character set. SAS Viya platform 4.0.1 supports Unicode 13.0 characters.

  provides up to 143,859 character combinations. Unicode can accommodate basically all of the world's languages. Unicode is a coded character set. The following are encodings 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 most common encoding. Storing text as a UTF-8 encoding might take more space than storing it in legacy encodings. The expansion amount depends on the language and text. Here are some possible expansions for some common legacy encodings:

Table 3.1 *Examples of Storage Size Increases in UTF-8*

Encoding	Languages	Storage Size Increase in UTF-8
ASCII	English, Malay	0%
ISO-8859-1	Western European	10%
ISO-8859-7, plain text	Greek	90%
ISO-8859-7, 50% markup	Greek	45%

Encoding	Languages	Storage Size Increase in UTF-8
TIS-620, plain text	Thai	190%
TIS-620, 50% markup	Thai	95%
EUC-KR, plain text	Korean	50%
EUC-KR, 50% markup	Korean	55%

Footnote: Adapted from W3C Internationalization (I18n) Activity

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 LINUX computers.

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. HTTP and HTML protocols are based on Unicode.

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 3.2 *Other Encodings Approved by ISO*

ISO Standard	Name of Encoding	Description
ISO 8859-1	LATIN 1	US and Western European

ISO Standard	Name of Encoding	Description
ISO 8859-2	LATIN 2	Central and Eastern European
ISO 8859-3	LATIN 3	Southern European, Maltese, and Esperanto
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

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

Table 3.3 *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

Standard	Name of Encoding	Description
KS C 5601	Korean National Standard	Korea
JIS	Japan Industry Standard	Japan
SHIFT-JIS	Japan Industry Standard multibyte encoding	Japan

There are other encodings that support different languages and locales.

See "Encodings and Their Aliases and Encoding Character Set Compatibility" 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. 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
: that develops and promotes the Unicode standard, which provides a unique number for every character.

Collating Sequence

Overview of 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 session encoding is the default encoding for a SAS session. The default encoding can be specified by using various SAS language elements.) The sort order corresponds directly to the arrangement of the code points within the code page.

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. It lists words beginning with accented characters after words beginning with unaccented characters. Therefore, for ASCII-based encodings, the capital letter `Z` `a`. `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
- linguistic collation

This example 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
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 third column shows the results of specifying the ASCII-based, double-byte latin1 encoding.

The last column shows the results of linguistic collation for the session locale fr_CA (French_Canada), 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.

Request Alternate Collating Sequence

To request an alternate collating sequence, use the following SAS language element:

- SORTSEQ= option in the PROC SORT statement. See “Collating Sequence Option”.

Note that neither method supports all of the collating sequences. For example, the SORTSEQ= option in the PROC SORT statement supports linguistic collation. However, both the SORTSEQ= option in the PROC SORT statement and the SORTSEQ= system option support translation table collating sequences.

PROC SQL also supports linguistic collation. See “Sorting Data” in *SAS SQL Procedure User’s Guide*

The BASE (V9) engine and the REMOTE engine for SAS/SHARE support all alternate collating sequences. The V9TAPE sequential engine supports the use of a translation table and an encoding value to sort data, but the V9TAPE engine does not support linguistic collation.

Specifying a Translation Table

A translation table is a SAS catalog entry that transcodes data from one single-byte encoding to another single-byte encoding. A translation table also reorders characters when sorting them. A translation table can be one that SAS provides, such as a standard collating sequence like ASCII, or DANISH; or it can be a user-defined translation table.

When you specify a translation table for an alternate collating sequence, the characters are reordered by mapping the code point of each character to an integer weight value in the range of 0 to 255. A binary collation is then performed.

For collating purposes, you can create translation tables that order characters so that lowercase and uppercase characters alternate. For example, you can create a translation table to correct the situation in which **Z** precedes **a** in an ASCII-based encoding. (However, regardless of the weight assignments in the translation table, it is difficult to achieve a true alphabetic ordering that takes the character case into account.) You can also create a translation table that orders alphabetic characters of a particular language in their expected order.

The TRANTAB procedure creates, edits, and displays translation tables. For example, you can display a translation table to view the character-weight values. The translation tables that are supplied by SAS are stored in the SASHELP.HOST catalog. Any translation table that you create or customize is stored in your SASUSER.PROFILE catalog. Translation tables have an entry type of TRANTAB. See Chapter 20, “TRANTAB Procedure” for more information about translation tables.

You can specify a translation table with the SORTSEQ= option in the PROC SORT statement or with the SORTSEQ= system option. For example, if your operating environment sorts with the ASCII-based Wlatin1 encoding by default, and you want to sort with a translation table that alternates uppercase and lowercase characters, issue the following statements to specify the SAS translation table FRSOLAT1:

```
proc sort data=myfiles.test sortseq=FRSOLAT1;
   by name;
run;
```

A SAS data set that is sorted with a translation table contains a sort indicator that displays the specified translation table name as the collating sequence in CONTENTS procedure output.

Specifying an Encoding Value

An *encoding* is a set of characters (letters, logograms, digits, punctuation marks, symbols, and control characters) that have been mapped to hexadecimal values, called code points, that computers use. When you specify an encoding value for an alternate collating sequence, the characters are transcoded from the SAS session

encoding to the specified encoding. Then a binary collation is performed. You can specify all encoding values that are supported by the ENCODING= option, including multi-byte encodings. Note that specifying a translation table can transcode data, but translation tables are limited to single-byte encodings.

You can specify an encoding value with the SORTSEQ= option in the PROC SORT statement, but you cannot specify an encoding value in the SORTSEQ= system option. For example, you want to sort a SAS data set and then transport it to a Japanese Windows environment. If your session encoding is ASCII-based and binary collation is in effect, you can issue the following statements to specify the ASCII-based double-byte encoding SHIFT-JIS:

```
proc sort data=myfiles.test sortseq='shift-jis';
   by name;
run;
```

Note that SAS checks the encoding value for any translation tables with the same name. If a translation table name exists, SAS uses the translation table.

A SAS data set that is sorted with an encoding value contains a sort indicator that displays the specified encoding value as the collating sequence in CONTENTS procedure output.

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). ICU 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, the following statements cause the SORT procedure to collate linguistically, in accordance with the French_Canada locale:

```
options locale=fr_CA;

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

Note: Starting in 2021.2.4, ICU libraries have been upgraded to 70.1. With the upgrade, we might see incompatibilities in the linguistic collation for the Chinese locales such as zh_CN and zh_TW.

Existing datasets that were sorted by the linguistic collator for the Chinese locale must be re-sorted.

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, local conventions such as data formatting, and 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" or the SORTSEQ= option in the PROC SORT statement in *Base SAS Procedures Guide*.

A SAS data set that is sorted linguistically contains a sort indicator that displays the collating sequence LINGUISTIC in 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. The descriptor information is also displayed in 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)
```

Default SAS Session Encoding

The ENCODING= system option is used to specify the SAS session encoding. The session encoding establishes the environment to process SAS syntax and SAS data sets, and to read and write external files. If neither the LOCALE= nor ENCODING= options is set, a default value is set.

The ENCODING system option is set explicitly in all SASFoundation sasv9.cfg configuration files.

The BI server configuration process uses the default SASFoundation sasv9.cfg file (!SASROOT/sasv9.cfg). The default sasv9.cfg file points to the sasv9.cfg that contains the LOCALE and ENCODING option that you select. This action usually happens during deployment. You can deploy multiple server contexts at the site that uses different SAS configurations.

Table 3.5 *Default SAS Session Encoding Values*

Operating Environment	Default ENCODING= Value	Description
LINUX	Latin1	Western (ISO)

For a complete list of supported encoding values for a SAS session, see Chapter 24, "Encoding Values for a SAS Session," on page 1139.

Setting the Encoding of a SAS Session

You can set the session encoding by using the ENCODING= system option, the DBCS options, or the LOCALE= system option.

Note: Values for the ENCODING= system option depend on the operating environment.

The priority order for setting the encoding is as follows:

1 ENCODING= system option

The SAS session encoding is determined by the ENCODING= option regardless of whether the DBCS or LOCALE= options are specified. If the ENCODING= option is specified, a set of valid DBCS options is set regardless of whether the user has specified those options. Also, if the ENCODING= option is specified,

the LOCALE= option is set to an appropriate value unless a value has been specified by the user.

Note: If the ENCODING= option is specified, the TRANTAB= option is implicitly set. TRANTABS apply only to external files.

2 DBCS options

Most North and South American, European, Middle Eastern, African, and Australian users use the SAS SBCS environment and do not use the DBCS environment.

If the ENCODING= option is not specified, the SAS session encoding is determined by the DBCS options regardless of whether the LOCALE= option is specified. The LOCALE= option is set to an appropriate value unless a value has been specified by the user.

The encoding is determined by the values of the DBCSLANG and DBCSTYPE options for DBCS languages, such as Japanese, Korean, Simplified Chinese, and Traditional Chinese.

The DBCS options are valid only when the DBCS extension directory is included in the path option list. The path of the DBCS extension dynamic link library (DLLs) must be located at the top of the pathname list of the path option for the DBCS languages when you want to invoke a DBCS SAS session. The DBCS extension DLLs are located in the directory **!SASROOT/dbcs/sasexe** by default.

Also, you might have to specify the resourcesloc, msg, and sashelp options to use localized resources even if the SAS session encoding is not a DBCS language (for example, Polish, German, and French). The localized resources are located under **!SASROOT/nls/<language identifier>/<sasmsg, sashelp, sasmacro, resource>**. The values for language identifiers are: cs, de, en, es, fr, hu, it, ja, ko, pl, ru, sv, zh, and zt.

You can specify a sasv9.cfg file located in the localized directories such as **!SASROOT/nls/<language identifier>** so that you do not have to consider using the **path, resourcesloc, sasmsg**, and **sashelp** options.

If DBCS (which specifies that SAS process DBCS encodings) is specified, DBCSLANG= and DBCSTYPE= options are implicitly set. The default values for DBCSTYPE= and DBCSLANG= match those values for the DBCS environment on the host (for example, Japanese, Korean, or Chinese).

3 LOCALE= system option

The SAS session encoding is determined by the LOCALE= option and the platform, if the ENCODING= or DBCS options are not specified.

The following example shows that encoding is explicitly set by default for the Spanish_Spain locale:

```
sas9 -locale Spanish_Spain
```

The WLATIN1 encoding is the default encoding for the Spanish_Spain locale.

The following example shows that the WLATIN2 encoding is set explicitly when SAS is invoked:

```
sas9 -encoding wlatin2
```

Note: Setting DBCS encodings, DBCS options, or a CJK (Chinese, Japanese, Korean) locale on SAS if the DBCS extensions are not available fails to successfully invoke SAS.

Note: Changing the encoding for a SAS session does not affect SAS keywords or SAS log output, which remain in English.

In Table 3.6 on page 22, the following values for the CJK locales are based on locale and platform:

Table 3.6 *Default Encoding Values Based on the LOCALE= Option*

Locales	LINUX
zh_TW zh_HK zh_MO	Solaris on X64, Solaris on SPARC, EUC-TW (yeuc) others: MS-950 (ywin)
zh_CN zh_SG	EUC-CN (zeuc)
ja_JP	h64, h6i, AIX on Power, SHIFT-JIS (sjis) others: EUC-JP (jeuc)
ko_KR	EUC-KR (keuc)

Encoding Behavior in a SAS Session

Encoding Support for Data Sets by SAS Release

For SAS files, there are three categories of encoding support. Encoding support is based on the version of SAS that created the file:

- Data sets that are created in Viya automatically have an encoding attribute, which is specified in the descriptor portion of the file. In Viya, DBCS by default recognizes the DBCSTYPE value and converts it to the encoding value and specifies it in the descriptor portion of the field.

- Data sets that are created in SAS 7 and SAS 8 do not have an encoding value that is specified in the file. It is assumed that SAS 7 and SAS 8 data sets were created in the SAS session encoding of the operating environment. However, the descriptor portion of the file does support an encoding value. When you replace or update a SAS 7 or SAS 8 file in a SAS 9 session, SAS specifies the current session encoding in the descriptor portion of the file, by default. In SAS 8, DBCS has the DBCSTYPE field, instead of the encoding field.
- Data sets created in SAS 6 do not have an encoding value that is associated with the file. An encoding value cannot be specified in the file.

Output Processing

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

- If a new output file is created, the data is written to the file using the current session encoding.
- 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. The OUTREP= option specifies a data representation that is different from the current session. For more information, see "OUTREP= Data Set Option".
- 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 that is specified in it, then the current session encoding is used.

Input Processing

For input (read) processing, 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.
- If a file does not contain a specified encoding, SAS transcodes the data only if the file's data representation is different from the session encoding.

Reading and Writing External Files

SAS reads and writes external files using the current session encoding. Viya assumes that the external file has the same encoding as the session encoding. For

example, if you are creating a new data set by reading an external file, Viya 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 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".

Migrating Data to UTF-8

UTF-8 is part of the Unicode coded character set. UTF-8 is the preferred and most-used encoding, and it is the recommended encoding for using Unicode with operating systems like Linux.

You can migrate your data that use other encodings to UTF-8 encoding in order to support multilingual data and to support Viya. There are several methods of migrating data: Cross-environment data access (CEDA), Character Variable Padding (CVP) engine, and the %COPY_TO_NEW_ENCODING macro. Instructions on how to use these methods are described in *Migrating Data to UTF-8 for the SAS Viya Platform*.

Migrating Data from WLATIN1 to UTF-8

The common encodings for Western European character data are ISO 8859-1 (LATIN1) and Windows cp1252 (WLATIN1). If your character data is encoded as LATIN1 or WLATIN1, there are several factors that you must consider when you migrate to UTF-8. This section focuses on WLATIN1 characters, which are a superset of LATIN1 characters.

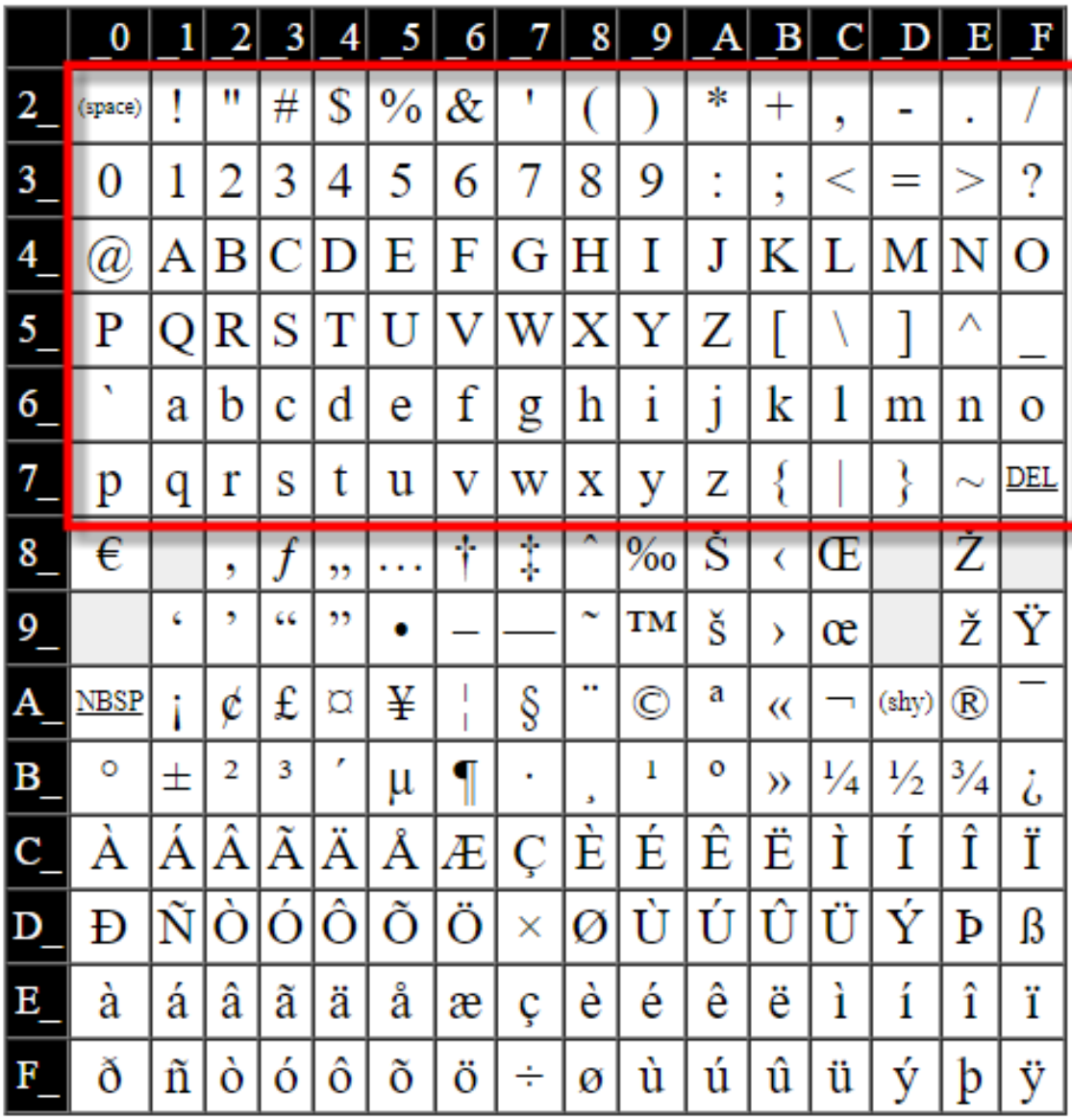
All WLATIN1 characters are available in UTF-8. Here is an example of the WLATIN1 code page:

Figure 3.1 *WLATIN1 Code Page*

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
2_	(space)	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8_	€		,	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
9_		‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ
A_	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	(shy)	®	¯
B_	°	±	2	3	´	µ	¶	·	¸	1	º	»	¼	½	¾	¿
C_	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D_	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E_	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F_	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Characters in WLATIN1 require 1 byte, and UTF-8 characters require 1–4 bytes. In Figure 3.2, the characters in the highlighted rows 2–7 require 1 byte in UTF-8. These characters are also referred to as ASCII characters.

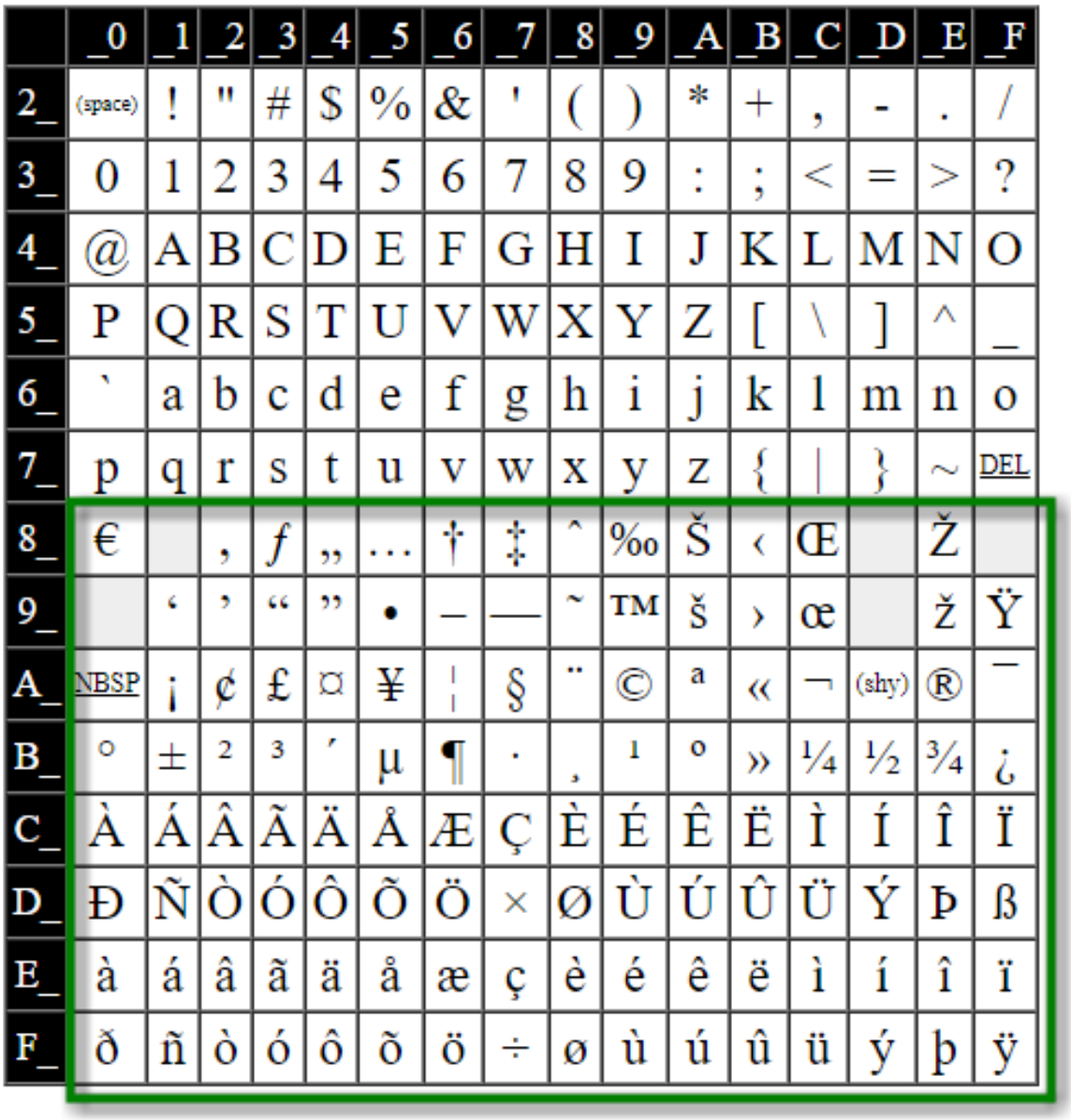
Figure 3.2 *Rows 2–7 in the WLATIN1 Code Page*



	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
2_	(space)	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8_	€		,	*f*	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
9_		‘	’	“	”	•	–	—	˜	TM	š	›	œ		ž	Ÿ
A_	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	(shy)	®	¯
B_	°	±	2	3	´	µ	¶	·	¸	1	º	»	¼	½	¾	¿
C_	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D_	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E_	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F_	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

In Figure 3.3, the characters in the highlighted rows 8–F require 1 byte in WLATIN1 and 2–3 bytes in UTF-8.

Figure 3.3 *Rows 8–F in the WLATIN1 Code Page*



	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
2_	(space)	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8_	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
9_		‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ
A_	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	(shy)	®	¯
B_	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C_	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D_	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E_	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F_	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

For errors or incorrect data that result from migrating characters that are 1 byte in WLATIN1 but 2–4 bytes in UTF-8, follow these recommendations to resolve the problem:

Truncation
: All WLATIN1 characters can be transcoded to UTF-8. A transcoding error or warning means that the character variable is not long enough to hold the UTF-8 representation of those characters. To resolve the problem, you can expand the size of your character columns using CVP.

Unexpected characters
: If you are expecting all characters in your WLATIN1 data to be 1 byte in UTF-8, you might be surprised to see a transcoding error when you migrate your data to UTF-8. The WLATIN1 encoding includes two versions of some punctuation characters:

- ASCII version in rows 2–7
- 8-bit characters in rows 8–F

Some text editors, such as Microsoft Word, support an AutoFormat feature that converts the ASCII characters that are entered from the keyboard to a version of the character that is produced by a typographical device.

For example, the quotation mark character (") (also known as a straight quotation mark) has a hexadecimal representation of 22 in both WLATIN1 and UTF-8. However, AutoFormat might convert the character to left and right double quotation characters (also referred to as curly quotation marks or smart quotation marks). The hexadecimal values for the left and right double quotation marks in WLATIN1 are 93 and 94. But, in UTF-8, these characters are E2809C and E2809D, and each character requires 3 bytes.

You can use the KPROPDATA function with the PUNC option to convert smart quotation marks and other typographical punctuation characters to a 1-byte character in UTF-8.

Here are the punctuation characters that are 1 byte in WLATIN1 and 2–4 bytes in UTF-8:

Table 3.7 *Smart Quotation Marks and Punctuation Characters*

	Punctuation				**Convert To**		
Category	**Punctuation character**	**Description**	**WLATIN1 code point (Hex)**	**UTF-8 code point (Hex)**	**ASCII Character**	**Description**	**ASCII code point (Hex)**
Quotation marks	‚	single low-9 quotation mark	82	E2 80 9A	,	comma	2C
	„	double low-9 quotation mark	84	E2 80 9E	,,	quotation mark	22
	‘	left single quotation mark	91	E2 80 98	'	apostrophe	27
	’	right single quotation mark	92	E2 80 99	'	apostrophe	27
	“	left double quotation mark	93	E2 80 9C	"	quotation mark	22

Category	Punctuation				Convert To		
	Punctuation character	**Description**	**WLATIN1 code point (Hex)**	**UTF-8 code point (Hex)**	**ASCII Character**	**Description**	**ASCII code point (Hex)**
	”	right double quotation mark	94	E2 80 9D	"	quotation mark	22
Angle brackets	‹	single left-pointing angle quotation mark	8B	E2 80 B9	<	less-than sign	3C
	›	single right-pointing angle quotation mark	9B	E2 80 BA	>	greater-than sign	3E
Ellipsis	…	horizontal ellipsis	85	E2 80 A6	...	full stop	2E 2E 2E [1]
Bullet	•	bullet	95	E2 80 A2	*	asterisk	2A
Hyphen	–	en hyphen	96	E2 80 93	-	hyphen-minus	2D
	—	em hyphen	97	E2 80 94	-	hyphen-minus	2D

1 Three ASCII period characters.

4

Transcoding for NLS

Overview of Transcoding 31
Common Reasons for Transcoding 32
Transcoding and Translation Tables 32
SAS Options That Transcode SAS Data 34
Transcoding Considerations 34
Compatible and Incompatible Encodings 35
Overview of Compatible and Incompatible Encodings 35
Session Encodings and Data Set Encodings, Notes, Warnings, and Errors 39
Troubleshooting Truncation and Data Loss Issues during Transcoding 40
Avoiding Character Data Truncation By Using the CVP Engine 41
Avoiding Character Truncation Using the %COPY_TO_NEW_ENCODING Macro 46
Avoiding Character Data Loss during CEDA Transcoding by Using the KPROPDATA Function 47

Overview of 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.

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, then moved to an IBM mainframe that uses the German EBCDIC encoding. When the file is processed on the IBM mainframe, the data is remapped from the LATIN1 encoding to the German EBCDIC encoding. If the data contains an uppercase letter Ä, the hexadecimal number is converted from C4 to 4A.

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 details, see Chapter 23, "Encoding Values in SAS Language Elements".

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 and the file's encoding is incompatible with the session encoding

Transcoding and Translation Tables

Translation tables are implicitly set by the LOCALE= system option. They are used only for transcoding external files. There is direct transcoding for SAS files between the session encodings.

Specifying LOCALE= or ENCODING= indirectly sets the appropriate translation table values in the TRANTAB= option. Translation tables are used for transcoding one SBCS encoding to another and back again. For example, there is a specific translation table that maps Windows LATIN2 to ISO LATIN2. Translation tables apply only to external files.

The following figure shows a translation table. The area of a translation table for mapping from Windows LATIN2 (wlt2) to ISO LATIN 2 (lat2) is named "table 1," and the area for mapping characters from ISO LATIN2 to Windows LATIN2 is named "table 2."

Figure 4.1 *SAS Windows LATIN 2 to ISO LATIN 2 Translation Table*

```
WLT2LAT2 table 1:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '606162636465666768696A6B6C6D6E6F'x
     70 '707172737475767778797A7B7C7D7E7F'x
     80 '808182838485868788898A98AA6ABAEAC'x
     90 '908B8C8D8E8F91929893B994B6BBBEBC'x
     A0 'A0B7A2A3A4A195A7A896AA9799AD9AAF'x
     B0 'B09BB2B3B49C9D9EB8B1BA9FA5BDB5BF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x


WLT2LAT2 table 2:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '606162636465666768696A6B6C6D6E6F'x
     70 '707172737475767778797A7B7C7D7E7F'x
     80 '808182838485868788898B9192939495'x
     90 '909697999BA6A9AB98ACAEB1B5B6B7BB'x
     A0 'A0A5A2A3A4BC8CA7A88AAA8D8FAD8EAF'x
     B0 'B0B9B2B3B4BE9CA1B89ABA9D9FBD9EBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

The LOCALE= or ENCODING= system option and other encoding options (statements, commands, or procedures) eliminate the need to directly create or manage translation tables.

CAUTION

Do not change a translation table unless you are familiar with its purpose. Translation tables are used internally by the SAS supervisor to implement NLS. If you are unfamiliar with the purpose of translation tables, do not change the specifications without proper technical advice.

The TRANTAB= option specifies the translation table to be used in the SAS session. For details, see "TRANTAB= System Option" . The TRANTAB procedure is used to create, edit, and display customized translation tables. For details, see Chapter 20, "TRANTAB Procedure".

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=	in a 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".

Transcoding Considerations

Although transcoding usually occurs with no problems, there are situations that can affect your data and produce unsatisfactory results. Here are several examples:

- 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 a single-byte character set (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. See "SAS File Processing with CEDA " in *SAS Programmer's Guide: Essentials*.
- Incorrect encoding can be stamped on a SAS 7 or SAS 8 data set if it is copied or replaced in a SAS®9 session with a different session encoding from the data. 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.
- Data set and variable labels have a maximum of 256 bytes and variable names have a maximum of 32 characters. If a data set contains long labels with international characters, the transcoded labels might exceed the maximum, resulting in a transcoding error. You can shorten the long labels in the original data set to resolve the transcoding issue.

Compatible and Incompatible Encodings

Overview of 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.

Table 4.2 lists encodings that do not require CEDA processing for transcoding. For example, the first row is for the encoding method SBCS ASCII. If the session

encoding is LATIN1, and the data set's encoding is LATIN9, CEDA processing does not occur. Another example is the DBCS ASCII encoding method. Any combination of encodings in the first line of that row, SHIFT-JIS, MS-932, IBM-942, MACOS-1 does not invoke CEDA processing. If the session encoding is IBM-942, and the data set's encoding is MACOS-1, CEDA processing does not occur.

See "Definitions for Cross-Environment Data Access (CEDA)" in *SAS Programmer's Guide: Essentials* for information about CEDA.

In order to handle character data correctly, SAS requires that data be encoded using the session encoding. When SAS reads a data set that has a different encoding, CEDA compares the data set encoding with the SAS session encoding. The encodings might be incompatible, meaning that transcoding is required in order for SAS to safely handle the characters that are read from the data set.

ASCII is the foundation for many encodings, and it is used by most personal computers and workstations. However, the IBM mainframe uses an EBCDIC encoding. ASCII and EBCDIC machines, and the encodings that are used to store their data, are incompatible. Therefore, transcoding is required. For example, data must be transcoded if the SAS file is WLATIN1 and the session encoding is OPEN_ED-1047. Note that SAS CEDA processing is used only in a mainframe environment when the data sets reside in a hierarchical file system such as z/OS or USS. Encodings that support characters for different languages are also incompatible. For example, WLATIN1 is a single-byte (SBCS) encoding that supports characters for Western European languages. The WCYRILLIC is also SBCS, but it has Russian characters and does not have many of the Western European characters. Therefore, WLATIN1 and WCYRILLIC are incompatible.

You can disable CEDA transcoding. For example, if the data set encoding is not compatible with the session encoding, but you know that all of the characters in the data set are compatible, transcoding occurs even though it is not necessary. You might prefer to transcode the data using a function such as KPROPDATA or KCVT instead of CEDA transcoding the data. You can prevent CEDA from transcoding by specifying one of these encoding names in either the ENCODING= data set option, or in either the INENCODING= or OUTENCODING= option in the LIBNAME statement.

CAUTION

Misuse of these encoding values could result in data corruption. ASCIIANY prevents transcoding.

- 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. The encoding value that is set for the SAS data set using ASCIIANY is US-ASCII.
- EBCDICANY enables you to create a data set that is compatible with all EBCDIC-based encodings.

It is important for you to know the data in your data set before using one of these encodings to disable transcoding. For example, if the encoding of your data set is WLATIN1 and the session encoding is WLATIN2, CEDA transcodes the data because the encodings are not compatible. However, if you know that all of the characters in the data set are ASCII characters, specify ASCIIANY as the INENCODING option in the LIBNAME statement to prevent transcoding.

Note: If you are running SAS in a UTF-8 session encoding, any data set that is not UTF-8 or US-ASCII is incompatible.

```
libname myfiles 'path to data sets' inencoding=asciiany;
```

The encoding of the data set is not changed, but CEDA assumes that the data set encoding is the encoding that is specified in the LIBNAME statement.

As an alternative, you can prevent transcoding by changing the encoding in the data set header. To do so, use the CORRECTENCODING option in the PROC DATASETS MODIFY statement.

```
libname myfiles "path to data sets";
proc datasets library=myfiles;
   modify olddata / correctencoding=ASCIIANY;
quit;
```

You can use these functions to verify that encodings are compatible:

- The ENCODCOMPAT function verifies the transcoding compatibility between two encodings. For more information, see "ENCODCOMPAT Function".
- The ENCODISVALID function specifies a valid encoding name. For more information, see "ENCODCOMPAT Function".

For a list of the encodings, by operating environment, see Chapter 24, "Encoding Values for a SAS Session".

Table 4.2 *Encoding Combinations That Do Not Need CEDA Processing for Transcoding*

Encoding Method	Encoding
SBCS ASCII	LATIN1, LATIN9
	LATIN1, WLATIN1 **TIP** Transcoding is not performed in SAS using CEDA between these two encodings. WLATIN1 differs from LATIN1 by using displayable characters rather than control characters in the 80 to 9F (hexadecimal) range. These differences can lead to problems when cutting and pasting from MS WORD documents.
SBCS EBCDIC	EBCDIC037, OPEN_ED-037
	EBCDIC1025, OPEN_ED-1025

Encoding Method	Encoding
	EBCDIC1026, OPEN_ED-1026
	EBCDIC1047, OPEN_ED-1047
	EBCDIC1112, OPEN_ED-1112
	EBCDIC1122, OPEN_ED-1122
	EBCDIC1130, OPEN_ED-1130
	EBCDIC1140, OPEN_ED-1140
	EBCDIC1141, OPEN_ED-1141
	EBCDIC1142, OPEN_ED-1142
	EBCDIC1144, OPEN_ED-1144
	EBCDIC1145, OPEN_ED-1145
	EBCDIC1146, OPEN_ED-1146
	EBCDIC1147, OPEN_ED-1147
	EBCDIC1148, OPEN_ED-1148
	EBCDIC1160, OPEN_ED-1160
	EBCDIC1164, OPEN_ED-1164
	EBCDIC037, OPEN_ED-037
	EBCDIC275, OPEN_ED-275
	EBCDIC424, OPEN_ED-424
	EBCDIC425, OPEN_ED-425
	EBCDIC838, OPEN_ED-838
	EBCDIC870, OPEN_ED-870
	EBCDIC875, OPEN_ED-875
	EBCDIC924, OPEN_ED-924

Encoding Method	Encoding
DBCS ASCII	SHIFT-JIS, MS-932, IBM-942, MACOS-1
	EUC-KR, MS-949, MACOS-3
	EUC-CN, MS-936, DEC-CN, MACOS-25
	EUC-TW, DEC-TW, EUC-TW
	BIG5, MACOS-2, MS-950
DBCS EBCDIC	IBM-930, OPEN_ED-930
	IBM-933, OPEN_ED-933
	IBM-935, OPEN_ED-935
	IBM-937, OPEN_ED-937
	IBM-939, OPEN_ED-939
	IBM-939E, OPEN_ED-939E
	IBM-930E, OPEN_ED930E

Session Encodings and Data Set Encodings, Notes, Warnings, and Errors

The encoding used to create a SAS data set is saved in the data set header. For example, if you create a data set when you are running SAS with a session encoding of WLATIN1, the WLATIN1 encoding is stored in the data set header. SAS expects the character data encoding to be the same as the encoding in your SAS session. When the encodings do not match, CEDA must be used to transcode the data.

If the data set encoding and session encoding do not match, and the message is a NOTE, the character data should be valid. For example, when a UTF-8 session reads a data set created with a WLATIN1 encoding, you will see the following note in your log:

```
NOTE: Data file MYLIB.CARS.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.
```

If the data set encoding and the session encoding are not compatible, transcoding fails, and the message is a WARNING or an ERROR:

```
WARNING: Some character data was lost during transcoding in the data set
"MYLIB.CARS".
It contains one or more characters that are not available in the new encoding.
To avoid the transcoding error, please refer to "Troubleshooting Truncation and
Data loss Issue during Transcoding"
in SAS National Language Support (NLS): reference guide.
```

This error can mean there is not enough space in one or more character columns in the data set's observation buffer to convert the data to the session encoding or that the characters cannot be represented in the session encoding.

If the data set contains characters that are not supported by the session encoding, the message means that characters were lost during transcoding. For example, if the session encoding is WLATIN1, which supports characters for Western European languages, and SAS reads a data set that is SHIFT-JIS and contains Japanese characters, the message in the SAS log means that the Japanese characters are lost during transcoding.

Troubleshooting Truncation and Data Loss Issues during Transcoding

Transcoding usually occurs with no problems. However, data loss or truncation can occur. One example that can result in data truncation is when the number of bytes for a character in one encoding are different from the number of bytes for the same character in another encoding. Transcoding from single-byte character set (SBCS) to multibyte encoding (UTF-8) can result in character value truncation if the destination buffer is not long enough.

Here is an example of a character having a different number of bytes in two encodings. In this example, the quotation mark has one byte in WLATIN1 and three bytes in UTF-8.

Note: The "Migrating Data from WLATIN1 to UTF-8" section has more information about converting data from WLATIN1 to UTF-8.

Table 4.3 *Character Bytes for the Quotation Mark in WLATIN1 and UTF-8*

Character	Description	WLATIN1 code point (hexadecimal representation)	UTF-8 code point (hexadecimal representation)
„	double low-9 quotation mark	84	E2 80 9E

A character in one encoding might not be supported in another encoding. For example, a specific encoding might not include a character for the dollar sign ($). Transcoding the data to an encoding that does not support the dollar sign would result in character data loss. This loss occurs when transcoding between incompatible encodings. See "Compatible and Incompatible Encodings" for information about encodings that incompatible and can cause character data loss.

See "Transcoding Considerations" for more information about transcoding issues.

There are several methods that help in resolving truncation and data loss issues:

Table 4.4 *Methods for Avoiding Data Truncation*

Method	Description
CVP Engine	You can use the CVP engine to avoid character loss and truncation. For more information and examples see "CVPBYTES=, CVPENGINE=, CVPFORMATWIDTH=, CVPMULTIPLIER=,CVPVARCHAR=, CVPEXCLUDE=, CVPINCLUDE= Options".
%COPY_TO_NEW_ENCODING Macro	You can use the %COPY_TO_NEW_ENCODING macro to avoid character loss and truncation. For more information and examples see "%COPY_TO_NEW_ENCODING Macro".
%COPY_TO_UTF8 Macro	You can use the %COPY_TO_UTF8 macro to create a new version of a data set in UTF-8 encoding.
KPROPDATA Function	You can use the KPROPDATA function to avoid character loss and truncation. For more information and examples see "KPROPDATA Function".

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 one 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 one byte, and a DBCS represents each character in one or two bytes. An MBCS represents characters in a varying length from one to four bytes. For example, when transcoding from WLATIN1 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, then the data is processed. Think of the CVP engine as an intermediate engine that is used to prepare the data for transcoding. After the lengths are increased, the primary engine, such as the default base engine, is used to do the actual file processing.

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 using 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 use for processing 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 "CASNCHARMULTIPLIER= System Option" in *SAS System Options: Reference* increases the number of bytes when transcoding to UTF-8 in the CAS server.

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

Note: Data set and variable labels have a maximum of 256 bytes and variable names have a maximum of 32 characters. If a data set contains long labels with international characters, the transcoded labels might exceed the maximum, resulting in a transcoding error. You can shorten the long labels in the original data set to resolve the transcoding issue.

The CVPFORMATWIDTH option specifies whether to expand the character format width. The CVPFORMATWIDTH default is YES.

This example shows the difference in the PROC CONTENTS output of the format width when using the CVP engine.

The CVP engine is not specified in the first example: `libname in 'C:\temp\datalatin';` In the second example, the CVP engine is specified: `libname in1 cvp 'c:\temp\datalatin';` Notice that the format for the code is $CNTRY24. After running the code, the format result where CVP is specified, *in1* is $CNTRY36 because the CVPFORMATWIDTH is set to YES by default. The format output for the code where CVP is not specified, *in* is $CNTRY24 because CVPFORMATWIDTH is not available since CVP was not specified.

```
options dlcreatedir;
libname in 'C:\temp\datalatin';
proc format;
   value $cntry
   'USA' = 'United States of America'
   'Mexico' = 'United Mexican States';
run;

data IN.COUNTRY;
   length country $10;
   country = 'USA'; output;
   country = 'Mexico'; output;
   format country $cntry24.;
run;

libname in1 cvp 'c:\temp\datalatin';
title 'cvp engine';
proc contents data=in1.country;run;

libname in 'c:\temp\datalatin';
title 'no cvp engine';
proc contents data=in.country;
run;
```

This PROC CONTENTS output shows the results of *in1* that specifies CVP support, so by default format expansion is available. Notice that the format is $CNTRY36 for IN1.COUNTRY.

cvp engine

The CONTENTS Procedure

Data Set Name	IN1.COUNTRY	Observations	2
Member Type	DATA	Variables	1
Engine	CVP	Indexes	0
Created	08/17/2020 10:22:07	Observation Length	16
Last Modified	08/17/2020 10:22:07	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Underlying Engine Accessing Data	V9
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	6470
Obs in First Data Page	2
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\temp\datalatin\country.sas7bdat
Release Created	9.0401M7
Host Created	X64_10PRO
Owner Name	CARYNT\daharr
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
1	country	Char	15	$CNTRY36.

This output shows the results of *in* that does not specify CVP support so that no format expansion is available. Notice that the format is $CNTRY24 for IN.COUNTRY.

no cvp engine

The CONTENTS Procedure

Data Set Name	IN.COUNTRY	**Observations**	2
Member Type	DATA	**Variables**	1
Engine	V9	**Indexes**	0
Created	08/17/2020 10:22:07	**Observation Length**	10
Last Modified	08/17/2020 10:22:07	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	6470
Obs in First Data Page	2
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\temp\datalatin\country.sas7bdat
Release Created	9.0401M7
Host Created	X64_10PRO
Owner Name	CARYNT\daharr
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes				
#	**Variable**	**Type**	**Len**	**Format**
1	country	Char	10	$CNTRY24.

For more information and examples, see “CVPBYTES=, CVPENGINE=, CVPFORMATWIDTH=, CVPMULTIPLIER=,CVPVARCHAR=, CVPEXCLUDE=, CVPINCLUDE= Options” on page 1024.

Avoiding Character Truncation Using the %COPY_TO_NEW_ENCODING Macro

If you write a CNTLOUT= data set using a different encoding than the session encoding, you might receive an error for truncating character variables. If this error occurs, write the CNTLOUT= data set using the current session encoding, and then use the %COPY_TO_NEW_ENCODING macro to create the CNTLOUT= data set with the new encoding.

Here is an example of characters not being supported between encodings. If you have a data set that contains monetary values in Euros, the session encoding is WLATIN1, and you specify UTF-8 encoding for the CNTLOUT= data set, some characters are not compatible. In WLATIN1, the Label variable is predetermined to be 5 bytes long, with a value of €1234 (in hexadecimal, '803132334'x). When you attempt to store the variable in the CNTLOUT= data set with UTF-8 encoding, the length of that string must be 7 bytes (in hexadecimal, 'E282AC31323334'x), so it truncates. You can use the %COPY_TO_NEW_ENCODING macro to prevent this error.

Here is another example of how you can avoid character loss by using the %COPY_TO_NEW_ENCODING macro.

Run the following code from a LATIN1 SAS session.

```
data indata;                          1
   price = "8031323334"x;
run;

libname outlib "myutf8lib" INENCODING='UTF-8' OUTENCODING='UTF-8';
                                                2

proc copy out=outlib in=work noclone;      3
   select indata;
run;

 proc contents data=work.indata; run;       4

%macro copy_to_new_encoding(indata,outdata,UTF8);    5

proc copy in=work out=outlib noclone;      6
   select outdata;
run;

proc contents data=outlib.outdata;         7
run;

libname out2 "outlib" INENCODING='binary' OUTENCODING='binary';
                                                8
data _null_;                                    9
```

```
   set out2.outdata;
   put price=$HEX10.;
run;
```

1 Create the data set INDATA, which contains the Euro character.

2 Assign a libref that has the UTF-8 encoding.

3 Copy the data set INDATA (WLATIN1) to OUTLIB (UTF-8). This error message is displayed because some characters are not compatible between WLATIN1 and UTF-8.

```
NOTE: Data file OUTLIB.INDATA.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.
NOTE: There were 1 observations read from the data set WORK.INDATA.
ERROR: Some character data was lost during transcoding in the dataset OUTLIB.INDATA.
Either the data contains characters that are not representable in the new encoding or
 truncation occurred during transcoding.
ERROR: File OUTLIB.INDATA.DATA has not been saved because copy could not be completed.
```

4 PROC CONTENTS displays the encoding value as UTF-8 and the variable length of price as 5.

5 The %macro copy_to_new_encoding copies the data from INDATA to OUTDATA and converts the data to UTF-8.

6 PROC COPY copies the data set OUTDATA to the OUTLIB library. CEDA is not invoked and data loss does not occur.

7 PROC CONTENTS displays the OUTDATA information. The encoding is UTF-8 and the variable length of price is 7.

8 LIBNAME creates the libref out2.

9 Now the value of price is not truncated. The hexadecimal representation of price in UTF-8 is E282AC31323334.

For more information about the %COPY_TO_NEW_ENCODING macro, see "%COPY_TO_NEW_ENCODING Macro".

Avoiding Character Data Loss during CEDA Transcoding by Using the KPROPDATA Function

When the source and session encodings are not the same, CEDA reads and converts the data. When you read or write a data set that has a different encoding, character loss can occur during the transcoding process if they are not compatible.

You can use the KPROPDATA function to avoid character loss.

Here is an example of using the KPROPDATA function to avoid character loss when LATIN1 characters are converted into EUC-JP encoding.

Run the following code from a LATIN1 SAS session.

```
libname mylib 'my_library';        /* 1

data mylib.indata;                 /* 2
   a='My Text 2021PßCß$N/A';       /* 3
run;

proc contents data=mylib.indata;   /* 4
run;
```

1 Create a libref: mylib.

2 Create the data set INDATA.

3 Assign a character string **`My Text 2021PßCß$N/A`** to variable a.

4 Print information about the data set, INDATA, by using PROC CONTENTS. The encoding is LATIN1.

Here is the output of the data set.

```
The CONTENTS Procedure

      Data Set Name     MYLIB.INDATA         Observations          1
      Member Type       DATA                 Variables             1
      Engine            V9                   Indexes               0
      Created           05/12/2021 16:09:54  Observation Length    20
      Last Modified     05/12/2021 16:09:54  Deleted Observations  0
      Protection                             Compressed            NO
      Data Set Type                          Sorted                NO
      Label
      Data Representation  SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64,
LINUX_POWER_64
      Encoding          latin1  Western (ISO)
```

If the data set INDATA is printed from an EUC-JP session, CEDA warnings are issued because there are unsupported characters.

```
options msglevel=i;
proc print data=mylib.indata;
run;
```

SAS writes the following output to the log:

```
INFO: Data file MYLIB.INDATA.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.
WARNING: Some character data was lost during transcoding in the data set
"MYLIB.INDATA".
It contains one or more characters that are not available in the new encoding.
To avoid the transcoding error, please refer to "Troubleshooting Truncation and Data
loss Issue during Transcoding" in SAS National Language Support (NLS): reference
guide.
NOTE: There were 1 observations read from the data set MYLIB.INDATA.
INFO: One or more characters in the variable "a" at row number 1 in the
encoding "latin1" are not available in the encoding "euc-jp".
```

You can use the “KPROPDATA Function” to correct the unsupported characters.

```
data indata(encoding='euc-jp');
set mylib.indata(encoding=binary);              /*1
a=KPROPDATA(a,"QUESTION","LATIN1","EUC-JP");    /*2
run;

proc print data=indata;                         /*3
run;
```

1 Access the data set INDATA.

2 Replace ß with ? in the character string in variable a using the KPROPDATA function. LATIN1 is the input encoding and EUC-JP is the output encoding.

The KPROPDATA function replaces the unsupported characters, so there are no CEDA warnings in the log.

SAS writes the following output to the log:

```
NOTE: There were 1 observations read from the data set WORK.INDATA.
NOTE: The PROCEDURE PRINT printed page 2.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

The unsupported characters are replaced with a **?**.

```
data _null_;
set indata;              /*1
   put a=;               /*2
run;
```

1 Access the data set INDATA.

2 print the value of **a** to the log. The unsupported character has been replaced with a **?** because the character string was processed by the KPROPDATA function.

SAS writes the following output to the log:

```
a=201743WP?C?$
```

5

Double-Byte Character Sets (DBCS)

Overview to Double-Byte Character Sets (DBCS) 51
East Asian Languages 52
Specifying DBCS 52
Requirements for Displaying DBCS Character Sets 53
When You Can Use DBCS Features 53
DBCS and SAS on a Mainframe 54
SAS Data Conversion between DBCS Encodings 54
Avoiding Problems with Split DBCS Character Strings 55

Overview to Double-Byte Character Sets (DBCS)

Because East Asian languages have thousands of characters, double (two) bytes of information are needed to represent most characters.

Each East Asian language usually has more than one DBCS encoding system, due to nonstandardization among computer manufacturers. SAS processes the DBCS encoding information that is unique to each manufacturer for the major East Asian languages.

With the proper software extensions, you can use SAS for the following functions:

- display any of the major East Asian languages in the DBCS version of the SAS System
- import data from East Asian language computers and move the data from one application or operating environment to another (which might require SAS ACCESS or other SAS products)

- convert standard East Asian date and time notation to SAS date values, SAS time values, and SAS datetime values
- create data sets and various types of output (such as reports and graphs) that contain East Asian language characters.

East Asian Languages

East Asian languages include:

- Chinese, which is written in Simplified Chinese script, and is used in the People's Republic of China and Singapore
- Chinese, which is written in Traditional Chinese script, and is used in Hong Kong Special Administrative Region of the People's Republic of China (SAR), Macau SAR, and Taiwan
- Japanese
- Korean

Specifying DBCS

To specify DBCS, use the following SAS system options:

ENCODING
: recognizes default character-est encoding.

DBCS
: recognizes DBCS characters.

You can set the session encoding by using the ENCODING= system option, the DBCS options, or the LOCALE= system option. The ENCODING= system option has priority. For more information, see "Setting the Encoding of a SAS Session". Please see "ENCODING System Option".

Requirements for Displaying DBCS Character Sets

In order to display data sets that contain DBCS characters, you must have the following resources:

- system support for multiple code pages
- DBCS fonts that correspond to the language that you intend to use

If you need to create a user-defined character for use with SAS software, your computer must support DBCS. These computers have a limited availability in the U.S. and Europe. These East Asian language computer systems use various methods of creating the characters. In one popular method, the user enters the phonetic pronunciation of the character, often using Latin characters. The computer presents a menu of characters whose sounds are similar to the phonetic pronunciation and prompts the user to select one of them.

When You Can Use DBCS Features

After you have set up your SAS session to recognize a specific DBCS language and operating environment, you can work with your specified language in these general areas:

- the DATA step and batch-oriented procedures
- windowing and interactive capabilities
- cross-system connectivity and compatibility
- access to databases
- graphics

In a DATA step and in batch-oriented procedures, you can use DBCS wherever a text string within quotation marks is allowed. Variable values, variable labels, and data set labels can all be in DBCS. DBCS can also be used as input data and with range and label specifications in the FORMAT procedure. In WHERE expression processing, you can search for embedded DBCS text.

DBCS and SAS on a Mainframe

Another type of DBCS encoding exists on mainframe systems, which combine DBCS support with the 3270-style data stream. Each DBCS character string is surrounded by escape codes called shift out/shift in, or SO/SI. These codes originated from the need for the old-style printers to shift out from the EBCDIC character set, to the DBCS character set. The major manufacturers have different encodings for SO/SI; some manufacturers pad DBCS code with one byte of shift code information while others pad the DBCS code with two bytes of shift code information. These differences can cause problems in reading DBCS information about mainframes.

PCs, minicomputers, and workstations do not have SO/SI but have their own types of DBCS encodings that differ from manufacturer to manufacturer. SAS has several formats and informats that can read DBCS on SO/SI systems:

Table 5.1 *SAS Formats and Informats That Support DBCS on SO/SI Systems*

Keyword	Language Element	Description
$KANJI	informat	Removes SO/SI from Japanese kanji DBCS
$KANJIX	informat	Adds SO/SI to Japanese kanji DBCS
$KANJI	format	Adds SO/SI to Japanese kanji DBCS
$KANJIX	format	Removes SO/SI from Japanese kanji DBCS

SAS Data Conversion between DBCS Encodings

Normally, DBCS data that is generated on one computer system is incompatible with data generated on another computer system. SAS has features that allow conversion from one DBCS source to another, as shown in the following table.

Table 5.2 DBCS Conversions

Language Element	Type	Use	See
KCVT	function	Converts DBCS data from one operating environment to another	"KCVT Function" (p. 633)
CPORT	procedure	Moves files from one environment to another	*Base SAS Procedures Guide*
CIMPORT	procedure	Imports a transport file created by CPORT	*Base SAS Procedures Guide*

Avoiding Problems with Split DBCS Character Strings

- When working with DBCS characters, review your data to make sure that SAS recognizes the entire character string when data is imported or converted or used in a DATA step or a PROC step.
- On mainframe systems that use shift out/shift in escape codes, DBCS character strings can become truncated during conversion across operating environments.
- There is a possibility that DBCS character strings can be split when working with the PRINT, REPORT, TABULATE, and FREQ procedures. If undesirable splitting occurs, you might have to add spaces on either side of your DBCS character string to force the split to occur in a better place. The SPLIT= option can also be used with PROC REPORT and PROC PRINT to force string splitting in a better location.

6

Time Zones

Overview of SAS Time Zones 57

Specifying Time Zones in SAS 57
Definitions 57
About Time Zones in SAS 58
Tasks and SAS Language Elements for Time Zone Processing 60
Time Zone Example 65

Overview of SAS Time Zones

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
: Specify 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".

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".

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

Universal Coordinated Time (UTC)
: Specify 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".

When you specify a time zone, SAS shifts the datetime value according to the time zone and stores the new value. The time-zone-specific timestamp is displayed to the SAS log. 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.

This example shows how the datetime values change according to the time zone.

```
options tz='America/New_York';  1
data NY;
   NY_DT=datetime();  2
   format NY_DT datetime20.;  3
run;
options tz='Asia/Shanghai';  4
data SH;
   SH_DT=datetime();  5
   format SH_DT datetime20.;  6
run;

options tz='America/New_York';  7
proc contents data = NY nodetails;  8
run;
options tz='Asia/Shanghai';  9
proc contents data = NY nodetails;  10
run;
```

These statements correspond to the lines that are numbered in the preceding DATA and PROC steps.

1. Set the TZ option to America/New_York.
2. Assign the date and time to the variable NY_DT using the DATETIME function.
3. Modify the value of NY_DT using the DATETIME format.
4. Set the TZ option for the second DATA step to Asia/Shanghai.
5. Assign the date and time to the variable SH_DT using the DATETIME function.

6 Modify the value of SH_DT using the DATETIME format.

7 Set the TZ option for the first PROC CONTENTS step to America/New_ York.

8 Display information for the data set NY. The TZ option is set to America/New_York.

9 Set the TZ option for the second PROC CONTENTS step to Asia/Shanghai.

10 Display information for the data set NY with the TZ option that is set to Asia/Shanghai. Notice the time zone shift in the Created and Last Modified rows. You are displaying information about the same data set but with different time zones.

These statements produce this output. The first output is from the data set using the America/New_York time zone. The second output is from the data set using the Asia/Shanghai time zone.

Data Set Name	WORK.NY	Observations	1
Member Type	DATA	Variables	1
Engine	V9	Indexes	0
Created	27/02/2019 17:17:42	Observation Length	8
Last Modified	27/02/2019 17:17:42	Deleted Observations	0

Data Set Name	WORK.NY	Observations	1
Member Type	DATA	Variables	1
Engine	V9	Indexes	0
Created	28/02/2019 06:17:42	Observation Length	8
Last Modified	28/02/2019 06:17:42	Deleted Observations	0

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". For more information, see "TIMEZONE= System Option".

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. Here is the output in the SAS log:

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

```
-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".

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" and "TZONENAME Function" on page 740.

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 743.

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 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 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 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 "B8601DXw. Format", "B8601LXw. Format", "B8601TXw. Format", "E8601DXw. Format", "E8601LXw. Format", and "E8601TXw. Format".

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 "NLDATMTZw. Format", "NLDATMZw. Format", and "NLDATMWZw. Format" on page 246.

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 7
Dictionary of Autocall Macros for NLS 69

7

Dictionary of Autocall Macros for NLS

Autocall Macro Entries by Category 69

Dictionary 70
%COPY_TO_NEW_ENCODING Macro 70
%COPY_TO_UTF8 Macro 73
%KLOWCASE Autocall Macro 74
%QKLOWCAS Autocall Macro 75
%KTRIM Autocall Macro 75
%QKTRIM Autocall Macro 76
%KVERIFY Autocall Macro 77

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 Autocall Macro (p. 74)	Change uppercase characters to lowercase.
	%QKLOWCAS Autocall Macro (p. 75)	Change uppercase characters to lowercase.
	%KTRIM Autocall Macro (p. 75)	Trim trailing blanks.
	%QKTRIM Autocall Macro (p. 76)	Trim trailing blanks.

Category	Language Elements	Description
	%KVERIFY Autocall Macro (p. 77)	Returns the position of the first character unique to an expression.
Encoding	%COPY_TO_NEW_ENCODING Macro (p. 70)	Creates a new version of a data set with a specified encoding.
	%COPY_TO_UTF8 Macro (p. 73)	Creates a new version of a data set in UTF-8 encoding.

Dictionary

%COPY_TO_NEW_ENCODING Macro

Creates a new version of a data set with a specified encoding.

Category: Encoding

Notes: %COPY_TO_NEW_ENCODING must be run on a system that matches the source-data encoding.

Starting in 2022.1.2 the %COPY_TO_NEW_ENCODING macro is in the AUTOCALL library. Prior versions can access the source code in the SAS® 9.4 National Language Support (NLS): Reference Guide, Fifth Edition.

%COPY_TO_UTF8 does not transcode variables that have the transcode property set to NO.

Syntax

%COPY_TO_NEW_ENCODING(*from_dsname*, *to_dsname*, *new_encoding*)

Arguments

from_dsname
specifies the original data set name.

to_dsname
specifies the new data set name.

new_encoding
specifies the encoding of the new data set.

Note: Common encodings would be UTF-8 and LATIN1. See "Encodings and Their Aliases and Encoding Character Set Compatibility" for a list of encodings.

Details

Basics

The %COPY_TO_NEW_ENCODING macro creates a new version of a data set with a specified encoding. If the data set contains character variables whose values need larger lengths when transcoded to the specified encoding, then the DATA step creates a data set with the proper lengths. If the %COPY_TO_NEW_ENCODING macro is not used, the copy might fail because truncation of non-blanks is not allowed. For example, the following code, with a wlatin1 encoding, fails because the variable x needs three bytes to hold the transcoded UTF-8 representation:

Note: %COPY_TO_NEW_ENCODING must be run on a system that matches the source-data encoding.

For example, the following code, with a wlatin1 encoding, fails because the variable x needs three bytes to hold the transcoded UTF-8 representation:

```
data temp;
   length x  $1;
   x='€';
run;
data temp(encoding=utf8);
   set temp;
run;
```

The %COPY_TO_NEW_ENCODING macro produces this type of code:

```
   data temp(encoding=utf8);
    length x $3;
   set temp;
   run;
```

Examples

This example uses the SAS data set SASHELP.CLASS and creates a converted copy of that data set to be stored in the WORK library as the data set CLASS. The data set CLASS will have a UTF8 encoding.

Example 1: LATIN9 DATA to UTF-8

```
   %COPY_TO_NEW_ENCODING(sashelp.class, class, utf8);
```

This example shows how data is converted from LATIN9 to UTF-8.

```
   data indata;
```

```
   cost = 'A4392E3939'x;                        /*1*/
run;

%copy_to_new_encoding(indata, outdata, utf8);  /*2*/
```

1 Create a data set indata with a value of cost=A4392E3939. This value is the hexadecimal representation of €9.99 in LATIN9.

   The Euro(€) character in LATIN9 takes up a single byte 'A4'x and the length of the variable cost is 5 in the input data set. In UTF-8 encoding the same Euro character occupies 3 bytes 'E282AC'x so the length of the variable changes to 7.

2 The %copy_to_new_encoding macro converts the data in the input data set, indata, to a UTF-8 encoding and outputs the data to the output data set, outdata.

Example 2: %COPY_TO_UTF8 Macro Functionality

This example demonstrates the functionality of the %COPY_TO_UTF8 macro.

```
data mydsU8;                /*1*/
   set wlatin1.myds;
run;                                        /*2*/


%COPY_TO_UTF8(wlatin1,tempU8);              /*3*/

libname INU8CVP CVP "path_to_wlatin1_library"
CVPMULTIPLIER=4;    /*4*/
data tempU8;
length variable_name $ new_length;
set INU8CVP.wlatin1;
run;

libname INU8CVP clear ;
```

1 The code reads a WLATIN1 encoded data set from a UTF-8 SAS session.

2 SAS displays an error message if the variable lengths are not large enough to hold the transcoded values.

   Note: ERROR: Some character data was lost during transcoding in the data set "WLATIN1.MYDS". It contains one or more characters that are not available in the new encoding. To avoid the transcoding error, please refer to "Troubleshooting Truncation and Data loss Issue during Transcoding" in SAS National Language Support (NLS): reference guide.

3 %COPY_TO_UTF8 generates the data set by assigning a new length to the variables based on their actual data size and avoiding data truncation.

4 Generated code that is executed by %COPY_TO_UTF8.

%COPY_TO_UTF8 Macro

Creates a new version of a data set in UTF-8 encoding.

Category: Encoding

Notes: %COPY_TO_UTF8 must be run on a UTF-8 SAS session.

Starting in 2022.1.2 the %COPY_TO_UTF8 macro is in the AUTOCALL library. Prior versions can access the source code in the SAS® 9.4 National Language Support (NLS): Reference Guide, Fifth Edition

%COPY_TO_UTF8 uses the CVP engine to read a data set.

Truncation warnings might be displayed during execution of the %COPY_TO_UTF8 macro. These warnings are normal and expected.

%COPY_TO_UTF8 does not transcode variables that have the transcode property set to NO.

See: See "%COPY_TO_NEW_ENCODING Macro" for more information.

Syntax

%COPY_TO_UTF8(*from_dsname*, *to_dsname*);

Required Arguments

from_dsname
specifies the original data set name.

to_dsname
specifies the new data set name.

Note: %COPY_TO_UTF8 must be run from a UTF-8 session.

Details

The %COPY_TO_UTF8 macro creates a new version of a data set in UTF-8 encoding.

If the data set contains character variables whose values need larger lengths when transcoded to UTF-8, then the macro generates a SAS program that re-creates the data set with the proper lengths. If the data set has variables that do not need longer lengths when transcoding to UTF-8, then the data set remains the same.

%COPY_TO_UTF8 uses the CVP engine with the CVPMULTIPLIER=4 to read the specified data set to avoid truncation issues. Then, the macro transcodes each

character variable to determine the maximum length, in bytes, needed by the variable.

You might receive this warning when using the %COPY_TO_UTF8 macro: 'Libref _INU8CVP refers to the same physical library as IN.' 'IN' is a libref defined for the input dataset.

For example, the following code, with a WLATIN1 encoding, fails because the variable x needs three bytes to hold the transcoded UTF-8 representation:

```
data temp(encoding=wlatin1);
   length x  $1;
   x='€';
run;
data temp(encoding=utf8);
   set temp;
run;
```

%COPY_TO_UTF8 produces this type of code:

```
libname INU8CVP CVP path to work library;
data tempU8;
   length x  $3;
   set INU8CVP.temp;
run;
```

%KLOWCASE Autocall Macro

Change uppercase characters to lowercase.

Category: DBCS

Requirement: MAUTOSOURCE system option

See: "%QKLOWCAS Autocall Macro"

Syntax

%KLOWCASE (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.

%QKLOWCAS Autocall Macro

Change uppercase characters to lowercase.

Category: DBCS

Requirement: MAUTOSOURCE system option

See: "%KLOWCASE Autocall Macro"

Syntax

%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

Note: 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 Autocall Macro

Trim trailing blanks.

Category: DBCS

Requirement: MAUTOSOURCE system option

See: "%QKTRIM Autocall Macro"

Syntax

%KTRIM (*text | text expression*)

Required Argument

text
text or text expression.

Details

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

%QKTRIM Autocall Macro

Trim trailing blanks.

Category: DBCS

Requirement: MAUTOSOURCE system option

See: "%KTRIM Autocall Macro"

Syntax

%QKTRIM (*text | text expression*)

Required Argument

text
text or text expression.

Details

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

QKTRIM produces a result with these 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 8
Dictionary of Data Set Options for NLS 81

8

Dictionary of Data Set Options for NLS

Data Set Options by Category 81

Dictionary 82
ENCODING= Data Set Option 82
OUTREP= Data Set Option 85

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. 82)	Overrides the encoding to use for reading or writing a SAS data set.
	OUTREP= Data Set Option (p. 85)	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

Syntax

ENCODING= ANY | ASCIIANY | EBCDICANY | SESSION | *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.

SESSION
specifies the locale of the current Compute Server session.

The session value is supported in the OVERRIDE= option in the COPY procedure or COPY statement. For more information, see OVERRIDE=

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.

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.
- 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 normal access to the file uses a different session encoding.

For example, if the current session encoding is Wlatin1, 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;
```

Example 3: Overriding Encoding for Input Processing

For input processing, you can override the encoding that is specified in the file, and specify a different encoding.

For this example, the current session encoding is EBCDIC-870, but the file has the encoding value EBCDIC-1047 in the descriptor information. By specifying ENCODING=EBCDIC-870, SAS does not transcode the data, but instead displays the data using EBCDIC-870 encoding.

```
proc print data=myfiles.mixed (encoding=ebcdic870);
run;
```

See Also

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

Options in Statements and Commands:

- “ENCODING= Option” on page 1035
- “INENCODING=, OUTENCODING= Options” on page 1038

System Options:

- “ENCODING System Option” on page 982
- “LOCALE System Option” on page 987

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 Data Set Options: Reference*.

Syntax

OUTREP= *format*

PART 4

Formats for NLS

Chapter 9
Overview to NLS Formats 89

Chapter 10
Dictionary of Formats for NLS 107

9

Overview to NLS Formats

International Date and Datetime Formats 89

Currency Representation 90
- Overview to Currency 90
- U.S. Dollars 90
- Euros 91
- Customized Currency Representations 91
- Localized National and International Currency Representations 92
- Unique National and International Monetary Representations 94
- Example: Representing Currency in National and International Formats 97

Exceptions for Date and Time Default Widths 98

NL Formats that Use Uniform Patterns 100

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 9.1 *International Date and Datetime Formats*

English Format	International Format	Min	Max	Default
DATE.	NLDATE.	10	200	20
DATETIME.	NLDATM.	10	200	30
DOWNAME.	NLDATEWN.	4	200	9

English Format	International Format	Min	Max	Default
MONNAME.	NLDATEMN.	4	200	9
MONYY.	NLDATEYM.	6	200	10
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 DOLLARw.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 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:

- the lack of support for all currency symbols

- the reversal of the dot and comma for currency formatting is not used by all European countries.
- the appearance of the currency symbol varies by computer.

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.

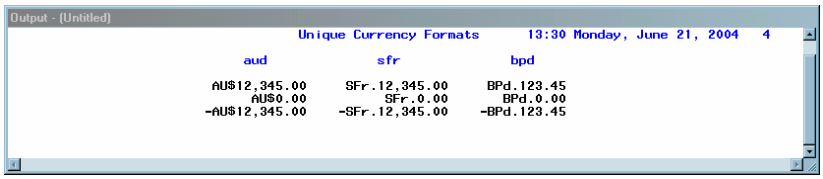
Customized Currency Representations

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 details about the FORMAT procedure, see *Base SAS Procedures Guide*.

Example Code 9.1 *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;
```



Customizing currency representations offers flexibility, but requires 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 9.2 *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	12 345.00 CHF
German_Germany	Euros	12.345,00 €
German_Luxembourg	Euros	12.345,00 €
Spanish_Spain	Euros	12.345,00 €
Spanish_Venezuela	Venezuelan bolivars	Bs.S12.345,00

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. Whereas French_France uses no thousands separator but uses a comma as a decimal separator, 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. Examples are 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.

Example:

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

```
Output:
USD12,345.00
```

Selected international currency representations follow:

Table 9.3 *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	VES12.345,00

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. Whereas the EMU countries put the currency code after the currency, 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 NLMNY*ISO*w.d and NLMNYI *ISO*w.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.

NLMNY and NLMNYI formats return results that display the number that is found in the data with a currency symbol that matches the current SAS LOCALE option. SAS does not convert the value from one currency to another. If the LOCALE changes, the same number is displayed with a different currency symbol. A changed currency symbol does not guarantee that the value has changed.

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_;
   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 9.4 *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

A currency symbol or a currency code precedes most currencies. Also, a comma is used 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 3-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 forms 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.

```
data _null_;
   x=12345;
   put x nlmniusd15.2;
run;
```

Output:

```
   USD12,345.00
```

Selected international currency representations follow:

Table 9.5 *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

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 9.2 *SAS Code That Formats National and International Currency Formats*

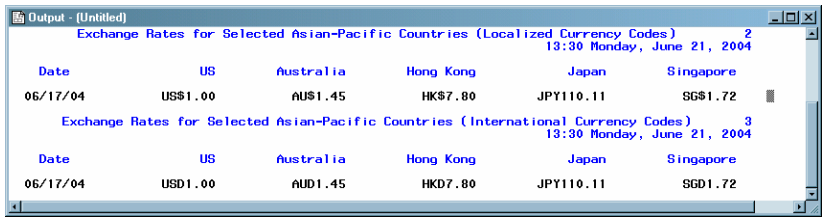
```
data curr;
input ex_date mmddyy. usd aud hkd jpy sgd 12.2;
datalines;
061704 1.00000 1.45349 7.79930 110.110 1.71900    1
;
proc print data=curr noobs label;
   var ex_date usd aud hkd jpy sgd;
   format ex_date mmddyy. usd nlmnlusd15.2 aud nlmnlaud15.2 hkd
nlmnlhkd15.2
jpy nlmnljpy15.2 sgd nlmnlsgd15.2;    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 9.1 National and International Format Output



Exceptions for Date and Time Default Widths

The length of the *month-name* and the *week-name* in some locale and encoding combinations might exceed the default width in some formats. Do not use the default width for these combinations. Refer to the following tables for the affected locales, encodings, and formats. Use the recommended widths for accurate output.

The following table lists the encoding and locale combinations where the localized output might exceed the default width.

Encoding	Locale
WLATIN1	FR_FR, IT_IT, DE_DE, ES_ES, EN_US,EN_GB
WLATIN2	CS_CS, HU_HU, PL_PL
WCYRILLIC	RU_RU
EUC_CN	EN_US & ZH_CN
SHIFT-JIS	EN_US JA_JP4-6
EUC-KR	EN_US KO_KR
BIG5	EN_US ZT_TW ZT_HK
UTF-8	all listed locales

The following table lists the recommended widths for encoding and locale combinations where the localized output might exceed the default width. You might need to use the recommended width in the specified formats.

Encoding	Locale	Format	Default Width	Recommended Width (>=)
WLATIN1	EN_GB	nldatmw	30	32
		nltimap	10	16
	EN_US	nltimap	10	11
WLATIN2	CS_CS	nltimap	10	11
EUC-CN	EN_US	nltimap	10	11
	ZH_CN	nldatmw	30	38
		nldatmwz	40	44
		nltimap	10	16
SHIFT-JIS	EN_US	nltimap	10	11
	JA_JP	nldatmw	30	34
		nltimap	10	16
EUC-KR	EN_US	nltimap	10	11
	KO_KR	nldatmw	30	37
		nldatmwz	40	43
		nltimap	10	17
BIG5	EN_US	nltimap	10	11
	ZT_HK	nltimap	10	11
	ZT_TW	nltimap	10	11
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

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

NL Formats that Use Uniform Patterns

NL formats that use uniform patterns help customers output data in different locales. You can specify regular, long, medium, and short patterns for output. If a formatted value cannot fit within the length specified for the NL format, a shorter pattern is used resulting in a different appearance. A uniform series is provided for

several of the NL formats, which selects the appropriate pattern to ensure that all data is presented in the same style within the specified width. See Table 9.7 for a list of NL formats that use uniform patterns. For example:

NLDATE
Writes a SAS date value as a date that is appropriate for the current SAS locale.

NLDATEL
Writes a SAS date value as a long-uniform pattern date in the form month, date, year that is appropriate for the current SAS locale.

NLDATEM
Writes a SAS date value as a date in a medium-uniform pattern that is appropriate for the current SAS locale.

NLDATES
Writes a SAS date value as a date string as a short-uniform pattern that is appropriate for the current SAS locale.

In this example NLDATE format writes the output for the dates specified in datalines. A width of 17 is specified in the NLDATE format.

```
options locale = English_UnitedStates;     1
data dval;                                 2
infile datalines;
input date mmddyy8.;
datalines;
01/25/02
02/25/02
03/24/02
04/25/02
05/25/02
06/24/02
07/25/02
08/25/02
09/24/02
10/25/02
11/25/02
12/24/02
;
run;
proc print data = dval;                    3
format date nldate17.;
run;
```

1 Specify the English_United States locale.

2 For data set dval, the contents of datalines are input as the variable *date* using the informat, MMDDYY with a width of 8.

3 Print the contents of dval using the format nldate with a width of 17.

These statements produce these results. The ninth Obs is 09/24/2023 because the ninth month is September. With a fixed width of 17, the format of the date value in the ninth observation differs from the others. This occurs because the format length is not long enough to display the month name, September, and the other elements of the date value. The NLDATE format selects a shorter pattern so that all elements can be displayed.

The SAS System

Obs	date
1	January 25, 2023
2	February 25, 2023
3	March 24, 2023
4	April 25, 2023
5	May 25, 2023
6	June 24, 2023
7	July 25, 2023
8	August 25, 2023
9	09/24/2023
10	October 25, 2023
11	November 25, 2023
12	December 24, 2023

The NL uniform series of formats are designed to assist with these scenarios. The format names of the uniform formats end with L, which use a long pattern, M for a medium length pattern, or S for a short pattern. The uniform formats ensure a consistent layout. They precalculate and predict all possible output lengths before determining the most appropriate pattern.

Table 9.6 shows results from the previous code where NLDATE is replaced with NLDATEL, NLDATEM, and NLDATES. The following proc print code uses the NLDATEL format instead of using NLDATE17. The width is not necessary since NLDATEL is used. The output for the ninth observation is September 24, 2023 and a width is not specified.

```
proc print data = dval;
format date nldatel.;
run;
```

Table 9.6 *NLDATEL, NLDATEM, NLDATES Output*

NLDATEL	NLDATEM	NLDATES

The SAS System

Obs	date
1	January 25, 2023
2	February 25, 2023
3	March 24, 2023
4	April 25, 2023
5	May 25, 2023
6	June 24, 2023
7	July 25, 2023
8	August 25, 2023
9	September 24, 2023
10	October 25, 2023
11	November 25, 2023
12	December 24, 2023

The SAS System

Obs	date
1	Jan 25, 2023
2	Feb 25, 2023
3	Mar 24, 2023
4	Apr 25, 2023
5	May 25, 2023
6	Jun 24, 2023
7	Jul 25, 2023
8	Aug 25, 2023
9	Sep 24, 2023
10	Oct 25, 2023
11	Nov 25, 2023
12	Dec 24, 2023

The SAS System

Obs	date
1	01/25/2023
2	02/25/2023
3	03/24/2023
4	04/25/2023
5	05/25/2023
6	06/24/2023
7	07/25/2023
8	08/25/2023
9	09/24/2023
10	10/25/2023
11	11/25/2023
12	12/24/2023

Here is a list of the NL formats that have uniform-pattern versions.

Table 9.7 *Formats That Have Uniform-Pattern Versions*

Format	Short Description	L (long-width format)	M (medium-width format)	S (short-width format)
NLDATE	Writes a SAS date value as a date that is appropriate for the current SAS locale.	NLDATEL	NLDATEM	NLDATES
NLDATEMD	Writes the SAS date value as the name of the month and the day of the month that is appropriate for the current SAS locale.	NLDATEMDL	NLDATEMDM	NLDATEMDS

Format	Short Description	L (long-width format)	M (medium-width format)	S (short-width format)
NLDATEYM	Writes the SAS date value as the year and the name of the month that is appropriate for the current SAS locale.	NLDATEYML	NLDATEYMM	NLDATEYMS
NLDATEYQ	Writes the SAS date value as the year and the quarter that is appropriate for the current SAS locale.	NLDATEYQL	NLDATEYQM	NLDATEYQS
NLDATM	Writes a SAS datetime value as a datetime that is appropriate for the current SAS locale.	NLDATML on page 231	NLDATMM	NLDATMS
NLDATMMD	Writes the SAS datetime value as the name of the month and the day of the month that is appropriate for the current SAS locale.	NLDATMMDL	NLDATMMDM	NLDATMMDS
NLDATMYM	Writes the SAS datetime value as the month and year that is appropriate for the current SAS locale.	NLDATMYML	NLDATMYMM	NLDATMYMS
NLDATMYQ	Writes the SAS datetime value as the quarter and the year that is appropriate for the current SAS locale.	NLDATMYQL	NLDATMYQM	NLDATMYQS

10

Dictionary of Formats for NLS

Categories of NLS Formats **112**

Dictionary **144**
B8601DXw. Format 144
B8601LXw. Format 146
B8601TXw. Format 148
BESTDOTXw. Format 150
$BIDIw. Format 151
$CPTDWw. Format 153
$CPTWDw. Format 154
DTWEEKVw. Format 155
$EBCDICw. Format 157
E8601DXw. Format 158
E8601LXw. Format 160
E8601TXw. Format 162
EUROw.d Format 164
EUROXw.d Format 167
HDATEw. Format 170
HEBDATEw. Format 171
JNENGOw.d Format 173
JNENGOTw. Format 174
JNENGOTW w. Format 175
JNENGOWw.d Format 176
$KANJIw. Format 178
$KANJIXw. Format 179
$LOGVSw. Format 180
$LOGVSRw. Format 182
MINGUOw. Format 183
NENGOw. Format 185
NLBESTw. Format 187
NLDw.p Format 189
NLDATEw. Format 191
NLDATECPw.p Format 193
NLDATELw. Format 196
NLDATEMw. Format 197
NLDATEMDw. Format 199
NLDATEMDLw. Format 200

NLDATEMDMw. Format 201
NLDATEMDSw. Format 202
NLDATEMNw. Format 203
NLDATESw. Format 205
NLDATEWw. Format 207
NLDATEWNw. Format 209
NLDATEYMw. Format 210
NLDATEYMLw. Format 212
NLDATEYMMw. Format 213
NLDATEYMSw. Format 214
NLDATEYQw. Format 215
NLDATEYQLw. Format 217
NLDATEYQMw. Format 218
NLDATEYQSw. Format 219
NLDATEYRw. Format 220
NLDATEYWw. Format 221
NLDATMw. Format 223
NLDATMAPw. Format 224
NLDATMCPw.p Format 226
NLDATMDTw. Format 230
NLDATMLw. Format 231
NLDATMMw. Format 232
NLDATMMDw. Format 233
NLDATMMDLw. Format 235
NLDATMMDMw. Format 236
NLDATMMDSw. Format 237
NLDATMMNw. Format 238
NLDATMSw. Format 240
NLDATMTMw. Format 241
NLDATMTZw. Format 242
NLDATMWw. Format 243
NLDATMWNw. Format 245
NLDATMWZw. Format 246
NLDATMYMw. Format 247
NLDATMYMLw. Format 248
NLDATMYMMw. Format 249
NLDATMYMSw. Format 251
NLDATMYQw. Format 252
NLDATMYQLw. Format 253
NLDATMYQMw. Format 254
NLDATMYQSw. Format 256
NLDATMYRw. Format 257
NLDATMYWw. Format 258
NLDATMZw. Format 259
NLMNACIAED w.d Format 260
NLMNACIAUD w.d Format 261
NLMNACIBGN w.d Format 263
NLMNACIBRL w.d Format 264
NLMNACICAD w.d Format 266
NLMNACICHF w.d Format 267
NLMNACICNY w.d Format 269
NLMNACICZK w.d Format 270
NLMNACIDKK w.d Format 272
NLMNACIEEK w.d Format 273

NLMNACIEGPw.d Format 275
NLMNACIEURw.d Format 276
NLMNACIGBPw.d Format 278
NLMNACIHKDw.d Format 279
NLMNACIHRKw.d Format 281
NLMNACIHUFw.d Format 282
NLMNACIIDRw.d Format 284
NLMNACIILSw.d Format 285
NLMNACIINRw.d Format 287
NLMNACIJPYw.d Format 288
NLMNACIKRWw.d Format 290
NLMNACILTLw.d Format 291
NLMNACILVLw.d Format 293
NLMNACIMOPw.d Format 294
NLMNACIMXNw.d Format 296
NLMNACIMYRw.d Format 297
NLMNACINOKw.d Format 299
NLMNACINZDw.d Format 300
NLMNACIPLNw.d Format 302
NLMNACIRUBw.d Format 303
NLMNACISEKw.d Format 305
NLMNACISGDw.d Format 306
NLMNACITHBw.d Format 308
NLMNACITRYw.d Format 309
NLMNACITWDw.d Format 311
NLMNACIUSDw.d Format 312
NLMNACIZARw.d Format 314
NLMNACLAEDx.d Format 315
NLMNACLAUDw.d Format 317
NLMNACLBGNw.d Format 318
NLMNACLBRLw.d Format 320
NLMNACLCADw.d Format 321
NLMNACLCHFw.d Format 323
NLMNACLCNYw.d Format 324
NLMNACLCZKw.d Format 326
NLMNACLDKKw.d Format 327
NLMNACLEEKw.d Format 329
NLMNACLEGPw.d Format 330
NLMNACLEURw.d Format 332
NLMNACLGBPw.d Format 333
NLMNACLHKDw.d Format 335
NLMNACLHRKw.d Format 336
NLMNACLHUFw.d Format 338
NLMNACLIDRw.d Format 339
NLMNACLILSw.d Format 341
NLMNACLINRw.d Format 342
NLMNACLJPYw.d Format 344
NLMNACLKRWw.d Format 345
NLMNACLLTLw.d Format 347
NLMNACLLVLw.d Format 348
NLMNACLMOPw.d Format 350
NLMNACLMXNw.d Format 351
NLMNACLMYRw.d Format 353
NLMNACLNOKw.d Format 354

NLMNACLNZDw.d Format 356
NLMNACLPLNw.d Format 357
NLMNACLRUBw.d Format 359
NLMNACLSEKw.d Format 360
NLMNACLSGDw.d Format 362
NLMNACLTHBw.d Format 363
NLMNACLTRYw.d Format 365
NLMNACLTWDw.d Format 366
NLMNACLUSDw.d Format 368
NLMNACLZARw.d Format 369
NLMNIAEDw.d Format 371
NLMNIAUDw.d Format 372
NLMNIBGNw.d Format 373
NLMNIBRLw.d Format 375
NLMNICADw.d Format 376
NLMNICHFw.d Format 378
NLMNICNYw.d Format 379
NLMNICZKw.d Format 380
NLMNIDKKw.d Format 382
NLMNIEEKw.d Format 383
NLMNIEGPw.d Format 385
NLMNIEURw.d Format 386
NLMNIGBPw.d Format 388
NLMNIHKDw.d Format 389
NLMNIHRKw.d Format 390
NLMNIHUFw.d Format 392
NLMNIIDRw.d Format 393
NLMNIILSw.d Format 395
NLMNIINRw.d Format 396
NLMNIJPYw.d Format 397
NLMNIKRWw.d Format 399
NLMNILTLw.d Format 400
NLMNILVLw.d Format 402
NLMNIMOPw.d Format 403
NLMNIMXNw.d Format 404
NLMNIMYRw.d Format 406
NLMNINOKw.d Format 407
NLMNINZDw.d Format 409
NLMNIPLNw.d Format 410
NLMNIRUBw.d Format 411
NLMNISEKw.d Format 413
NLMNISGDw.d Format 414
NLMNITHBw.d Format 416
NLMNITRYw.d Format 417
NLMNITWDw.d Format 418
NLMNIUSDw.d Format 420
NLMNIZARw.d Format 421
NLMNLAEDx.d Format 423
NLMNLAUDw.d Format 424
NLMNLBGNw.d Format 425
NLMNLBRLw.d Format 427
NLMNLCADw.d Format 428
NLMNLCHFw.d Format 430
NLMNLCNYw.d Format 431

NLMNLCZKw.d Format ... 432
NLMNLDKKw.d Format ... 434
NLMNLEEKw.d Format ... 435
NLMNLEGPw.d Format ... 437
NLMNLEURw.d Format ... 438
NLMNLGBPw.d Format ... 440
NLMNLHKDw.d Format ... 441
NLMNLHRKw.d Format ... 443
NLMNLHUFw.d Format ... 444
NLMNLIDRw.d Format ... 445
NLMNLILSw.d Format ... 447
NLMNLINRw.d Format ... 448
NLMNLJPYw.d Format ... 450
NLMNLKRWw.d Format ... 451
NLMNLLTLw.d Format ... 452
NLMNLLVLw.d Format ... 454
NLMNLMOPw.d Format ... 455
NLMNLMXNw.d Format ... 457
NLMNLMYRw.d Format ... 458
NLMNLNOKw.d Format ... 459
NLMNLNZDw.d Format ... 461
NLMNLPLNw.d Format ... 462
NLMNLRUBw.d Format ... 464
NLMNLSEKw.d Format ... 465
NLMNLSGDw.d Format ... 467
NLMNLTHBw.d Format ... 468
NLMNLTRYw.d Format ... 469
NLMNLTWDw.d Format ... 471
NLMNLUSDw.d Format ... 472
NLMNLZARw.d Format ... 474
NLMNYw.d Format ... 475
NLMNYACw.d Format ... 477
NLMNYIw.d Format ... 479
NLMNYIACw.d Format ... 481
NLNUMw.d Format ... 483
NLNUMIw.d Format ... 485
NLPCTw.d Format ... 487
NLPCTIw.d Format ... 489
NLPCTNw.d Format ... 490
NLPCTPw.d Format ... 492
NLPVALUEw.d Format ... 493
NLSTRMONw.d Format ... 494
NLSTRQTRw.d Format ... 496
NLSTRWKw.d Format ... 497
NLTIMAPw. Format ... 499
NLTIMEw. Format ... 500
NLTIMELw. Format ... 502
NLTIMEMw. Format ... 504
NLTIMESw. Format ... 505
$UCS2Bw. Format ... 507
$UCS2BEw. Format ... 508
$UCS2Lw. Format ... 510
$UCS2LEw. Format ... 511
$UCS2Xw. Format ... 513

$UCS2XEw. Format 514
$UCS4Bw. Format 516
$UCS4BEw. Format 517
$UCS4Lw. Format 519
$UCS4LEw. Format 521
$UCS4Xw. Format 522
$UCS4XEw. Format 524
$UESCw. Format 525
$UESCEw. Format 527
$UNCRw. Format 528
$UNCREw. Format 529
$UPARENw. Format 531
$UPARENEw. Format 532
$UTF8Xw. Format 533
$UTF8XEw. Format 534
$VSLOGw. Format 536
$VSLOGRw. Format 537
WEEKUw. Format 539
WEEKVw. Format 541
WEEKWw. Format 543
YENw.d Format 545
YYWEEKUw. Format 546
YYWEEKVw. Format 548
YYWEEKWw. Format 550

Categories of NLS Formats

The following categories relate to NLS issues:

Category	Description
BIDI Text Handling	Instructs SAS to write bidirectional data values from data variables. See BIDI Text Handling for a list of formats.
CAS	Instructs SAS that these formats run on the CAS server. See CAS for a list of formats.
Character	Instructs SAS to write character data values from character variables. See Character for a list of formats.
DBCS	Instructs SAS to translate double-byte-character sets that are used in Asian languages. See DBCS for a list of formats.

Category	Description
Hebrew Text Handling	Instructs SAS to read Hebrew data from data variables. See Hebrew Text Handling for a list of formats.
Numeric	Instructs SAS to write numeric data values from numeric variables. See Numeric for a list of formats.

Category	Language Elements	Description
BIDI Text Handling	$BIDIw. Format (p. 151)	Converts between a logically ordered string and a visually ordered string, by reversing the order of Hebrew and Arabic characters while preserving the order of Latin words and numbers.
	$LOGVSw. Format (p. 180)	Processes a character string that is in left-to-right-logical order, and then writes the character string in visual order.
	$LOGVSRw. Format (p. 182)	Processes a character string that is in right-to-left-logical order, and then writes the character string in visual order.
	$VSLOGw. Format (p. 536)	Processes a character string that is in visual order, and then writes the character string in left-to-right logical order.
	$VSLOGRw. Format (p. 537)	Processes a character string that is in visual order, and then writes the character string in right-to-left logical order.
CAS	DTWEEKVw. Format (p. 155)	Writes a week and datetime number in decimal format by using the V algorithm.
	EUROw.d Format (p. 164)	Writes numeric values with a leading euro symbol (E), a comma that separates every three digits, and a period that separates the decimal fraction.
	EUROXw.d Format (p. 167)	Writes numeric values with a leading euro symbol (E), a period that separates every three digits, and a comma that separates the decimal fraction.
	$LOGVSw. Format (p. 180)	Processes a character string that is in left-to-right-logical order, and then writes the character string in visual order.
	$LOGVSRw. Format (p. 182)	Processes a character string that is in right-to-left-logical order, and then writes the character string in visual order.
	NENGOw. Format (p. 185)	Writes date values as Japanese dates in the form e.yymmdd.
	NLBESTw. Format (p. 187)	Writes the best numerical notation based on the locale.
	NLDw.p Format (p. 189)	Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude.

Category	Language Elements	Description
	NLDATEw. Format (p. 191)	Writes a SAS date value as a date that is appropriate for the current SAS locale.
	NLDATECPw.p Format (p. 193)	Converts a SAS date value to a locale-sensitive-compact-interval date format.
	NLDATELw. Format (p. 196)	Writes a SAS date value as a date in the form month, date, year that is appropriate for the current SAS locale.
	NLDATEMw. Format (p. 197)	Writes a SAS date value as a date in a medium-uniform pattern that is appropriate for the current SAS locale.
	NLDATEMDw. Format (p. 199)	Writes the SAS date value as the name of the month and the day of the month that is appropriate for the current SAS locale.
	NLDATEMDLw. Format (p. 200)	Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale.
	NLDATEMDMw. Format (p. 201)	Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale.
	NLDATEMDSw. Format (p. 202)	Writes a SAS date value as the month and day of the month in a short-uniform pattern that is appropriate for the current SAS locale.
	NLDATEMNw. Format (p. 203)	Writes a SAS date value as the name of the month that is appropriate for the current SAS locale.
	NLDATESw. Format (p. 205)	Writes a SAS date value as a date string that is appropriate for the current SAS locale.
	NLDATEWw. Format (p. 207)	Writes a SAS date value as the date and the day of the week that is appropriate for the current SAS locale.
	NLDATEWNw. Format (p. 209)	Writes the SAS date value as the day of the week that is appropriate for the current SAS locale.
	NLDATEYMw. Format (p. 210)	Writes the SAS date value as the year and the name of the month that is appropriate for the current SAS locale.
	NLDATEYMLw. Format (p. 212)	Writes a SAS date value as the month and year that is appropriate for the current SAS locale.
	NLDATEYMMw. Format (p. 213)	Writes a SAS date value as the month and year with abbreviations that is appropriate for the current SAS locale.
	NLDATEYMSw. Format (p. 214)	Writes a SAS date value as a date and year that is appropriate for the current SAS locale.
	NLDATEYQw. Format (p. 215)	Writes the SAS date value as the year and the quarter that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLDATEYQLw. Format (p. 217)	Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale.
	NLDATEYQMw. Format (p. 218)	Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale.
	NLDATEYQSw. Format (p. 219)	Writes a SAS date value as the year and the year's quarter value (1–4) with numbers and delimiters that is appropriate for the current SAS locale.
	NLDATEYRw. Format (p. 220)	Writes the SAS date value as the year that is appropriate for the current SAS locale.
	NLDATEYWw. Format (p. 221)	Writes the SAS date value as the year and the week that is appropriate for the current SAS locale.
	NLDATMw. Format (p. 223)	Writes a SAS datetime value as a date time that is appropriate for the current SAS locale.
	NLDATMAPw. Format (p. 224)	Writes a SAS datetime value as a datetime with a.m. or p.m.that is appropriate for the current SAS locale.
	NLDATMCPw.p Format (p. 226)	Converts a SAS datetime value to a locale-sensitive-compact-interval datetime format.
	NLDATMDTw. Format (p. 230)	Writes the SAS datetime value as the name of the month, day of the month and year that is appropriate for the current SAS locale.
	NLDATMLw. Format (p. 231)	Writes a SAS datetime value as a long representation of the date that is appropriate for the current SAS locale.
	NLDATMMw. Format (p. 232)	Writes a SAS datetime value as a medium representation of the date that is appropriate for the current SAS locale.
	NLDATMMDw. Format (p. 233)	Writes the SAS datetime value as the name of the month and the day of the month that is appropriate for the current SAS locale.
	NLDATMMDLw. Format (p. 235)	Writes a SAS datetime value as the full-length of the month and day of the month that is appropriate for the current SAS locale.
	NLDATMMDMw. Format (p. 236)	Writes a SAS datetime value as the month and day of the month using abbreviations that is appropriate for the current SAS locale.
	NLDATMMDSw. Format (p. 237)	Writes a SAS datetime value as the month and day of the month using numbers and delimiters that is appropriate for the current SAS locale.
	NLDATMMNw. Format (p. 238)	Writes the SAS datetime value as the name of the month that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLDATMSw. Format (p. 240)	Writes a SAS datetime value as the short representation of the date that is appropriate for the current SAS locale.
	NLDATMTMw. Format (p. 241)	Writes the time portion of a SAS datetime value as the time of day that is appropriate for the current SAS locale.
	NLDATMTZw. Format (p. 242)	Writes the time portion of the SAS datetime value as the time of day and time zone that is appropriate for the current SAS locale.
	NLDATMWw. Format (p. 243)	Writes SAS datetime values as day of the week and the datetime that is appropriate for the current SAS locale.
	NLDATMWNw. Format (p. 245)	Writes a SAS datetime value as the day of the week that is appropriate for the current SAS locale.
	NLDATMWZw. Format (p. 246)	Writes SAS datetime values as a day-of-week, datetime, and time zone value that is appropriate for the current SAS locale.
	NLDATMYMw. Format (p. 247)	Writes the SAS datetime value as the month and year that is appropriate for the current SAS locale.
	NLDATMYMLw. Format (p. 248)	Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale.
	NLDATMYMMw. Format (p. 249)	Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale.
	NLDATMYMSw. Format (p. 251)	Writes a SAS datetime value as the month and year with numbers and a delimiter that is appropriate for the current SAS locale.
	NLDATMYQw. Format (p. 252)	Writes the SAS datetime value as the quarter and the year that is appropriate for the current SAS locale.
	NLDATMYQLw. Format (p. 253)	Writes a SAS datetime value as the year's quarter value (1–4) and the year that is appropriate for the current SAS locale.
	NLDATMYQMw. Format (p. 254)	Writes a SAS datetime value as the year's quarter (1–4) and year that is appropriate for the current SAS locale.
	NLDATMYQSw. Format (p. 256)	Writes a SAS datetime value as the year and the quarter (1-4) using numbers and a delimiter that is appropriate for the current SAS locale.
	NLDATMYRw. Format (p. 257)	Writes the SAS datetime value as the year that is appropriate for the current SAS locale.
	NLDATMYWw. Format (p. 258)	Writes the SAS datetime value as the week number and the year that is appropriate for the current SAS locale.
	NLDATMZw. Format (p. 259)	Writes SAS datetime values as datetime and time zone that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLMNACIAED w.d Format (p. 260)	Writes the monetary format of the international accounting expression for the United Arab Emirates.
	NLMNACIAUD w.d Format (p. 261)	Writes the monetary format of the international accounting expression for Australia.
	NLMNACIBGN w.d Format (p. 263)	Writes the monetary format of the international accounting expression for Bulgaria.
	NLMNACIBRL w.d Format (p. 264)	Writes the monetary format of the international accounting expression for Brazil.
	NLMNACICAD w.d Format (p. 266)	Writes the monetary format of the international accounting expression for Canada.
	NLMNACICHF w.d Format (p. 267)	Writes the monetary format of the international accounting expression for Liechtenstein and Switzerland.
	NLMNACICNY w.d Format (p. 269)	Writes the monetary format of the international accounting expression for China.
	NLMNACICZK w.d Format (p. 270)	Writes the monetary format of the international accounting expression for Czechia.
	NLMNACIDKK w.d Format (p. 272)	Writes the monetary format of the international accounting expression for Denmark, Faroe Island, and Greenland.
	NLMNACIEEK w.d Format (p. 273)	Writes the monetary format of the international accounting expression for Estonia.
	NLMNACIEGPw.d Format (p. 275)	Writes the monetary format of the international accounting expression for Egypt.
	NLMNACIEURw.d Format (p. 276)	Writes the monetary format of the international accounting expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNACIGBPw.d Format (p. 278)	Writes the monetary format of the international accounting expression for the United Kingdom.
	NLMNACIHKDw.d Format (p. 279)	Writes the monetary format of the international accounting expression for Hong Kong.
	NLMNACIHRKw.d Format (p. 281)	Writes the monetary format of the international accounting expression for Croatia.
	NLMNACIHUFw.d Format (p. 282)	Writes the monetary format of the international accounting expression for Hungary.
	NLMNACIIDRw.d Format (p. 284)	Writes the monetary format of the international accounting expression for Indonesia.

Category	Language Elements	Description
	NLMNACIILSw.d Format (p. 285)	Writes the monetary format of the international accounting expression for Israel.
	NLMNACIINRw.d Format (p. 287)	Writes the monetary format of the international accounting expression for India.
	NLMNACIJPYw.d Format (p. 288)	Writes the monetary format of the international accounting expression for Japan.
	NLMNACIKRWw.d Format (p. 290)	Writes the monetary format of the international accounting expression for South Korea.
	NLMNACILTLw.d Format (p. 291)	Writes the monetary format of the international accounting expression for Lithuania.
	NLMNACILVLw.d Format (p. 293)	Writes the monetary format of the international accounting expression for Latvia.
	NLMNACIMOPw.d Format (p. 294)	Writes the monetary format of the international accounting expression for Macau.
	NLMNACIMXNw.d Format (p. 296)	Writes the monetary format of the international accounting expression for Mexico.
	NLMNACIMYRw.d Format (p. 297)	Writes the monetary format of the international accounting expression for Malaysia.
	NLMNACINOKw.d Format (p. 299)	Writes the monetary format of the international accounting expression for Norway.
	NLMNACINZDw.d Format (p. 300)	Writes the monetary format of the international accounting expression for New Zealand.
	NLMNACIPLNw.d Format (p. 302)	Writes the monetary format of the international accounting expression for Poland.
	NLMNACIRUBw.d Format (p. 303)	Writes the monetary format of the international accounting expression for Russia.
	NLMNACISEKw.d Format (p. 305)	Writes the monetary format of the international accounting expression for Sweden.
	NLMNACISGDw.d Format (p. 306)	Writes the monetary format of the international accounting expression for Singapore.
	NLMNACITHBw.d Format (p. 308)	Writes the monetary format of the international accounting expression for Thailand.
	NLMNACITRYw.d Format (p. 309)	Writes the monetary format of the international accounting expression for Turkey.
	NLMNACITWDw.d Format (p. 311)	Writes the monetary format of the international accounting expression for Taiwan.

Category	Language Elements	Description
	NLMNACIUSDw.d Format (p. 312)	Writes the monetary format of the international accounting expression for Puerto Rico and the United States.
	NLMNACIZARw.d Format (p. 314)	Writes the monetary format of the international accounting expression for South Africa.
	NLMNACLAEDx.d Format (p. 315)	Writes the monetary format of the local accounting expression for the United Arab Emirates.
	NLMNACLAUDw.d Format (p. 317)	Writes the monetary format of the local accounting expression for Australia.
	NLMNACLBGNw.d Format (p. 318)	Writes the monetary format of the local standard expression for Bulgaria.
	NLMNACLBRLw.d Format (p. 320)	Writes the monetary format of the local expression for Brazil.
	NLMNACLCADw.d Format (p. 321)	Writes the monetary format of the local accounting expression for Canada.
	NLMNACLCHFw.d Format (p. 323)	Writes the monetary format of the local accounting expression for Liechtenstein and Switzerland.
	NLMNACLCNYw.d Format (p. 324)	Writes the monetary format of the local accounting expression for China.
	NLMNACLCZKw.d Format (p. 326)	Writes the monetary format of the local accounting expression for Czechia.
	NLMNACLDKKw.d Format (p. 327)	Writes the monetary format of the local accounting expression for Denmark, Faroe Island, and Greenland.
	NLMNACLEEKw.d Format (p. 329)	Writes the monetary format of the local accounting expression for Estonia.
	NLMNACLEGPw.d Format (p. 330)	Writes the monetary format of the local accounting expression for Egypt.
	NLMNACLEURw.d Format (p. 332)	Writes the monetary format of the local accounting expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNACLGBPw.d Format (p. 333)	Writes the monetary format of the local accounting expression for the United Kingdom.
	NLMNACLHKDw.d Format (p. 335)	Writes the monetary format of the local accounting expression for Hong Kong.
	NLMNACLHRKw.d Format (p. 336)	Writes the monetary format of the local accounting expression for Croatia.

Category	Language Elements	Description
	NLMNACLHUFw.d Format (p. 338)	Writes the monetary format of the local accounting expression for Hungary.
	NLMNACLIDRw.d Format (p. 339)	Writes the monetary format of the local accounting expression for Indonesia.
	NLMNACLILSw.d Format (p. 341)	Writes the monetary format of the local accounting expression for Israel.
	NLMNACLINRw.d Format (p. 342)	Writes the monetary format of the local accounting expression for India.
	NLMNACLJPYw.d Format (p. 344)	Writes the monetary format of the local accounting expression for Japan.
	NLMNACLKRWw.d Format (p. 345)	Writes the monetary format of the local accounting expression for South Korea.
	NLMNACLLTLw.d Format (p. 347)	Writes the monetary format of the local accounting expression for Lithuania.
	NLMNACLLVLw.d Format (p. 348)	Writes the monetary format of the local accounting expression for Latvia.
	NLMNACLMOPw.d Format (p. 350)	Writes the monetary format of the local accounting expression for Macau.
	NLMNACLMXNw.d Format (p. 351)	Writes the monetary format of the local accounting expression for Mexico.
	NLMNACLMYRw.d Format (p. 353)	Writes the monetary format of the local accounting expression for Malaysia.
	NLMNACLNOKw.d Format (p. 354)	Writes the monetary format of the local accounting expression for Norway.
	NLMNACLNZDw.d Format (p. 356)	Writes the monetary format of the local accounting expression for New Zealand.
	NLMNACLPLNw.d Format (p. 357)	Writes the monetary format of the local accounting expression for Poland.
	NLMNACLRUBw.d Format (p. 359)	Writes the monetary format of the local accounting expression for Russia.
	NLMNACLSEKw.d Format (p. 360)	Writes the monetary format of the local accounting expression for Sweden.
	NLMNACLSGDw.d Format (p. 362)	Writes the monetary format of the local accounting expression for Singapore.
	NLMNACLTHBw.d Format (p. 363)	Writes the monetary format of the local accounting expression for Thailand.

Category	Language Elements	Description
	NLMNACLTRYw.d Format (p. 365)	Writes the monetary format of the local accounting expression for Turkey.
	NLMNACLTWDw.d Format (p. 366)	Writes the monetary format of the local accounting expression for Taiwan.
	NLMNACLUSDw.d Format (p. 368)	Writes the monetary format of the local accounting expression for Puerto Rico and the United States.
	NLMNACLZARw.d Format (p. 369)	Writes the monetary format of the local accounting expression for South Africa.
	NLMNIAEDw.d Format (p. 371)	Writes the monetary format of the international standard expression for the United Arab Emirates.
	NLMNIAUDw.d Format (p. 372)	Writes the monetary format of the international expression for Australia.
	NLMNIBGNw.d Format (p. 373)	Writes the monetary format of the international standard expression for Bulgaria.
	NLMNIBRLw.d Format (p. 375)	Writes the monetary format of the international standard expression for Brazil.
	NLMNICADw.d Format (p. 376)	Writes the monetary format of the international standard expression for Canada.
	NLMNICHFw.d Format (p. 378)	Writes the monetary format of the international standard expression for Liechtenstein and Switzerland.
	NLMNICNYw.d Format (p. 379)	Writes the monetary format of the international standard expression for China.
	NLMNICZKw.d Format (p. 380)	Writes the monetary format of the international standard expression for Czechia.
	NLMNIDKKw.d Format (p. 382)	Writes the monetary format of the international standard expression for Denmark, Faroe Island, and Greenland.
	NLMNIEEKw.d Format (p. 383)	Writes the monetary format of the international standard expression for Estonia.
	NLMNIEGPw.d Format (p. 385)	Writes the monetary format of the international standard expression for Egypt.
	NLMNIEURw.d Format (p. 386)	Writes the monetary format of the international standard expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNIGBPw.d Format (p. 388)	Writes the monetary format of the international standard expression for the United Kingdom.

Category	Language Elements	Description
	NLMNIHKDw.d Format (p. 389)	Writes the monetary format of the international standard expression for Hong Kong.
	NLMNIHRKw.d Format (p. 390)	Writes the monetary format of the international standard expression for Croatia.
	NLMNIHUFw.d Format (p. 392)	Writes the monetary format of the international standard expression for Hungary.
	NLMNIIDRw.d Format (p. 393)	Writes the monetary format of the international standard expression for Indonesia.
	NLMNIILSw.d Format (p. 395)	Writes the monetary format of the international standard expression for Israel.
	NLMNIINRw.d Format (p. 396)	Writes the monetary format of the international standard expression for India.
	NLMNIJPYw.d Format (p. 397)	Writes the monetary format of the international standard expression for Japan.
	NLMNIKRWw.d Format (p. 399)	Writes the monetary format of the international standard expression for South Korea.
	NLMNILTLw.d Format (p. 400)	Writes the monetary format of the international standard expression for Lithuania.
	NLMNILVLw.d Format (p. 402)	Writes the monetary format of the international standard expression for Latvia.
	NLMNIMOPw.d Format (p. 403)	Writes the monetary format of the international standard expression for Macau.
	NLMNIMXNw.d Format (p. 404)	Writes the monetary format of the international standard expression for Mexico.
	NLMNIMYRw.d Format (p. 406)	Writes the monetary format of the international standard expression for Malaysia.
	NLMNINOKw.d Format (p. 407)	Writes the monetary format of the international standard expression for Norway.
	NLMNINZDw.d Format (p. 409)	Writes the monetary format of the international standard expression for New Zealand.
	NLMNIPLNw.d Format (p. 410)	Writes the monetary format of the international standard expression for Poland.
	NLMNIRUBw.d Format (p. 411)	Writes the monetary format of the international standard expression for Russia.
	NLMNISEKw.d Format (p. 413)	Writes the monetary format of the international standard expression for Sweden.

Category	Language Elements	Description
	NLMNISGDw.d Format (p. 414)	Writes the monetary format of the international standard expression for Singapore.
	NLMNITHBw.d Format (p. 416)	Writes the monetary format of the international standard expression for Thailand.
	NLMNITRYw.d Format (p. 417)	Writes the monetary format of the international standard expression for Turkey.
	NLMNITWDw.d Format (p. 418)	Writes the monetary format of the international standard expression for Taiwan.
	NLMNIUSDw.d Format (p. 420)	Writes the monetary format of the international standard expression for Puerto Rico and the United States.
	NLMNIZARw.d Format (p. 421)	Writes the monetary format of the international standard expression for South Africa.
	NLMNLAEDx.d Format (p. 423)	Writes the monetary format of the local standard expression for the United Arab Emirates.
	NLMNLAUDw.d Format (p. 424)	Writes the monetary format of the local standard expression for Australia.
	NLMNLBGNw.d Format (p. 425)	Writes the monetary format of the local standard expression for Bulgaria.
	NLMNLBRLw.d Format (p. 427)	Writes the monetary format of the local standard expression for Brazil.
	NLMNLCADw.d Format (p. 428)	Writes the monetary format of the local standard expression for Canada.
	NLMNLCHFw.d Format (p. 430)	Writes the monetary format of the local standard expression for Liechtenstein and Switzerland.
	NLMNLCNYw.d Format (p. 431)	Writes the monetary format of the local standard expression for China.
	NLMNLCZKw.d Format (p. 432)	Writes the monetary format of the local standard expression for Czechia.
	NLMNLDKKw.d Format (p. 434)	Writes the monetary format of the local standard expression for Denmark, Faroe Island, and Greenland.
	NLMNLEEKw.d Format (p. 435)	Writes the monetary format of the local standard expression for Estonia.
	NLMNLEGPw.d Format (p. 437)	Writes the monetary format of the local standard expression for Egypt.
	NLMNLEURw.d Format (p. 438)	Writes the monetary format of the local standard expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania,

Category	Language Elements	Description
		Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNLGBPw.d Format (p. 440)	Writes the monetary format of the local standard expression for the United Kingdom.
	NLMNLHKDw.d Format (p. 441)	Writes the monetary format of the local standard expression for Hong Kong.
	NLMNLHRKw.d Format (p. 443)	Writes the monetary format of the local standard expression for Croatia.
	NLMNLHUFw.d Format (p. 444)	Writes the monetary format of the local standard expression for Hungary.
	NLMNLIDRw.d Format (p. 445)	Writes the monetary format of the local standard expression for Indonesia.
	NLMNLILSw.d Format (p. 447)	Writes the monetary format of the local standard expression for Israel.
	NLMNLINRw.d Format (p. 448)	Writes the monetary format of the local standard expression for India.
	NLMNLJPYw.d Format (p. 450)	Writes the monetary format of the local standard expression for Japan.
	NLMNLKRWw.d Format (p. 451)	Writes the monetary format of the local standard expression for South Korea.
	NLMNLLTLw.d Format (p. 452)	Writes the monetary format of the local standard expression for Lithuania.
	NLMNLLVLw.d Format (p. 454)	Writes the monetary format of the local standard expression for Latvia.
	NLMNLMOPw.d Format (p. 455)	Writes the monetary format of the local standard expression for Macau.
	NLMNLMXNw.d Format (p. 457)	Writes the monetary format of the local standard expression for Mexico.
	NLMNLMYRw.d Format (p. 458)	Writes the monetary format of the local standard expression for Malaysia.
	NLMNLNOKw.d Format (p. 459)	Writes the monetary format of the local standard expression for Norway.
	NLMNLNZDw.d Format (p. 461)	Writes the monetary format of the local standard expression for New Zealand.
	NLMNLPLNw.d Format (p. 462)	Writes the monetary format of the local standard expression for Poland.

Category	Language Elements	Description
	NLMNLRUBw.d Format (p. 464)	Writes the monetary format of the local standard expression for Russia.
	NLMNLSEKw.d Format (p. 465)	Writes the monetary format of the local standard expression for Sweden.
	NLMNLSGDw.d Format (p. 467)	Writes the monetary format of the local standard expression for Singapore.
	NLMNLTHBw.d Format (p. 468)	Writes the monetary format of the local standard expression for Thailand.
	NLMNLTRYw.d Format (p. 469)	Writes the monetary format of the local standard expression for Turkey.
	NLMNLTWDw.d Format (p. 471)	Writes the monetary format of the local standard expression for Taiwan.
	NLMNLUSDw.d Format (p. 472)	Writes the monetary format of the local standard expression for Puerto Rico and the United States.
	NLMNLZARw.d Format (p. 474)	Writes the monetary format of the local standard expression for South Africa.
	NLMNYw.d Format (p. 475)	Writes the monetary format of the local standard expression using local currency that is appropriate for the current SAS locale.
	NLMNYACw.d Format (p. 477)	Writes the monetary format of the local accounting expression using local currency that is appropriate for the current SAS locale.
	NLMNYIw.d Format (p. 479)	Writes the monetary format of the international standard expression that is appropriate for the current SAS locale.
	NLMNYIACw.d Format (p. 481)	Writes the monetary format of the international accounting expression that is appropriate for the current SAS locale.
	NLNUMw.d Format (p. 483)	Writes the numeric format of the local expression that is appropriate for the current SAS locale.
	NLNUMIw.d Format (p. 485)	Writes the numeric format of the international expression that is appropriate for the current SAS locale.
	NLPCTw.d Format (p. 487)	Writes percentage data of the local expression that is appropriate for the current SAS locale.
	NLPCTIw.d Format (p. 489)	Writes percentage data of the international expression that is appropriate for the current SAS locale.
	NLPCTNw.d Format (p. 490)	Writes percentages, using a minus sign for negative values that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLPCTPw.d Format (p. 492)	Writes locale-specific numeric values as percentages that is appropriate for the current SAS locale.
	NLPVALUEw.d Format (p. 493)	Writes p-values of the local expression that is appropriate for the current SAS locale.
	NLSTRMONw.d Format (p. 494)	Writes the month name that is appropriate for the current SAS locale.
	NLSTRQTRw.d Format (p. 496)	Writes a numeric value as the quarter-of-the-year that is appropriate for the current SAS locale.
	NLSTRWKw.d Format (p. 497)	Writes a numeric value as the day-of-the-week that is appropriate for the current SAS locale.
	NLTIMAPw. Format (p. 499)	Writes a SAS time value as a time value with a.m. or p.m. that is appropriate for the current SAS locale.
	NLTIMEw. Format (p. 500)	Writes a SAS time value as a time value that is appropriate for the current SAS locale. NLTIME also writes SAS date-time values.
	NLTIMELw. Format (p. 502)	Writes a SAS time value as a time value in a long-uniform pattern that is appropriate for the current SAS locale.
	NLTIMEMw. Format (p. 504)	Writes a SAS time value as a time value in a medium-uniform pattern that is appropriate for the current SAS locale.
	NLTIMESw. Format (p. 505)	Writes a SAS time value as a time value in a short-uniform pattern that is appropriate for the current SAS locale.
	$VSLOGw. Format (p. 536)	Processes a character string that is in visual order, and then writes the character string in left-to-right logical order.
	$VSLOGRw. Format (p. 537)	Processes a character string that is in visual order, and then writes the character string in right-to-left logical order.
	WEEKUw. Format (p. 539)	Writes a week number in decimal format by using the U algorithm.
	WEEKVw. Format (p. 541)	Writes a week number in decimal format by using the V algorithm.
	WEEKWw. Format (p. 543)	Writes a week number in decimal format by using the W algorithm.
	YENw.d Format (p. 545)	Writes numeric values with yen signs, commas, and decimal points.
	YYWEEKUw. Format (p. 546)	Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.

Category	Language Elements	Description
	YYWEEKVw. Format (p. 548)	Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.
	YYWEEKWw. Format (p. 550)	Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.
Character	$EBCDICw. Format (p. 157)	Converts native format character data to EBCDIC representation.
	$UCS2Bw. Format (p. 507)	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.
	$UCS2BEw. Format (p. 508)	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.
	$UCS2Lw. Format (p. 510)	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.
	$UCS2LEw. Format (p. 511)	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.
	$UCS2Xw. Format (p. 513)	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.
	$UCS2XEw. Format (p. 514)	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.
	$UCS4Bw. Format (p. 516)	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.
	$UCS4BEw. Format (p. 517)	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.
	$UCS4Lw. Format (p. 519)	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.
	$UCS4LEw. Format (p. 521)	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.
	$UCS4Xw. Format (p. 522)	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	Language Elements	Description
	$UCS4XEw. Format (p. 524)	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.
	$UESCw. Format (p. 525)	Processes a character string that is encoded in the current SAS session, and then writes the character string in Unicode escape (UESC) representation.
	$UESCEw. Format (p. 527)	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.
	$UNCRw. Format (p. 528)	Processes a character string that is encoded in the current SAS session, and then writes the character string in numeric character representation (NCR).
	$UNCREw. Format (p. 529)	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.
	$UPARENw. Format (p. 531)	Processes a character string that is encoded in the current SAS session, and then writes the character string in Unicode parenthesis (UPAREN) representation.
	$UPARENEw. Format (p. 532)	Processes a character string that is in Unicode parenthesis (UPAREN), and then writes the character string in the encoding of the current SAS session.
	$UTF8Xw. Format (p. 533)	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.
	$UTF8XEw. Format (p. 534)	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	B8601DXw. Format (p. 144)	Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+hhmm.
	B8601LXw. Format (p. 146)	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.
	B8601TXw. Format (p. 148)	Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss+|-hhmm.
	DTWEEKVw. Format (p. 155)	Writes a week and datetime number in decimal format by using the V algorithm.
	E8601DXw. Format (p. 158)	Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and

Category	Language Elements	Description
		time zone extended notation yyyy-mm-ddThh:mm:ss +hh:mm.
	E8601LXw. Format (p. 160)	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.
	E8601TXw. Format (p. 162)	Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 extended time notation hh:mm:ss+|−hh:mm.
	HDATEw. Format (p. 170)	Writes date values in the form yyyy mmmmm dd where dd is the day-of-the-month, mmmmm represents the month's name in Hebrew, and yyyy is the year.
	HEBDATEw. Format (p. 171)	Writes date values according to the Jewish calendar.
	JNENGOw.d Format (p. 173)	Writes SAS date values as Japanese dates as reign, year, month, and day.
	JNENGOTw. Format (p. 174)	Writes SAS datetime values as Japanese datetimes as reign, year, month, day, hour, and minute, and with narrow characters for numbers.
	JNENGOTW w. Format (p. 175)	Writes SAS datetime values as Japanese datetimes as reign, year, month, day, hour, and minute, and with wide characters for numbers.
	JNENGOWw.d Format (p. 176)	Writes SAS date values as Japanese dates as reign, year, month, and day, and with wide character for numbers.
	MINGUOw. Format (p. 183)	Writes date values as Taiwanese dates in the form yyyymmdd.
	NENGOw. Format (p. 185)	Writes date values as Japanese dates in the form e.yymmdd.
	NLDATEw. Format (p. 191)	Writes a SAS date value as a date that is appropriate for the current SAS locale.
	NLDATECPw.p Format (p. 193)	Converts a SAS date value to a locale-sensitive-compact-interval date format.
	NLDATELw. Format (p. 196)	Writes a SAS date value as a date in the form month, date, year that is appropriate for the current SAS locale.
	NLDATEMw. Format (p. 197)	Writes a SAS date value as a date in a medium-uniform pattern that is appropriate for the current SAS locale.
	NLDATEMDw. Format (p. 199)	Writes the SAS date value as the name of the month and the day of the month that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLDATEMDLw. Format (p. 200)	Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale.
	NLDATEMDMw. Format (p. 201)	Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale.
	NLDATEMDSw. Format (p. 202)	Writes a SAS date value as the month and day of the month in a short-uniform pattern that is appropriate for the current SAS locale.
	NLDATEMNw. Format (p. 203)	Writes a SAS date value as the name of the month that is appropriate for the current SAS locale.
	NLDATESw. Format (p. 205)	Writes a SAS date value as a date string that is appropriate for the current SAS locale.
	NLDATEWw. Format (p. 207)	Writes a SAS date value as the date and the day of the week that is appropriate for the current SAS locale.
	NLDATEWNw. Format (p. 209)	Writes the SAS date value as the day of the week that is appropriate for the current SAS locale.
	NLDATEYMw. Format (p. 210)	Writes the SAS date value as the year and the name of the month that is appropriate for the current SAS locale.
	NLDATEYMLw. Format (p. 212)	Writes a SAS date value as the month and year that is appropriate for the current SAS locale.
	NLDATEYMMw. Format (p. 213)	Writes a SAS date value as the month and year with abbreviations that is appropriate for the current SAS locale.
	NLDATEYMSw. Format (p. 214)	Writes a SAS date value as a date and year that is appropriate for the current SAS locale.
	NLDATEYQw. Format (p. 215)	Writes the SAS date value as the year and the quarter that is appropriate for the current SAS locale.
	NLDATEYQLw. Format (p. 217)	Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale.
	NLDATEYQMw. Format (p. 218)	Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale.
	NLDATEYQSw. Format (p. 219)	Writes a SAS date value as the year and the year's quarter value (1–4) with numbers and delimiters that is appropriate for the current SAS locale.
	NLDATEYRw. Format (p. 220)	Writes the SAS date value as the year that is appropriate for the current SAS locale.
	NLDATEYWw. Format (p. 221)	Writes the SAS date value as the year and the week that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLDATMw. Format (p. 223)	Writes a SAS datetime value as a date time that is appropriate for the current SAS locale.
	NLDATMAPw. Format (p. 224)	Writes a SAS datetime value as a datetime with a.m. or p.m.that is appropriate for the current SAS locale.
	NLDATMCPw.p Format (p. 226)	Converts a SAS datetime value to a locale-sensitive-compact-interval datetime format.
	NLDATMDTw. Format (p. 230)	Writes the SAS datetime value as the name of the month, day of the month and year that is appropriate for the current SAS locale.
	NLDATMLw. Format (p. 231)	Writes a SAS datetime value as a long representation of the date that is appropriate for the current SAS locale.
	NLDATMMw. Format (p. 232)	Writes a SAS datetime value as a medium representation of the date that is appropriate for the current SAS locale.
	NLDATMMDw. Format (p. 233)	Writes the SAS datetime value as the name of the month and the day of the month that is appropriate for the current SAS locale.
	NLDATMMDLw. Format (p. 235)	Writes a SAS datetime value as the full-length of the month and day of the month that is appropriate for the current SAS locale.
	NLDATMMDMw. Format (p. 236)	Writes a SAS datetime value as the month and day of the month using abbreviations that is appropriate for the current SAS locale.
	NLDATMMDSw. Format (p. 237)	Writes a SAS datetime value as the month and day of the month using numbers and delimiters that is appropriate for the current SAS locale.
	NLDATMMNw. Format (p. 238)	Writes the SAS datetime value as the name of the month that is appropriate for the current SAS locale.
	NLDATMSw. Format (p. 240)	Writes a SAS datetime value as the short representation of the date that is appropriate for the current SAS locale.
	NLDATMTMw. Format (p. 241)	Writes the time portion of a SAS datetime value as the time of day that is appropriate for the current SAS locale.
	NLDATMTZw. Format (p. 242)	Writes the time portion of the SAS datetime value as the time of day and time zone that is appropriate for the current SAS locale.
	NLDATMWw. Format (p. 243)	Writes SAS datetime values as day of the week and the datetime that is appropriate for the current SAS locale.
	NLDATMWNw. Format (p. 245)	Writes a SAS datetime value as the day of the week that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLDATMWZw. Format (p. 246)	Writes SAS datetime values as a day-of-week, datetime, and time zone value that is appropriate for the current SAS locale.
	NLDATMYMw. Format (p. 247)	Writes the SAS datetime value as the month and year that is appropriate for the current SAS locale.
	NLDATMYMLw. Format (p. 248)	Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale.
	NLDATMYMMw. Format (p. 249)	Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale.
	NLDATMYMSw. Format (p. 251)	Writes a SAS datetime value as the month and year with numbers and a delimiter that is appropriate for the current SAS locale.
	NLDATMYQw. Format (p. 252)	Writes the SAS datetime value as the quarter and the year that is appropriate for the current SAS locale.
	NLDATMYQLw. Format (p. 253)	Writes a SAS datetime value as the year's quarter value (1–4) and the year that is appropriate for the current SAS locale.
	NLDATMYQMw. Format (p. 254)	Writes a SAS datetime value as the year's quarter (1–4) and year that is appropriate for the current SAS locale.
	NLDATMYQSw. Format (p. 256)	Writes a SAS datetime value as the year and the quarter (1-4) using numbers and a delimiter that is appropriate for the current SAS locale.
	NLDATMYRw. Format (p. 257)	Writes the SAS datetime value as the year that is appropriate for the current SAS locale.
	NLDATMYWw. Format (p. 258)	Writes the SAS datetime value as the week number and the year that is appropriate for the current SAS locale.
	NLDATMZw. Format (p. 259)	Writes SAS datetime values as datetime and time zone that is appropriate for the current SAS locale.
	NLTIMAPw. Format (p. 499)	Writes a SAS time value as a time value with a.m. or p.m. that is appropriate for the current SAS locale.
	NLTIMEw. Format (p. 500)	Writes a SAS time value as a time value that is appropriate for the current SAS locale. NLTIME also writes SAS date-time values.
	NLTIMELw. Format (p. 502)	Writes a SAS time value as a time value in a long-uniform pattern that is appropriate for the current SAS locale.
	NLTIMEMw. Format (p. 504)	Writes a SAS time value as a time value in a medium-uniform pattern that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLTIMESw. Format (p. 505)	Writes a SAS time value as a time value in a short-uniform pattern that is appropriate for the current SAS locale.
	WEEKUw. Format (p. 539)	Writes a week number in decimal format by using the U algorithm.
	WEEKVw. Format (p. 541)	Writes a week number in decimal format by using the V algorithm.
	WEEKWw. Format (p. 543)	Writes a week number in decimal format by using the W algorithm.
	YYWEEKUw. Format (p. 546)	Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.
	YYWEEKVw. Format (p. 548)	Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.
	YYWEEKWw. Format (p. 550)	Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.
DBCS	$KANJIw. Format (p. 178)	Adds shift-code data to DBCS data.
	$KANJIXw. Format (p. 179)	Removes shift-code data from DBCS data.
Hebrew Text Handling	$CPTDWw. Format (p. 153)	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).
	$CPTWDw. Format (p. 154)	Processes a character string that is encoded in Windows (cp1255), and then writes the character string in Hebrew DOS (cp862) encoding.
ISO 8601	B8601DXw. Format (p. 144)	Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+hhmm.
	B8601LXw. Format (p. 146)	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.
	B8601TXw. Format (p. 148)	Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss+|-hhmm.
	E8601DXw. Format (p. 158)	Adjusts a Coordinated Universal Time (UTC) datetime value to the user's 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.
	E8601LXw. Format (p. 160)	Writes datetime values as local time by appending a time zone offset difference between the local time and UTC,

Category	Language Elements	Description
		using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss+|−hh:mm.
	E8601TXw. Format (p. 162)	Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 extended time notation hh:mm:ss+|−hh:mm.
Numeric	BESTDOTXw. Format (p. 150)	Specifies that SAS choose the best notation and use a dot as a decimal separator.
	EUROw.d Format (p. 164)	Writes numeric values with a leading euro symbol (E), a comma that separates every three digits, and a period that separates the decimal fraction.
	EUROXw.d Format (p. 167)	Writes numeric values with a leading euro symbol (E), a period that separates every three digits, and a comma that separates the decimal fraction.
	NLBESTw. Format (p. 187)	Writes the best numerical notation based on the locale.
	NLDw.p Format (p. 189)	Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude.
	NLMNACIAED w.d Format (p. 260)	Writes the monetary format of the international accounting expression for the United Arab Emirates.
	NLMNACIAUD w.d Format (p. 261)	Writes the monetary format of the international accounting expression for Australia.
	NLMNACIBGN w.d Format (p. 263)	Writes the monetary format of the international accounting expression for Bulgaria.
	NLMNACIBRL w.d Format (p. 264)	Writes the monetary format of the international accounting expression for Brazil.
	NLMNACICAD w.d Format (p. 266)	Writes the monetary format of the international accounting expression for Canada.
	NLMNACICHF w.d Format (p. 267)	Writes the monetary format of the international accounting expression for Liechtenstein and Switzerland.
	NLMNACICNY w.d Format (p. 269)	Writes the monetary format of the international accounting expression for China.
	NLMNACICZK w.d Format (p. 270)	Writes the monetary format of the international accounting expression for Czechia.
	NLMNACIDKK w.d Format (p. 272)	Writes the monetary format of the international accounting expression for Denmark, Faroe Island, and Greenland.
	NLMNACIEEK w.d Format (p. 273)	Writes the monetary format of the international accounting expression for Estonia.
	NLMNACIEGPw.d Format (p. 275)	Writes the monetary format of the international accounting expression for Egypt.

Category	Language Elements	Description
	NLMNACIEURw.d Format (p. 276)	Writes the monetary format of the international accounting expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNACIGBPw.d Format (p. 278)	Writes the monetary format of the international accounting expression for the United Kingdom.
	NLMNACIHKDw.d Format (p. 279)	Writes the monetary format of the international accounting expression for Hong Kong.
	NLMNACIHRKw.d Format (p. 281)	Writes the monetary format of the international accounting expression for Croatia.
	NLMNACIHUFw.d Format (p. 282)	Writes the monetary format of the international accounting expression for Hungary.
	NLMNACIIDRw.d Format (p. 284)	Writes the monetary format of the international accounting expression for Indonesia.
	NLMNACIILSw.d Format (p. 285)	Writes the monetary format of the international accounting expression for Israel.
	NLMNACIINRw.d Format (p. 287)	Writes the monetary format of the international accounting expression for India.
	NLMNACIJPYw.d Format (p. 288)	Writes the monetary format of the international accounting expression for Japan.
	NLMNACIKRWw.d Format (p. 290)	Writes the monetary format of the international accounting expression for South Korea.
	NLMNACILTLw.d Format (p. 291)	Writes the monetary format of the international accounting expression for Lithuania.
	NLMNACILVLw.d Format (p. 293)	Writes the monetary format of the international accounting expression for Latvia.
	NLMNACIMOPw.d Format (p. 294)	Writes the monetary format of the international accounting expression for Macau.
	NLMNACIMXNw.d Format (p. 296)	Writes the monetary format of the international accounting expression for Mexico.
	NLMNACIMYRw.d Format (p. 297)	Writes the monetary format of the international accounting expression for Malaysia.
	NLMNACINOKw.d Format (p. 299)	Writes the monetary format of the international accounting expression for Norway.
	NLMNACINZDw.d Format (p. 300)	Writes the monetary format of the international accounting expression for New Zealand.

Category	Language Elements	Description
	NLMNACIPLNw.d Format (p. 302)	Writes the monetary format of the international accounting expression for Poland.
	NLMNACIRUBw.d Format (p. 303)	Writes the monetary format of the international accounting expression for Russia.
	NLMNACISEKw.d Format (p. 305)	Writes the monetary format of the international accounting expression for Sweden.
	NLMNACISGDw.d Format (p. 306)	Writes the monetary format of the international accounting expression for Singapore.
	NLMNACITHBw.d Format (p. 308)	Writes the monetary format of the international accounting expression for Thailand.
	NLMNACITRYw.d Format (p. 309)	Writes the monetary format of the international accounting expression for Turkey.
	NLMNACITWDw.d Format (p. 311)	Writes the monetary format of the international accounting expression for Taiwan.
	NLMNACIUSDw.d Format (p. 312)	Writes the monetary format of the international accounting expression for Puerto Rico and the United States.
	NLMNACIZARw.d Format (p. 314)	Writes the monetary format of the international accounting expression for South Africa.
	NLMNACLAEDx.d Format (p. 315)	Writes the monetary format of the local accounting expression for the United Arab Emirates.
	NLMNACLAUDw.d Format (p. 317)	Writes the monetary format of the local accounting expression for Australia.
	NLMNACLBGNw.d Format (p. 318)	Writes the monetary format of the local standard expression for Bulgaria.
	NLMNACLBRLw.d Format (p. 320)	Writes the monetary format of the local expression for Brazil.
	NLMNACLCADw.d Format (p. 321)	Writes the monetary format of the local accounting expression for Canada.
	NLMNACLCHFw.d Format (p. 323)	Writes the monetary format of the local accounting expression for Liechtenstein and Switzerland.
	NLMNACLCNYw.d Format (p. 324)	Writes the monetary format of the local accounting expression for China.
	NLMNACLCZKw.d Format (p. 326)	Writes the monetary format of the local accounting expression for Czechia.
	NLMNACLDKKw.d Format (p. 327)	Writes the monetary format of the local accounting expression for Denmark, Faroe Island, and Greenland.

Category	Language Elements	Description
	NLMNACLEEKw.d Format (p. 329)	Writes the monetary format of the local accounting expression for Estonia.
	NLMNACLEGPw.d Format (p. 330)	Writes the monetary format of the local accounting expression for Egypt.
	NLMNACLEURw.d Format (p. 332)	Writes the monetary format of the local accounting expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNACLGBPw.d Format (p. 333)	Writes the monetary format of the local accounting expression for the United Kingdom.
	NLMNACLHKDw.d Format (p. 335)	Writes the monetary format of the local accounting expression for Hong Kong.
	NLMNACLHRKw.d Format (p. 336)	Writes the monetary format of the local accounting expression for Croatia.
	NLMNACLHUFw.d Format (p. 338)	Writes the monetary format of the local accounting expression for Hungary.
	NLMNACLIDRw.d Format (p. 339)	Writes the monetary format of the local accounting expression for Indonesia.
	NLMNACLILSw.d Format (p. 341)	Writes the monetary format of the local accounting expression for Israel.
	NLMNACLINRw.d Format (p. 342)	Writes the monetary format of the local accounting expression for India.
	NLMNACLJPYw.d Format (p. 344)	Writes the monetary format of the local accounting expression for Japan.
	NLMNACLKRWw.d Format (p. 345)	Writes the monetary format of the local accounting expression for South Korea.
	NLMNACLLTLw.d Format (p. 347)	Writes the monetary format of the local accounting expression for Lithuania.
	NLMNACLLVLw.d Format (p. 348)	Writes the monetary format of the local accounting expression for Latvia.
	NLMNACLMOPw.d Format (p. 350)	Writes the monetary format of the local accounting expression for Macau.
	NLMNACLMXNw.d Format (p. 351)	Writes the monetary format of the local accounting expression for Mexico.
	NLMNACLMYRw.d Format (p. 353)	Writes the monetary format of the local accounting expression for Malaysia.

Category	Language Elements	Description
	NLMNACLNOKw.d Format (p. 354)	Writes the monetary format of the local accounting expression for Norway.
	NLMNACLNZDw.d Format (p. 356)	Writes the monetary format of the local accounting expression for New Zealand.
	NLMNACLPLNw.d Format (p. 357)	Writes the monetary format of the local accounting expression for Poland.
	NLMNACLRUBw.d Format (p. 359)	Writes the monetary format of the local accounting expression for Russia.
	NLMNACLSEKw.d Format (p. 360)	Writes the monetary format of the local accounting expression for Sweden.
	NLMNACLSGDw.d Format (p. 362)	Writes the monetary format of the local accounting expression for Singapore.
	NLMNACLTHBw.d Format (p. 363)	Writes the monetary format of the local accounting expression for Thailand.
	NLMNACLTRYw.d Format (p. 365)	Writes the monetary format of the local accounting expression for Turkey.
	NLMNACLTWDw.d Format (p. 366)	Writes the monetary format of the local accounting expression for Taiwan.
	NLMNACLUSDw.d Format (p. 368)	Writes the monetary format of the local accounting expression for Puerto Rico and the United States.
	NLMNACLZARw.d Format (p. 369)	Writes the monetary format of the local accounting expression for South Africa.
	NLMNIAEDw.d Format (p. 371)	Writes the monetary format of the international standard expression for the United Arab Emirates.
	NLMNIAUDw.d Format (p. 372)	Writes the monetary format of the international expression for Australia.
	NLMNIBGNw.d Format (p. 373)	Writes the monetary format of the international standard expression for Bulgaria.
	NLMNIBRLw.d Format (p. 375)	Writes the monetary format of the international standard expression for Brazil.
	NLMNICADw.d Format (p. 376)	Writes the monetary format of the international standard expression for Canada.
	NLMNICHFw.d Format (p. 378)	Writes the monetary format of the international standard expression for Liechtenstein and Switzerland.
	NLMNICNYw.d Format (p. 379)	Writes the monetary format of the international standard expression for China.

Category	Language Elements	Description
	NLMNICZKw.d Format (p. 380)	Writes the monetary format of the international standard expression for Czechia.
	NLMNIDKKw.d Format (p. 382)	Writes the monetary format of the international standard expression for Denmark, Faroe Island, and Greenland.
	NLMNIEEKw.d Format (p. 383)	Writes the monetary format of the international standard expression for Estonia.
	NLMNIEGPw.d Format (p. 385)	Writes the monetary format of the international standard expression for Egypt.
	NLMNIEURw.d Format (p. 386)	Writes the monetary format of the international standard expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNIGBPw.d Format (p. 388)	Writes the monetary format of the international standard expression for the United Kingdom.
	NLMNIHKDw.d Format (p. 389)	Writes the monetary format of the international standard expression for Hong Kong.
	NLMNIHRKw.d Format (p. 390)	Writes the monetary format of the international standard expression for Croatia.
	NLMNIHUFw.d Format (p. 392)	Writes the monetary format of the international standard expression for Hungary.
	NLMNIIDRw.d Format (p. 393)	Writes the monetary format of the international standard expression for Indonesia.
	NLMNIILSw.d Format (p. 395)	Writes the monetary format of the international standard expression for Israel.
	NLMNIINRw.d Format (p. 396)	Writes the monetary format of the international standard expression for India.
	NLMNIJPYw.d Format (p. 397)	Writes the monetary format of the international standard expression for Japan.
	NLMNIKRWw.d Format (p. 399)	Writes the monetary format of the international standard expression for South Korea.
	NLMNILTLw.d Format (p. 400)	Writes the monetary format of the international standard expression for Lithuania.
	NLMNILVLw.d Format (p. 402)	Writes the monetary format of the international standard expression for Latvia.
	NLMNIMOPw.d Format (p. 403)	Writes the monetary format of the international standard expression for Macau.

Category	Language Elements	Description
	NLMNIMXNw.d Format (p. 404)	Writes the monetary format of the international standard expression for Mexico.
	NLMNIMYRw.d Format (p. 406)	Writes the monetary format of the international standard expression for Malaysia.
	NLMNINOKw.d Format (p. 407)	Writes the monetary format of the international standard expression for Norway.
	NLMNINZDw.d Format (p. 409)	Writes the monetary format of the international standard expression for New Zealand.
	NLMNIPLNw.d Format (p. 410)	Writes the monetary format of the international standard expression for Poland.
	NLMNIRUBw.d Format (p. 411)	Writes the monetary format of the international standard expression for Russia.
	NLMNISEKw.d Format (p. 413)	Writes the monetary format of the international standard expression for Sweden.
	NLMNISGDw.d Format (p. 414)	Writes the monetary format of the international standard expression for Singapore.
	NLMNITHBw.d Format (p. 416)	Writes the monetary format of the international standard expression for Thailand.
	NLMNITRYw.d Format (p. 417)	Writes the monetary format of the international standard expression for Turkey.
	NLMNITWDw.d Format (p. 418)	Writes the monetary format of the international standard expression for Taiwan.
	NLMNIUSDw.d Format (p. 420)	Writes the monetary format of the international standard expression for Puerto Rico and the United States.
	NLMNIZARw.d Format (p. 421)	Writes the monetary format of the international standard expression for South Africa.
	NLMNLAEDx.d Format (p. 423)	Writes the monetary format of the local standard expression for the United Arab Emirates.
	NLMNLAUDw.d Format (p. 424)	Writes the monetary format of the local standard expression for Australia.
	NLMNLBGNw.d Format (p. 425)	Writes the monetary format of the local standard expression for Bulgaria.
	NLMNLBRLw.d Format (p. 427)	Writes the monetary format of the local standard expression for Brazil.
	NLMNLCADw.d Format (p. 428)	Writes the monetary format of the local standard expression for Canada.

Category	Language Elements	Description
	NLMNLCHFw.d Format (p. 430)	Writes the monetary format of the local standard expression for Liechtenstein and Switzerland.
	NLMNLCNYw.d Format (p. 431)	Writes the monetary format of the local standard expression for China.
	NLMNLCZKw.d Format (p. 432)	Writes the monetary format of the local standard expression for Czechia.
	NLMNLDKKw.d Format (p. 434)	Writes the monetary format of the local standard expression for Denmark, Faroe Island, and Greenland.
	NLMNLEEKw.d Format (p. 435)	Writes the monetary format of the local standard expression for Estonia.
	NLMNLEGPw.d Format (p. 437)	Writes the monetary format of the local standard expression for Egypt.
	NLMNLEURw.d Format (p. 438)	Writes the monetary format of the local standard expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.
	NLMNLGBPw.d Format (p. 440)	Writes the monetary format of the local standard expression for the United Kingdom.
	NLMNLHKDw.d Format (p. 441)	Writes the monetary format of the local standard expression for Hong Kong.
	NLMNLHRKw.d Format (p. 443)	Writes the monetary format of the local standard expression for Croatia.
	NLMNLHUFw.d Format (p. 444)	Writes the monetary format of the local standard expression for Hungary.
	NLMNLIDRw.d Format (p. 445)	Writes the monetary format of the local standard expression for Indonesia.
	NLMNLILSw.d Format (p. 447)	Writes the monetary format of the local standard expression for Israel.
	NLMNLINRw.d Format (p. 448)	Writes the monetary format of the local standard expression for India.
	NLMNLJPYw.d Format (p. 450)	Writes the monetary format of the local standard expression for Japan.
	NLMNLKRWw.d Format (p. 451)	Writes the monetary format of the local standard expression for South Korea.
	NLMNLLTLw.d Format (p. 452)	Writes the monetary format of the local standard expression for Lithuania.

Category	Language Elements	Description
	NLMNLLVLw.d Format (p. 454)	Writes the monetary format of the local standard expression for Latvia.
	NLMNLMOPw.d Format (p. 455)	Writes the monetary format of the local standard expression for Macau.
	NLMNLMXNw.d Format (p. 457)	Writes the monetary format of the local standard expression for Mexico.
	NLMNLMYRw.d Format (p. 458)	Writes the monetary format of the local standard expression for Malaysia.
	NLMNLNOKw.d Format (p. 459)	Writes the monetary format of the local standard expression for Norway.
	NLMNLNZDw.d Format (p. 461)	Writes the monetary format of the local standard expression for New Zealand.
	NLMNLPLNw.d Format (p. 462)	Writes the monetary format of the local standard expression for Poland.
	NLMNLRUBw.d Format (p. 464)	Writes the monetary format of the local standard expression for Russia.
	NLMNLSEKw.d Format (p. 465)	Writes the monetary format of the local standard expression for Sweden.
	NLMNLSGDw.d Format (p. 467)	Writes the monetary format of the local standard expression for Singapore.
	NLMNLTHBw.d Format (p. 468)	Writes the monetary format of the local standard expression for Thailand.
	NLMNLTRYw.d Format (p. 469)	Writes the monetary format of the local standard expression for Turkey.
	NLMNLTWDw.d Format (p. 471)	Writes the monetary format of the local standard expression for Taiwan.
	NLMNLUSDw.d Format (p. 472)	Writes the monetary format of the local standard expression for Puerto Rico and the United States.
	NLMNLZARw.d Format (p. 474)	Writes the monetary format of the local standard expression for South Africa.
	NLMNYw.d Format (p. 475)	Writes the monetary format of the local standard expression using local currency that is appropriate for the current SAS locale.
	NLMNYACw.d Format (p. 477)	Writes the monetary format of the local accounting expression using local currency that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLMNYIw.d Format (p. 479)	Writes the monetary format of the international standard expression that is appropriate for the current SAS locale.
	NLMNYIACw.d Format (p. 481)	Writes the monetary format of the international accounting expression that is appropriate for the current SAS locale.
	NLNUMw.d Format (p. 483)	Writes the numeric format of the local expression that is appropriate for the current SAS locale.
	NLNUMIw.d Format (p. 485)	Writes the numeric format of the international expression that is appropriate for the current SAS locale.
	NLPCTw.d Format (p. 487)	Writes percentage data of the local expression that is appropriate for the current SAS locale.
	NLPCTIw.d Format (p. 489)	Writes percentage data of the international expression that is appropriate for the current SAS locale.
	NLPCTNw.d Format (p. 490)	Writes percentages, using a minus sign for negative values that is appropriate for the current SAS locale.
	NLPCTPw.d Format (p. 492)	Writes locale-specific numeric values as percentages that is appropriate for the current SAS locale.
	NLPVALUEw.d Format (p. 493)	Writes p-values of the local expression that is appropriate for the current SAS locale.
	NLSTRMONw.d Format (p. 494)	Writes the month name that is appropriate for the current SAS locale.
	NLSTRQTRw.d Format (p. 496)	Writes a numeric value as the quarter-of-the-year that is appropriate for the current SAS locale.
	NLSTRWKw.d Format (p. 497)	Writes a numeric value as the day-of-the-week that is appropriate for the current SAS locale.
	YENw.d Format (p. 545)	Writes numeric values with yen signs, commas, and decimal points.

Dictionary

B8601DXw. Format

Adjusts a Coordinated Universal Time (UTC) datetime value to the user's 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

Restriction: This format is not supported in a DATA step that runs in CAS.

Supports: ISO 8601 Element 5.4.1, complete representation

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.

```
data _null_ ;
   t='01Feb2021T12:34:56'dt ;
   put t b8601dx.;
run;
```

SAS writes the following output to the log:

```
20210201T083456-0400
```

This example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

```
options timezone='America/Adak';
data data _null_ ;
   t='01Feb2021T12:34:56'dt ;
   put t b8601dx.;
run;
```

SAS writes the following output to the log:

```
20210201T023456-1000
```

See Also

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

B8601LXw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

Supports: ISO 8601 Elements 5.3.3 and 5.3.4.2

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

```
data a;
   blx='01Feb2021T12:34:56'dt;
   put blx=;
   put blx b8601lx.;
run;
```

SAS writes the following output to the log:

```
blx=1927802096
      20210201T123456-0500
```

See Also

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

B8601TXw. Format

Adjusts a Coordinated Universal Time (UTC) value to the user's 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

Restriction: This format is not supported in a DATA step that runs in CAS.

Supports: ISO 8601 Elements 5.3.3 and 5.3.4

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 Formats and Informats: Reference*

BESTDOTXw. Format

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

Category: Numeric

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

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

LOCALE option	Default decimal separator character for the locale	NLDECSEPARATOR option	Separator character used by BEST*w.*	Separator character used by BESTDOTX*w.*
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

- "BESTw. Format" in *SAS Formats and Informats: Reference*
- "NLDECSEPARATOR System Option" on page 994

$BIDIw. Format

Converts between a logically ordered string and a visually ordered string, by reversing the order of Hebrew and Arabic characters while preserving the order of Latin words and numbers.

Category: BIDI Text Handling

Alignment: Left

Restriction: CAS

Syntax

$BIDI*w.*

Syntax Description

w

specifies the width of the output field.

Default 1 if *w* is not specified

Range 1–32767

Details

In the Windows operating environment, Hebrew and Arabic text is stored in logical order. The text is stored in the order in which it is written and not necessarily as it is displayed. However, in other operating environments, Hebrew text is stored in the same order it is displayed. SAS users can encounter Hebrew and Arabic text that is reversed. Such situations can occur when you use SAS/CONNECT or other software to transfer SAS data sets or reports with Hebrew and Arabic text from a visual operating environment to a logical one. The $BIDI format is a format that reverses Hebrew and Arabic text while maintaining the order of numbers and Latin-1 words.

Operating Environment Information: In mainframe operating environments, this format is designed to work with NewCode Hebrew and Arabic. Some mainframe operating environments might experience unsatisfactory results, because they use the OldCode Hebrew or Arabic encoding. There is a hotfix for this encoding on :SAS Institute's Web site.

Comparisons

The $BIDI*w*. format performs a reversing function similar to the $REVERJ*w*. format, which writes character data in reverse order and preserves blanks. $BIDI*w*. behaves in the following way:

- $BIDI*w*. reverses the order of words and numbers in a specified string, preserving blanks. Latin-1 words and numbers themselves are not reversed, only their order in the string.
- When $BIDI encounters a word consisting of Hebrew or Arabic characters in the text string, the characters in the Hebrew or Arabic word are reversed. The position of the Hebrew or Arabic word is reversed in the string.

Example

This example demonstrates how $BIDI*w.* reverses Hebrew characters. The Hebrew is reversed in the string. The Hebrew characters in the words are also reversed.

```
data;
   a='שלום כיתה א abc 123';
   b1 = put (a,$bidi20.);
   put b=;
   b2 = put (b,$bidi20.);
   put b=;
run;
```

The following lines are written to the SAS log:

```
b1=123 abc א הטיכ מולש
b2=שלום כיתה א abc 123
```

$CPTDWw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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:

- “$CPTWDw. Format” on page 154

Informats:

- “$CPTDWw. Informat” on page 781
- “$CPTWDw. Informat” on page 782

$CPTWDw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 "JJN".

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

See Also

Format:

- "$CPTDWw. Format" on page 153

Informats:

- "$CPTDWw. Informat" on page 781
- "$CPTWDw. Informat" on page 782

DTWEEKVw. Format

Writes a week and datetime number in decimal format by using the V algorithm.

Categories: CAS
Date and Time

Alignment: Left

Syntax

DTWEEKV*w.*

Syntax Description

w

specifies the width of the output field.

Default 18

Range 3–200

Details

This table has information on widths, formats, and examples:

Width	Format	Example
3–4	*Www*	w01
5–6	*yyWww*	03w01
7–8	*yyWwwdd*	03w0101
9–11	*yyyyWwwdd*	2003w0101
12–14	*yyyyWwwdd:hh*	2003w0101:09
15–17	*yyyyWwwdd:hh:mm*	2003w0101:09:10
18–19	*yyyyWwwdd:hh:mm:ss*	2003w0101:09:10:56
20–200	*yyyy-Www-dd:hh:mm:ss*	2003-w01-01:09:10:56

Example

Here is an example of the DTWEEKV*w*. format:

Statements	Result

`DATA _NULL_;`
`  t='01Feb2017:1:2:3'dt;`
`  put t= DTWEEKV30.;`
`RUN;`
	`t=2017-W05-03:01:02:03`

$EBCDICw. Format

Converts native format character data to EBCDIC representation.

Category: Character

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Note: UTF-8 is the only supported session encoding. $EBCDIC format works only for 7-bit 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 21.1 on page 1109.

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 994.

Comparisons

- On ASCII systems, $EBCDIC*w*. converts ASCII character data to EBCDIC.

- On all other systems, $EBCDICw behaves like the $CHARw. format.

Example

```
data one;
   input x $;
   datalines;
ABC
{ä}
[@]
;
run;

data two;
  set one;
  y=put(x,$ebcdic3.);
  put y $HEX6.;
run;
```

Table 10.1 *Output Locale Values for American English, French, and German*

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 pair of hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one character.

E8601DXw. Format

Adjusts a Coordinated Universal Time (UTC) datetime value to the user's 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

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—

stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 this 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='15Sep2018T12:34:56'dt ;`
`put t e8601dx.;`
`run;`
	`2018-09-15T08:34:56-04:00`

`options timezone='America/Adak';`
`data _null_ ;`
`t='15Sep2018T12:34:56'dt ;`
`put t e8601dx.;`
`run;`
	`2018-09-15T03:34:56-09:00`

See Also

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

E8601LXw. 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

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 this 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 example writes the time for the Eastern time zone:

```
data one;
   mydt='15sep2018 T12:34:56'dt;
   put mydt e8601lx.
run;
```

Value of mydt	Result
1852634096	2018-09-15T12:34:56-05:00

See Also

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

E8601TXw. Format

Adjusts a Coordinated Universal Time (UTC) value to the user's 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

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 this 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.;`
	`07:34:56-05:00`

Statement	Result
run;	
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 Formats and Informats: Reference*

EUROw.d 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: CAS
Numeric

Alignment: Right

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.;`	`1,255`
`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
1,255
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:

- “EUROXw.d Format” on page 167

Informats:

- “EUROw.d Informat” on page 784
- “EUROXw.d Informat” on page 786

EUROXw.d 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: CAS
Numeric

Alignment: Right

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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`
`put amount eurox5.;`	`1.255`
`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
1.255
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:

- "EUROw.d Format" on page 164

Informats:

- "EUROw.d Informat" on page 784
- "EUROXw.d Informat" on page 786

HDATEw. Format

Writes date values in the form *yyyy mmmmm dd* where *dd* is the day-of-the-month, *mmmmm* represents the month's name in Hebrew, and *yyyy* is the year.

Category: Date and Time

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

HDATE*w*.

Syntax Description

w
specifies the width of the output field.

Note: Use widths 9, 11, 15, or 17 for the best view.

Default 17

Range 9–17

Details

The HDATE*w*. format writes the SAS date value in the form *yyyy mmmmm dd*:

yyyy
is the year

mmmmm
is English month name written in Hebrew letters

dd
is the day-of-the-month

Example

The following example uses the input value of 15780, which is the SAS date of March 16, 2003.

Statements	Results
	`----+----1----+----2----+`
`put day hdate9.;`	`03 אדר 16`
`put day hdate11.;`	`2003 אדר 16`
`put day hdate17.;`	`2003 אדר 16`

See Also

Format:

- “HEBDATEw. Format” on page 171

HEBDATEw. Format

Writes date values according to the Jewish calendar.

Category: Date and Time

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

HEBDATE*w*.

Syntax Description

w

specifies the width of the output field.

Default 16

Range 7–24

Tip When using a non-Hebrew encoding, asterisks appear instead of the Hebrew letters.

Details

The Jewish calendar is a combined solar and lunar calendar. Years are counted from the creation of the world, which according to Jewish history, occurred 3760 years and three months before the commencement of the Christiar. You must add 3761, beginning in the autumn of a specified year in the Gregorian calendar to calculate the Hebrew year.

The HEBDATE*w*. format writes the SAS date value according to the Jewish calendar. The date is written in one of the following formats:

long

ראשון י' אדר ה'תשס"ג

default

י' אדר תשס"ג

short

י'/ו'/תשס"ג

Example

The following example uses the input value of 15780, which is the SAS date of March 16, 2003.

Statements	Results
	`----+----1----+`
`put day hebdate13.;`	י"ב/ו2'/תשס"ג
`put day hebdate16.;`	י"ב אדר-ב' תשס"ג
`put day hebdate24.;`	ראשון י"ב אדר-ב' ה'תשס"ג

See Also

Informat:

- “HDATEw. Format” on page 170

JNENGOw.d Format

Writes SAS date values as Japanese dates as reign, year, month, and day.

Category: Date and Time

Alignment: Left

Restrictions: This format is not supported in a DATA step that runs in CAS.

This format is supported only in a Japanese environment.

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

JNENGO*w.d*

Optional Arguments

w

specifies the width of the input field.

Default 21

Range 6–200

d

specifies these values:

.0 writes all information.

.1 writes Reign, Year, and Month information.

.2 writes Reign and Year information.

.3 writes Reign information only.

Default 0

Range 0–3

Comparisons

JNENGO*w.d* is similar to JNENGOW*w.d* except that JNENGO*w.d* uses narrow characters for numbers.

Example

This example uses the Japanese_Japan session encoding. It uses the input value of 15760, which is the SAS date value that corresponds to February 24, 2003.

```
Data _null_;
   x = 15760 ;
   put x JNENGO.3;
   put x JNENGO.2;
   put x JNENGO.1;
   put x JNENGO.;
run;
```

```
平成
平成 15 年
平成 15 年 2 月
平成 15 年 2 月 24 日
```

JNENGOTw. Format

Writes SAS datetime values as Japanese datetimes as reign, year, month, day, hour, and minute, and with narrow characters for numbers.

Category: Date and Time

Alignment: Left

Restrictions: This format is not supported in a DATA step that runs in CAS.

This format is supported only in a Japanese environment.

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

JNENGOT*w.*

Optional Argument

w

specifies the width of the input field.

Default 32

Range 27–200

Comparisons

JNENGOT*w*. is similar to JNENGOTW*w*. except that JNENGOT*w*. uses narrow characters for numbers.

Example

This example uses the Japanese_Japan session encoding. It uses the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003.

```
data _null_;
   x = 1361709583 ;
   put x JNENGOT32.;
   put x JNENGOT40.;
run;
```

```
平成 15 年 2 月 24 日 12:39
平成 15 年 2 月 24 日 12 時 39 分
```

JNENGOTW w. Format

Writes SAS datetime values as Japanese datetimes as reign, year, month, day, hour, and minute, and with wide characters for numbers.

Category: Date and Time

Alignment: Left

Restrictions: This format is supported only in a Japanese environment.

This format is not supported in a DATA step that runs in CAS.

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

JNENGOTW*w.*

Optional Argument

w
specifies the width of the input field.

Default 52

Range 49–200

Comparisons

JNENGOTW*w*. is similar to JNENGOT*w*. except that JNENGOTW*w*. uses wide characters for numbers.

Example

This example uses the Japanese_Japan session encoding. It uses the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003.

```
Data _null_;
   x = 1361709583 ;
   put x JNENGOTW49.;
   put x JNENGOTW80.;
run;
```

```
平成１５年　２月２４日　１２：３９
平成１５年　２月２４日　１２時３９分
```

JNENGOWw.d Format

Writes SAS date values as Japanese dates as reign, year, month, and day, and with wide character for numbers.

Category: Date and Time

Alignment: Left

Restrictions: This format is not supported in a DATA step that runs in CAS.
This format runs only in a Japanese environment.

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

JNENGOW*w.d*

Optional Arguments

w

specifies the width of the input field.

Default 33

Range 6–200

d

specifies these values:

.0 writes all information.

.1 writes Reign, Year, and Month information.

.2 writes Reign and Year information.

.3 writes Reign information only.

Default 0

Range 0–3

Comparisons

JNENGOW*w.d* is similar to JNENGO*w.d* except that JNENGOW*w.d* uses wide characters for numbers.

Example

This example uses the Japanese_Japan session encoding. It uses the input value of 15760, which is the SAS date value that corresponds to February 24, 2003.

```
data _null;
   x =15760 ;
   put x JNENGOW.3;
   put x JNENGOW.2;
   put x JNENGOW.1;
```

```
   put x JNENGOW.;
run;
```

```
平成
 平成１５年
平成１５年　２月
平成１５年　２月２４日
```

$KANJIw. Format

Adds shift-code data to DBCS data.

Category: DBCS

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Syntax

$KANJI*w.*

Syntax Description

w
specifies the width of the output field.

Range: The minimum width of the format is *2 + (length of shift code used on the current DBCSTYPE= setting)*2*

Restriction: The width must be an even number. If it is an odd number, it is truncated. The width must be equal to or greater than the length of the shift-code data.

Details

The $KANJI format adds shift-code data to DBCS data that does not have shift-code data. If the input data is blank, shift-code data is not added.

The $KANJI format processes host-mainframe data, but $KANJI can be used on other platforms. If you use the $KANJI format on non-EBCDIC (non-modal encoding) hosts, the data does not change.

See Also

Formats:

- “$KANJIXw. Format” on page 179

Informat:

- “$KANJIw. Informat” on page 789
- “$KANJIXw. Informat” on page 790

System Option:

- “DBCSTYPE System Option” on page 978

$KANJIXw. Format

Removes shift-code data from DBCS data.

Category: DBCS

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Syntax

$KANJIX*w.*

Syntax Description

w
specifies the width of the output field.

Range The minimum width of the format is 2.

Restriction The width must be an even number. If it is an odd number, it is truncated. The width must be equal to or greater than the length of the shift-code data.

Details

The $KANJIX format removes shift-code data from DBCS data. The input data length must be `2 + (SO/SI length)*2`. The data must start with SO and end with SI, unless single-byte data is returned.

The $KANJIX format processes host mainframe data, but $KANJIX can be used on other platforms. If you use the $KANJIX format on non-EBCDIC (non-modal encoding) hosts, the data does not change.

See Also

Format:

- “$KANJIw. Format” on page 178

Informats:

- “$KANJIw. Informat” on page 789
- “$KANJIXw. Informat” on page 790

System Option:

- “DBCSTYPE System Option” on page 978

$LOGVSw. Format

Processes a character string that is in left-to-right-logical order, and then writes the character string in visual order.

Categories: BIDI Text Handling
CAS

Alignment: Left

Syntax

$LOGVS*w*.

Syntax Description

w
specifies the width of the output field.

Default 200

Range 1–32767

Details

The $LOGVS*w*. format is used when you store logical-ordered text on a visual server.

Note: If the $LOGVS*w*. format is not accessible, then the Hebrew or Arabic portion of the data is reversed.

Comparisons

The $LOGVS*w*. format performs processing that is the opposite of the $LOGVSR*w*. format.

Example

The following example uses the Hebrew input value of " flight". טיסה

Statements	Results
	`----+----1----+----2----+`
`put text $logvs12.;`	`טיסה flight`

The following example uses the Arabic input value of " ذات " computer.

Statements	Results
	`----+----1----+----2----+`
`put text $logvs12.;`	ذات computer

See Also

Formats:

- "$LOGVSRw. Format" on page 182

Informats:

- "$LOGVSRw. Informat" on page 793
- "$LOGVSw. Informat" on page 791

$LOGVSRw. Format

Processes a character string that is in right-to-left-logical order, and then writes the character string in visual order.

Categories: BIDI Text Handling
CAS

Alignment: Left

Syntax

$LOGVSR*w*.

Syntax Description

w
specifies the width of the output field.

Default 200

Range 1–32767

Details

The $LOGVSR*w*. format is used when you store logical-ordered text on a visual server. The Hebrew or Arabic portion of the text is reversed if the $LOGVS*w*. format is not on the server.

Comparisons

The $LOGVSR*w*. format performs processing that is opposite of the $LOGVS*w*. format.

Example

The following example uses the Hebrew input value of "טיסה flight".

Statements	Results
	`----+----1----+`
`put text $logvsr12.;`	flight הסיט

The following example uses the Arabic input value of “ تاذ ” computer.

Statements	Results
	`----+----1----+`
`put text $logvsr12.;`	ذات computer

See Also

Formats:

- “$LOGVSw. Format” on page 180

Informats:

- “$LOGVSw. Informat” on page 791
- “$LOGVSRw. Informat” on page 793

MINGUOw. Format

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

Category: Date and Time

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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`

Statements	Results
`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:

- “MINGUOw. Informat” on page 794

NENGOw. Format

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

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 imperial era (Meiji, Taisho, Showa, Heisei, or Reiwa).

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;
```

```
H14
H14/01
H.140102
H14/01/02
H.14/01/02
```

See Also

Informat:

- “NENGOw. Informat” on page 796

NLBESTw. Format

Writes the best numerical notation based on the locale.

Categories: CAS
Numeric

Alignment: Right

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 Formats and Informats: Reference*.

Example

The following code produces results based on the locale:

```
options locale=English_UnitedStates;
data test;
   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.;
run;
```

```
-126E4
***
=====
  -0.1
-.1
=====
   0.1
0.1
=====
1.26E6
1E6
```

```
options locale=German_Germany;
data test;
   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.;
run;
```

```
-126E4
***
=====
  -0,1
-,1
=====
   0,1
0,1
=====
1,26E6
1E6
```

```
options locale=ar_BH;
data test;
   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.;
run;
```

```
-126E4
***
=====
  -0.1
-.1
=====
   0.1
0.1
=====
1.26E6
1E6
```

NLDw.p Format

Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude.

Categories: CAS
Numeric

Alignment: Right

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLD*w.p*

Optional Arguments

w

specifies the width of the output field.

Default 12

Range 1–32

p

specifies the precision.

Default 3

Range 0–16

Requirement *p* must be less than *w*.

Tip If *p* is omitted or is specified as 0, then *p* is set to 3.

Details

The NLD*w.p* format writes numbers so that the locale's decimal points align in groups of values with similar magnitude. Larger values of *p* print the data values with more precision and potentially more shifts in the locale-decimal-point alignment. Smaller values of *p* print the data values with less precision and a greater chance of locale-decimal-point alignment.

Examples

Example 1

These examples show the functionality of the NLD format and D format. The first example uses the English_United States locale. The second example uses the German_Germany locale.

```
options locale=en_US;
data _null_;
   x=.75;
   put 'NLD=<' x nld4.1 '>';
   put 'D=  <' x d4.1 '>';
run;
```

```
NLD=< 0.8>
D=  < 0.8>
```

Example 2

```
options locale=de_DE;
data _null_;
   x=.75;
   put 'NLD=<' x nld4.1 '>';
   put 'D=  <' x d4.1 '>';
run;
```

```
NLD=< 0.8>
D=  < 0.8>
```

NLDATEw. Format

Writes a SAS date value as a date that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATE*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 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. The first example specifies the english_United States locale. The second example specifies the german_Germany locale.

```
options locale=English_UnitedStates;
data test;
   day=15760;
   put day nldate.;
run;
```

```
February 24, 2003
```

```
options locale=German_Germany;
data test;
   day=15760;
   put day nldate.;
run;
```

```
24. Februar 2003
```

See Also

Formats:

- "NLDATEMNw. Format" on page 203
- "NLDATEWw. Format" on page 207
- "NLDATEWNw. Format" on page 209

NLDATECPw.p Format

Converts a SAS date value to a locale-sensitive-compact-interval date format.

Categories: CAS
Date and Time

Alignment: Left

Notes: The NLDATECP format is supported beginning in 2023.03.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATECP*w.p*

Optional Arguments

w

specifies the width of the output field. This argument is optional, and the default value is recommended. If the value of *w* is not large enough to accommodate the output, then a digital pattern is used, and the output will contain asterisks (*).

Default 20

Range 2–200

p

specifies the type of compact interval.

Default 9

Range 1–9

Details

The NLDATECPw.p format writes the date in a compact-uniform-interval pattern.

Examples

Example 1

This example specifies the input value of 18972 , which is the SAS date value that corresponds to December 11, 2011. The first example specifies the en_US locale. The second example specifies the fr_FR locale. The third example specifies the ja_JP locale. The fourth example demonstrates a digital pattern output.

```
option locale=en_US;
data _null_;
   dt = 18972;
   put dt nldatecp.1;
   put dt nldatecp.2;
   put dt nldatecp.3;
   put dt nldatecp.4;
   put dt nldatecp.5;
   put dt nldatecp.6;
   put dt nldatecp.7;
   put dt nldatecp.8;
   put dt nldatecp.9;
run;
```

```
11
Dec
11
2011
11Dec
Dec11
Dec2011
11Dec11
11Dec2011
```

Example 2

```
option locale=fr_FR;
data _null_;
   dt = 18972;
   put dt nldatecp.1;
   put dt nldatecp.2;
   put dt nldatecp.3;
   put dt nldatecp.4;
   put dt nldatecp.5;
   put dt nldatecp.6;
```

```
   put dt nldatecp.7;
   put dt nldatecp.8;
   put dt nldatecp.9;
run;
```

```
11
déc.
11
2011
11déc.
déc.11
déc.2011
11déc.11
11déc.2011
```

Example 3

```
option locale=ja_JP;
data _null_;
   dt = 18972;
   put dt nldatecp.1;
   put dt nldatecp.2;
   put dt nldatecp.3;
   put dt nldatecp.4;
   put dt nldatecp.5;
   put dt nldatecp.6;
   put dt nldatecp.7;
   put dt nldatecp.8;
   put dt nldatecp.9;
run;
```

```
11 日
w12 月
11 年
2011 年
12 月 11 日
11 年 12 月
2011 年 12 月
11 年 12 月 11 日
2011 年 12 月 11 日
```

Example 4

This example demonstrates when the value of *w* is not large enough to accommodate the output, a digital pattern is used. Note that astericks (*) are output in some of the output lines.

```
option locale=en_US;
data _null_;
   dt = 18972;
   put dt nldatecp2.9;
   put dt nldatecp3.9;
```

```
    put dt nldatecp4.9;
    put dt nldatecp5.9;
    put dt nldatecp6.9;
    put dt nldatecp7.9;
    put dt nldatecp8.9;
    put dt nldatecp9.9;
run;
```

These statements produce this output. Note that in the first six lines of the output, astericks (*) are displayed. These output lines are the result when the value *w* is not large enough to accommodate the output.

```
**
***
****
*****
******
*******
11122011
11Dec2011
```

See Also

"NLDATMCPw.p Format"

NLDATELw. Format

Writes a SAS date value as a date in the form month, date, year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 2–200

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.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   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=18max=200 ---+
November 19, 2012
11/19/2012
Nov 19, 2012
November 19, 2012
November 19, 2012
```

NLDATEMw. Format

Writes a SAS date value as a date in a medium-uniform pattern that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 specifies the date Nov 19, 2012.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   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
```

NLDATEMDw. Format

Writes the SAS date value as the name of the month and the day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATEMD*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 6-200

Example

This example specifies the english_United States locale option.

```
option locale=en_US;
data _null_;
   dy = date();
   put dy nldatemd.;
run;
```

```
April 03
```

See Also

Format:

- “NLDATEYMw. Format” on page 210

NLDATEMDLw. Format

Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 the full length of the month and the day such as November 19.

Example

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

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   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=12max=200 ---+
November 19
11/19
Nov 19
November 19
November 19
```

NLDATEMDMw. Format

Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 specifies the date of November 19, 2012.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put '+--- NLDATEMDL min=5 default=9
   max=200 ---+';
   put dy nldatemdm.;
   put dy nldatemdm5.;
   put dy nldatemdm9.;
   put dy nldatemdm200.;
run;
```

```
+--- NLDATEMDL min=5 default=9max=200 ---+
Nov 19
11/19
Nov 19
Nov 19
```

NLDATEMDSw. Format

Writes a SAS date value as the month and day of the month in a short-uniform pattern that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 specifies the date of November 19, 2012.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put '+--- NLDATEMDL min=5 default=5
   max=200 ---+';
   put dy nldatemds.;
   put dy nldatemds5.;
   put dy nldatemds5.;
   put dy nldatemds200.;
run;
```

```
+--- NLDATEMDL min=5 default=5max=200 ---+
11/19
11/19
11/19
11/19
```

NLDATEMNw. Format

Writes a SAS date value as the name of the month that is appropriate for the current SAS locale.

Categories: CAS

Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

Details

NLDATEMN writes values in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

This example demonstrates uniform pattern. This example uses a UTF-8 session and the width value is 6 for both statements.

```
option locale = zh_cn;
data test;
   x = '08jun91'd;
   put x nldatemn6.;
    = '08sep91'd;
   put x nldatemn6.;
run;
```

```
6
12
```

Comparisons

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

Example

These examples specify the input value of 15760, which is the SAS date value that corresponds to February 24, 2003. The first example specifies the en_US locale. The second example specifies the german_Germany locale.

```
option locale=en_US;
data _null_;
   month=15760;
   put month nldatemn.;
run;
```

```
February
```

```
option locale=german_germany;
data _null_;
   month=15760;
   put month nldatemn.;
run;
```

```
Februar
```

See Also

Formats:

- “NLDATEw. Format” on page 191
- “NLDATEWw. Format” on page 207
- “NLDATEWNw. Format” on page 209

NLDATESw. Format

Writes a SAS date value as a date string that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 specifies the date November 19, 2012. This example specifies the en_US locale.

```
option locale=en_US;
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put '+--- NLDATEMDL min=2 default=10
   max=200 ---+';
   put dy nldates.;
   put dy nldates8.;
   put dy nldates10.;
   put dy nldates200.;;
run;
```

```
+--- NLDATEMDL min=2 default=10max=200 ---+
11/19/2012
11/19/12
11/19/2012
11/19/2012
```

NLDATEWw. Format

Writes a SAS date value as the date and the day of the week that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 98 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

NLDATEW writes the date value in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

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. The first example specifies the en_US locale. The second example specifies the de_DE locale.

```
options locale=en_US;
data _null_;
dy=15760;
put dy nldatew.;
put dy nldatew20.;
put dy nldatew200.;
run;
```

```
Monday, February 24, 2003
Mon, Feb 24, 2003
Monday, February 24, 2003
```

```
options locale=de_DE;
data _null_;
   dy=15760;
   put dy nldatew.;
   put dy nldatew20.;
   put dy nldatew200.;
run;
```

```
Mo., 24. Feb 2003
Mo., 24. Feb 2003
Montag, 24. Februar 2003
```

See Also

Formats:

- “NLDATEw. Format” on page 191
- “NLDATEMNw. Format” on page 203
- “NLDATEWNw. Format” on page 209

NLDATEWNw. Format

Writes the SAS date value as the day of the week that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

Details

NLDATEWN writes the date value in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

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. The first example specifies the en_US locale. The second example specifies the de_DE locale.

```
options locale=en_US;
data _null_;
   dy=15760;
   put dy nldatewn.;
run;
```

```
Monday
```

```
options locale=de_DE;
data _null_;
   dy=15760;
   put dy nldatewn10.;
run;
```

```
Montag
```

See Also

Formats:

- “NLDATEw. Format” on page 191
- “NLDATEMNw. Format” on page 203
- “NLDATEWw. Format” on page 207

NLDATEYMw. Format

Writes the SAS date value as the year and the name of the month that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 specifies the spanish_Spain locale option. This example specifies the date of April 4, 2019.

```
options locale=spanish_Spain;
data _null_;
   dy=today();
   put dy nldateym.;
   put dy nldateym12.;
   put dy nldateym200.;
run;
```

```
abril de 2019
abr de 2019
abril de 2019
```

See Also

Format:

- “NLDATEMDw. Format” on page 199

NLDATEYMLw. Format

Writes a SAS date value as the month and year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 April 2019.

Example

This example specifies the date April 4, 2019. This example specifies the en_US locale.

```
options locale=en_US;
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 ---+
April 2019
04/2019
Apr 2019
April 2019
April 2019
```

NLDATEYMMw. Format

Writes a SAS date value as the month and year with abbreviations that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 Apr 2019.

Example

This example specifies the date April 4, 2019. This example specifies the en_US locale.

```
options locale=en_US;
data _null_;
   dt = datetime();
   dy = date();
   put "+--- NLDATEYML min=5 default=14 max=200 ---+";
   put dy nldateymm.;
   put dy nldateymm7.;
   put dy nldateymm11.;
   put dy nldateymm200.;
run;
```

```
+--- NLDATEYML min=5 default=14 max=200 ---+
Apr 2019
04/2019
Apr 2019
Apr 2019
```

NLDATEYMSw. Format

Writes a SAS date value as a date and year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 specifies the date April 4, 2019. This example specifies the en_US locale.

```
options locale=en_US;
data _null_;
   dt = datetime();
   dy = date();
   put "+--- NLDATEYMS min=5 default=14 max=200 ---+";
   put dy nldateyms.;
   put dy nldateyms5.;
   put dy nldateyms7.;
   put dy nldateyms200.;
run;
```

```
+--- NLDATEYMS min=5 default=14 max=200 ---+
04/2019
04/19
04/2019
04/2019
```

NLDATEYQw. Format

Writes the SAS date value as the year and the quarter that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 98 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

Example

This example specifies the date April 4, 2019. This example specifies the fr_FR locale option.

```
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     T2 2019
 4     ****
 14     T2 2019
 32     2e trimestre 2019
200
2e trimestre 2019
```

NLDATEYQLw. Format

Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 2nd quarter 2019.

Example

This example specifies the date April 4, 2019. This example specifies the en_US locale option.

```
options locale=en_US;
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 ---+
2nd quarter 2019
2019/2
Q2 2019
2nd quarter 2019
2nd quarter 2019
```

NLDATEYQMw. Format

Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 Q2 2019.

Example

The following example uses the date April 4, 2019.

```
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 ---+
Q2 2019
2019/2
Q2 2019
Q2 2019
```

NLDATEYQSw. Format

Writes a SAS date value as the year and the year's quarter value (1–4) with numbers and delimiters that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 2019/2.

Example

The following example specifies the date April 4, 2019.

```
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 ---+
2019/2
19/2
2019/2
2019/2
```

NLDATEYRw. Format

Writes the SAS date value as the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATEYR*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 2–200

Example

This example specifies the fr_FR locale option. This example specifies the date April 4, 2019.

```
options locale=fr_FR;
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATEYR min=4 default=6 max=200 ---+';
   put dy nldateyr.;
   put dy nldateyr2.;
   put dy nldateyr8.;
   put dy nldateyr200.;
run;
```

```
+--- NLDATEYR min=4 default=6 max=200 ---+
2019
19
2019
2019
```

NLDATEYWw. Format

Writes the SAS date value as the year and the week that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATEYW*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 5–200

Example

This example specifies the date April 4, 2019. This example specifies the fr_FR locale option.

```
options locale=fr_FR;
data _null_;
   dt = datetime();
   dy = date();
   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;
```

```
+--- NLDATEYW min=5 default=16 max=200 ---+
 16     Semaine 14 2019
 5     *****
 8     S14 19
 32     Semaine 14 2019
200
Semaine 14 2019
```

NLDATMw. Format

Writes a SAS datetime value as a date time that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 specify the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003. The first example specifies the en_US locale. The second example specifies the de_DE locale.

```
options locale=en_US;
data _null_;
   dy=1361709583;
   put dy nldatm.;
```

```
run;
```

```
24Feb2003:12:39:43
```

```
options locale=de_DE;
data _null_;
   dy=1361709583;
   put dy nldatm.;
run;
```

```
24. Februar 2003 12.39 Uhr
```

See Also

Formats:

- “NLDATMAPw. Format” on page 224
- “NLDATMTMw. Format” on page 241
- “NLDATMWw. Format” on page 243

NLDATMAPw. Format

Writes a SAS datetime value as a datetime with a.m. or p.m.that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 specify the input value of 1361709583, which is the SAS date-time value that corresponds to 12:39:43 p.m. on February 24, 2003. The first example specifies the en_US locale. The second example specifies the es_MX locale.

```
options locale=en_US;
data _null_;
dy=1361709583;
put dy nldatmap.;
run;
```

```
February 24, 2003 12:39:43 PM
```

```
options locale=es_MX;
data _null_;
dy=1361709583;
put dy nldatmap200.;
run;
```

```
24 de febrero de 2003 12:39:43 p.m.
```

See Also

Formats:

- “NLDATMw. Format” on page 223
- “NLDATMTMw. Format” on page 241
- “NLDATMWw. Format” on page 243

NLDATMCPw.p Format

Converts a SAS datetime value to a locale-sensitive-compact-interval datetime format.

Categories: CAS
Date and Time

Alignment: Left

Notes: The NLDATMCP format is supported beginning in 2023.03.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATMCP*w.p*

Optional Arguments

w

specifies the width of the output field. This argument is optional, and the default value is recommended. If the value of *w* is not large enough to accommodate the output, then a digital pattern is used, and the output will contain asterisks (*).

Default 33

Range 2–200

p

specifies the type of compact interval.

Default 12

Range 1–12

Details

The NLDATMCP*w.p* format writes the datetime in a compact-uniform-interval-pattern.

Examples

These examples specify the input value of 1361709583, which is the SAS datetime value that corresponds to February 24,2003 12:39:43. The first example specifies the en_US locale. The second example specifies the fr_FR locale. The third example specifies the ja_JP locale. The fourth example demonstrates a digital pattern output.

Example 1

```
option locale=en_US;
data _null_;
   dt = 1361709583;
   put dt nldatmcp.1;
   put dt nldatmcp.2;
   put dt nldatmcp.3;
   put dt nldatmcp.4;
   put dt nldatmcp.5;
   put dt nldatmcp.6;
   put dt nldatmcp.7;
   put dt nldatmcp.8;
   put dt nldatmcp.9;
   put dt nldatmcp.10;
   put dt nldatmcp.11;
   put dt nldatmcp.12;
run;
```

```
24
Feb
03
2003
24Feb
Feb03
Feb2003
24Feb03
24Feb2003
24Feb2003 12
24Feb2003 12:39
24Feb2003 12:39:43
```

Example 2

```
option locale=fr_FR;
data _null_;
   dt = 1361709583;
   put dt nldatmcp.1;
   put dt nldatmcp.2;
   put dt nldatmcp.3;
   put dt nldatmcp.4;
   put dt nldatmcp.5;
   put dt nldatmcp.6;
```

```
   put dt nldatmcp.7;
   put dt nldatmcp.8;
   put dt nldatmcp.9;
   put dt nldatmcp.10;
   put dt nldatmcp.11;
   put dt nldatmcp.12;
run;
```

```
24
févr.
03
2003
24févr.
févr.03
févr.2003
24févr.03
24févr.2003
24févr.2003 12h
24févr.2003 12:39
24févr.2003 12:39:43
```

Example 3

```
option locale=ja_JP;
data _null_;
   dt = 1361709583;
   put dt nldatmcp.1;
   put dt nldatmcp.2;
   put dt nldatmcp.3;
   put dt nldatmcp.4;
   put dt nldatmcp.5;
   put dt nldatmcp.6;
   put dt nldatmcp.7;
   put dt nldatmcp.8;
   put dt nldatmcp.9;
   put dt nldatmcp.10;
   put dt nldatmcp.11;
   put dt nldatmcp.12;
run;
```

```
24 日
2 月
03 年
2003 年
2 月 24 日
03 年 2 月
2003 年 2 月
03 年 2 月 24 日
2003 年 2 月 24 日
2003 年 2 月 24 日 12 時
2003 年 2 月 24 日 12:39
2003 年 2 月 24 日 12:39:43
```

Example 4

This example demonstrates when the value of *w* is not large enough to accommodate the output, a digital pattern is used. Note that astericks (*) are output in some of the output lines.

```
option locale=en_US;
data _null_;
   dt = 1361709583;
   put dt nldatmcp2.12;
   put dt nldatmcp3.12;
   put dt nldatmcp4.12;
   put dt nldatmcp5.12;
   put dt nldatmcp6.12;
   put dt nldatmcp7.12;
   put dt nldatmcp8.12;
   put dt nldatmcp9.12;
   put dt nldatmcp10.12;
   put dt nldatmcp11.12;
   put dt nldatmcp12.12;
   put dt nldatmcp13.12;
   put dt nldatmcp14.12;
   put dt nldatmcp15.12;
   put dt nldatmcp16.12;
   put dt nldatmcp17.12;
   put dt nldatmcp18.12;
run;
```

These statements produce this output. Note that in the first fourteen lines of the output, astericks (*) are displayed. These output lines are the result when the value *w* is not large enough to accommodate the output.

```
**
***
****
*****
******
*******
********
*********
**********
***********
************
*************
**************
***************
****************
24022003 12:39:43
24Feb2003 12:39:43
```

See Also

“NLDATECPw.p Format”

NLDATMDTw. Format

Writes the SAS datetime value as the name of the month, day of the month and year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATMDT*w*.

Syntax Description

w
specifies the width of the output field

Default 18

Range 10-200

Details

NLDATMDT writes values in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

Example

This example specifies the value 86400, which corresponds to January 2, 1960. The first example specifies the english_United States locale. The second example specifies the turkish_Turkey locale.

```
options locale=en_US;
data _null_;
   x=86400;
   put x nldatmdt.;
```

```
run;
```

```
January 2, 1960
```

```
options locale=tr_TR;
data _null_;
x=86400;
put x nldatmdt200.;
run;
```

```
2 Ocak 1960
```

See Also

Formats:

- “NLDATMMDw. Format” on page 233

NLDATMLw. Format

Writes a SAS datetime value as a long representation of the date that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 April 10, 2019 03:13:27 PM.

Example

```
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=30max=200 ---+
April 10, 2019 03:13:27 PM
04/10/2019 15:13:27
Apr 10, 2019 03:13:27 PM
April 10, 2019 03:13:27 PM
April 10, 2019 03:13:27 PM
```

NLDATMMw. Format

Writes a SAS datetime value as a medium representation of the date that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 Apr 10, 2019 03:23:34 PM.

Example

```
data _null_;
  dt = datetime();
  dy = date();
  put '+--- NLDATMM min=9 default=30
max=200 ---+';
  put dt NLDATMM.;
  put dt NLDATMM19.;
  put dt NLDATMM24.;
  put dt NLDATMM30.;
  put dt NLDATMM200.;
run;
```

```
+--- NLDATMM min=9 default=30max=200 ---+
Apr 10, 2019 03:23:34 PM
04/10/2019 15:23:34
Apr 10, 2019 03:23:34 PM
Apr 10, 2019 03:23:34 PM
Apr 10, 2019 03:23:34 PM
```

NLDATMMDw. Format

Writes the SAS datetime value as the name of the month and the day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.

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

```
x=January 02
```

See Also

Format:

- “NLDATMYMw. Format” on page 247

NLDATMMDLw. Format

Writes a SAS datetime value as the full-length of the month and day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 April 12, 2019.

```
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=12max=200 ---+
April 12
04/12
Apr 12
April 12
April 12
```

NLDATMMDMw. Format

Writes a SAS datetime value as the month and day of the month using abbreviations that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 April 12, 2019.

```
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=9max=200 ---+
Apr 12
04/12
Apr 12
Apr 12
```

NLDATMMDSw. Format

Writes a SAS datetime value as the month and day of the month using numbers and delimiters that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 4/12.

Example

The following example uses the date of April 12, 2019.

```
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=5max=200 ---+
04/12
04/12
04/12
04/12
```

NLDATMMNw. Format

Writes the SAS datetime value as the name of the month that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—

stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATMMN*w*.

Syntax Description

w

specifies the width of the output field.

Default 9

Range 4–200

Details

NLDATMMN writes values in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

Example

This example uses the en_US locale option.

```
options locale=en_US;
data _null_;
  dt = datetime();
  dy = date();
  put '+--- NLDATMMN min=5 default=10 max=200 ---+';
  put dt nldatmmn.;
  put dt nldatmmn4.;
  put dt nldatmmn10.;
  put dt nldatmmn200.;
run;
```

```
+--- NLDATMMN min=5 default=10max=200 ---+
April
Apr
April
April
```

NLDATMSw. Format

Writes a SAS datetime value as the short representation of the date that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 April 12, 2019.

```
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---+
04/12/2019 11:50:53
041219 11
04/12/2019 11:50:53
04/12/2019 11:50:53
```

NLDATMTMw. Format

Writes the time portion of a SAS datetime value as the time of day that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.

This example specifies the English_United_States locale.

```
options locale=en_US;
data one;
   event=1361709583;
   put event nldatmtm.;
run;
```

```
12:39:43
```

This example specifies the German_Germany locale.

```
options locale=de_DE;
data one;
   event=1361709583;
   put event nldatmtm.;
run;
```

```
12.39 Uhr
```

See Also

Formats:

- “NLDATMw. Format” on page 223
- “NLDATMAPw. Format” on page 224
- “NLDATMWw. Format” on page 243

NLDATMTZw. Format

Writes the time portion of the SAS datetime value as the time of day and time zone that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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. This example specifies the French_France locale.

```
options locale=fr_FR;
data test;
   x=datetime();
   put x=nldatmtz.;
run;
```

```
x=10 h 08 -0400
```

NLDATMWw. Format

Writes SAS datetime values as day of the week and the datetime that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 98 for information about recommended widths for locale and encoding combinations. You might need to use the suggested width for the NLDATMW format.

NLDATMW writes values in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

Comparisons

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

Example

This example uses the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003. This example specifies the English_United States locale.

```
options locale=English_UnitedStates;
data one;
   x=put(1361709583,nldatmw.);
   y=put(1361709583,nldatmw30.);
   z=put(1361709583,nldatmw200.);
   put x=;
   put y=;
   put z=;
run;
```

```
x=Monday, February 24, 2003 12:39:43 PM
y=Mon, Feb 24, 2003 12:39:43 PM
z=Monday, February 24, 2003 12:39:43 PM
```

See Also

Formats:

- “NLDATMw. Format” on page 223
- “NLDATMAPw. Format” on page 224
- “NLDATMTMw. Format” on page 241

NLDATMWNw. Format

Writes a SAS datetime value as the day of the week that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATMWN*w*.

Syntax Description

w
specifies the width of the output field.

Default 9

Range 4–200

Details

NLDATMWN writes values in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

Example

This example writes the SAS datetime value as a day of the week. This example specifies the English_United States locale, which is the default SAS session encoding.

```
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 ---+
Tuesday
Tue
Tuesday
Tuesday
```

NLDATMWZw. Format

Writes SAS datetime values as a day-of-week, datetime, and time zone value that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 98 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. This example specifies the French_France locale.

```
options locale=fr_FR;
data test;
   x=datetime();
   put x=nldatmwz.;
run;
```

```
x=mardi 16 avril 2019 10:35:06 -0400
```

NLDATMYMw. Format

Writes the SAS datetime value as the month and year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.

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

```
x=January 1960
y=January 1960
```

See Also

Format:

- “NLDATMMDw. Format” on page 233

NLDATMYMLw. Format

Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 April 16, 2019.

```
data _null_;
   dt = datetime();
   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=14max=200 ---+
April 2019
04/2019
Apr 2019
April 2019
April 2019
```

NLDATMYMMw. Format

Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale.

Categories: CAS

Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 April 16, 2019.

```
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 ---+
Apr 2019
04/2019
Apr 2019
Apr 2019
```

NLDATMYMSw. Format

Writes a SAS datetime value as the month and year with numbers and a delimiter that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 a delimiter for the month and year, such as 11/2012.

Example

The following example uses the date April 16, 2019.

```
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 ---+
04/2019
04/19
04/2019
04/2019
```

NLDATMYQw. Format

Writes the SAS datetime value as the quarter and the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 98 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

Example

This example uses the en_US locale option.

```
options locale=en_US;
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     2nd quarter 2019
  4     ****
 14     Q2 2019
 32     2nd quarter 2019
200
2nd quarter 2019
```

NLDATMYQLw. Format

Writes a SAS datetime value as the year's quarter value (1–4) and the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 April 16, 2019. This example specifies the en_US locale.

```
options locale=en_US;
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 ---+
2nd quarter 2019
2019/2
Q2 2019
2nd quarter 2019
2nd quarter 2019
```

NLDATMYQMw. Format

Writes a SAS datetime value as the year's quarter (1–4) and year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 Q2 2019.

Example

The following example uses the date of April 16, 2019.

```
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 ---+
Q2 2019
2019/2
Q2 2019
Q2 2019
```

NLDATMYQSw. Format

Writes a SAS datetime value as the year and the quarter (1-4) using numbers and a delimiter that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 a delimiter, such as 2012/4.

Example

The following example uses the date April 16, 2019.

```
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 ---+
2019/2
19/2
2019/2
2019/2
```

NLDATMYRw. Format

Writes the SAS datetime value as the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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, which is the default SAS session encoding.

```
data _null_;
   dt = datetime();
   dy = date();
```

```
   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 ---+
2019
19
2019
2019
```

NLDATMYWw. Format

Writes the SAS datetime value as the week number and the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATMYW*w*.

Syntax Description

w
specifies the width of the output field.

Default 16

Range 5–200

Example

This example uses the en_US locale option, which is the default SAS session encoding.

```
data _null_;
   dt = datetime();
   dy = date();
   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 16 2019
  5     *****
  8     W16 19
 32     Week 16 2019
200
Week 16 2019
```

NLDATMZw. Format

Writes SAS datetime values as datetime and time zone that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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. This example specifies the fr_FR locale.

```
options locale=fr_FR;
data test;
   x=datetime();
   put x=nldatmz.;
run;
```

```
x=16 avril 2019 11 h 40 -0400
```

NLMNACIAED w.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIAED *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_AE;
x=put(-1234.56789,nlmnaciaed32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.2 *NLMNACIAED Output*

Locale	Formatted value
ar_AE	(1,234.57AED)
en_US	(AED1,234.57)
es_ES	-1.234,57 AED

See Also

Format:

- “NLMNLAEDx.d Format” on page 423
- “NLMNACLAEDx.d Format”
- “NLMNIAEDw.d Format”

NLMNACIAUD w.d Format

Writes the monetary format of the international accounting expression for Australia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIAUD *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_AU;
x=put(-1234.56789,nlmnaciaud32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.3 *NLMNACIAUD Output*

Locale	Formatted value
en_AU	(AUD1,234.57)
en_US	(AUD1,234.57)
es_ES	-1.234,57 AUD

See Also

Format:

- “NLMNLAEDx.d Format” on page 423
- “NLMNIAUDw.d Format”
- “NLMNACLAUDw.d Format”

NLMNACIBGN w.d Format

Writes the monetary format of the international accounting expression for Bulgaria.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.
Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIBGN *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=bg_BG;
x=put(-1234.56789,nlmnacibgn32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.4 *NLMNACIBGN Output*

Locale	Formatted value
bg_BG	(1 234,57 BGN)
en_US	(BGN1,234.57)
es_ES	-1.234,57 BGN

See Also

Format:

- “NLMNLBGNw.d Format” on page 425
- “NLMNIBGNw.d Format”
- “NLMNACLBGNw.d Format”

NLMNACIBRL w.d Format

Writes the monetary format of the international accounting expression for Brazil.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.
Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIBRL *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pt_BR;
x=put(-1234.56789,nlmnacibrl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.5 *NLMNACIBRL Output*

Locale	Formatted value
pt_BR	-BRL 1.234,57
en_US	(BRL1,234.57)
es_ES	-1.234,57 BRL

See Also

Format:

- "NLMNLBRLw.d Format" on page 427
- "NLMNIBRLw.d Format"
- "NLMNACLBRLw.d Format"

NLMNACICAD w.d Format

Writes the monetary format of the international accounting expression for Canada.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACICAD *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_CA;
x=put(-1234.56789,nlmnacicad32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.6 *NLMNACICAD Output*

Locale	Formatted value
en_CA	(CAD1,234.57)
en_US	(CAD1,234.57)
es_ES	-1.234,57 CAD

See Also

Format:

- “NLMNLCADw.d Format” on page 428
- “NLMNICADw.d Format”
- “NLMNACLCADw.d Format”

NLMNACICHF w.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACICHF *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=fr_CH;
x=put(-1234.56789,nlmnacichf32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.7 *NLMNACICHF Output*

Locale	Formatted value
fr_CH	(1 234.57 CHF)
en_US	(CHF1,234.57)
es_ES	-1.234,57 CHF

See Also

Format:

- “NLMNLCHFw.d Format” on page 430
- “NLMNICHFw.d Format”
- “NLMNACLCHFw.d Format”

NLMNACICNY w.d Format

Writes the monetary format of the international accounting expression for China.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACICNY *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_CN;
x=put(-1234.56789,nlmnacicny32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.8 *NLMNACICNY Output*

Locale	Formatted value
zh_CN	(CNY1,234.57)
en_US	(CNY1,234.57)
es_ES	-1.234,57 CNY

See Also

Format:

- “NLMNLCNYw.d Format” on page 431
- “NLMNICNYw.d Format”
- “NLMNACLCNYw.d Format”

NLMNACICZK w.d Format

Writes the monetary format of the international accounting expression for Czechia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACICZK *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=cs_CZ;
x=put(-1234.56789,nlmnaciczk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.9 *NLMNACICZK Output*

Locale	Formatted value
cs_CZ	-1 234,57 CZK
en_US	(CZK1,234.57)
es_ES	-1.234,57 CZK

See Also

Format:

- "NLMNLCZKw.d Format" on page 432
- "NLMNICZKw.d Format"
- "NLMNACLCZKw.d Format"

NLMNACIDKK w.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIDKK *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=da_dk;
x=put(-1234.56789,nlmnacidkk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.10 *NLMNACIDKK Output*

Locale	Formatted value
da_DK	-1.234,57 DKK
en_US	(DKK1,234.57)
es_ES	-1.234,57 DKK

See Also

Format:

- “NLMNLDKKw.d Format” on page 434
- “NLMNIDKKw.d Format”
- “NLMNACLDKKw.d Format”

NLMNACIEEK w.d Format

Writes the monetary format of the international accounting expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

Notes: Estonia uses the Euro for currency. NLMNACIEEK is documented for historical purposes.

This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIEEK *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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=et_EE;
x=put(-1234.56789,nlmnacieek32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.11 *NLMNACIEEK Output*

Locale	Formatted value
et_EE	(1 234,57 EEK)
en_US	(EEK1,234.57)
es_ES	-1.234,57 EEK

See Also

Format:

- "NLMNLEEKw.d Format" on page 435
- "NLMNIEEKw.d Format"
- "NLMNACLEEKw.d Format"

NLMNACIEGPw.d Format

Writes the monetary format of the international accounting expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIEGP*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=ar_EG;
x=put(-1234.56789,nlmnaciegp32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.12 *NLMNACIEGP Output*

Locale	Formatted value
ar_EG	(1,234.57EGP)
en_US	(EGP1,234.57)
es_ES	-1.234,57 EGP

See Also

Format:

- “NLMNLEGPw.d Format” on page 437
- “NLMNIEGPw.d Format”
- “NLMNACLEGPw.d Format”

NLMNACIEURw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIEUR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code. In the following example, the LOCALE= system option is set to de_DE.

```
option locale=de_DE;
x=put(-1234.56789,nlmnacieur32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.13 *NLMNACIEUR Output*

Locale	Formatted value
de_DE	-1.234,57 EUR
en_US	(EUR1,234.57)
es_ES	-1.234,57 EUR

See Also

Format:

- "NLMNLEURw.d Format" on page 438
- "NLMNIEURw.d Format"
- "NLMNACLEURw.d Format"

NLMNACIGBPw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIGBP*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=en_GB;
x=put(-1234.56789,nlmnacigbp32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.14 *NLMNACIGBP Output*

Locale	Formatted value
en_AU	(GBP1,234.57)
en_US	(GBP1,234.57)
es_ES	-1.234,57 GBP

See Also

Format:

- “NLMNLGBPw.d Format” on page 440
- “NLMNIGBPw.d Format”
- “NLMNACLGBPw.d Format”

NLMNACIHKDw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIHKD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=en_HK;
x=put(-1234.56789,nlmnacihkd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.15 *NLMNACIHKD Output*

Locale	Formatted value
en_AU	(HKD1,234.57)
en_US	(HKD1,234.57)
es_ES	-1.234,57 HKD

See Also

Format:

- "NLMNLHKDw.d Format" on page 441
- "NLMNIHKDw.d Format"
- "NLMNACLHKDw.d Format"

NLMNACIHRKw.d Format

Writes the monetary format of the international accounting expression for Croatia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIHRK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=hr_HR;
x=put(-1234.56789,nlmnacihrk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.16 *NLMNACIHRK Output*

Locale	Formatted value
hr_HR	-1.234,57 HRK
en_US	(HRK1,234.57)
es_ES	-1.234,57 HRK

See Also

Format:

- “NLMNLHRKw.d Format” on page 443
- “NLMNIHRKw.d Format”
- “NLMNACLHRKw.d Format”

NLMNACIHUFw.d Format

Writes the monetary format of the international accounting expression for Hungary.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIHUF*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=hu_HU
x=put(-1234.56789,nlmnacihuf32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.17 *NLMNACIHUF Output*

Locale	Formatted value
hu_HU	-1.234,57 HRK
en_US	(HUF1,234.57)
es_ES	-1.234,57 HUF

See Also

Format:

- "NLMNLHUFw.d Format" on page 444
- "NLMNIHUFw.d Format"
- "NLMNACLHUFw.d Format"

NLMNACIIDRw.d Format

Writes the monetary format of the international accounting expression for Indonesia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIIDR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=id_ID;
x=put(-1234.56789,nlmnaciidr32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.18 *NLMNACIIDR Output*

Locale	Formatted value
id_ID	-IDR1.234,57
en_US	(IDR1,234.57)
es_ES	-1.234,57 IDR

See Also

Format:

- “NLMNLIDRw.d Format” on page 445
- “NLMNIIDRw.d Format”
- “NLMNACLIDRw.d Format”

NLMNACIILSw.d Format

Writes the monetary format of the international accounting expression for Israel.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIILS*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=he_IL;
x=put(-1234.56789,nlmnaciils32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.19 *NLMNACIILS Output*

Locale	Formatted value
he_IL	-1,234.57 ILS
en_US	(ILS1,234.57)
es_ES	-1.234,57 ILS

See Also

Format:

- "NLMNLILSw.d Format" on page 447
- "NLMNIILSw.d Format"
- "NLMNACLILSw.d Format"

NLMNACIINRw.d Format

Writes the monetary format of the international accounting expression for India.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIINR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=hi_IN;
x=put(-1234.56789,nlmnaciinr32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.20 *NLMNACIINR Output*

Locale	Formatted value
hi_IN	-INR1,234.57
en_US	(INR1,234.57)
es_ES	-1.234,57 INR

See Also

Format:

- “NLMNLINRw.d Format” on page 448
- “NLMNIINRw.d Format”
- “NLMNACLINRw.d Format”

NLMNACIJPYw.d Format

Writes the monetary format of the international accounting expression for Japan.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIJPY*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ja_JP;
x=put(-1234.56789,nlmnacijpy32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.21 *NLMNACIJPY Output*

Locale	Formatted value
ja_JP	(JPY1,234.57)
en_US	(JPY1,234.57)
es_ES	-1.234,57 JPY

See Also

Format:

- "NLMNLJPYw.d Format" on page 450
- "NLMNIJPYw.d Format"
- "NLMNACLJPYw.d Format"

NLMNACIKRWw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIKRW*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ko_KR;
x=put(-1234.56789,nlmnacikrw32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.22 *NLMNACIKRW Output*

Locale	Formatted value
ko_KR	(KRW1,234.57)
en_US	(KRW1,234.57)
es_ES	-1.234,57 KRW

See Also

Format:

- “NLMNLKRWw.d Format” on page 451
- “NLMNIKRWw.d Format”
- “NLMNACLKRWw.d Format”

NLMNACILTLw.d Format

Writes the monetary format of the international accounting expression for Lithuania.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACILTL*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale-lt_LT;
x=put(-1234.56789,nlmnaciltl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.23 *NLMNACILTL Output*

Locale	Formatted value
lt_LT	-1 234,57 LTL
en_US	(LTL1,234.57)
es_ES	-1.234,57 LTL

See Also

Format:

- “NLMNLLTLw.d Format” on page 452
- “NLMNILTLw.d Format”
- “NLMNACLLTLw.d Format”

NLMNACILVLw.d Format

Writes the monetary format of the international accounting expression for Latvia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACILVL*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=lv_LV;
x=put(-1234.56789,nlmnacilvl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.24 *NLMNACILVL Output*

Locale	Formatted value
lv_LV	-1 234,57 LVL
en_US	(LVL1,234.57)
es_ES	-1.234,57 LVL

See Also

Format:

- “NLMNLLVLw.d Format” on page 454
- “NLMNILVLw.d Format”
- “NLMNACLLVLw.d Format”

NLMNACIMOPw.d Format

Writes the monetary format of the international accounting expression for Macau.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIMOP*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=zh_MO;
x=put(-1234.56789,nlmnacimop32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.25 *NLMNACIMOP Output*

Locale	Formatted value
zh_MO	(MOP1,234.57)
en_US	(MOP1,234.57)
es_ES	-1.234,57 MOP

See Also

Format:

- "NLMNLMOPw.d Format" on page 455
- "NLMNIMOPw.d Format"
- "NLMNACLMOPw.d Format"

NLMNACIMXNw.d Format

Writes the monetary format of the international accounting expression for Mexico.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIMXN*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=es_MX;
x=put(-1234.56789,nlmnacimxn32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.26 *NLMNACIMXN Output*

Locale	Formatted value
es_MX	-MXN1,234.57
en_US	(MXN1,234.57)
es_ES	-1.234,57 MXN

See Also

Format:

- “NLMNLMXNw.d Format” on page 457
- “NLMNIMXNw.d Format”
- “NLMNACLMXNw.d Format”

NLMNACIMYRw.d Format

Writes the monetary format of the international accounting expression for Malaysia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIMYR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ms_MY;
x=put(-1234.56789,nlmnacimyr32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.27 *NLMNACIMYR Output*

Locale	Formatted value
ms_MY	(MYR1,234.57)
en_US	(MYR1,234.57)
es_ES	-1.234,57 MYR

See Also

Format:

- "NLMNLMYRw.d Format" on page 458
- "NLMNIMYRw.d Format"
- "NLMNACLMYRw.d Format"

NLMNACINOKw.d Format

Writes the monetary format of the international accounting expression for Norway.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACINOK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=no_NO;
x=put(-1234.56789,nlmnacinok32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.28 *NLMNACINOK Output*

Locale	Formatted value
no_NO	(NOK 1 234,57)
en_US	(NOK1,234.57)
es_ES	-1.234,57 NOK

See Also

Format:

- “NLMNLNOKw.d Format” on page 459
- “NLMNINOKw.d Format”
- “NLMNACLNOKw.d Format”

NLMNACINZDw.d Format

Writes the monetary format of the international accounting expression for New Zealand.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACINZD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_NZ;
x=put(-1234.56789,nlmnacinzd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.29 *NLMNACINZD Output*

Locale	Formatted value
en_NZ	(NZD1,234.57)
en_US	(NZD1,234.57)
es_ES	-1.234,57 NZD

See Also

Format:

- "NLMNLNZDw.d Format" on page 461
- "NLMNINZDw.d Format"
- "NLMNACLNZDw.d Format"

NLMNACIPLNw.d Format

Writes the monetary format of the international accounting expression for Poland.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIPLN*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pl_PL;
x=put(-1234.56789,nlmnacipln32.2;
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.30 *NLMNACIPLN Output*

Locale	Formatted value
pl_PL	(1 234,57 PLN)
en_US	(PLN1,234.57)
es_ES	-1.234,57 PLN

See Also

Format:

- “NLMNLPLNw.d Format” on page 462
- “NLMNIPLNw.d Format”
- “NLMNACLPLNw.d Format”

NLMNACIRUBw.d Format

Writes the monetary format of the international accounting expression for Russia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIRUB*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ru_RU;
x=put(-1234.56789,nlmnacirub32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.31 *NLMNACIRUB Output*

Locale	Formatted value
ru_RU	-1 234,57 RUB
en_US	(RUB1,234.57)
es_ES	-1.234,57 RUB

See Also

Format:

- “NLMNLRUBw.d Format” on page 464
- “NLMNIRUBw.d Format”
- “NLMNACLRUBw.d Format”

NLMNACISEKw.d Format

Writes the monetary format of the international accounting expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACISEK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=sv_SE;
x=put(-1234.56789,nlmnacisek32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.32 *NLMNACISEK Output*

Locale	Formatted value
sv_SE	-1 234,57 SEK
en_US	(SEK1,234.57)
es_ES	-1.234,57 SEK

See Also

Format:

- “NLMNLSEKw.d Format” on page 465
- “NLMNISEKw.d Format”
- “NLMNACLSEKw.d Format”

NLMNACISGDw.d Format

Writes the monetary format of the international accounting expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACISGD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_SG;
x=put(-1234.56789,nlmnacisgd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.33 *NLMNACISGD Output*

Locale	Formatted value
zh_SG	(SGD1,234.57)
en_US	(SGD1,234.57)
es_ES	-1.234,57 SGD

See Also

Format:

- "NLMNLSGDw.d Format" on page 467
- "NLMNISGDw.d Format"
- "NLMNACLSGDw.d Format"

NLMNACITHBw.d Format

Writes the monetary format of the international accounting expression for Thailand.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACITHB*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=th_TH;
x=put(-1234.56789,nlmnacithb32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.34 *NLMNACITHB Output*

Locale	Formatted value
th_TH	(THB1,234.57)
en_US	(THB1,234.57)
es_ES	-1.234,57 THB

See Also

Format:

- “NLMNLTHBw.d Format” on page 468
- “NLMNITHBw.d Format”
- “NLMNACLTHBw.d Format”

NLMNACITRYw.d Format

Writes the monetary format of the international accounting expression for Turkey.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACITRY*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=tr_TR;
x=put(-1234.56789,nlmnacitry32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.35 *NLMNACITRY Output*

Locale	Formatted value
tr_TR	(TRY1.234,57)
en_US	(TRY1,234.57)
es_ES	-1.234,57 TRY

See Also

Format:

- "NLMNLTRYw.d Format" on page 469
- "NLMNITRYw.d Format"
- "NLMNACLTRYw.d Format"

NLMNACITWDw.d Format

Writes the monetary format of the international accounting expression for Taiwan.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACITWD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_TW;
x=put(-1234.56789,nlmnacitwd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.36 *NLMNACITWD Output*

Locale	Formatted value
zh_TW	(TWD1,234.57)
en_US	(TWD1,234.57)
es_ES	-1.234,57 TWD

See Also

Format:

- “NLMNLTWDw.d Format” on page 471
- “NLMNITWDw.d Format”
- “NLMNACLTWDw.d Format”

NLMNACIUSDw.d Format

Writes the monetary format of the international accounting expression for Puerto Rico and the United States.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.
Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIUSD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=es_US;
x=put(-1234.56789,nlmnaciusd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.37 *NLMNACIUSD Output*

Locale	Formatted value
es_US	-USD1,234.57
en_US	(USD1,234.57)
es_ES	-1.234,57 USD

See Also

Format:

- "NLMNLUSDw.d Format" on page 472
- "NLMNIUSDw.d Format"
- "NLMNACLUSDw.d Format"

NLMNACIZARw.d Format

Writes the monetary format of the international accounting expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACIZAR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=af_ZA;
x=put(-1234.56789,nlmnacizar32.2;
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.38 *NLMNACIZAR Output*

Locale	Formatted value
af_ZA	(ZAR1 234,57)
en_US	(ZAR1,234.57)
es_ES	-1.234,57 ZAR

See Also

Format:

- “NLMNLZARw.d Format” on page 474
- “NLMNIZARw.d Format”
- “NLMNACLZARw.d Format”

NLMNACLAEDx.d Format

Writes the monetary format of the local accounting expression for the United Arab Emirates.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLAED*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_AE
x=put(-1234.56789,nlmnaclaed32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.39 *NLMNACLAED Output*

Locale	Formatted value
ar_AE	(1,234.57د.إ.)
en_US	(1,234.57.إ.د)
es_ES	-1.234,57 د.إ.

See Also

Format:

- "NLMNIAEDw.d Format" on page 371
- "NLMNLAEDx.d Format"
- "NLMNACIAED w.d Format"

NLMNACLAUDw.d Format

Writes the monetary format of the local accounting expression for Australia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLAUD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_AU;
x=put(-1234.56789,nlmnaclaud32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.40 *NLMNACLAUD Output*

Locale	Formatted value
en_AU	(AU$1,234.57)
en_US	(AU$1,234.57)
es_ES	-1.234,57 AU$

See Also

Format:

- “NLMNIAUDw.d Format” on page 372
- “NLMNLAUDw.d Format”
- “NLMNACIAUD w.d Format”

NLMNACLBGNw.d Format

Writes the monetary format of the local standard expression for Bulgaria.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLBGN*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=bg_BG;
x=put(-1234.56789,nlmnaclbgn32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.41 *NLMNACLBGN Output*

Locale	Formatted value
bg_BG	(1 234,57 лв)
en_US	-лв1,234.57
es_ES	-1.234,57 лв

See Also

Format:

- "NLMNIBGNw.d Format" on page 373
- "NLMNLBGNw.d Format"
- "NLMNACIBGN w.d Format"

NLMNACLBRLw.d Format

Writes the monetary format of the local expression for Brazil.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLBRL*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pt_BR;
x=put(-1234.56789,nlmnaclbrl32.2);
```

Table 10.42 *NLMNACLBRL Output*

Locale	Formatted value
pt_BR	-R$ 1.234,57
en_US	(R$1,234.57)
es_ES	-1.234,57 R$

See Also

Format:

- “NLMNIBRLw.d Format” on page 375
- “NLMNLBRLw.d Format”
- “NLMNACIBRL w.d Format”

NLMNACLCADw.d Format

Writes the monetary format of the local accounting expression for Canada.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLCAD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_CA;
x=put(-1234.56789,nlmnaclcad32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.43 *NLMNACLCAD Output*

Locale	Formatted value
en_CA	(CA$1,234.57)
en_US	(CA$1,234.57)
es_ES	-1.234,57 CA$

See Also

Format:

- “NLMNICADw.d Format” on page 376
- “NLMNLCADw.d Format”
- “NLMNACICAD w.d Format”

NLMNACLCHFw.d Format

Writes the monetary format of the local accounting expression for Liechtenstein and Switzerland.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLCHF*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=fr_CH;
x=put(-1234.56789,nlmnaclchf32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.44 *NLMNACLCHF Output*

Locale	Formatted value
fr_CH	(1 234.57 SFr.)
en_US	(SFr.1,234.57)
es_ES	-1.234,57 SFr.

See Also

Format:

- “NLMNICHFw.d Format” on page 378
- “NLMNLCHFw.d Format”

NLMNACLCNYw.d Format

Writes the monetary format of the local accounting expression for China.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLCNY*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_CN;
x=put(-1234.56789,nlmnaclcny32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.45 *NLMNACLCNY Output*

Locale	Formatted value
zh_CN	(RMB1,234.57)
en_US	(RMB1,234.57)
es_ES	-1.234,57 RMB

See Also

Format:

- “NLMNICNYw.d Format” on page 379
- “NLMNLCNYw.d Format”
- “NLMNACICNY w.d Format”

NLMNACLCZKw.d Format

Writes the monetary format of the local accounting expression for Czechia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLCZK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=cs_CZ;
x=put(-1234.56789,nlmnaclczk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.46 *NLMNACLCZK Output*

Locale	Formatted value
cs_CZ	-1 234,57 Kč
en_US	(Kč1,234.57)
es_ES	-1.234,57 Kč

See Also

Format:

- “NLMNICZKw.d Format” on page 380
- “NLMNLCZKw.d Format”
- “NLMNACICZK w.d Format”

NLMNACLDKKw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.
Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLDKK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=da_dk;
x=put(-1234.56789,nlmnacldkk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.47 *NLMNACLDKK Output*

Locale	Formatted value
da_dk	-1.234,57 kr
en_US	(kr1,234.57)
es_ES	-1.234,57 kr

See Also

Format:

- “NLMNIDKKw.d Format” on page 382
- “NLMNLDKKw.d Format”
- “NLMNACIDKK w.d Format”

NLMNACLEEKw.d Format

Writes the monetary format of the local accounting expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Estonia uses the Euro for curency. NLMNACLEEK is documented for historical purposes.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLEEK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=et_EE;
x=put(-1234.56789,nlmnacleek32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.48 *NLMNACLEEK Output*

Locale	Formatted value
et_EE	(1 234,57 kr)
en_US	(kr1,234.57)
es_ES	-1.234,57 kr

See Also

Format:

- “NLMNIEEKw.d Format” on page 383
- “NLMNLEEKw.d Format”
- “NLMNACIEEK w.d Format”

NLMNACLEGPw.d Format

Writes the monetary format of the local accounting expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLEGP*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_EG;
x=put(-1234.56789,nlmnaclegp32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.49 *NLMNACLEGP Output*

Locale	Formatted value
ar_EG	(1,234.57ج.م.)
en_US	(1,234.57ج.م)
es_ES	-1.234,57 ج.م.

See Also

Format:

- “NLMNIEGPw.d Format” on page 385
- “NLMNLEGPw.d Format”
- “NLMNACIEGPw.d Format”

NLMNACLEURw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLEUR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=de_DE;
```

```
y=put(-1234.56789,nlmnacleur32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.50 *NLMNACLEUR Output*

Locale	Formatted value
de_DE	-1.234,57 €
en_US	(€1,234.57)
es_ES	-1.234,57 €

See Also

Format:

- “NLMNIEURw.d Format” on page 386
- “NLMNLEURw.d Format”
- “NLMNACIEURw.d Format”

NLMNACLGBPw.d Format

Writes the monetary format of the local accounting expression for the United Kingdom.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLGBP*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_GB;
x=put(-1234.56789,nlmnaclgbp32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.51 *NLMNACLGBP Output*

Locale	Formatted value
en_GB	(£1,234.57)
en_US	(£1,234.57)
es_ES	-1.234,57 £

See Also

Format:

- “NLMNIGBPw.d Format” on page 388
- “NLMNLGBPw.d Format”
- “NLMNACIGBPw.d Format”

NLMNACLHKDw.d Format

Writes the monetary format of the local accounting expression for Hong Kong.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLHKD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_HK;
x=put(-1234.56789,nlmnaclhkd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.52 *NLMNACLHKD Output*

Locale	Formatted value
en_HK	(HK$1,234.57)
en_US	(HK$1,234.57)
es_ES	-1.234,57 HK$

See Also

Format:

- “NLMNIHKDw.d Format” on page 389
- “NLMNLHKDw.d Format”
- “NLMNACIHKDw.d Format”

NLMNACLHRKw.d Format

Writes the monetary format of the local accounting expression for Croatia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLHRK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hr_HR;
x=put(-1234.56789,nlmnaclhrk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.53 *NLMNACLHRK Output*

Locale	Formatted value
hr_HR	-1.234,57 Kn
en_US	(Kn1,234.57)
es_ES	-1.234,57 Kn

See Also

Format:

- “NLMNIHRKw.d Format” on page 390
- “NLMNLHRKw.d Format”
- “NLMNACIHRKw.d Format”

NLMNACLHUFw.d Format

Writes the monetary format of the local accounting expression for Hungary.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLHUF*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hu_HU;
x=put(-1234.56789,nlmnaclhuf32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.54 *NLMNACLHUF Output*

Locale	Formatted value
hu_HU	-1 234,57 Ft
en_US	(Ft1,234.57)
es_ES	-1.234,57 Ft

See Also

Format:

- "NLMNIHUFw.d Format" on page 392
- "NLMNLHUFw.d Format"
- "NLMNACIHUFw.d Format"

NLMNACLIDRw.d Format

Writes the monetary format of the local accounting expression for Indonesia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLIDR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=id_ID;
x=put(-1234.56789,nlmnaclidr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.55 *NLMNACLIDR Output*

Locale	Formatted value
id_ID	-Rp1.234,57
en_US	(Rp1,234.57)
es_ES	-1.234,57 Rp

See Also

Format:

- “NLMNIIDRw.d Format” on page 393
- “NLMNLIDRw.d Format”
- “NLMNACIIDRw.d Format”

NLMNACLILSw.d Format

Writes the monetary format of the local accounting expression for Israel.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLILS*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=he_IL;
x=put(-1234.56789,nlmnaclils32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.56 *NLMNACLILS Output*

Locale	Formatted value
he_IL	-1,234.57 ח"ש
en_US	(1,234.57ח"ש)
es_ES	-1.234,57 ח"ש

See Also

Format:

- "NLMNIILSw.d Format" on page 395
- "NLMNLILSw.d Format"
- "NLMNACIILSw.d Format"

NLMNACLINRw.d Format

Writes the monetary format of the local accounting expression for India.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLINR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hi_IN;
x=put(-1234.56789,nlmnaclinr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.57 *NLMNACLINR Output*

Locale	Formatted value
hi_IN	-₹s1,234.57
en_US	(₹s1,234.57)
es_ES	-1.234,57 ₹s

See Also

Format:

- “NLMNIINRw.d Format” on page 396
- “NLMNLINRw.d Format”
- “NLMNACIINRw.d Format”

NLMNACLJPYw.d Format

Writes the monetary format of the local accounting expression for Japan.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLJPY*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ja_JP;
x=put(-1234.56789,nlmnacljpy32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.58 *NLMNACLJPY Output*

Locale	Formatted value
ja_JP	(￥ 1,234.57)
en_US	(￥ 1,234.57)
es_ES	-1.234,57 ￥

See Also

Format:

- “NLMNIJPYw.d Format” on page 397
- “NLMNLJPYw.d Format”
- “NLMNACIJPYw.d Format”

NLMNACLKRWw.d Format

Writes the monetary format of the local accounting expression for South Korea.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLKRW*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ko_KR;
x=put(-1234.56789,nlmnaclkrw32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.59 *NLMNACLKRW Output*

Locale	Formatted value
ko_KR	(₩1,234.57)
en_US	(₩1,234.57)
es_ES	-1.234,57 ₩

See Also

Format:

- “NLMNIKRWw.d Format” on page 399
- “NLMNLKRWw.d Format”
- “NLMNACIKRWw.d Format”

NLMNACLLTLw.d Format

Writes the monetary format of the local accounting expression for Lithuania.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLLTL*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=lt_LT;
x=put(-1234.56789,nlmnaclltl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.60 *NLMNACLLTL Output*

Locale	Formatted value
lt_LT	-1 234,57 Lt
en_US	(Lt1,234.57)
es_ES	-1.234,57 Lt

See Also

Format:

- “NLMNILTLw.d Format” on page 400
- “NLMNLLTLw.d Format”
- “NLMNACILTLw.d Format”

NLMNACLLVLw.d Format

Writes the monetary format of the local accounting expression for Latvia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLLVL*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=lv_LV;
x=put(-1234.56789,nlmnacllvl32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.61 *NLMNACLLVL Output*

Locale	Formatted value
lv_LV	-1 234,57 Ls
en_US	(Ls1,234.57)
es_ES	-1.234,57 Ls

See Also

Format:

- “NLMNILVLw.d Format” on page 402
- “NLMNLLVLw.d Format”
- “NLMNACILVLw.d Format”

NLMNACLMOPw.d Format

Writes the monetary format of the local accounting expression for Macau.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLMOP*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_MO;
x=put(-1234.56789,nlmnaclmop32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.62 *NLMNACLMOP Output*

Locale	Formatted value
zh_MO	(P1,234.57)
en_US	(P1,234.57)
es_ES	-1.234,57 P

See Also

Format:

- "NLMNIMOPw.d Format" on page 403
- "NLMNLMOPw.d Format"
- "NLMNACIMOPw.d Format"

NLMNACLMXNw.d Format

Writes the monetary format of the local accounting expression for Mexico.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLMXN*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=es_MX;
x=put(-1234.56789,nlmnaclmxn32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.63 *NLMNACLMXN Output*

Locale	Formatted value
es_MX	-MX$1,234.57
en_US	(MX$1,234.57)
es_ES	-1.234,57 MX$

See Also

Format:

- “NLMNIMXNw.d Format” on page 404
- “NLMNLMXNw.d Format”
- “NLMNACIMXNw.d Format”

NLMNACLMYRw.d Format

Writes the monetary format of the local accounting expression for Malaysia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLMYR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ms_MY;
x=put(-1234.56789,nlmnaclmyr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.64 *NLMNACLMYR Output*

Locale	Formatted value
ms_MY	(R1,234.57)
en_US	(R1,234.57)
es_ES	-1.234,57 R

See Also

Format:

- “NLMNIMYRw.d Format” on page 406
- “NLMNLMYRw.d Format”
- “NLMNACIMYRw.d Format”

NLMNACLNOKw.d Format

Writes the monetary format of the local accounting expression for Norway.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLNOK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=no_NO;
x=put(-1234.56789,nlmnaclnok32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.65 *NLMNACLNOK Output*

Locale	Formatted value
no_NO	(kr 1 234,57)
en_US	(kr1,234.57)
es_ES	-1.234,57 kr

See Also

Format:

- “NLMNINOKw.d Format” on page 407
- “NLMNLNOKw.d Format”
- “NLMNACINOKw.d Format”

NLMNACLNZDw.d Format

Writes the monetary format of the local accounting expression for New Zealand.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLNZD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_NZ;
x=put(-1234.56789,nlmnaclnzd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.66 *NLMNACLNZD Output*

Locale	Formatted value
en_NZ	(NZ$1,234.57)
en_US	(NZ$1,234.57)
es_ES	-1.234,57 NZ$

See Also

Format:

- “NLMNINZDw.d Format” on page 409
- “NLMNLNZDw.d Format”
- “NLMNACINZDw.d Format”

NLMNACLPLNw.d Format

Writes the monetary format of the local accounting expression for Poland.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLPLN*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pl_PL;
x=put(-1234.56789,nlmnaclpln32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.67 *NLMNACLPLN Output*

Locale	Formatted value
pl_PL	(1 234,57 zł)
en_US	(zł1,234.57)
es_ES	-1.234,57 zł

See Also

Format:

- “NLMNIPLNw.d Format” on page 410
- “NLMNLPLNw.d Format”
- “NLMNACIPLNw.d Format”

NLMNACLRUBw.d Format

Writes the monetary format of the local accounting expression for Russia.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLRUB*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ru_RU;
x=put(-1234.56789,nlmnaclrub32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.68 *NLMNACLRUB Output*

Locale	Formatted value
ru_RU	-1 234,57 руб
en_US	(руб1,234.57)
es_ES	-1.234,57 руб

See Also

Format:

- “NLMNIRUBw.d Format” on page 411
- “NLMNLRUBw.d Format”
- “NLMNACIRUBw.d Format”

NLMNACLSEKw.d Format

Writes the monetary format of the local accounting expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLSEK*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=sv_SE;
x=put(-1234.56789,nlmnaclsek32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.69 *NLMNACLSEK Output*

Locale	Formatted value
sv_SE	-1 234,57 kr
en_US	(kr1,234.57)
es_ES	-1.234,57 kr

See Also

Format:

- “NLMNISEKw.d Format” on page 413
- “NLMNLSEKw.d Format”
- “NLMNACISEKw.d Format”

NLMNACLSGDw.d Format

Writes the monetary format of the local accounting expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLSGD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_SG;
x=put(-1234.56789,nlmnaclsgd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.70 *NLMNACLSGD Output*

Locale	Formatted value
zh_SG	(SG$1,234.57)
en_US	(SG$1,234.57)
es_ES	-1.234,57 SG$

See Also

Format:

- “NLMNISGDw.d Format” on page 414
- “NLMNLSGDw.d Format”
- “NLMNACISGDw.d Format”

NLMNACLTHBw.d Format

Writes the monetary format of the local accounting expression for Thailand.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLTHB*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=th_TH;
x=put(-1234.56789,nlmnaclthb32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.71 *NLMNACLTHB Output*

Locale	Formatted value
th_TH	(฿1,234.57)
en_US	(฿1,234.57)
es_ES	-1.234,57 ฿

See Also

Format:

- “NLMNITHBw.d Format” on page 416
- “NLMNLTHBw.d Format”
- “NLMNACITHBw.d Format”

NLMNACLTRYw.d Format

Writes the monetary format of the local accounting expression for Turkey.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLTRY*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=tr_TR;
x=put(-1234.56789,nlmnacltry32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.72 *NLMNACLTRY Output*

Locale	Formatted value
tr_TR	(YTL1.234,57)
en_US	(YTL1,234.57)
es_ES	-1.234,57 YTL

See Also

Format:

- “NLMNITRYw.d Format” on page 417
- “NLMNLTRYw.d Format”
- “NLMNACITRYw.d Format”

NLMNACLTWDw.d Format

Writes the monetary format of the local accounting expression for Taiwan.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLTWD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_TW;
x=put(-1234.56789,nlmnacltwd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.73 *NLMNACLTWD Output*

Locale	Formatted value
zh_TW	(NT$1,234.57)
en_US	(NT$1,234.57)
es_ES	-1.234,57 NT$

See Also

Format:

- “NLMNITWDw.d Format” on page 418
- “NLMNLTWDw.d Format”
- “NLMNACITWDw.d Format”

NLMNACLUSDw.d Format

Writes the monetary format of the local accounting expression for Puerto Rico and the United States.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLUSD*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=es_US;
x=put(-1234.56789,nlmnaclusd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.74 *NLMNACLUSD Output*

Locale	Formatted value
es_US	-US$1,234.57
en_US	(US$1,234.57)
es_ES	-1.234,57 US$

See Also

Format:

- "NLMNIUSDw.d Format" on page 420
- "NLMNLUSDw.d Format"
- "NLMNACIUSDw.d Format"

NLMNACLZARw.d Format

Writes the monetary format of the local accounting expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

Notes: This format is available beginning in 2023.11.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNACLZAR*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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=af_ZA;
x=put(-1234.56789,nlmnaclzar32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.75 *NLMNACLZAR Output*

Locale	Formatted value
af_ZA	(R1 234,57)
en_US	(R1,234.57)
es_ES	-1.234,57 R

See Also

Format:

- “NLMNIZARw.d Format” on page 421
- “NLMNLZARw.d Format”
- “NLMNACIZARw.d Format”

NLMNIAEDw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_AE;
x=put(-1234.56789,nlmniaed32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.76 *NLMNIAED Output*

Locale	Formatted value
ar_AE	-1,234.57 AED
en_US	-AED1,234.57
es_ES	-1.234,57 AED

See Also

Format:

- “NLMNLAEDx.d Format” on page 423

NLMNIAUDw.d Format

Writes the monetary format of the international expression for Australia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_AU;
x=put(-1234.56789,nlmniaud32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Locale	Formatted value
en_AU	(AUD1,234.57)
en_US	(AUD1,234.57)
es_ES	-1.234,57 AUD

See Also

Format:

- “NLMNLAUDw.d Format” on page 424

NLMNIBGNw.d Format

Writes the monetary format of the international standard expression for Bulgaria.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=bg_BG;
x=put(-1234.56789,nlmnibgn32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.77 *NLMNIBGN Output*

Locale	Formatted value
bg_BG	-1 234,57 BGN
en_US	-BGN1,234.57
es_ES	-1.234,57 BGN

See Also

Format:

- “NLMNLBGNw.d Format” on page 425

NLMNIBRLw.d Format

Writes the monetary format of the international standard expression for Brazil.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pt_BR;
x=put(-1234.56789,nlmnibrl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.78 *NLMNIBRL Output*

Locale	Formatted value
pt_BR	-BRL 1.234,57
en_US	-BRL1,234.57
es_ES	-1.234,57 BRL

See Also

Format:

- “NLMNLBRLw.d Format” on page 427

NLMNICADw.d Format

Writes the monetary format of the international standard expression for Canada.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_CA;
x=put(-1234.56789,nlmnicad32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.79 *NLMNICAD Output*

Locale	Formatted value
en_CA	-CAD1,234.57
en_US	-CAD1,234.57
es_ES	-1.234,57 CAD

See Also

Format:

- “NLMNLCADw.d Format” on page 428

NLMNICHFw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=fr_CH;
x=put(-1234.56789,nlmnichf32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.80 *NLMNICHF Output*

Locale	Formatted value
fr_CH	-1 234.57 CHF
en_US	-CHF1,234.57
es_ES	-1.234,57 CHF

See Also

Format:

- "NLMNLCHFw.d Format" on page 430

NLMNICNYw.d Format

Writes the monetary format of the international standard expression for China.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_CN;
x=put(-1234.56789,nlmnicny32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.81 *NLMNICNY Output*

Locale	Formatted value
zh_CN	-CNY1,234.57
en_US	-CNY1,234.57
es_ES	-1.234,57 CNY

See Also

Format:

- “NLMNLCNYw.d Format” on page 431

NLMNICZKw.d Format

Writes the monetary format of the international standard expression for Czechia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=cs_CZ;
x=put(-1234.56789,nlmniczk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.82 *NLMNICZK Output*

Locale	Formatted value
cs_CZ	-1 234,57 CZK)
en_US	-CZK1,234.57

Locale	Formatted value
es_ES	-1.234,57 CZK

See Also

Format:

- “NLMNLCZKw.d Format” on page 432

NLMNIDKKw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=da_dk;
x=put(-1234.56789,nlmnidkk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.83 *NLMNIDKK Output*

Locale	Formatted value
da_dk	-1.234,57 DKK
en_US	-DKK1,234.57
es_ES	-1.234,57 DKK

See Also

Format:

- “NLMNLDKKw.d Format” on page 434

NLMNIEEKw.d Format

Writes the monetary format of the international standard expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

Notes: Estonia uses the Euro for curency. NLMNIEEK is documented for historical purposes.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=et_EE;
x=put(-1234.56789,nlmnieek32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.84 *NLMNIEEK Output*

Locale	Formatted value
et_EE	-1 234,57 EEK
en_US	-EEK1,234.57
es_ES	-1.234,57 EEK

See Also

Format:

- “NLMNLEEKw.d Format” on page 435

NLMNIEGPw.d Format

Writes the monetary format of the international standard expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=ar_EG;
x=put(-1234.56789,nlmniegp32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.85 *NLMNIEGP Output*

Locale	Formatted value
ar_EG	-1,234.57 EGP
en_US	-EGP1,234.57
es_ES	-1.234,57 EGP

See Also

Format:

- “NLMNLEGPw.d Format” on page 437

NLMNIEURw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code. In the following example, the LOCALE= system option is set to de_DE.

```
option locale=de_DE;
x=put(-1234.56789,nlmnieur32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.86 *NLMNIEUR Output*

Locale	Formatted value
de_DE	-1.234,57 EUR
en_US	-EUR1,234.57
es_ES	-1.234,57 EUR

See Also

Format:

- “NLMNLEURw.d Format” on page 438

NLMNIGBPw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=en_GB;
x=put(-1234.56789,nlmnigbp32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.87 *NLMNIGBP Output*

Locale	Formatted value
en_GB	-GBP1,234.57
en_US	-GBP1,234.57
es_ES	-1.234,57 GBP

See Also

Format:

- "NLMNLGBPw.d Format" on page 440

NLMNIHKDw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=en_HK;
x=put(-1234.56789,nlmnihkd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.88 *NLMNIHKD Output*

Locale	Formatted value
en_HK	-HKD1,234.57
en_US	-HKD1,234.57
es_ES	-1.234,57 HKD

See Also

Format:

- “NLMNLHKDw.d Format” on page 441

NLMNIHRKw.d Format

Writes the monetary format of the international standard expression for Croatia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=hr_HR;
x=put(-1234.56789,nlmnihrk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.89 *NLMNIHRK Output*

Locale	Formatted value
hr_HR	-1.234,57 HRK
en_US	-HRK1,234.57

Locale	Formatted value
es_ES	-1.234,57 HRK

See Also

Format:

- “NLMNLHRKw.d Format” on page 443

NLMNIHUFw.d Format

Writes the monetary format of the international standard expression for Hungary.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=hu_HU
x=put(-1234.56789,nlmnihuf32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.90 *NLMNIHUF Output*

Locale	Formatted value
hu_HU	-1 234,57 HUF
en_US	-HUF1,234.57
es_ES	-1.234,57 HUF

See Also

Format:

- “NLMNLHUFw.d Format” on page 444

NLMNIIDRw.d Format

Writes the monetary format of the international standard expression for Indonesia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=id_ID;
x=put(-1234.56789,nlmniidr32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.91 *NLMNIIDR Output*

Locale	Formatted value
id_ID	-IDR1.234,57
en_US	-IDR1,234.57
es_ES	-1.234,57 IDR

See Also

Format:

■ “NLMNLIDRw.d Format” on page 445

NLMNIILSw.d Format

Writes the monetary format of the international standard expression for Israel.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=he_IL;
x=put(-1234.56789,nlmniils32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.92 *NLMNIILS Output*

Locale	Formatted value
he_IL	-1,234.57 ILS
en_US	-ILS1,234.57
es_ES	-1.234,57 ILS

See Also

Format:

- “NLMNLILSw.d Format” on page 447

NLMNIINRw.d Format

Writes the monetary format of the international standard expression for India.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=hi_IN;
x=put(-1234.56789,nlmniinr32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.93 *NLMNIINR Output*

Locale	Formatted value
hi_IN	-INR1,234.57
en_US	-INR1,234.57
es_ES	-1.234,57 INR

See Also

Format:

- “NLMNLINRw.d Format” on page 448

NLMNIJPYw.d Format

Writes the monetary format of the international standard expression for Japan.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ja_JP;
x=put(-1234.56789,nlmnijpy32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.94 *NLMNIJPY Output*

Locale	Formatted value
ar_AE	-JPY1,234.57
en_US	-JPY1,234.57

Locale	Formatted value
es_ES	-1.234,57 JPY

See Also

Format:

- “NLMNLJPYw.d Format” on page 450

NLMNIKRWw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ko_KR;
x=put(-1234.56789,nlmnikrw32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.95 *NLMNIKRW Output*

Locale	Formatted value
ko_KR	-KRW1,234.57
en_US	-KRW1,234.57
es_ES	-1.234,57 KRW

See Also

Format:

- "NLMNLKRWw.d Format" on page 451

NLMNILTLw.d Format

Writes the monetary format of the international standard expression for Lithuania.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale-lt_LT;
x=put(-1234.56789,nlmniltl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.96 *NLMNILTL Output*

Locale	Formatted value
lt_LT	-1 234,57 LTL
en_US	-LTL1,234.57
es_ES	-1.234,57 LTL

See Also

Format:

■ “NLMNLLTLw.d Format” on page 452

NLMNILVLw.d Format

Writes the monetary format of the international standard expression for Latvia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=lv_LV;
x=put(-1234.56789,nlmnilvl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.97 *NLMNILVL Output*

Locale	Formatted value
lv_LV	-1 234,57 LVL
en_US	-LVL1,234.57
es_ES	-1.234,57 LVL

See Also

Format:

- "NLMNLLVLw.d Format" on page 454

NLMNIMOPw.d Format

Writes the monetary format of the international standard expression for Macau.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=zh_MO;
x=put(-1234.56789,nlmnimop32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.98 *NLMNIMOP Output*

Locale	Formatted value
zh_MO	-MOP1,234.57
en_US	-MOP1,234.57
es_ES	-1.234,57 MOP

See Also

Format:

- “NLMNLMOPw.d Format” on page 455

NLMNIMXNw.d Format

Writes the monetary format of the international standard expression for Mexico.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=es_MX;
x=put(-1234.56789,nlmnimxn32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.99 *NLMNIMXN Output*

Locale	Formatted value
es_MX	-MXN1,234.57
en_US	-MXN1,234.57

Locale	Formatted value
es_ES	-1.234,57 MXN

See Also

Format:

- “NLMNLMXNw.d Format” on page 457

NLMNIMYRw.d Format

Writes the monetary format of the international standard expression for Malaysia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ms_MY;
x=put(-1234.56789,nlmnimyr32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.100 *NLMNIMYR Output*

Locale	Formatted value
ms_MY	-MYR1,234.57
en_US	-MYR1,234.57
es_ES	-1.234,57 MYR

See Also

Format:

- “NLMNLMYRw.d Format” on page 458

NLMNINOKw.d Format

Writes the monetary format of the international standard expression for Norway.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=no_NO;
x=put(-1234.56789,nlmninok32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.101 *NLMNINOK Output*

Locale	Formatted value
no_NO	NOK -1 234,57
en_US	-NOK1,234.57
es_ES	-1.234,57 NOK

See Also

Format:

■ “NLMNLNOKw.d Format” on page 459

NLMNINZDw.d Format

Writes the monetary format of the international standard expression for New Zealand.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_NZ;
x=put(-1234.56789,nlmninzd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.102 *NLMNINZD Output*

Locale	Formatted value
en_NZ	-NZD1,234.57
en_US	-NZD1,234.57
es_ES	-1.234,57 NZD

See Also

Format:

- "NLMNLNZDw.d Format" on page 461

NLMNIPLNw.d Format

Writes the monetary format of the international standard expression for Poland.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pl_PL;
x=put(-1234.56789,nlmnipln32.2;
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.103 *NLMNIPLN Output*

Locale	Formatted value
pl_PL	-1 234,57 PLN
en_US	-PLN1,234.57
es_ES	-1.234,57 PLN

See Also

Format:

- “NLMNLPLNw.d Format” on page 462

NLMNIRUBw.d Format

Writes the monetary format of the international standard expression for Russia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ru_RU;
x=put(-1234.56789,nlmnirub32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.104 *NLMNIRUB Output*

Locale	Formatted value
ru_RU	-1 234,57 RUB
en_US	-RUB1,234.57

Locale	Formatted value
es_ES	-1.234,57 RUB

See Also

Format:

- “NLMNLRUBw.d Format” on page 464

NLMNISEKw.d Format

Writes the monetary format of the international standard expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=sv_SE;
x=put(-1234.56789,nlmnisek32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.105 *NLMNISEK Output*

Locale	Formatted value
sv_SE	-1 234,57 SEK
en_US	-SEK1,234.57
es_ES	-1.234,57 SEK

See Also

Format:

- “NLMNLSEKw.d Format” on page 465

NLMNISGDw.d Format

Writes the monetary format of the international standard expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_SG;
x=put(-1234.56789,nlmnisgd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.106 *NLMNISGD Output*

Locale	Formatted value
zh_SG	-SGD1,234.57
en_US	-SGD1,234.57
es_ES	-1.234,57 SGD

See Also

Format:

■ “NLMNLSGDw.d Format” on page 467

NLMNITHBw.d Format

Writes the monetary format of the international standard expression for Thailand.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=th_TH;
x=put(-1234.56789,nlmnithb32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.107 *NLMNITHB Output*

Locale	Formatted value
th_TH	-THB1,234.57
en_US	-THB1,234.57
es_ES	-1.234,57 THB

See Also

Format:

- "NLMNLTHBw.d Format" on page 468

NLMNITRYw.d Format

Writes the monetary format of the international standard expression for Turkey.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=tr_TR;
x=put(-1234.56789,nlmnitry32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.108 *NLMNITRY Output*

Locale	Formatted value
tr_TR	-TRY1.234,57
en_US	-TRY1,234.57
es_ES	-1.234,57 TRY

See Also

Format:

- “NLMNLTRYw.d Format” on page 469

NLMNITWDw.d Format

Writes the monetary format of the international standard expression for Taiwan.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_TW;
x=put(-1234.56789,nlmnitwd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.109 *NLMNITWD Output*

Locale	Formatted value
zh-TW	-TWD1,234.57
en_US	-TWD1,234.57

Locale	Formatted value
es_ES	-1.234,57 TWD

See Also

Format:

- “NLMNLTWDw.d Format” on page 471

NLMNIUSDw.d Format

Writes the monetary format of the international standard expression for Puerto Rico and the United States.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=es_US;
x=put(-1234.56789,nlmniusd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.110 *NLMNIUSD Output*

Locale	Formatted value
es_US	-USD1,234.57
en_US	-USD1,234.57
es_ES	-1.234,57 USD

See Also

Format:

- “NLMNLUSDw.d Format” on page 472

NLMNIZARw.d Format

Writes the monetary format of the international standard expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=af_ZA;
x=put(-1234.56789,nlmnizar32.2;
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.111 *NLMNIZAR Output*

Locale	Formatted value
af_ZA	-ZAR1 234,57
en_US	-ZAR1,234.57
es_ES	-1.234,57 ZAR

See Also

Format:

■ “NLMNLZARw.d Format” on page 474

NLMNLAEDx.d Format

Writes the monetary format of the local standard expression for the United Arab Emirates.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_AE
x=put(-1234.56789,nlmnlaed32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.112 *NLMNIAED Output*

Locale	Formatted value
ar_AE	-1,234.57 د.إ.
en_US	-1,234.57 د.إ.
es_ES	-1.234,57 د.إ.

See Also

Format:

- “NLMNIAEDw.d Format” on page 371

NLMNLAUDw.d Format

Writes the monetary format of the local standard expression for Australia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_AU;
x=put(-1234.56789,nlmnlaud32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.113 *NLMNIAUD Output*

Locale	Formatted value
en_AU	-AU$1,234.57
en_US	-AU$1,234.57
es_ES	-1.234,57 AU$

See Also

Format:

- “NLMNIAUDw.d Format” on page 372

NLMNLBGNw.d Format

Writes the monetary format of the local standard expression for Bulgaria.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=bg_BG;
x=put(-1234.56789,nlmnlbgn32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.114 *NLMNLBGN Output*

Locale	Formatted value
bg_BG	-1 234,57 лв
en_US	-лв1,234.57

Locale	Formatted value
es_ES	-1.234,57 лв

See Also

Format:

- “NLMNIBGNw.d Format” on page 373

NLMNLBRLw.d Format

Writes the monetary format of the local standard expression for Brazil.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pt_BR;
x=put(-1234.56789,nlmnlbrl32.2);
```

Table 10.115 *NLMNLBRL Output*

Locale	Formatted value
pt_BR	-R$ 1.234,57
en_US	-R$1,234.57
es_ES	-1.234,57 R$

See Also

Format:

- “NLMNIBRLw.d Format” on page 375

NLMNLCADw.d Format

Writes the monetary format of the local standard expression for Canada.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_CA;
x=put(-1234.56789,nlmnlcad32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.116 *NLMNLCAD Output*

Locale	Formatted value
en_CA	-CA$1,234.57
en_US	-CA$1,234.57
es_ES	-1.234,57 CA$

See Also

Format:

■ “NLMNICADw.d Format” on page 376

NLMNLCHFw.d Format

Writes the monetary format of the local standard expression for Liechtenstein and Switzerland.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=fr_CH;
x=put(-1234.56789,nlmnlchf32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.117 *NLMNLCHF Output*

Locale	Formatted value
fr_CH	-1 234.57 SFr.
en_US	-SFr.1,234.57
es_ES	-1.234,57 SFr.

See Also

Format:

- “NLMNICHFw.d Format” on page 378

NLMNLCNYw.d Format

Writes the monetary format of the local standard expression for China.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_CN;
x=put(-1234.56789,nlmnlcny32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.118 *NLMNLCNY Output*

Locale	Formatted value
zh_CN	-RMB1,234.57
en_US	-RMB1,234.57
es_ES	-1.234,57 RMB

See Also

Format:

- “NLMNICNYw.d Format” on page 379

NLMNLCZKw.d Format

Writes the monetary format of the local standard expression for Czechia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=cs_CZ;
x=put(-1234.56789,nlmnlczk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.119 *NLMNLCZK Output*

Locale	Formatted value
cs_CZ	-1 234,57 Kč
en_US	-Kč1,234.57

Locale	Formatted value
es_ES	-1.234,57 Kč

See Also

Format:

- “NLMNICZKw.d Format” on page 380

NLMNLDKKw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=da_dk;
x=put(-1234.56789,nlmnldkk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.120 *NLMNLDKK Output*

Locale	Formatted value
da_dk	-1.234,57 kr
en_US	-kr1,234.57
es_ES	-1.234,57 kr

Locale	Formatted value
da_DK	-1.234,57 kr
en_US	(kr1,234.57)
es_ES	-1.234,57 kr

See Also

Format:

- “NLMNIDKKw.d Format” on page 382

NLMNLEEKw.d Format

Writes the monetary format of the local standard expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

Notes: Estonia uses the Euro for curency. NLMNIEEK is documented for historical purposes.

Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=et_EE;
x=put(-1234.56789,nlmnleek32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.121 *NLMNLEEK Output*

Locale	Formatted value
et_EE	-1 234,57 kr

Locale	Formatted value
en_US	-kr1,234.57
es_ES	-1.234,57 kr

See Also

Format:

- “NLMNIEEKw.d Format” on page 383

NLMNLEGPw.d Format

Writes the monetary format of the local standard expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_EG;
x=put(-1234.56789,nlmnlegp32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.122 *NLMNLEGP Output*

Locale	Formatted value
ar_EG	-1,234.57 ج.م.
en_US	-1.234,57 ج.م.
es_ES	-1.234,57 ج.م.

See Also

Format:

- “NLMNIEGPw.d Format” on page 385

NLMNLEURw.d Format

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=de_DE;
y=put(-1234.56789,nlmnleur32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.123 *NLMNLEUR Output*

Locale	Formatted value
de_DE	-1.234,57 €
en_US	-€1,234.57
es_ES	-1.234,57 €

See Also

Format:

- “NLMNIEURw.d Format” on page 386

NLMNLGBPw.d Format

Writes the monetary format of the local standard expression for the United Kingdom.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_GB;
x=put(-1234.56789,nlmnlgbp32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.124 *NLMNLGBP Output*

Locale	Formatted value
en_GB	-£1,234.57
en_US	-£1,234.57
es_ES	-1.234,57 £

See Also

Format:

- "NLMNIGBPw.d Format" on page 388

NLMNLHKDw.d Format

Writes the monetary format of the local standard expression for Hong Kong.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_HK;
x=put(-1234.56789,nlmnlhkd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.125 *NLMNLHKD Output*

Locale	Formatted value
en_HK	-HK$1,234.57
en_US	-HK$1,234.57
es_ES	-1.234,57 HK$

See Also

Format:

- “NLMNIHKDw.d Format” on page 389

NLMNLHRKw.d Format

Writes the monetary format of the local standard expression for Croatia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hr_HR;
x=put(-1234.56789,nlmnlhrk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.126 *NLMNLHRK Output*

Locale	Formatted value
hr_HR	-1.234,57 Kn
en_US	-Kn1,234.57
es_ES	-1.234,57 Kn

See Also

Format:

- “NLMNIHRKw.d Format” on page 390

NLMNLHUFw.d Format

Writes the monetary format of the local standard expression for Hungary.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hu_HU;
x=put(-1234.56789,nlmnlhuf32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.127 *NLMNLHUF Output*

Locale	Formatted value
hu_HU	-1 234,57 Ft
en_US	-Ft1,234.57
es_ES	-1.234,57 Ft

See Also

Format:

- “NLMNIHUFw.d Format” on page 392

NLMNLIDRw.d Format

Writes the monetary format of the local standard expression for Indonesia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=id_ID;
x=put(-1234.56789,nlmnlidr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.128 *NLMNLIDR Output*

Locale	Formatted value
af_ZA	-Rp1.234,57
en_US	-Rp1,234.57

Locale	Formatted value
es_ES	-1.234,57 Rp

See Also

Format:

- “NLMNIIDRw.d Format” on page 393

NLMNLILSw.d Format

Writes the monetary format of the local standard expression for Israel.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=he_IL;
x=put(-1234.56789,nlmnlils32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.129 *NLMNLILS Output*

Locale	Formatted value
he_IL	-1,234.57ח"ש
en_US	-1,234.57ח"ש
es_ES	-1,234.57ח"ש

See Also

Format:

- "NLMNIILSw.d Format" on page 395

NLMNLINRw.d Format

Writes the monetary format of the local standard expression for India.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hi_IN;
x=put(-1234.56789,nlmnlinr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.130 *NLMNLINR Output*

Locale	Formatted value
hi_IN	-₹s1,234.57
en_US	-₹s1,234.57
es_ES	-1.234,57 ₹s

See Also

Format:

■ “NLMNIINRw.d Format” on page 396

NLMNLJPYw.d Format

Writes the monetary format of the local standard expression for Japan.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ja_JP;
x=put(-1234.56789,nlmnljpy32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.131 *NLMNLJPY Output*

Locale	Formatted value
ja_JP	-￥ 1,234.57
en_US	-￥ 1,234.57
es_ES	-1.234,57 ￥

See Also

Format:

- “NLMNIJPYw.d Format” on page 397

NLMNLKRWw.d Format

Writes the monetary format of the local standard expression for South Korea.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ko_KR;
x=put(-1234.56789,nlmnlkrw32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.132 *NLMNLKRW Output*

Locale	Formatted value
ko_KR	-₩1,234.57
en_US	-₩1,234.57
es_ES	-1.234,57 ₩

See Also

Format:

- “NLMNIKRWw.d Format” on page 399

NLMNLLTLw.d Format

Writes the monetary format of the local standard expression for Lithuania.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=lt_LT;
x=put(-1234.56789,nlmnlltl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.133 *NLMNLLTL Output*

Locale	Formatted value
lt_LT	-1 234,57 Lt
en_US	-Lt1,234.57

Locale	Formatted value
es_ES	-1.234,57 Lt

See Also

Format:

- “NLMNILTLw.d Format” on page 400

NLMNLLVLw.d Format

Writes the monetary format of the local standard expression for Latvia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=lv_LV;
x=put(-1234.56789,nlmnllvl32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.134 *NLMNLLVL Output*

Locale	Formatted value
lv_LV	-1 234,57 Ls
en_US	-Ls1,234.57
es_ES	-1.234,57 Ls

See Also

Format:

- “NLMNILVLw.d Format” on page 402

NLMNLMOPw.d Format

Writes the monetary format of the local standard expression for Macau.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_MO;
x=put(-1234.56789,nlmnlmop32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.135 *NLMNLMOP Output*

Locale	Formatted value
zh_MO	-P1,234.57
en_US	-P1,234.57
es_ES	-1.234,57 P

See Also

Format:

■ “NLMNIMOPw.d Format” on page 403

NLMNLMXNw.d Format

Writes the monetary format of the local standard expression for Mexico.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=es_MX;
x=put(-1234.56789,nlmnlmxn32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.136 *NLMNLMXN Output*

Locale	Formatted value
es_MX	-MX$1,234.57
en_US	-MX$1,234.57
es_ES	-1.234,57 MX$

See Also

Format:

- “NLMNIMXNw.d Format” on page 404

NLMNLMYRw.d Format

Writes the monetary format of the local standard expression for Malaysia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ms_MY;
x=put(-1234.56789,nlmnlmyr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.137 *NLMNLMYR Output*

Locale	Formatted value
ms_MY	-R1,234.57
en_US	-R1,234.57
es_ES	-1.234,57 R

See Also

Format:

- “NLMNIMYRw.d Format” on page 406

NLMNLNOKw.d Format

Writes the monetary format of the local standard expression for Norway.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=no_NO;
x=put(-1234.56789,nlmnlnok32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.138 *NLMNLNOK Output*

Locale	Formatted value
no_NO	kr -1 234,57
en_US	-kr1,234.57

Locale	Formatted value
es_ES	-1.234,57 kr

See Also

Format:

- “NLMNINOKw.d Format” on page 407

NLMNLNZDw.d Format

Writes the monetary format of the local standard expression for New Zealand.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_NZ;
x=put(-1234.56789,nlmnlnzd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.139 *NLMNLNZD Output*

Locale	Formatted value
en_NZ	-NZ$1,234.57
en_US	-NZ$1,234.57
es_ES	-1.234,57 NZ$

Locale	Formatted value
en_NZ	(NZ$1,234.57)
en_US	(NZ$1,234.57)
es_ES	-1.234,57 NZ$

See Also

Format:

- “NLMNINZDw.d Format” on page 409

NLMNLPLNw.d Format

Writes the monetary format of the local standard expression for Poland.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pl_PL;
x=put(-1234.56789,nlmnlpln32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.140 *NLMNLPLN Output*

Locale	Formatted value
pl_PL	-1 234,57 zł
en_US	-zł1,234.57
es_ES	-1.234,57 zł

See Also

Format:

- “NLMNIPLNw.d Format” on page 410

NLMNLRUBw.d Format

Writes the monetary format of the local standard expression for Russia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ru_RU;
x=put(-1234.56789,nlmnlrub32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.141 *NLMNLRUB Output*

Locale	Formatted value
ru_RU	-1 234,57 руб
en_US	-руб1,234.57
es_ES	-1.234,57 руб

See Also

Format:

- “NLMNIRUBw.d Format” on page 411

NLMNLSEKw.d Format

Writes the monetary format of the local standard expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=sv_SE;
x=put(-1234.56789,nlmnlsek32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.142 *NLMNLSEK Output*

Locale	Formatted value
sv_SE	-1 234,57 kr
en_US	-kr1,234.57
es_ES	-1.234,57 kr

See Also

Format:

- “NLMNISEKw.d Format” on page 413

NLMNLSGDw.d Format

Writes the monetary format of the local standard expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_SG;
x=put(-1234.56789,nlmnlsgd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.143 *NLMNLSGD Output*

Locale	Formatted value
zh_SG	-SG$1,234.57
en_US	-SG$1,234.57
es_ES	-1.234,57 SG$

See Also

Format:

- “NLMNISGDw.d Format” on page 414

NLMNLTHBw.d Format

Writes the monetary format of the local standard expression for Thailand.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=th_TH;
x=put(-1234.56789,nlmnlthb32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.144 *NLMNLTHB Output*

Locale	Formatted value
th_TH	-฿1,234.57
en_US	-฿1,234.57
es_ES	-1.234,57 ฿

See Also

Format:

- “NLMNITHBw.d Format” on page 416

NLMNLTRYw.d Format

Writes the monetary format of the local standard expression for Turkey.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=tr_TR;
x=put(-1234.56789,nlmnltry32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.145 *NLMNLTRY Output*

Locale	Formatted value
tr_TR	-YTL1.234,57
en_US	-YTL1,234.57

Locale	Formatted value
es_ES	-1.234,57 YTL

See Also

Format:

- “NLMNITRYw.d Format” on page 417

NLMNLTWDw.d Format

Writes the monetary format of the local standard expression for Taiwan.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_TW;
x=put(-1234.56789,nlmnltwd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.146 *NLMNLTWD Output*

Locale	Formatted value
zh_TW	-NT$1,234.57
en_US	-NT$1,234.57
es_ES	-1.234,57 NT$

See Also

Format:

- “NLMNITWDw.d Format” on page 418

NLMNLUSDw.d Format

Writes the monetary format of the local standard expression for Puerto Rico and the United States.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=es_US;
x=put(-1234.56789,nlmnlusd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.147 *NLMNLUSD Output*

Locale	Formatted value
es_US	-US$1,234.57
en_US	-US$1,234.57
es_ES	-1.234,57 US$

See Also

Format:

■ “NLMNIUSDw.d Format” on page 420

NLMNLZARw.d Format

Writes the monetary format of the local standard expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=af_ZA;
x=put(-1234.56789,nlmnlzar32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

Table 10.148 *NLMNLZAR Output*

Locale	Formatted value
af_ZA	-R1 234,57
en_US	-R1,234.57
es_ES	-1.234,57 R

See Also

Format:

- “NLMNIZARw.d Format” on page 421

NLMNYw.d Format

Writes the monetary format of the local standard expression using local currency that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 in standard pattern 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*, NLMNYAC*w.d*, NLMNYI*w.d* and NLMNYIAC*w.d* formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNIAC*w.d* and NLMNYAC*w.d* formats use accounting pattern, while NLMNY*w.d* and NLMNYI*w.d* formats use standard pattern; the NLMNYI*w.d* and NLMNYIAC*w.d* formats uses three-letter international currency codes, such as USD, while NLMNY*w.d* and NLMNYAC*w.d* formats 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,nlmnyi32.2);
y=put(-1234.56789,nlmny32.2);
x2=put(-1234.56789,nlmnyiac32.2);
y2=put(-1234.56789,nlmnyac32.2);
```

```
z=put(-1234.56789,dollar32.2);
```

Statements	Results
	----+----1----+
put x=;	-USD1,234.57
put y=;	$-1,234.57
put x2;	(USD1,234.57)
put y2;	($1,234.57)
put z;	$-1,234.57

See Also

Format:

- “NLMNYIw.d Format” on page 479

Informats:

- “NLMNYw.d Informat” on page 899
- “NLMNYIw.d Informat” on page 901

NLMNYACw.d Format

Writes the monetary format of the local accounting expression using local currency that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: This format is available beginning in 2023.11.

Syntax

NLMNYAC*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 NLMNYAC*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLMNYAC*w.d* format writes numeric values in an accounting pattern by using the currency symbol, the thousands separator, and the decimal separator that is used by the locale.

Note: The NLMNYAC*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*, NLMNYAC*w.d*, NLMNYI*w.d* and NLMNYIAC*w.d* formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNIAC*w.d* and NLMNYAC*w.d* formats use accounting pattern, while NLMNY*w.d* and NLMNYI*w.d* formats use standard pattern; the NLMNYI*w.d* and NLMNYIAC*w.d* formats uses three-letter international currency codes, such as USD, while NLMNY*w.d* and NLMNYAC*w.d* formats uses local currency symbols, such as $.

The NLMNYAC*w.d* format is similar to the DOLLAR*w.d* format except that the NLMNYAC*w.d* format is locale-specific and uses an accounting pattern.

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);
x2=put(-1234.56789,nlmnyiac32.2);
y2=put(-1234.56789,nlmnyac32.2);
```

```
z=put(-1234.56789,dollar32.2);
```

Statements	Results
	----+----1----+
put x=; put y=;	-USD1,234.57 -$1,234.57
put x2=;	(USD1,234.57)
put y2=;	($1,234.57)
put z=;	$-1,234.57

See Also

Format:

- “NLMNYIw.d Format” on page 479
- “NLMNYw.d Format”
- “NLMNYIACw.d Format”

Informats:

- “NLMNYw.d Informat” on page 899
- “NLMNYIw.d Informat” on page 901

NLMNYIw.d Format

Writes the monetary format of the international standard expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 in standard pattern 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*, NLMNYIAC*w.d* and NLMNYI*w.d* formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNIAC*w.d* and NLMNYAC*w.d* formats use accounting pattern, while NLMNY*w.d* and NLMNYI*w.d* formats use standard pattern; the NLMNYI*w.d* and NLMNYIAC*w.d* formats use three-letter international currency codes, such as USD, while NLMNY*w.d* and NLMNYAC*w.d* formats use 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);
x2=put(-1234.56789,nlmnyiac32.2);
y2=put(-1234.56789,nlmnyac32.2);
z=put(-1234.56789,dollar32.2);
```

Statements	Results
	----+----1----+
put x=;	-USD1,234.57
put y=;	-$1,234.57
put x2;	(USD1,234.57)
put y2;	($1,234.57)
put z=;	$-1,234.57

See Also

Format:

- “NLMNYw.d Format” on page 475

Informats:

- “NLMNYw.d Informat” on page 899
- “NLMNYIw.d Informat” on page 901

NLMNYIACw.d Format

Writes the monetary format of the international accounting expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: This format is available beginning in 2023.11.

Syntax

NLMNYIAC*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 NLMNYIAC*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLMNYIAC*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 NLMNYIAC*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* NLMNYAC*w d*, NLMNYIAC*w d* and NLMNYI*w.d* formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNYIAC*w.d* and NLMNYAC*w.d*formats use accounting pattern, while NLMNY*w. d* and NLMNYI*w. d* formats use standard pattern; the NLMNYI*w. d* and NLMNYIAC*w. d* formats use three-letter international currency codes, such as USD, while NLMNY*w.d* and NLMNYAC*w.d* formats use 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);
x2=put(-1234.56789,nlmnyiac32.2);
y2=put(-1234.56789,nlmnyac32.2);
z=put(-1234.56789,dollar32.2);
```

Statements	Results
put x=;	-USD1,234.57
put y=;	-$1,234.57
put x2=;	(USD1,234.57)
put y2=;	($1,234.57)
put z=;	$-1,234.57

See Also

Format:

- “NLMNYw.d Format” on page 475
- “NLMNYACw.d Format”
- “NLMNYIw.d Format”

Informats:

- “NLMNYw.d Informat” on page 899
- “NLMNYIw.d Informat” on page 901

NLNUMw.d Format

Writes the numeric format of the local expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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* informat 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:

- “NLNUMIw.d Format” on page 485

Informats:

- “NLNUMw.d Informat” on page 902
- “NLNUMIw.d Informat” on page 904

NLNUMIw.d Format

Writes the numeric format of the international expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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;`	`-1,234,356.79`

Statements	Results
`put x=;`	

See Also

Format:

- “NLNUMw.d Format” on page 483

Informats:

- “NLNUMw.d Informat” on page 902
- “NLNUMIw.d Informat” on page 904

NLPCTw.d Format

Writes percentage data of the local expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “NLPCTIw.d Format” on page 489

Informats:

- “NLPCTw.d Informat” on page 906
- “NLPCTIw.d Informat” on page 907

NLPCTIw.d Format

Writes percentage data of the international expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “NLPCTw.d Format” on page 487

Informats:

- “NLPCTIw.d Informat” on page 906
- “NLPCTIw.d Informat” on page 907

NLPCTNw.d Format

Writes percentages, using a minus sign for negative values that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Right

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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%

NLPCTPw.d Format

Writes locale-specific numeric values as percentages that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Right

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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%)

NLPVALUEw.d Format

Writes p-values of the local expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- "PVALUEw.d Format" in *SAS Formats and Informats: Reference*

NLSTRMONw.d Format

Writes the month name that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 10th month (October)
- 11 = the 11th month (November)
- 12 = the 12th month (December)

Example

This example uses the English_UnitedStates session encoding.

```
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
```

NLSTRQTRw.d Format

Writes a numeric value as the quarter-of-the-year that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

NLSTRWKw.d Format

Writes a numeric value as the day-of-the-week that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

NLTIMAPw. Format

Writes a SAS time value as a time value with a.m. or p.m. that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 98 for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

NLTIMAP also writes date time values.

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:

- “NLTIMEw. Format” on page 500

NLTIMEw. Format

Writes a SAS time value as a time value that is appropriate for the current SAS locale. NLTIME also writes SAS date-time values.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “NLTIMAPw. Format” on page 499

NLTIMELw. Format

Writes a SAS time value as a time value in a long-uniform pattern that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLTIMEL*w*.

Optional Argument

w
specifies the width of the input field. If necessary, SAS abbreviates the time to fit the format width.

Default 30

Range 10–200

Details

The NLTIMEL*w*. format writes the time in a long-uniform pattern.

Examples

Example 1

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

```
data a;
options locale=English_UnitedStates;
options timezone='America/Chicago';
   time=59083;
   put time nltimel.;
   put time nltimel10.;
   put time nltimel200.;
run;
```

```
04:24:43 PM CDT
04:24 PM
04:24:43 PM CDT
```

Example 2

```
data b;
options locale=German_Germany;
options timezone='Europe/Berlin';
   time=59083;
   put time nltimel.;
   put time nltimel10.;
   put time nltimel12.;
   put time nltimel18.;
   put time nltimel200.;
run;
```

```
16:24:43 CEST
16:24:43
16:24:43
16:24:43 CEST
16:24:43 CEST
```

Example 3

```
data c;
options locale=English_UnitedStates;
options timezone='Etc/GMT-6';
   time=59083;
   put time nltimel.;
   put time nltimel10.;
```

```
   put time nltime1200.;
run;
```

```
04:24:43 PM GMT+6
04:24 PM
04:24:43 PM GMT+6
```

NLTIMEMw. Format

Writes a SAS time value as a time value in a medium-uniform pattern that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLTIMEM*w*.

Optional Argument

w
specifies the width of the input field. If necessary, SAS abbreviates the time to fit the format width.

Default 14

Range 10–200

Details

The NLTIMEM*w*. format writes the time in a medium-uniform pattern.

Example

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

```
data a;
options locale=English_UnitedStates;
   time=59083;
   put time nltimem.;
   put time nltimem10.;
   put time nltimem200.;
run;
```

```
04:24:43 PM
04:24 PM
04:24:43 PM
```

```
data b;
options locale=German_Germany;
   time=59083;
   put time nltimem.;
   put time nltimem10.;
   put time nltimem12.;
   put time nltimem18.;
   put time nltimem200.;
run;
```

```
16:24:43
16:24:43
16:24:43
16:24:43
16:24:43
```

NLTIMESw. Format

Writes a SAS time value as a time value in a short-uniform pattern that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLTIMES*w*.

Optional Argument

w.
specifies the width of the input field. If necessary, SAS abbreviates the time to fit the format width.

Default 11

Range 10–200

Details

The NLTIMES*w*. format writes the time in a short-uniform pattern.

Example

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

```
data a;
options locale=English_UnitedStates;
   time=59083;
   put time nltimes.;
   put time nltimes10.;
   put time nltimes200.;
run;
```

```
04:24 PM
04:24 PM
04:24 PM
```

```
data b;
options locale=German_Germany;
   time=59083;
   put time nltimes.;
   put time nltimes10.;
   put time nltimes12.;
   put time nltimes18.;
   put time nltimes200.;
run;
```

```
16:24
16:24
16:24
16:24
16:24
```

$UCS2Bw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

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

`data _null_;`
`x = '大';`
`y=put(x,$ucs2b2.);`
`put y $hex.;`
`run;`
	`5927`

See Also

Formats:

- "$UCS2Lw. Format" on page 510
- "$UCS2Xw. Format" on page 513
- "$UTF8Xw. Format" on page 533
- "$UCS2BEw. Format" on page 508

Informats:

- "$UCS2Bw. Informat" on page 915
- "$UCS2BEw. Informat" on page 917
- "$UCS2Lw. Informat" on page 918
- "$UCS2Xw. Informat" on page 921
- "$UTF8Xw. Informat" on page 938

$UCS2BEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Results
	`----+----1`
`x ='592700410042'x;` `put x $ucs2be.;`	丂AB

See Also

Formats:

- “$UCS2Bw. Format” on page 507

Informats:

- "$UCS2Bw. Informat" on page 915
- "$UCS2BEw. Informat" on page 917

$UCS2Lw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1`

`data _null_;`
`x ='太';`
`y=put(x,$ucs2l2.);`
`put y $hex.;`
`run;`
	`2759`

See Also

Formats:

- "$UCS2Bw. Format" on page 507
- "$UCS2LEw. Format" on page 511
- "$UCS2Xw. Format" on page 513
- "$UTF8Xw. Format" on page 533

Informats:

- "$UCS2Bw. Informat" on page 915
- "$UCS2Lw. Informat" on page 918
- "$UCS2LEw. Informat" on page 919
- "$UCS2Xw. Informat" on page 921
- "$UTF8Xw. Informat" on page 938

$UCS2LEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1`
`x ='275941004200'x;` `put x $ucs2le.;`	人AB

See Also

Format:

- “$UCS2Lw. Format” on page 510

Informats:

- “$UCS2Lw. Informat” on page 918
- “$UCS2LEw. Informat” on page 919

$UCS2Xw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1`
`x ='大';` `put x $ucs2x2.;`	`'5927'x (binary) or '2759'x (little endian)`

See Also

Formats:

- “$UCS2Bw. Format” on page 507
- “$UCS2XEw. Format” on page 514
- “$UCS2Lw. Format” on page 510
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Lw. Informat” on page 918
- “$UCS2Xw. Informat” on page 921
- “$UCS2XEw. Informat” on page 922
- “$UTF8Xw. Informat” on page 938

$UCS2XEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1`
`x ='e5a4a7'x; /* Japanese '大' in UTF8 */;` `put x $utf8xe10.;`	大

See Also

Format:

- "$UCS2Xw. Format" on page 513

Informats:

- "$UCS2Xw. Informat" on page 921
- "$UCS2XEw. Informat" on page 922

$UCS4Bw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1`
`x ='大';` `put x $ucs4b4.;`	`'00005927'x (binary)`

See Also

Formats:

- “$UCS2Lw. Format” on page 510
- “$UCS2Xw. Format” on page 513
- “$UCS4BEw. Format” on page 517
- “$UCS4Lw. Format” on page 519
- “$UCS4Xw. Format” on page 522
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Lw. Informat” on page 918
- “$UCS2Xw. Informat” on page 921
- “$UCS4Bw. Informat” on page 923
- “$UCS4Lw. Informat” on page 924
- “$UCS4Xw. Informat” on page 926
- “$UTF8Xw. Informat” on page 938

$UCS4BEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1`
`x ='00005927000004100000042'x;` `put x $ucs4be.;`	丈AB

See Also

Format:

- "$UCS4Bw. Format" on page 516

Informat:

- "$UCS4Bw. Informat" on page 923

$UCS4Lw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1`

`data _null_;`
`x = '大';`
`y=put(x,$ucs4l4.);`
`put y $hex.;`
`run;`
	`2759`

See Also

Formats:

- “$UCS2Bw. Format” on page 507
- “$UCS2Xw. Format” on page 513
- “$UCS4Bw. Format” on page 516
- “$UCS4LEw. Format” on page 521
- “$UCS4Xw. Format” on page 522
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Lw. Informat” on page 918
- “$UCS2Xw. Informat” on page 921
- “$UCS4Bw. Informat” on page 923
- “$UCS4Lw. Informat” on page 924
- “$UCS4Xw. Informat” on page 926
- “$UTF8Xw. Informat” on page 938

$UCS4LEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	----+----1
x ='27590000410000042000000'x; put x $ucs4le.;	大AB

See Also

Format:

- “$UCS4Lw. Format” on page 519

Informat:

- “$UCS4Lw. Informat” on page 924

$UCS4Xw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating environment.

Statements	Results
	`----+----1`
`x = '大';` `put x $ucs4x4.;`	`'00005927'x (binary) or '27590000'x (little endian)`

See Also

Formats:

- “$UCS2Lw. Format” on page 510
- “$UCS4XEw. Format” on page 524
- “$UCS2Xw. Format” on page 513
- “$UCS4Bw. Format” on page 516
- “$UCS4Lw. Format” on page 519
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Lw. Informat” on page 918
- “$UCS2Xw. Informat” on page 921

- "$UCS4Bw. Informat" on page 923
- "$UCS4Bw. Format" on page 516
- "$UCS4Lw. Informat" on page 924
- "$UCS4Xw. Informat" on page 926
- "$UTF8Xw. Informat" on page 938

$UCS4XEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1`
`x ='27590000410000042000000'x;` `put x $ucs4be4.;`	大AB (little endian)

See Also

Format:

- "$UCS4Xw. Format" on page 522

Informat:

- "$UCS4Xw. Informat" on page 926

$UESCw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating system.

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

`x='犬' ;`
`y='u5927'`
`z='uu5927';`
`put x = $uesc10. ;`
`put y = $uesc10. ;`
`put z = $uesc10. ;`
	`¥u5927`
	`¥uu5927`
	`¥uuu5927`

See Also

Formats:

- “$UESCEw. Format” on page 527

Informats:

- “$UESCw. Informat” on page 928
- “$UESCEw. Informat” on page 930

$UESCEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating system.

Statements	Results
	----+----1----+----2
x=put('¥u5927',$uesce10.) ; x=put('¥uu5927',$uesce10.) ; x=put('¥uuu5927',$uesce10.) ;	x=大 x=¥u5927 x=¥uu5927

See Also

Format:

- “$UESCw. Format” on page 525

Informats:

- “$UESCw. Informat” on page 928
- “$UESCEw. Informat” on page 930

$UNCRw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating system.

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:

- “$UNCREw. Format” on page 529

Informats:

- “$UNCRw. Informat” on page 931
- “$UNCREw. Informat” on page 933

$UNCREw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating system.

Statements	Results
	`----+----1`
`x='大abc';` `put x $uncre10.;`	大abc

See Also

Formats:

- “$UNCRw. Format” on page 528

Informats:

- "$UNCRw. Informat" on page 931
- "$UNCREw. Informat" on page 933

$UPARENw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating system.

Statements	Results
	----+----1----+----2----+----3----+
x='大'; y='abc3'; put x $uparen7.; put y $uparen28.;	<u5927> <u0061> <u0062> <u0063> <u0033>

See Also

Formats:

- “$UPARENEw. Format” on page 532

Informats:

- “$UPARENw. Informat” on page 934
- “$UPARENEw. Informat” on page 936

$UPARENEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating system.

Statements	Results
	`----+`
`x='<u0061><u0062><u0063><u0033>';` `put x $uparene4.;`	`abc3`

See Also

Formats:

- “$UPARENw. Format” on page 531

Informats:

- “$UPARENw. Informat” on page 934
- “$UPARENEw. Informat” on page 936

$UTF8Xw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating environment.

Statements	Results
	`----+----1`
`x ='91E5'x; ; /* Japanese '木' in Shift-JIS */` `put x $utf8x10.;`	`x='e5a4a7'x`

See Also

Formats:

- “$UCS2Bw. Format” on page 507
- “$UCS2Lw. Format” on page 510
- “$UCS2Xw. Format” on page 513

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Lw. Informat” on page 918
- “$UCS2Xw. Informat” on page 921

$UTF8XEw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating environment.

Statements	Results
	`----+----1`
x = unicode('u5927'); `put x $utf8xe10.;`	大

See Also

Formats:

- “$UTF8Xw. Format” on page 533

Informats:

- "$UTF8Xw. Informat" on page 938

$VSLOGw. Format

Processes a character string that is in visual order, and then writes the character string in left-to-right logical order.

Categories: BIDI Text Handling
CAS

Alignment: Left

Syntax

$VSLOG*w*.

Syntax Description

w
specifies the width of the output field.

Default 200

Range 1–32767

Details

The $VSLOG*w*. format is used when transferring data that is stored in visual order.

Note: The $VSLOG*w*. format does not correctly process all combinations of data strings.

Comparisons

The $VSLOG*w*. format performs processing that is opposite to the $VSLOGR*w*. format.

Example

The following example uses the Hebrew input value of "טיסה flight".

Statements	Results
	----+----1----+----2----+
put text $vslog12.;	טיסה flight

The following example uses the Arabic input value of " ذاتي " computer.

Statements	Results
	----+----1----+----2----+
put text $vslog12.;	ذات computer

See Also

Format:

- "$VSLOGRw. Format" on page 537

Informats:

- "$VSLOGw. Informat" on page 940
- "$VSLOGRw. Informat" on page 941

$VSLOGRw. Format

Processes a character string that is in visual order, and then writes the character string in right-to-left logical order.

Categories: BIDI Text Handling
CAS

Alignment: Left

Syntax

$VSLOGR*w*.

Syntax Description

w
specifies the width of the output field.

Default 200

Range 1–32767

Details

The $VSLOGR*w*. format is used when transferring data that is stored in visual order.

Note: The $VSLOGR*w*. format does not correctly process all combinations of data strings.

Comparisons

The $VSLOGR*w*. format performs processing that is opposite to the $VSLOG*w*. format.

Example

The following example uses the Hebrew input value of "טיסה flight."

Statements	Results
	`----+----1----+`
`put text $logvs12;`	`flight` טיסה

The following example uses the Arabic input value of " كمبيوتر " computer.

Statements	Results
	----+----1----+
put text $logvs12;	ذات computer

See Also

Informats:

- “$VSLOGw. Informat” on page 940
- “$VSLOGRw. Informat” on page 941

WEEKUw. Format

Writes a week number in decimal format by using the U algorithm.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.);`	`W00` `03W00`

Statements	Results

x=put(sasdate,weeku7.);
y=put(sasdate,weeku9.);
z=put(sasdate,weeku11.);
put v;
put w;
put x;
put y;
put z;
	03W0004
	2003W0004
	2003-W00-04

See Also

Formats:

- “WEEKVw. Format” on page 541
- “WEEKWw. Format” on page 543

WEEKVw. Format

Writes a week number in decimal format by using the V algorithm.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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----+`
`v=put(sasdate,weekv3.);` `w=put(sasdate,weekv5.);`	`W01` `03W01`

Statements	Results

x=put(sasdate,weekv7.);
y=put(sasdate,weekv9.);
z=put(sasdate,weekv11.);
put v;
put w;
put x;
put y;
put z;
	03W0103
	2003W0103
	2003-W01-03

See Also

Formats:

- “WEEKUw. Format” on page 539
- “WEEKWw. Format” on page 543

WEEKWw. Format

Writes a week number in decimal format by using the W algorithm.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.);`	`W03`
`w=put(sasdate,weekw5.);`	`03W03`
`x=put(sasdate,weekw7.);`	`03W0003`
`y=put(sasdate,weekw9.);`	`2003W0003`

Statements	Results

`z=put(sasdate,weekw11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	`2003-W00-03`

See Also

Formats:

- “WEEKUw. Format” on page 539
- “WEEKVw. Format” on page 541

YENw.d Format

Writes numeric values with yen signs, commas, and decimal points.

Categories: CAS
Numeric

Alignment: Right

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.

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

See Also

Informat:

- “YENw.d Informat” on page 943

YYWEEKUw. Format

Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “WEEKUw. Format” on page 539

YYWEEKVw. Format

Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 0 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:

- “WEEKVw. Format” on page 541

YYWEEKWw. Format

Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this format supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “WEEKWw. Format” on page 543

PART 5

Functions for NLS

Chapter 11
Internationalization Compatibility for SAS String Functions **555**

Chapter 12
VARCHAR Support in Functions **583**

Chapter 13
Dictionary of Functions for NLS **587**

11

Internationalization Compatibility for SAS String Functions

Internationalization Compatibility for SAS String Functions 555

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 in a string. Table 11.4 on page 557 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.

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 11.1 *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 11.2 *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 11.3 on page 557 solves this issue by substituting the SUBSTR() function, which works only 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 11.3 *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.

String functions are assigned I18N levels depending on whether the functions can process DBCS, MBCS, or SBCS. 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, DBCS, and MBCS (UTF-8) data.

Table 11.4 *SAS String Functions*

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"ANORM420 Function" (p. 595)	Returns a normalized string from an input string		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	encoded in EBCDIC420.			
"ANYALNUM Function" in *SAS 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	
"ANYALPHA Function" in *SAS 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 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 Functions and CALL Routines: Reference*	Searches a character string for a digit, and returns the first position at which the digit is found. **Note:** This function is assigned an I18N Level 1 status unless a VARCHAR variable is used, or if the function is threaded or runs in DS2. If these exceptions occur, then this function is assigned an I18N Level 2 status.		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"ANYFIRST Function" in *SAS 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. **Note:** This function is assigned an I18N Level 1 status unless a VARCHAR variable is used, or if the function is threaded or runs in DS2. If these exceptions occur, then this function is assigned an I18N Level 2 status.		X	
"ANYGRAPH Function" in *SAS 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 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 Functions and CALL Routines: Reference*	Searches a character string for a character that is valid in a SAS variable name		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	under VALIDVARNAME= V7, and returns the first position at which that character is found. **Note:** This function is assigned an I18N Level 1 status unless a VARCHAR variable is used, or if the function is threaded or runs in DS2. If these exceptions occur, then this function is assigned an I18N Level 2 status.			
"ANYPRINT Function" in *SAS 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 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 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		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	which that character is found.			
"ANYUPPER Function" in *SAS 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 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. **Note:** This function is assigned an I18N Level 1 status unless a VARCHAR variable is used, or if the function is threaded or runs in DS2. If these exceptions occur, then this function is assigned an I18N Level 2 status.		X	
"BASECHAR Function" (p. 606)	Converts characters to base characters.			X
"BYTE Function" in *SAS Functions and CALL Routines: Reference*	Returns one character in the ASCII or the EBCDIC collating sequence.	X		
"CAT Function" in *SAS Functions and CALL Routines: Reference*	Does not remove leading or trailing blanks, and returns			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	a concatenated character string.			
"CATQ Function" in *SAS 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 Functions and CALL Routines: Reference*	Removes leading and trailing blanks, and returns a concatenated character string.			X
"CATT Function" in *SAS Functions and CALL Routines: Reference*	Removes trailing blanks, and returns a concatenated character string.			X
"CATX Function" in *SAS Functions and CALL Routines: Reference*	Removes leading and trailing blanks, inserts delimiters, and returns a character string.			X
"CHAR Function" in *SAS Functions and CALL Routines: Reference*	Returns a single character from a specified position in a character string.			X
"CHOOSEC Function" in *SAS 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 Functions and CALL Routines: Reference*	Returns a numeric value that represents the results of choosing			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	from a list of arguments.			
"COALESCEC Function" in *SAS Functions and CALL Routines: Reference*	Returns the first nonmissing value from a list of numeric arguments.			X
"COLLATE Function" in *SAS Functions and CALL Routines: Reference*	Returns a character string in ASCII or EBCDIC collating sequence.	X		
"COMPARE Function" in *SAS 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 Functions and CALL Routines: Reference*	Removes multiple blanks from a character string.			X
"COMPGED Function" in *SAS Functions and CALL Routines: Reference*	Returns the generalized edit distance between two strings.	X		
"COMPLEV Function" in *SAS Functions and CALL Routines: Reference*	Returns the Levenshtein edit distance between two strings.	X		
"COMPRESS Function" in *SAS Functions and CALL Routines: Reference*	Returns a character string with specified characters removed from the original string.	X		
"COUNT Function" in *SAS Functions and CALL Routines: Reference*	Counts the number of times that a specified substring appears within a character string.	X		

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"COUNTC Function" in *SAS 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 Functions and CALL Routines: Reference*	Counts the number of words in a character string.	X		
"DEQUOTE Function" in *SAS 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	
"FIND Function" in *SAS Functions and CALL Routines: Reference*	Searches for a specific substring of characters within a character string.	X		
"FINDC Function" in *SAS Functions and CALL Routines: Reference*	Searches a string for any character in a list of characters.	X		
"FINDW Function" in *SAS 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 Functions and CALL Routines: Reference*	Returns the first character in a character string.			X
"GETLOCENV Function" (p. 611)	Returns the current locale/language environment.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"GETPXLANGUAGE Function" (p. 612)	Returns the current two-letter language code.			X
"GETPXLOCALE Function" (p. 613)	Returns the POSIX locale value for a SAS locale.			X
"GETPXREGION Function" (p. 614)	Returns the current two-letter region code.			X
"HTMLDECODE Function" in *SAS 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 Functions and CALL Routines: Reference*	Encodes characters using HTML character entity references, and returns the encoded string.			X
"IFC Function" in *SAS Functions and CALL Routines: Reference*	Returns a character value based on whether an expression is true, false, or missing.			X
"IFN Function" in *SAS Functions and CALL Routines: Reference*	Returns a numeric value based on whether an expression is true, false, or missing.			X
"INDEX Function" in *SAS Functions and CALL Routines: Reference*	Searches a character expression for a string of characters, and returns the position of the string's first	X		

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	character for the first occurrence of the string.			
"INDEXC Function" in *SAS 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 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. 619)	Returns the result of a comparison of character expressions.			X
"KCOMPRESS Function" (p. 621)	Removes specified characters from a character expression.			X
"KCOUNT Function" (p. 623)	Returns the number of double-byte characters in an expression.			X
"KCOUNTC Function" (p. 624)	counts individual characters in a character string.			X
"KCOUNTW Function" (p. 627)	Counts the number of words in a character string.			X
"KCOUNTX Function" (p. 630)	Counts the number of times that a specified substring			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	appears within a character string.			
"KCVT Function" (p. 633)	Converts data from one type of encoding data to another encoding data.			X
"KFIND Function" (p. 635)	Searches for a specific substring of characters within a character string.			X
"KFINDC Function" (p. 638)	Searches a string for any character in a list of characters.			X
"KFINDW Function" (p. 642)	Returns the character position of a word in a string or the number of the word in a string.			X
"KINDEX Function" (p. 647)	Searches a character expression for a string of characters.			X
"KINDEXB Function" (p. 649)	Searches a character expression for a string of characters.			X
"KINDEXCB Function" (p. 651)	Searches a character expression for specified characters.			X
"KINDEXC Function" (p. 652)	Searches a character expression for specified characters.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"KLEFT Function" (p. 655)	Left-aligns a character expression by removing unnecessary leading DBCS blanks and SO-SI.			X
"KLENGTH Function" (p. 657)	Returns the length of an argument.			X
"KLOWCASE Function" (p. 659)	Converts all letters in an argument to lowercase.			X
"KPROPCASE Function" (p. 664)	Converts Chinese, Japanese, Korean, Taiwanese (CJKT) characters.			X
"KPROPCHAR Function" (p. 667)	Converts special characters to normal characters.			X
"KPROPDATA Function" (p. 669)	Removes or converts unprintable characters.			X
"KREVERSE Function" (p. 674)	Reverses a character expression.			X
"KRIGHT Function" (p. 675)	Right-aligns a character expression by trimming trailing DBCS blanks and SO-SI.			X
"KSCAN Function" (p. 676)	Selects a specified word from a character expression.			X
"KSCANX Function" (p. 679)	Selects a specified word from a			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	character expression.			
"KSTRCAT Function" (p. 685)	Concatenates two or more character expressions.			X
"KSTRIP Function" (p. 686)	Removes leading and trailing blanks from a character string.			X
"KSUBSTR Function" (p. 688)	Extracts a substring from an argument.			X
"KSUBSTRB Function" (p. 690)	Extracts a substring from an argument according to the byte position of the substring in the argument.			X
"KSUBSTRN Function" (p. 692)	Returns a substring, allowing a result with a length of zero.			X
"KTRANSLATE Function"	Replaces specified characters in a character expression.			X
"KTRIM Function" (p. 696)	Removes trailing DBCS blanks and SO-SI from character expressions.			X
"KTRUNCATE Function" (p. 697)	Truncates a character string to a specified length in byte units without breaking multibyte characters.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"KUPCASE Function" (p. 699)	Converts all letters in an argument to uppercase.			X
"KUPDATE Function" (p. 701)	Inserts, deletes, and replaces character value contents.			X
"KUPDATEB Function" (p. 704)	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. 706)	Inserts, deletes, and replaces character value contents.			X
"KVERIFYB Function" (p. 710)	Returns the position of the first character that is unique to an expression.			X
"KVERIFY Function" (p. 709)	Returns the position of the first character that is unique to an expression.			X
"LEFT Function" in *SAS Functions and CALL Routines: Reference*	Left-aligns a character string.			X
"LENGTH Function" in *SAS 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 Functions and CALL Routines: Reference*	Returns the length of a character			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	string, including trailing blanks.			
"LENGTHM Function" in *SAS Functions and CALL Routines: Reference*	Returns the amount of memory (in bytes) that is allocated for a character string.			X
"LENGTHN Function" in *SAS Functions and CALL Routines: Reference*	Returns the length of a character string, excluding trailing blanks.			X
"LOWCASE Function" in *SAS Functions and CALL Routines: Reference*	Converts all letters in an argument to lowercase.			X
"MD5 Function" in *SAS Functions and CALL Routines: Reference*	Returns the result of the message digest of a specified string.			X
"MISSING Function" in *SAS Functions and CALL Routines: Reference*	Returns a numeric result that indicates whether the argument contains a missing value.			X
"MVALID Function" in *SAS Functions and CALL Routines: Reference*	Checks the validity of a character string for use as a SAS member name.			X
"NLDATE Function" (p. 712)	Converts the SAS date value to the date value of the specified locale by using the date format descriptors.			X
"NLDATM Function" (p. 715)	Converts the SAS datetime value to the time value of the specified locale by using the datetime-format descriptors.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"NLTIME Function" (p. 718)	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 Functions and CALL Routines: Reference*	Converts a character string that you specify to a SAS name literal.			X
"NOTALNUM Function" in *SAS 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 Functions and CALL Routines: Reference*	Searches a character string for a nonalphabetic character, and returns the first position at which the character is found.		X	
"NOTCNTRL Function" in *SAS 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 Functions and CALL Routines: Reference*	Searches a character string for any character that is not a digit, and returns the first		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	position at which that character is found. **Note:** This function is assigned an I18N Level 1 status unless a VARCHAR variable is used, or if the function is threaded or runs in DS2. If these exceptions occur, then this function is assigned an I18N Level 2 status.			
"NOTFIRST Function" in *SAS 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. **Note:** This function is assigned an I18N Level 1 status unless a VARCHAR variable is used, or if the function is threaded or runs in DS2. If these exceptions occur, then this function is assigned an I18N Level 2 status.		X	
"NOTGRAPH Function" in *SAS Functions and CALL Routines: Reference*	Searches a character string for a non-graphical character, and returns the first		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	position at which that character is found.			
"NOTLOWER Function" in *SAS 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	
"NOTNAME Function" in *SAS 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. **Note:** This function is assigned an I18N Level 1 status unless a VARCHAR variable is used, or if the function is threaded or runs in DS2. If these exceptions occur, then this function is assigned an I18N Level 2 status.		X	
"NOTPRINT Function" in *SAS Functions and CALL Routines: Reference*	Searches a character string for a nonprintable character, and returns the first position at which that character is found.	X		

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"NOTPUNCT Function" in *SAS 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 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 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 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. **Note:** This function is assigned an I18N Level 1 status unless a VARCHAR variable is used, or		X	

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	if the function is threaded or runs in DS2. If these exceptions occur, then this function is assigned an I18N Level 2 status.			
"NVALID Function" in *SAS Functions and CALL Routines: Reference*	Checks the validity of a character string for use as a SAS variable name.			X
"PROPCASE Function" in *SAS Functions and CALL Routines: Reference*	Converts all words in an argument to proper case.			X
"PRXCHANGE Function" in *SAS Functions and CALL Routines: Reference*	Performs a pattern-matching replacement.			X
"PRXMATCH Function" in *SAS 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 Functions and CALL Routines: Reference*	Returns the last bracket match for which there is a match in a pattern.			X
"PRXPARSE Function" in *SAS 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 Functions and CALL Routines: Reference*	Returns a character string that contains the value for a capture buffer.			X
"PUT Function" in *SAS Functions and CALL Routines: Reference*	Returns a value using a specified format.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"PUTC Function" in *SAS Functions and CALL Routines: Reference*	Enables you to specify a character format at run time.			X
"PUTN Function" in *SAS Functions and CALL Routines: Reference*	Enables you to specify a numeric format at run time.			X
"QUOTE Function" in *SAS Functions and CALL Routines: Reference*	Adds double quotation marks to a character value.			X
"RANK Function" in *SAS Functions and CALL Routines: Reference*	Returns the position of a character in the ASCII or EBCDIC collating sequence.	X		
"REPEAT Function" in *SAS Functions and CALL Routines: Reference*	Returns a character value that consists of the first argument repeated n+1 times.			X
"REVERSE Function" in *SAS Functions and CALL Routines: Reference*	Reverses a character string.	X		
"RIGHT Function" in *SAS Functions and CALL Routines: Reference*	Right-aligns a character expression.			X
"SASMSG Function" (p. 720)	Specifies a message from a data set. The returned message is based on the current locale and a specified key.			X
"SASMSGL Function" (p. 723)	Specifies a message from a data set. The message is based on a specified locale value and a specified key value.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"SCAN Function" in *SAS Functions and CALL Routines: Reference*	Returns the nth word from a character string.	X		
"SETLOCALE Function" (p. 725)	Specifies the locale keys for the current SAS locale.			X
"SHA256 Function" in *SAS Functions and CALL Routines: Reference*	Returns the result of the message digest of a specified string.			X
"SORTKEY Function" (p. 733)	Creates a linguistic sort key.			X
"SOUNDEX Function" in *SAS Functions and CALL Routines: Reference*	Encodes a string to facilitate searching.	X		
"SPEDIS Function" in *SAS Functions and CALL Routines: Reference*	Determines the likelihood of two words matching, expressed as the asymmetric spelling distance between the two words.	X		
"STRIP Function" in *SAS Functions and CALL Routines: Reference*	Returns a character string with all leading and trailing blanks removed.	X		
"SUBPAD Function" in *SAS 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 Functions and CALL Routines: Reference*	Replaces character value contents	X		
"SUBSTR (right of =) Function" in *SAS Functions and CALL Routines: Reference*	Extracts a substring from an argument.	X		

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"SUBSTRN Function" in *SAS Functions and CALL Routines: Reference*	Returns a substring, allowing a result with a length of zero.	X		
"TRANSLATE Function" in *SAS Functions and CALL Routines: Reference*	Replaces specific characters in a character string.	X		
"TRANSTRN Function" in *SAS Functions and CALL Routines: Reference*	Replaces or removes all occurrences of a substring in a character string.			X
"TRANTAB Function" on page 737	Transcodes data by using the specified translation table.	X		
"TRANWRD Function" in *SAS Functions and CALL Routines: Reference*	Replaces or removes all occurrences of a substring in a character string.			X
"TRIM Function" in *SAS 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 Functions and CALL Routines: Reference*	Removes trailing blanks from character expressions, and returns a string with a length of zero if the expression is missing.			X
"TZONEDSTNAME Function" (p. 744)	Returns a daylight savings time name.			X
"TZONEDSTOFF Function" (p. 745)	Returns the time zone offset value for the specified			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
	daylight savings time.			
"TZONEID Function" (p. 738)	Returns the current time zone ID.			X
"TZONENAME Function" (p. 740)	Returns the current standard or daylight savings time, time zone name.			X
"TZONEOFF Function" (p. 741)	Returns the user time zone offset.			X
"TZONES2U Function" (p. 743)	Converts a SAS date time value to a UTC date time value.			X
"TZONESTTNAME Function" (p. 746)	Returns a standard time zone name.			X
"TZONESTTOFF Function" (p. 747)	Returns the time zone offset value for the specified standard time.			X
"TZONEU2S Function" (p. 748)	Converts a UTC date time value to a SAS date time value.			X
"UNICODE Function" (p. 749)	Converts Unicode characters to the current SAS session encoding.			X
"UNICODEC Function" (p. 752)	Converts characters in the current SAS session encoding to Unicode characters.			X
"UNICODELEN Function" (p. 755)	Specifies the length of the character unit for the Unicode data.			X

Function	Description	I18N Level 0	I18N Level 1	I18N Level 2
"UNICODEWIDTH Function" (p. 756)	Specifies the length of a display unit for the Unicode data.			X
"UPCASE Function" in *SAS Functions and CALL Routines: Reference*	Converts all letters in an argument to uppercase.			X
"URLDECODE Function" in *SAS Functions and CALL Routines: Reference*	Returns a string that was decoded using the URL escape syntax.		X	
"URLENCODE Function" in *SAS Functions and CALL Routines: Reference*	Returns a string that was encoded using the URL escape syntax.		X	
"VERIFY Function" in *SAS 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 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

12

VARCHAR Support in Functions

VARCHAR Data Type in String Functions 583

VARCHAR Data Type in String Functions

SAS supports the VARCHAR data type for character-data processing in the DATA step and PROC step. The CAS engine is required to read or write VARCHAR data in order to preserve the variable's type. 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 platform 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 12.1 on page 584 shows results from the fixed-width variable using the data €123. Table 12.2 on page 585 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.

K functions should be used for fixed-width character data that might contain multi-byte characters:

Table 12.1 *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=;`
	`x=€123 x=E282AC`
	`xsub=¬123 xsub=82AC31323320`
	`xidx=4`

Statements	Results
`run;`	

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 12.2 *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

13

Dictionary of Functions for NLS

Functions by Category 588

Dictionary 595
ANORM420 Function 595
CALL KSCANX Routine 601
BASECHAR Function 606
ENCODCOMPAT Function 608
ENCODISVALID Function 610
GETLOCENV Function 611
GETPXLANGUAGE Function 612
GETPXLOCALE Function 613
GETPXREGION Function 614
KCHARLIST Function 616
KCOMPARE Function 619
KCOMPRESS Function 621
KCOUNT Function 623
KCOUNTC Function 624
KCOUNTW Function 627
KCOUNTX Function 630
KCVT Function 633
KFIND Function 635
KFINDC Function 638
KFINDW Function 642
KINDEX Function 647
KINDEXB Function 649
KINDEXCB Function 651
KINDEXC Function 652
KLEFT Function 655
KLENGTH Function 657
KLOWCASE Function 659
KNEXTCHAR Function 660
KOFFSET Function 662
KPROPCASE Function 664
KPROPCHAR Function 667
KPROPDATA Function 669
KREVERSE Function 674
KRIGHT Function 675

KSCAN Function ... 676
KSCANX Function ... 679
KSTRCAT Function ... 685
KSTRIP Function ... 686
KSUBSTR Function ... 688
KSUBSTRB Function ... 690
KSUBSTRN Function ... 692
KTRANSLATE Function ... 693
KTRIM Function ... 696
KTRUNCATE Function ... 697
KUPCASE Function ... 699
KUPDATE Function ... 701
KUPDATEB Function ... 704
KUPDATES Function ... 706
KVERIFY Function ... 709
KVERIFYB Function ... 710
NLDATE Function ... 712
NLDATM Function ... 715
NLTIME Function ... 718
SASMSG Function ... 720
SASMSGL Function ... 723
SETLOCALE Function ... 725
SORTKEY Function ... 733
TRANTAB Function ... 737
TZONEID Function ... 738
TZONENAME Function ... 740
TZONEOFF Function ... 741
TZONES2U Function ... 743
TZONEDSTNAME Function ... 744
TZONEDSTOFF Function ... 745
TZONESTTNAME Function ... 746
TZONESTTOFF Function ... 747
TZONEU2S Function ... 748
UNICODE Function ... 749
UNICODEC Function ... 752
UNICODELEN Function ... 755
UNICODEWIDTH Function ... 756
VARTRANSCODE Function ... 757
VTRANSCODE Function ... 759
VTRANSCODEX Function ... 760

Functions by Category

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.

The following categories relate to NLS issues:

Table 13.1 *Categories of NLS Functions*

Category	Description
CAS	Functions that run on the CAS server. See CAS for a list of functions.
Character	processes character data See Character for a list of functions.
Character Manipulation	processes character data in any encoding and processes multibyte characters. See Character Manipulation for a list of functions.
DBCS	processes double-byte character set. See DBCS for a list of functions.
Date and Time	processes data and time data. See Date and Time for a list of functions.
Locale	processes data based on the specified locale. See Locale for a list of functions.
Variable Information	processes variable information. See Variable Information for a list of functions.

Category	Language Elements	Description
CAS	CALL KSCANX Routine (p. 601)	Returns the position and length of the nth word from a character string.
	BASECHAR Function (p. 606)	Converts characters to base characters.
	KCHARLIST Function (p. 616)	Returns the character list according to the modifier.
	KCOMPARE Function (p. 619)	Returns the result of a comparison of character expressions.
	KCOMPRESS Function (p. 621)	Removes specified characters from a character expression.
	KCOUNT Function (p. 623)	Returns the number of double-byte characters in an expression.

Category	Language Elements	Description
	KCOUNTC Function (p. 624)	counts individual characters in a character string.
	KCOUNTW Function (p. 627)	Counts the number of words in a character string.
	KCOUNTX Function (p. 630)	Counts the number of times that a specified substring appears within a character string.
	KCVT Function (p. 633)	Converts data from one type of encoding data to another type of encoding data.
	KFIND Function (p. 635)	Searches for a specific substring of characters within a character string.
	KFINDC Function (p. 638)	Searches a string for any character in a list of characters.
	KFINDW Function (p. 642)	Returns the character position of a word in a string or the number of the word in a string.
	KINDEX Function (p. 647)	Searches a character expression for a string of characters.
	KINDEXC Function (p. 652)	Searches a character expression for specified characters and returns character-based values.
	KLEFT Function (p. 655)	Left-aligns a character expression by removing unnecessary leading DBCS blanks and SO/SI.
	KLENGTH Function (p. 657)	Returns the length, in characters, of an argument.
	KLOWCASE Function (p. 659)	Converts the uppercase alphabetic letters to lowercase letters.
	KNEXTCHAR Function (p. 660)	Returns a character from a string at a specified byte offset.
	KOFFSET Function (p. 662)	Returns the byte offset from the character position.
	KREVERSE Function (p. 674)	Reverses a character expression.
	KRIGHT Function (p. 675)	Right-aligns a character expression by trimming trailing DBCS blanks and SO/SI.
	KSCAN Function (p. 676)	Selects a specified word from a character expression.
	KSCANX Function (p. 679)	Selects a specified word from a character expression using a modifier to process the function's action.
	KSTRCAT Function (p. 685)	Concatenates two or more character expressions.
	KSTRIP Function (p. 686)	Removes leading and trailing blanks from a character string.

Category	Language Elements	Description
	KSUBSTR Function (p. 688)	Extracts a substring from an argument.
	KTRANSLATE Function (p. 693)	Replaces specified characters in a character expression.
	KTRIM Function (p. 696)	Removes trailing DBCS blanks and SO/SI from character expressions.
	KUPCASE Function (p. 699)	Converts the lowercase alphabetic letters to uppercase letters.
	KUPDATE Function (p. 701)	Replaces character value contents.
	KUPDATES Function (p. 706)	Replaces character value contents.
	KVERIFY Function (p. 709)	Returns the position of the first character (character-based value) that is unique to an expression.
	UNICODE Function (p. 749)	Converts Unicode characters to the current SAS session encoding.
	UNICODEC Function (p. 752)	Converts characters in the current SAS session encoding to Unicode characters.
	UNICODELEN Function (p. 755)	Specifies the length of the character unit for the Unicode data.
	UNICODEWIDTH Function (p. 756)	Specifies the length of a display unit for the Unicode data.
	VTRANSCODE Function (p. 759)	Returns a value that indicates whether transcoding is enabled for the specified character variable.
	VTRANSCODEX Function (p. 760)	Returns a value that indicates whether transcoding is enabled for the specified argument.
Character	ANORM420 Function (p. 595)	Returns a normalized string from an input string encoded in EBCDIC420.
	BASECHAR Function (p. 606)	Converts characters to base characters.
	KCVT Function (p. 633)	Converts data from one type of encoding data to another type of encoding data.
	KSTRIP Function (p. 686)	Removes leading and trailing blanks from a character string.
	TRANTAB Function (p. 737)	Transcodes data by using the specified translation table.
	UNICODE Function (p. 749)	Converts Unicode characters to the current SAS session encoding.

Category	Language Elements	Description
	UNICODEC Function (p. 752)	Converts characters in the current SAS session encoding to Unicode characters.
	UNICODELEN Function (p. 755)	Specifies the length of the character unit for the Unicode data.
	UNICODEWIDTH Function (p. 756)	Specifies the length of a display unit for the Unicode data.
Character Manipulation	KCHARLIST Function (p. 616)	Returns the character list according to the modifier.
	KCOMPARE Function (p. 619)	Returns the result of a comparison of character expressions.
	KCOMPRESS Function (p. 621)	Removes specified characters from a character expression.
	KCOUNT Function (p. 623)	Returns the number of double-byte characters in an expression.
	KCOUNTC Function (p. 624)	counts individual characters in a character string.
	KCOUNTW Function (p. 627)	Counts the number of words in a character string.
	KCOUNTX Function (p. 630)	Counts the number of times that a specified substring appears within a character string.
	KFIND Function (p. 635)	Searches for a specific substring of characters within a character string.
	KFINDC Function (p. 638)	Searches a string for any character in a list of characters.
	KFINDW Function (p. 642)	Returns the character position of a word in a string or the number of the word in a string.
	KINDEX Function (p. 647)	Searches a character expression for a string of characters.
	KINDEXB Function (p. 649)	Searches a character expression for specified characters and returns byte-based values.
	KINDEXCB Function (p. 651)	Searches a character expression for specified characters and returns byte-based values.
	KINDEXC Function (p. 652)	Searches a character expression for specified characters and returns character-based values.
	KLENGTH Function (p. 657)	Returns the length, in characters, of an argument.
	KLOWCASE Function (p. 659)	Converts the uppercase alphabetic letters to lowercase letters.

Category	Language Elements	Description
	KNEXTCHAR Function (p. 660)	Returns a character from a string at a specified byte offset.
	KOFFSET Function (p. 662)	Returns the byte offset from the character position.
	KPROPCASE Function (p. 664)	Converts Chinese, Japanese, Korean, Taiwanese (CJKT) characters.
	KPROPCHAR Function (p. 667)	Converts Unicode to the corresponding characters enclosed in parenthesis.
	KPROPDATA Function (p. 669)	Converts text between encodings and handles characters that cannot be represented in the target encoding. KPROPDATA is designed for data cleansing and encoding compatibility in multilingual workflows.
	KREVERSE Function (p. 674)	Reverses a character expression.
	KRIGHT Function (p. 675)	Right-aligns a character expression by trimming trailing DBCS blanks and SO/SI.
	KSCAN Function (p. 676)	Selects a specified word from a character expression.
	KSCANX Function (p. 679)	Selects a specified word from a character expression using a modifier to process the function's action.
	KSTRCAT Function (p. 685)	Concatenates two or more character expressions.
	KSUBSTR Function (p. 688)	Extracts a substring from an argument.
	KSUBSTRB Function (p. 690)	Extracts a substring from an argument according to the byte position of the substring in the argument.
	KSUBSTRN Function (p. 692)	Returns a substring, allowing a result with a length of zero.
	KTRANSLATE Function (p. 693)	Replaces specified characters in a character expression.
	KTRUNCATE Function (p. 697)	Truncates a string to a specified length in byte unit without breaking multibyte characters.
	KUPCASE Function (p. 699)	Converts the lowercase alphabetic letters to uppercase letters.
	KUPDATE Function (p. 701)	Replaces character value contents.
	KUPDATEB Function (p. 704)	Replaces the contents of the character value according to the byte position of the character value in the argument.
	KUPDATES Function (p. 706)	Replaces character value contents.

Category	Language Elements	Description
	KVERIFY Function (p. 709)	Returns the position of the first character (character-based value) that is unique to an expression.
	KVERIFYB Function (p. 710)	Returns the position of the first character (byte-based value) that is unique to an expression.
Date and Time	NLDATE Function (p. 712)	Converts the SAS date value to the date value of the specified locale by using the date format descriptors.
	NLDATM Function (p. 715)	Converts the SAS datetime value to the time value of the specified locale by using the datetime-format descriptors.
	NLTIME Function (p. 718)	Converts the SAS time or the datetime value to the time value of the specified locale by using the NLTIME descriptors.
	TZONEID Function (p. 738)	Returns the current time zone ID.
	TZONENAME Function (p. 740)	Returns the current standard or daylight savings time, time zone name.
	TZONEOFF Function (p. 741)	Returns the user time zone offset.
	TZONES2U Function (p. 743)	Converts a SAS date time value to a UTC date time value.
	TZONEDSTNAME Function (p. 744)	Returns a daylight savings time name.
	TZONEDSTOFF Function (p. 745)	Returns the time zone offset value for the specified daylight savings time.
	TZONESTTNAME Function (p. 746)	Returns a standard time zone name.
	TZONESTTOFF Function (p. 747)	Returns the time zone offset value for the specified standard time.
	TZONEU2S Function (p. 748)	Converts a UTC date time value to a SAS date time value.
DBCS	CALL KSCANX Routine (p. 601)	Returns the position and length of the nth word from a character string.
	KLEFT Function (p. 655)	Left-aligns a character expression by removing unnecessary leading DBCS blanks and SO/SI.
	KTRIM Function (p. 696)	Removes trailing DBCS blanks and SO/SI from character expressions.
Encoding	ENCODCOMPAT Function (p. 608)	Verifies the transcoding compatibility between two encodings.

Category	Language Elements	Description
	ENCODISVALID Function (p. 610)	Verifies a valid encoding name.
Locale	GETLOCENV Function (p. 611)	Returns the current locale/language environment.
	GETPXLANGUAGE Function (p. 612)	Returns the current two-letter language code.
	GETPXLOCALE Function (p. 613)	Returns the POSIX locale value for a SAS locale.
	GETPXREGION Function (p. 614)	Returns the current two-letter region code.
	SASMSG Function (p. 720)	Specifies a message from a data set. The returned message is based on the current locale and a specified key.
	SASMSGL Function (p. 723)	Specifies a message from a data set. The message is based on a specified locale value and a specified key value.
	SETLOCALE Function (p. 725)	Specifies the locale keys for the current SAS locale.
	SORTKEY Function (p. 733)	Creates a linguistic sort key.
Variable Information	VARTRANSCODE Function (p. 757)	Returns the transcode attribute of a SAS data set variable.
	VTRANSCODE Function (p. 759)	Returns a value that indicates whether transcoding is enabled for the specified character variable.
	VTRANSCODEX Function (p. 760)	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 555 .

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 13.2 *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

Character	Unicode Value	From	To	Description
ﷵ	(U+FEF7)	0xB4	0xB1 + 0x49	Arabic Ligature Lam with Aleph with Hamza above - Isolated form
ﷶ	(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
ﻼ	(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 13.3 *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 13.4 *Modifier p: Shaped Characters Mapping*

From		To	
0x43	U+FE7D	0x42	U+0651
0x48	FE82	0x47	U+0622
0x51	U+FE84	0x49	U+0623
0x57	U+FE8E	0x56	U+0627
0x59	U+FE91	0x58	U+0628
0x64	U+FE97	0x63	U+062A

From		To	
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
0xCD	U+FEEC	0xBF	U+0647
0xDB	U+FEF0	0xDA	U+061F

From		To	
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 13.5 *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

```
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;
```

These statements produce these results:

```
s1=D8A8D987D8A7D985D8A7
s2=D8A8D987D8A7D985D8A7
s3=58BF56BB567740404040
```

CALL KSCANX Routine

Returns the position and length of the *n*th word from a character string.

Categories: CAS
DBCS

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 for SAS String Functions" on page 555.

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 SAS log.

Syntax

CALL KSCANX(<*string*>, *count*, *position*, *length*,<*character-list*>,<*modifier(s)*>);

Required Arguments

string
specifies a character constant, variable, or expression.

count
specifies 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 CALL 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, CALL KSCANX counts words from left to right in the character string.
- If *count* is negative, CALL KSCANX 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 0. If *count* is 0 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 0. If *count* is 0 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 CALL KSCANX. 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 0. 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.

In the CALL KSCANX routine, a 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 0 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 0 unless you specify the M modifier.

Using Default Delimiters in an ASCII Environment

If your computer uses ASCII characters, then the default delimiters are: blank ! $ % & ( ) * + , - . / ; < ^. In ASCII environments that do not contain the ^ character, the CALL KSCANX routine uses the ~ character instead.

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 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 0 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 0.

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 0.

Finding a Designated Word as a Character String

To find a designated word as a character string after calling the CALL KSCANX routine, use the SUBSTRN function with the *string*, *position*, and *length* arguments:

```
ksubstr(string, position, length);
```

Using Null Arguments

The CALL KSCANX routine allows character arguments to be null. Null arguments are treated as character strings with a length of 0. Numeric arguments cannot be null.

Example

This example uses Chinese characters:

```
data test;
   length word word_r $30;
   string='这个是字符函数的测试。';
   do until(position <= 0);
      count + 1;
      call kscanx(string, count, position, length, '是的。', 'oq');
      word=ksubstrn(string, position, length);
      if not position then leave;
      output;
   end;
run;
proc print data=test noobs;
   var count position length word;
run;
```

SAS writes the following output to the log:

The SAS System

count	position	length	word
1	1	2	这个
2	4	4	字符函数
3	9	2	测试

BASECHAR Function

Converts characters to base characters.

Categories: CAS
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 555.

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 SAS session encoding for this example is wlatin1.

Note: The BASECHAR function uses Unicode character decomposition mappings, so the sharp s, ß, does not have a Unicode decomposition. http://www.unicode.org/reports/tr44/#Character_Decomposition_Mappings

```
data cities;
   length name$24 name_ascii$24;
   input name;
   name_ascii = basechar(name);
   put name_ascii=;
   datalines;
Mühlenfließ
LaUña
ZielonaGóra
;
```

These statements produce these results:

```
name_ascii=Muhlenfließ
name_ascii=LaUna
name_ascii=ZielonaGora
```

The session encoding for this example is wlatin1 and the ESC option is specified.

```
data cities;
   length name$24 name_ascii$80;
   input name;
   name_ascii = basechar(name, "ESC");
   put name_ascii=;
   datalines;
Mühlenfließ
LaUña
ZielonaGóra
;
```

These statements produce these results:

```
name_ascii=Muhlenflie\u00df
name_ascii=LaUna
name_ascii=ZielonaGora
```

The session encoding for this example is wlatin1 and the NCR option is specified.

```
data cities;
   length name$24 name_ascii$80;
   input name;
   name_ascii = basechar(name, "NCR");
   put name_ascii=;
   datalines;
Mühlenfließ
LaUña
ZielonaGóra
```

```
;
```

These statements produce these results:

```
name_ascii=Muhlenfließ
name_ascii=LaUna
name_ascii=ZielonaGora
```

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 session encoding for these examples is UTF-8.

```
data encoding;
   isCompat= EncodCompat('xyz');
   put isCompat=;
run;
```

These statements produce this result:

```
isCompat=-1
```

```
data encoding;
   isCompat= EncodCompat ('ascii');
   put isCompat=;
run;
```

These statements produce this result:

```
isCompat=1
```

```
data encoding;
isCompat= EncodCompat
("ebcdic1149","open_ed-1149");
put isCompat;
run;
```

These statements produce this result:

```
isCompat=2
```

```
data encoding;
   isCompat= EncodCompat ('cp1251','ebcdic1149');
   put isCompat=;
run;
```

These statements produce this result:

```
isCompat=0;
```

This code compares the Windows encoding cp1251, also known as WLATIN2, with open_ed-1149, Iceland OpenEdition EBCDIC.

```
data encoding;
   isCompat= EncodCompat ('ebcdic1149','open_ed-1149');
   put isCompat=;
run;
```

These statements produce this result:

```
isCompat=2;
```

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

```
data valid;
   isValid1=EncodIsValid('xyz');
   isValid2=EncodIsValid('wlt2');
   isValid3=EncodIsValid('wlatin2');
   isValid4=EncodIsValid('cp1250');
   put isValid1=;
   put isValid2=;
   put isValid3=;
   put isValid4;
run;
```

These statements produce these results:

```
isValid1=0
isValid2=1
isValid3=2
isValid4=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 555.

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

```
Data test;
option locale=french_france;
   environ=getlocenv();
   put environ=;
```

```
run;
```

These statements produce this result:

```
environ=MBCS
```

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 555.

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();	de

put lang;	

option locale=en_US;
lang=getpxLanguage();
put lang;
	en

See Also

System Options:

- “LOCALE System Option” on page 987

Functions:

- “GETPXREGION Function” on page 614
- “GETPXLOCALE Function” on page 613

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 555.

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 987

Functions:

- "GETPXLANGUAGE Function" on page 612
- "GETPXREGION Function" on page 614

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 555.

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 this example, the LOCALE system option is set to French_France.

```
data test;
option locale=french_france;
   region=getpxRegion();
   put region;
run;
```

These statements produce this result:

```
FR
```

In this example, the LOCALE system option is set to German.

```
data test;
option locale=german;
   region=getpxRegion();
   put region;
run;
```

These statements produce this result:

```
DE
```

In this example, the LOCALE system option is set to English_United States.

```
data test;
option locale=en_US;
   region=getpxRegion();
   put region;
run;
```

These statements produce this result:

```
US
```

See Also

System Options:

- “LOCALE System Option” on page 987

Functions:

- “GETPXLOCALE Function” on page 613
- “GETPXLANGUAGE Function” on page 612

KCHARLIST Function

Returns the character list according to the modifier.

Categories: CAS
Character Manipulation

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 555.

Note: A returned character list or string is specified to a variable:
str=KCHARLIST(*modifier*)

Syntax

KCHARLIST(*modifier*)

Required Argument

modifier
specifies a character constant, variable, or expression in which each non-blank character modifies the action of the KCHARLIST function. Use the following characters as modifiers:

A or a	returns alphabetic characters.
C or c	returns control characters.
D or d	returns digital characters.
For f	returns the first characters in SAS names.

G or g	returns graphic characters.
L or l	returns lowercase characters.
P or p	returns punctuation characters.
S or s	returns all space characters.
U or u	returns uppercase characters.
X or x	returns hexadecimal characters.

Details

KCHARLIST returns encoding-specific characters that can be used with other K functions. Using KCHARLIST to determine the delimiters helps to avoid the ambiguity of the delimiters in SAS string functions.

KCHARLIST can return DBCS space and punctuation character lists.

Examples

Example 1

This example uses KCHARLIST to return alphabetic characters using the a modifier.

```
data _null_;
   str=kcharlist('a');
   put str=;
run;
```

The preceding statements produce these results:

```
str=ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

Example 2

This example demonstrates the functionality of using delimiters.

```
data _null_;
   str = 'Circumference = 2 × π × radius';
   result1 = kscan(str, 5);                  /*1*/
   put result1=;
   result2 = kscan(str, 2, kcharlist('p'));  /*2*/
   put result2=;
run;
```

1 KSCAN selects the fifth word using the delimiter, SBCS space, so 'π' is selected.

2 KCHARLIST selects 2 × π × radius because the puncutation delimiter, p, is specified. The = character is the first punctuation delimiter in the string.

The preceding statements produce these results:

```
result1=π
result2=2 × π × radius
```

Example 3

This example uses KCHARLIST and KSCAN to process a string with a DBCS space and an SBCS space.

```
data _null_;
   string = "北京　赛仕 软件";                          /*1*/
   result1 = KSCAN(string, 2);                          /*2*/
   put result1=;                                        /*3*/
   result2 = KSCAN(string, 2, KCHARLIST('S'));          /*4*/
   put result2=;                                        /*5*/
run;
```

1 Assign a value with a DBCS space and an SBCS space to the variable *string*. The input string contain two spaces. The first space is a DBCS space, and the second space is an SBCS space.

2 KSCAN selects the second word, but KSCAN ignores DBCS spaces, so 软件 is selected.

3 The value 软件 is written to the output.

4 KCHARLIST processes the DBCS spaces and passes the space list to KSCAN as delimiters. The correct value 赛仕 is selected.

5 The value 赛仕 is written to the output.

```
result1=软件
result2=赛仕
```

See Also

Functions

- "SCAN Function" in *SAS Functions and CALL Routines: Reference*
- "KSCAN Function" on page 676

KCOMPARE Function

Returns the result of a comparison of character expressions.

Categories: CAS
Character Manipulation

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 555.

Tip: Non-DBCS equivalent function is the "COMPARE Function" in *SAS Functions and CALL Routines: Reference*.

Syntax

KCOMPARE(*source*,<*pos*, *count*, > *findstr*,<*modifiers*>)

Required Arguments

source
specifies the character expression to be compared.

findstr
specifies the character expression to compare to *source*.

Optional Arguments

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.

modifiers

i or I	ignores the case in *source* and *findstr*.
l or L	removes leading blanks in *source* and *findstr* before comparing the values.
n or N	removes quotation marks from any argument that is a literal name and ignores the case of *source* and *findstr*. A literal name is a name token that is expressed as a string within quotation marks, followed by the uppercase or lowercase letter n. Literal names enable you to use special characters (including blanks), which are not allowed in SAS data sets or variable names. For KCOMPARE to recognize a string as a literal name, the first character must be a quotation mark.

: (colon) truncates the longer string in *source* or *findstr* to the length of the shorter string, whichever is greater. If ':' modifier is not specified, then use the longer string length, and the shorter one will be padded with blanks at the end of the string.

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.

Details

The order in which the modifiers appear in the COMPARE function is relevant.

- LN first removes leading blanks from each string, and then it removes quotation marks from name literals.
- NL first removes quotation marks from name literals, and then it removes leading blanks from each string.

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 Chinese characters.

```
options pageno=1 nodate ls=80 ps=60;
data test;
  rc1=kcompare('ABCDEF','ABC');
  rc2=kcompare('déclaré','declared');
  rc3=kcompare('déclaré',2,'declared');
  rc4=kcompare('漢字のテスト','漢字');
  rc5=kcompare('漢字のテスト',10,'テスト');
  rc6=kcompare('漢字のテスト',1,9,'漢字');
run;

proc print data=test; run;
```

The preceding statements produce these results:

```
                    The SAS System

 Obs    rc1    rc2    rc3    rc4    rc5    rc6

  1      4      2      1      3      0      3
```

KCOMPRESS Function

Removes specified characters from a character expression.

Categories: CAS
Character Manipulation

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 555.

Tip: Non-DBCS equivalent function is COMPRESS in *SAS Functions and CALL Routines: Reference*.

Syntax

KCOMPRESS(*source*,<*characters-to-remove*>, <*modifier(s)*>)

Required Argument

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.

Optional Arguments

characters-to-remove
specifies the character or characters that KCOMPRESS removes from the character. If characters-to-remove is omitted, KCOMPRESS removes all blank expressions.

Tip Enclose a literal string of characters in quotation marks.

modifier
specifies a character constant, variable, or expression in which each non-blank character modifies the action of the KCOMPRESS function. Full-width alphabetic and numeric characters are supported. Blanks are ignored. The following characters can be used as modifiers:

a or A adds alphabetic characters to the list of characters. English characters and full-width alphabetic and numeric characters are supported.

c or C adds control characters to the list of characters.

d or D adds numeric characters (or digits) to the list of characters. English characters and full-width alphabetic and numeric characters are supported.

f or F	adds the underscore character and English characters to the list of characters. Only English characters are supported.
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 and full-width characters to be kept or removed.
k or K	keeps the characters in the list instead of removing them.
l or L	adds lowercase characters to the list of characters. English characters and full-width alphabetic and numeric characters are supported.
n or N	adds numeric characters, the underscore character, and English characters to the list of characters. Only English characters are supported.
o or O	uses the O modifier in the DATA step (excluding WHERE clauses) or the SQL procedure to make KCOMPRESS run much faster when it is called in a loop, where the second and third arguments do not change.
p or P	adds punctuation marks to the list of characters.
s or S	removes all Unicode space characters in a Unicode SAS session.
t or T	trims trailing blanks from the first and second arguments. Only English characters are supported.
u or U	adds uppercase characters to the list of characters. English characters and full-width alphabetic and numeric characters are supported.
w or W	adds printable characters to the list of characters.
x or X	adds hexadecimal characters to the list of characters.

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 an expression.

Example

The following example uses Japanese characters.

```
data _null_;
  result1=kcompress('Renée','é');
  result2=kcompress('Renée!','é','a');
  result3=kcompress('漢字のテスト','の');
  put result1=/result2=/result3=;
run;
```

The preceding statements produce these results:

```
result1=Rene
result2=!
result3=漢字テスト
```

See Also

Functions:

- “KLEFT Function” on page 655
- “KTRIM Function” on page 696

KCOUNT Function

Returns the number of double-byte characters in an expression.

Categories: CAS
Character Manipulation

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 555.

Syntax

KCOUNT(*source*)

Required Argument

source
specifies the character expression to count.

Details

See “Internationalization Compatibility for SAS String Functions” on page 555 for restrictions and more information.

Example

These examples use UTF-8 encoding.

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. The following examples use an emoji.

```
data _null_;
  text=' abc ';         /*1*/
  result=kcount(text);  /*2*/
  put result=;
run;
```

1 Assign letters and an emoji to *text*.

2 KCOUNT counts the DBCS/MBCS characters and assigns that value to *result*.

The preceding statements produce these results:

```
result=1
```

KCOUNTC Function

counts individual characters in a character string.

Categories: CAS
Character Manipulation

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8).

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.

> **TIP** 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

modifer(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, 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.
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 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 KCOUNTC routine 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 (writable) 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.

Examples

Example 1

These examples use UTF=8 encoding.

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses emojis:

```
data _null_;
  text=' abc ';
  result=kcountc(text,'a '); /*1*/
  put result = ;
run;
```

1 KCOUNTC counts the occurrences of *a* and .

```
result=2
```

Example 2

This example uses Chinese characters:

```
data _null_ ;
  text='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。';
  result=kcountc(text,'软件'); /*1*/
  put result = ;
run;
```

1 KCOUNTC counts the occurrences of 软 and 件.

The preceding statements produce these results:

```
result=4
```

KCOUNTW Function

Counts the number of words in a character string.

Categories: CAS
Character Manipulation

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 for SAS String Functions" on page 555.

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, 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 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 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 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.
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.

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 Environments

If you use the KCOUNTW function with only two arguments, and your computer uses ASCII characters, the following default delimiters are used:

blank ! $ % & ( ) * + , - . / ; < ^ |

In ASCII environments that do not contain the ^ character, the SCAN function uses the ~ character instead.

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.

Examples

Example 1

These examples use UTF-8 encoding.

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses emojis:

```
data _null_;
  text='Renée Dupont? ';
  result1=kcountw(text);         /*1*/
  result2=kcountw(text,'','p'); /*2*/
  put result1=/result2=;
run;
```

1 KCOUNTW counts the words in *text*. The SBCS space is a delimter.

2 KCOUNTW counts the words in *text*. The SBCS space and ? are delimiters.

The preceding statements produce these results:

```
result1=2
result2=3
```

Example 2

These examples use UTF-8 encoding.

This example uses Chinese characters.

```
data _null_;
  text='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。';
  result=kcountw(text, ' ', 'p');  /*1*/
  put result = ;
run;
```

1 KCOUNTW counts the words in *text*. Punctuation marks are delimiters.

The preceding statements produce these results:

```
result=2
```

KCOUNTX Function

Counts the number of times that a specified substring appears within a character string.

Categories: CAS
Character Manipulation

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 for SAS String Functions" on page 555.

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 *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

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, kcountx('boobooboo', 'booboo') might return either a 1 or a 2.

Examples

Example 1

These examples use UTF-8 encoding.

This example uses accented characters.

```
data _null_;
  text='Renée Dupont';
  result=kcountx(text,'ée'); /*1*/
  put result=;
run;
```

1 KCOUNTX counts the occurrences of the accented character ée.

The preceding statements produce these results:

```
result=1
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text='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。';
  result=kcountx(text,'软件'); /*1*/
  put result = ;
run;
```

1 KCOUNTX counts the occurrences of 软件.

The preceding statements produce these results:

```
result=2
```

KCVT Function

Converts data from one type of encoding data to another type of encoding data.

Categories: CAS
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 555.

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 1127.

ASCIIANY and EBCDICANY 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 1127.

ASCIIANY and EBCDICANY 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 555 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.

In the SQL procedure, or in a WHERE clause in any procedure, the maximum length of a word that is returned by the KCVT function is 200 characters.

Example

These examples use UTF-8 encoding.

```
data _null_;
   src = '436972636F6E66E972656E636520'x; /*1*/
   text = kcvt(src, 'latin1','utf-8');    /*2*/
   put text= ;
run;
```

1 Assign the latin1 value for Circonférence to the variable src.

2 Convert the latin1 value for Circonférence to UTF-8 encoding.

The preceding statements produce these results:

```
text=Circonférence
```

See Also

System options:

- "DBCS System Option" on page 976

Procedure:

- Chapter 18, "DBCSTAB Procedure," on page 1055

KFIND Function

Searches for a specific substring of characters within a character string.

Categories: CAS
Character Manipulation

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 for SAS String Functions" on page 555.

Syntax

KFIND(*string*,*substring*,<*start-position*>,<*modifier(s)*>)

KFIND(*string*,*substring*,<*modifier(s)*>,<*start-position*>)

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:

i or I
ignores character case during the search. If this modifier is not specified, KFIND searches only for character substrings with the same case as the characters in *substring*.

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 KTRIM 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

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.

Value of *startpos*	Action
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.

Examples

Example 1

These examples use UTF-8 encoding.

```
data _null_;
  text='Renée Dupont';
  pos_of=kfind(text,'Dupont'); /*1*/
  put pos_of=;
run;
```

1 KFIND specifies the position of the first character of Dupont.

The preceding statements produce these results:

```
pos_of=7
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text='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(text,'软件'); /*1*/
  put pos_of=;
run;
```

1 KFIND specifies the position of the first character of 软件.

The preceding statements produce these results:

```
pos_of=10
```

KFINDC Function

Searches a string for any character in a list of characters.

Categories: CAS
Character Manipulation

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 for SAS String Functions" on page 555.

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	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, 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	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. 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 KFINDC routine 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 (writable) 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

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.

Examples

Example 1

These examples use UTF-8 encoding.

```
data _null_;
  text='Renée Dupont';
  pos_of=kfindc(text,'éut'); /*1*/
  put pos_of=;
run;
```

1 Specifies the position of the first character found in the string éut.

```
pos_of=4
```

Example 2

```
data _null_;
  text='Renée, Dupont';
  pos_of=kfindc(text,'ut','p'); /*1*/
  put pos_of=;
run;
```

1 Specifies the first position of the string ut or punctuation marks.

The preceding statements produce these results:

```
pos_of=6
```

Example 3

This example uses Chinese characters.

```
data _null_;
  text='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(text,'软件的'); /*1*/
  put pos_of=;
run;
```

1 Specifies the first character position of the string 软件的.

The preceding statements produce these results:

```
pos_of=9
```

KFINDW Function

Returns the character position of a word in a string or the number of the word in a string.

Categories: CAS

Character Manipulation

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 for SAS String Functions" on page 555.

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 these characters 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.
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, 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. 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.
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 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 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*, *word*, and *character-list* arguments.
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.

Details

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.

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 *start-position*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 Environments

If you use the KFINDW function with only two arguments, and your computer uses ASCII characters, the following default delimiters are used:

blank ! $ % & ( ) * + , - . / ; < ^ |

In ASCII environments that do not contain the ^ character, the KFINDW function uses the ~ character instead.

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.

Examples

Example 1

These examples use UTF-8 encoding.

```
data _null_;
  text='Renée Dupont';
  pos_of=kfindw(text,'Dupont'); /*1*/
  put pos_of=;
run;
```

1 KFINDW specifies the position of the text, Dupont, using the SBCS space as a delimiter.

The preceding statements produce these results:

```
pos_of=7
```

Example 2

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses an emoji.

```
data _null_;
  text='Renée Dupont? ';
  pos_of=kfindw(text,' ','','p'); /*1*/
  put pos_of=;
run;
```

1 KFINDW specifies the position of the emoji using SBCS space and punctuation marks as delimiters.

The preceding statements produce these results:

```
pos_of=14
```

Example 3

This example uses Chinese characters.

```
data _null_;
  text='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(text, 'sas', '', 'sp'); /*1*/
  put pos_of = ;
run;
```

1 KFINDW specfies the position of sas, using spaces and punctuation marks as delimiters.

The preceding statements produce these results:

```
pos_of=17
```

KINDEX Function

Searches a character expression for a string of characters.

Categories: CAS
Character Manipulation

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 555.

Tip: Non-DBCS equivalent function is INDEX in *SAS 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 555 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*, KINDEX returns a value of 0. If there are multiple occurrences of the string, KINDEX returns only the position of the first occurrence.

Examples

Example 1

These examples use UTF-8 encoding.

```
data _null_;
  text=' Renée Dupont';
  result1=kindex(text,'ée');   /*1*/
  result2=kindex(text,'éec');  /*2*/
  put result1=/result2=;
run;
```

1 Specifies the position of the character ée.

2 Specifies the position of the character éec.

The preceding statements produce these results:

```
result1=5
result2=0
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text='漢字文字列の検索';
  result1=kindex(text,'検索');  /*1*/
  text2='探す';
  result2=kindex(text,text2);   /*2*/
  put result1=/result2=;
run;
```

1 KINDEX specfies the position of character 検索.

2 KINDEX specfies the position of character 探す.

The preceding statements produce these results:

```
result1=7
result2=0
```

See Also

Functions:

- “KINDEXC Function” on page 652

KINDEXB Function

Searches a character expression for specified characters and returns byte-based values.

Category: Character Manipulation

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 555.

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. When processing an SBCS string (for example, wlatin1 encoding), KINDEXB and KINDEX return identical results. But in a DBCS session (for example, EUC-JP or SHIFT-JIS encoding), most CJK characters are 2 bytes wide, causing these functions to 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.

Examples

Example 1

These examples use UTF-8 encoding.

```
data _null_;
  text=' Renée Dupont';
  result1=kindexb(text,'ée');   /*1*/
  result2=kindexb(text,'éec');  /*2*/
  put result1=/result2=;
run;
```

1 KINDEXB specifies the byte position of ée.

2 KINDEXB specifies the byte position of éec.

The preceding statements produce these results:

```
result1=8
result2=0
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text='漢字文字列の検索';
  result1=kindexb(text,'検索');  /*1*/
  text2='探す';
  result2=kindexb(text,text2);   /*2*/
  put result1=/result2=;
run;
```

1 KINDEXB specifies the byte position of 検索.

2 KINDEXB specifies the byte position of 探す.

The preceding statements produce these results:

```
result1=19
result2=0
```

KINDEXCB Function

Searches a character expression for specified characters and returns byte-based values.

Category: Character Manipulation

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 555.

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. When processing SBCS strings (for example, wlatin1 encoding),both functions return identical results. But in a DBCS session (for example, EUC-JP or SHIFT-JIS encoding), most CJK characters are 2 bytes wide, causing these functions to 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.

Examples

Example 1

These examples use UTF-8 encoding.

```
data _null_;
  text=' Renée Dupont';
  result1=kindexcb(text,'ée');    /*1*/
  result2=kindexcb(text,'éec');   /*2*/
  put result1=/result2=;
run;
```

1 KINDEXCB specifies the first byte position for é and e.

2 KINDEXCB specifies the first byte position for é, e, and c.

The preceding statements produce these results:

```
result1=6
result2=6
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text= '感.def(漢字 123456)';
  result=kindexcb(text,'1234567890','中漢'); /*1*/
  put result=;
run;
```

1 KINDEXCB specifies the first position of 中, 漢 and 0 to 9.

The preceding statements produce these results:

```
result=9
```

KINDEXC Function

Searches a character expression for specified characters and returns character-based values.

Categories: CAS
Character Manipulation

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 555.

Tip: The Non-DBCS equivalent function is "INDEXC Function" in *SAS 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 555 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.

Examples

Example 1

These examples use UTF-8 encoding.

```
data _null_;
  text=' Renée Dupont';
  result1=kindexc(text,'ée');    /*1*/
  result2=kindexc(text,'éec');   /*2*/
  put result1=/result2=;
run;
```

1 KINDEXC specifies the first position of characters é and e.

2 KINDEXC specifies the first position of characters é, e, and c.

The preceding statements produce these results:

```
result1=3
result2=3
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text= '感.def(漢字 123456)';
  result=kindexc(text,'1234567890','中漢'); /*1*/
  put result=;
run;
```

1 KINDEXC specifies the first position of characters 中, 漢, and 0 to 9.

The preceding statements produce these results:

```
result=7
```

See Also

Function:

- “KINDEX Function” on page 647

KLEFT Function

Left-aligns a character expression by removing unnecessary leading DBCS blanks and SO/SI.

Categories: CAS
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 555.

Note: This function supports the VARCHAR data type.

Tip: Non-DBCS equivalent function is LEFT in *SAS 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 555 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 13.6 *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

Unicode Spaces Definition in ANYSPACE	Unicode Spaces That Are Removed by KLEFT, KRIGHT, and KTRIM	Comment
	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
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

Unicode Spaces Definition in ANYSPACE	Unicode Spaces That Are Removed by KLEFT, KRIGHT, and KTRIM	Comment
U+205f	U+205F	MEDIUM MATHEMATICAL SPACE
U+3000	U+3000	IDEOGRAPHIC SPACE

Example

The following examples use UTF-8 encoding..

This example uses Chinese characters.

```
data _null_;
  a='　漢字のテスト';  /*1*/
  b=kleft(a);
  put '---- +---- 1 ---- + ---- 2 ---- +';
  put a $25.;
  put b $25.;
run;
```

1 The first character is a DBCS space.

The preceding statements produce these results:

```
---- +---- 1 ---- + ---- 2 ---- +
  漢字のテスト
漢字のテスト
```

See Also

Functions:

- “KCOMPRESS Function” on page 621
- “KRIGHT Function” on page 675
- “KTRIM Function” on page 696

KLENGTH Function

Returns the length, in characters, of an argument.

Categories: CAS
Character Manipulation

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 555.

Tip: Non-DBCS equivalent function is LENGTH in *SAS 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 555 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));
   if NOT len then len = 1;
```

Example

The following examples use UTF-8 encoding.

```
data _null_;
  result1=klength(' Renée Dupont');
  result2=klength('abc あいう');
  put result1=/result2=;
run;
```

The preceding statements produce these results:

```
result1=13
result2=6
```

KLOWCASE Function

Converts the uppercase alphabetic letters to lowercase letters.

Categories: CAS
Character Manipulation

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 555.

Tip: Non-DBCS equivalent function is LOWCASE in *SAS Functions and CALL Routines: Reference*.

Syntax

KLOWCASE(*argument*)

Required Argument

argument
specifies any SAS character expression.

Details

The KLOWCASE function copies a character argument, converts the uppercase alphabetic letters to lowercase letters, and returns the altered value.

The definitions of the alphabetic letters differ based on the SAS session encoding. Here are typical SAS sessions and how KLOWCASE processes alphabetic letters for the different sessions.

UTF-8 (SAS Unicode server)
When the SAS session encoding is UTF-8, the Unicode character attributes apply. If a character is classified as a letter and has a lowercase equivalent, KLOWCASE returns the lowercase letter. Otherwise, the same letter is returned.

DBCS (For example, Chinese, Japanese and Korean languages)
KLOWCASE converts uppercase letters in the English alphabet to lowercase in a DBCS SAS session.

SBCS (For example, English or European languages)
When the SAS session encoding is SBCS, the TRANTAB option is updated to use the mapping tables for uppercase letters and the lowercase letter

equivalents. The mapping tables for lowercase letters use the naming pattern of *nnnn*_lcs, where *nnnn* matches the short encoding name. The KLOWCASE function uses these tables to convert the uppercase letters to the lowercase letters that are supported by the session encoding.

Examples

Example 1

In this example, the text contains a mix of English alphabetic uppercase letters plus the uppercase Ñ. The SAS session encoding is Windows code page 1252, which is WLATIN1. Notice that KLOWCASE returns all the letters in lowercase, including the ñ. This example demonstrates results from KLOWCASE when the SAS encoding is SBCS and supports Western European characters. The same result is returned for UTF-8 text when the SAS encoding is UTF-8.

```
options locale=en_US;
data one;
   x='NIÑO';
   y=klowcase(x);

   put x=;
   put y=;
run;
```

```
x=NIÑO
y=niño
```

Example 2

```
data _null_;
  result=klowcase('ABC あいうえお');
  put result =;
run;
```

The preceding statements produce these results:

```
result=abc あいうえお
```

KNEXTCHAR Function

Returns a character from a string at a specified byte offset.

Categories: CAS
Character Manipulation

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 555.

Note: A returned character at a specified offset is assigned to a variable:
`onechar=KNEXTCHAR(text,offset`

See: "KOFFSET Function"

Syntax

KNEXTCHAR(*text*,*offset*)

Required Arguments

text
specifies a character constant, variable, or expression.

offset
a number that represents the byte-based input and output value. The initial value of *offset* must be 1.

Details

The KNEXTCHAR function returns a character from a string at a specified byte offset. This function is used to efficiently process every character in linear time. Performance is much more acceptable when processing huge character strings.

The KOFFSET function can be used with KNEXTCHAR to obtain the initial value of *offset*. By utilizing the functionality of both functions and processing the data in a loop, iteration of the character string becomes very efficient.

Examples

Example 1

These examples use a UTF-8 encoding.

```
data _null_;
  text=' Renée Dupont';
  result=knextchar(text,5);  /*1*/
  put result =;
run;
```

1 5 is the byte offset.

The preceding statements produce these results:

```
result=R
```

Example 2

```
data _null_;
  length x varchar(18);
  x = ' Renée Dupont';
  n = length(x);
  offset = koffset(x);          /*1*/
  do i = 1 to n;
    c = knextchar(x, offset);   /*2*/
    put i= c= offset=;
  end;
run;
```

1 Initialize offset to 1.

2 The offset is set to the byte position of the next character after executing KNEXTCHAR.

The preceding statements produce these results:

```
i=1 c=  offset=5
i=2 c=R offset=6
i=3 c=e offset=7
i=4 c=n offset=8
i=5 c=é offset=10
i=6 c=e offset=11
i=7 c=  offset=12
i=8 c=D offset=13
i=9 c=u offset=14
i=10 c=p offset=15
i=11 c=o offset=16
i=12 c=n offset=17
i=13 c=t offset=18
```

KOFFSET Function

Returns the byte offset from the *character position*.

Categories: CAS
Character Manipulation

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 555.

See: "KNEXTCHAR Function"

Syntax

byte_offset=**KOFFSET**(*text*<,*character_position*>);

Required Argument

text
specifies a character constant, variable, or expression.

Optional Argument

character_position
specifies a position in the character string.

Details

The KOFFSET function specifies a byte offset. If *character_position* is not specified, then KOFFSET returns a value of 1.

Examples

Example 1

These examples use a UTF-8 encoding.

```
data _null_;
  text=' Renée Dupont';
  result=koffset(text,2);     /* 1 */
  put result =;
run;
```

1 The second character is R and the byte offset is 5.

The preceding statements produce these results:

```
result=5
```

Example 2

```
data _null_;
```

```
  length x varchar(18);
  x = ' Renée Dupont';
  n = length(x);
  offset = koffset(x);  /*1*/
  do i = 1 to n;
    c = knextchar(x, offset);  /*2*/
    put i= c= offset=;
  end;
run;
```

1 Initialize the offset to 1.

2 The offset is set to the byte position of the next character after executing KNEXTCHAR.

The preceding statements produce these results:

```
i=1 c=  offset=5
i=2 c=R offset=6
i=3 c=e offset=7
i=4 c=n offset=8
i=5 c=é offset=10
i=6 c=e offset=11
i=7 c=  offset=12
i=8 c=D offset=13
i=9 c=u offset=14
i=10 c=p offset=15
i=11 c=o offset=16
i=12 c=n offset=17
i=13 c=t offset=18
```

KPROPCASE Function

Converts Chinese, Japanese, Korean, Taiwanese (CJKT) characters.

Category: Character Manipulation

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 555.

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 555 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.

Comparisons

The PROPCASE function is designed for Latin-based languages. It operates by capitalizing the first letter of each word, with words being defined by delimiters such as spaces.

The KPROPCASE function is designed for double-byte languages like Simplified Chinese, Traditional Chinese, Japanese, and Korean. These languages also require proper case, but not in the form of initial capital as in Latin languages. Instead, they involve other types of conversions, such as full-width and half-width transformations. Additionally, double-byte languages do not use delimiters to separate characters into words as do Latin languages.

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=;
```

```
str1=C4DEB0C5AFC220
str2=8368815B83698362836320
str3=DO-NAXTUTSU
str4=8368815B83698362836320
str5=DO-NAXTUTSU
str6=8263826E817C826D8260827782738274827382728274 20
str7=DO-NAXTUTSU
str8=do-naxtutsu
```

KPROPCHAR Function

Converts Unicode to the corresponding characters enclosed in parenthesis.

Category: Character Manipulation

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 555.

Syntax

str=**KPROPCHAR**(<*instr*>)

Required Arguments

str
result string. Unicode is converted to characters.

instr
input data string.

Details

This function converts Unicode such as ① (0x2460) to an equivalent character (1). 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:

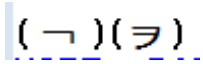
```
length in1 out1 $30 ;
  in1=unicode('\u2460\u2473\u277F\u325F');
  out1=KPROPCHAR(in1);
  put out1;
RESULTS:
(1)(20)(-10)(35)
```

Statements	Results

`data _null_;`
`length in1 out1 $30 ;`
`in1=unicode('\u2460\u2473\u277F\u325F');`
	`(1)(20)(-10)(35)`

Statements	Results
out1=KPROPCHAR(in1); put out1; run;	

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;
	

KPROPDATA Function

Converts text between encodings and handles characters that cannot be represented in the target encoding. KPROPDATA is designed for data cleansing and encoding compatibility in multilingual workflows.

Category: Character Manipulation

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 555.

Syntax

str=**KPROPDATA**(<*instr*>, <*options*>, <*input encode name*>, <*output encode name*>)

Required Arguments

str
specifies a data string that has been converted and is in the session encoding.

instr
specifies the 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.

PUNC

Note: The PUNC option is not supported in SAS 9.

Converts commonly used 8-bit punctuation marks to 7-bit characters.

WLATIN1 characters are 1-byte, but some of these characters are 2-4 bytes in UTF-8. This option converts punctuation marks that are 1-byte in WLATIN1, but are 2-4 bytes in UTF-8. Here are the punctuation marks that can be converted:

Table 13.7 *Smart Quotation Marks and Punctuation Characters*

	Punctuation				Convert To		
Category	**Punctuation character**	**Description**	**WLATIN1 code point (Hex)**	**UTF-8 code point (Hex)**	**ASCII Character**	**Description**	**ASCII code point (Hex)**
Quotation marks	‚	single low-9 quotation mark	82	E2 80 9A	,	comma	2C
	„	double low-9 quotation mark	84	E2 80 9E	,,	quotation mark	22

Category	Punctuation				Convert To		
	Punctuation character	Description	WLATIN1 code point (Hex)	UTF-8 code point (Hex)	ASCII Character	Description	ASCII code point (Hex)
	‘	left single quotation mark	91	E2 80 98	'	apostrophe	27
	’	right single quotation mark	92	E2 80 99	'	apostrophe	27
	“	left double quotation mark	93	E2 80 9C	"	quotation mark	22
	”	right double quotation mark	94	E2 80 9D	"	quotation mark	22
Angle brackets	‹	single left-pointing angle quotation mark	8B	E2 80 B9	<	less-than sign	3C
	›	single right-pointing angle quotation mark	9B	E2 80 BA	>	greater-than sign	3E
Ellipsis	…	horizontal ellipsis	85	E2 80 A6	...	full stop	2E 2E 2E [1]
Bullet	•	bullet	95	E2 80 A2	*	asterisk	2A
Hyphen	–	en hyphen	96	E2 80 93	-	hyphen-minus	2D
	—	em hyphen	97	E2 80 94	-	hyphen-minus	2D

1 Three ASCII period characters.

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 1127.

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 1127.

Details

Basics

This function converts the input data string to the current SAS session encoding and removes or replaces unprintable characters based on the options.

Unicode Escape Interpretation in Input

During processing, the implementation interprets recognized Unicode escape sequences in the input stream:

\uXXXX and \UXXXX style sequences are parsed.

If a Unicode Escape character is escaped with another backslash, \\uXXXX, one backslash is retained to preserve escaping semantics. \\ represents a single backslash only in front a the letter u or U.

Examples

Example 1: LATIN2 SAS Session Encoding

The following example demonstrates the functionality of the KPROPDATA function. This example uses a LATIN2 session encoding.

In this example the registered sign ('ae'x in latin1 session) is not supported in LATIN2 encoding. The example shows how KPROPDATA helps to represent the unsupported character.

- "ae"x is byte 0xAE in WLATIN1 (registered sign).
- In a LATIN2 session, that character is not representable, so it is treated as unprintable/problematic for conversion.
- Each option demonstrates a different recovery strategy.

```
data test;
   length instr $12;
   length str1 str2 str3 str4 str5 str6 str7 str8 $ 50;
   instr = "SAS" ||"ae"x || " System";
   put instr;
   str1 = kpropdata(instr, 'UESC','wlatin1');
```

```
   put str1= +2 str1= $hex34.;
   str2 = kpropdata(instr,'TRIM','wlatin1');
   put str2= +2 str2= $hex26.;
   str3 = kpropdata(instr,'BLANK', 'wlatin1');
   put str3= +2 str3= $hex26.;
   str4 = kpropdata(instr,'?', 'wlatin1');
   put str4= +2 str4= $hex26.;
   str5 = kpropdata(instr,'hex', 'wlatin1');
   put str5= +2 str5= $hex26.;
   str6 = kpropdata(instr,'TRUNC', 'wlatin1');
   put str6= +2 str6= $hex26.;
   str7 = kpropdata(instr,'REMOVE', 'wlatin1');
   put str7= +2 str7= $hex26.;
   str8 = kpropdata(instr,'NCR', 'wlatin1');
   put str8= +2 str8= $hex26.;
run;
```

The preceding statements produce these results:

```
SASŽ System
str1=SAS\u00ae System   str1=5341535C7530306165205379737465 6D20
str2=SAS System   str2=53415320537973746 56D202020
str3=SAS  System   str3=534153202053797374656D2020
str4=SAS? System   str4=5341533F2053797374656D2020
str5=SAS\xae System   str5=5341535C786165205379737465
str6=SAS   str6=534153202020202020202020
str7=    str7=20202020202020202020202020
str8=SAS&#174; System   str8=53415326233137343B20537973
```

Example 2: WLATIN1 SAS Session Encoding

This example shows the functionality of the PUNC option. The string assigned to *instr* contains several punctuation characters that are not ASCII characters. This example was run in SAS with a WLATIN1 session encoding.

```
data _null_;
   length instr $48;
   length outstr $14;
   instr = "––‘’,“”„•…‹›"; /* "969791928293948495858B9B"x */
   put instr= /instr=hex.;
   outstr = kpropdata(instr, 'PUNC');
   put outstr= /outstr=hex.;

run;
```

The preceding statements produce these results:

```
instr =––‘’,“”„•…‹›
instr =979691928293948495858B9B
outstr=--'',"""*...<>
outstr=2D2D27272C2222222A2E2E2E3C3E
\
```

Example 3: UTF-8 SAS Session Encoding

This example uses a UTF-8 session encoding. The value assigned to *instr* contains **'ae'x** which is an invalid character/byte. KPROPDATA processes the invalid UTF-8 character.

```
data test;
   length instr $12;
   length str1 str2  $50;
   instr = "SAS" || 'ae'x || " System"; /* 'ae'x represents here the
invalid character/byte */
   put instr;
   str1 = kpropdata(instr,'hex');
   put str1= +2 str1= $hex26.;
   str2 = kpropdata(instr,'?');
   put str2= +2 str2= $hex26.;
run;
```

These statements produce these results: Notice that by using KPROPDATA the invalid UTF-8 character **'ae'x** is processed.

```
SAS? System
str1=SAS\xae System str1=5341535C786165205379737465
str2=SAS? System str2=5341533F2053797374656D2020
```

KREVERSE Function

Reverses a character expression.

Categories: CAS
Character Manipulation

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 555.

Tip: Non-DBCS equivalent function is REVERSE in *SAS 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 555 for restrictions and more information.

Example

The following example uses a UTF-8 encoding.

```
data _null_;
  result1=kreverse('Renée Dupont');
  result2=kreverse('漢字のテスト');
  put result1=/result2=;
run;
```

The preceding statements produce these results:

```
result1=tnopuD eéneR
result2=トステの字漢
```

KRIGHT Function

Right-aligns a character expression by trimming trailing DBCS blanks and SO/SI.

Categories: CAS
Character Manipulation

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 555.

Tip: See “RIGHT Function” in *SAS 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 555 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 13.180 on page 655 for a list of the Unicode characters.

Example

The following example uses a UTF-8 encoding.

```
data _null_;
  a='漢字のテスト  ';  /*1*/
  b=kright(a);
  put '---- +---- 1 ---- + ---- 2 ---- +';
  put a $25.;
  put b $25.;
run;
```

1 The last character is a DBCS space.

The preceding statements produce these results:

```
---- +---- 1 ---- + ---- 2 ---- +
漢字のテスト
   漢字のテスト
```

See Also

Functions:

- “KCOMPRESS Function” on page 621
- “KLEFT Function” on page 655
- “KTRIM Function” on page 696

KSCAN Function

Selects a specified word from a character expression.

Categories: CAS
Character Manipulation

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 555.

Tip: Non-DBCS equivalent function is SCAN in *SAS 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 555 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.

The KSCAN function does not have a *modify* argument. Use the KSCANX function if you need the *modify* functionality. KSCANX is a complete NLS replacement for the SCAN function.

Examples

Example 1

The following examples use a UTF-8 encoding.

```
data _null_;
  text='Renée Dupont';
  result=kscan(text,2);     /*1*/
  put result=;
run;
```

1 KSCAN specifies the second word. The SBCS space is the delimiter.

The preceding statements produce these results:

```
result=Dupont
```

Example 2

```
data _null_;
  text='ReneéxDupont';
  result=kscan(text,2,'x');     /*1*/
  put result=;
run;
```

1 KSCAN specifies the second word. x is the delimiter.

The preceding statements produce these results:

```
result=Dupont
```

Example 3

This example uses Chinese characters.

```
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;
```

The preceding statements produce these results:

```
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=
```

See Also

- “KSCANX Function” on page 679
- “SCAN Function” in *SAS Functions and CALL Routines: Reference*

KSCANX Function

Selects a specified word from a character expression using a modifier to process the function’s action.

Categories: CAS

Character Manipulation

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 for SAS String Functions” on page 555.

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 an ASCII Environment" in *SAS 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 *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, 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 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 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 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 *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 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, 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

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, 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
- 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:

  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 *character-list* argument. Modifiers do not add to the list of default modifiers.

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.

Examples

Example 1

These examples use UTF-8 encoding.

```
data _null_;
  text='Renée,Dupont';
```

```
  result=kscanx(text,2);     /*1*/
  put result=;
run;
```

1 KSCANX specifies the second word. The SBCS space is the delimiter.

The preceding statements produce these results:

```
result=Dupont
```

Example 2

```
data _null_;
  text='RenéexDupont';
  result=kscan(text,2,'x');     /*1*/
  put result=;
run;
```

1 KSCANX specifies the second word. x is the delimiter.

The preceding statements produce these results:

```
result=Dupont
```

Example 3

This example uses Chinese characters.

```
data _null_;
text='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。';
result=kscanx(text, 2, '', 'sp'); /*1*/
put result=;
run;
```

1 Spaces and puncuation marks are the delimiters.

The preceding statements produce these results:

```
result=sas
```

See Also

- "SCAN Function" in *SAS Functions and CALL Routines: Reference*
- "KSCAN Function"

KSTRCAT Function

Concatenates two or more character expressions.

Categories: CAS
Character Manipulation

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 555.

Tip: Non-DBCS equivalent function is CAT in *SAS 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 555 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.

In the SQL procedure, or in a WHERE clause in any procedure, the maximum length of a word that is returned by the KSTRCAT function is 200 characters.

Examples

Example 1

The following examples use UTF-8 encoding.

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses an emoji.

```
data _null_;
  text1='Renée';
  text2='Dupont';
  result=kstrcat (text1, ' ', text2);
  put result=;
run;
```

The preceding statements produce these results:

```
result=Renée Dupont
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text1='漢字';
  text2='文字列';
  text3='連結';
  result=kstrcat (text1, text2,'の', text3);
  put result=;
run;
```

The preceding statements produce these results:

```
result=漢字文字列の連結
```

KSTRIP Function

Removes leading and trailing blanks from a character string.

Categories: CAS
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 555 .

Note: This function supports the VARCHAR data 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 truncates 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 normalization in a DBCS SAS session using this code:

```
long_char_variable = KSTRIP(long_char_variable);
```

Examples

Example 1

These examples use a UTF-8 encoding.

```
data _null_;
  str = ' Renée ';   /*1*/
  result1 = '*' ||str|| '*';
  result2 = '*' || kstrip(str)|| '*';
  put result1=/result2=;
run;
```

1 The character string contains leading and trailing SBCS blanks.

The preceding statements produce these results:

```
result1=* Renée *
result2=*Renée*
```

Example 2

The following example shows how the KSTRIP function deletes leading and trailing blanks, and how the DBCS character is truncated.

```
data _null_;
   length a $5;
   v = unicode('\u0020\u0020\u8404\u8404\u8404\u0020\u0020');
   put v = hex.;
   a = kstrip(v);
   put a = hex.;
run;
```

The preceding statements produce these results:

```
93B893B820
```

See Also

Function

- "STRIP Function" in *SAS Functions and CALL Routines: Reference*

KSUBSTR Function

Extracts a substring from an argument.

Categories: CAS
Character Manipulation

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 555.

Tip: See "SUBSTR (right of =) Function" in *SAS 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 555 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*.

Examples

Example 1

These examples use a UTF-8 encoding.

```
data _null_;
  text='Renée Dupont';
  result=ksubstr(text,3,5);  /*1*/
  put result=;
run;
```

1 KSUBSTR specifies characters from the character string beginning with the third character and then selecting five characters.

The preceding statements produce these results:

```
result=née D
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text='漢字文字列の抽出';
  result1=ksubstr(text, 1,2);
  result2=ksubstr(text,3,4);
  kstart=7;
  klen=2;
  result3=ksubstr(text,kstart,klen);
  put result1=/result2=/result3=;
run;
```

The preceding statements produce these results:

```
result1=漢字
result2=文字列の
result3=抽出
```

See Also

Function:

- “KSUBSTRB Function” on page 690

KSUBSTRB Function

Extracts a substring from an argument according to the byte position of the substring in the argument.

Category: Character Manipulation

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 555.

The KSUBSTRB function does not work with in-memory CAS tables or SAS files with .hdat extensions.

The KSUBSTRB function does not work with .hdat extensions.

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 555 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*.

Examples

Example 1

These examples use a UTF-8 encoding.

```
data _null_;
  text='Renée Dupont';
  result=ksubstrb(text,3,4);   /*1*/
  put result=;
run;
```

1 KSUBSTRB specifies a character string beginning with byte position 3 for a total of 4 bytes.

The preceding statements produce these results:

```
result=née
```

Example 2

This example uses Chinese characters.

```
data _null_;
  text='漢字文字列の抽出';
  result1=ksubstrb(text,1,9);
  result2=ksubstrb(text,4,6);
  put result1=/result2=;
run;
```

The preceding statements produce these results:

```
result1=漢字文
result2=字文
```

See Also

Function:

- “KSUBSTR Function” on page 688

KSUBSTRN Function

Returns a substring, allowing a result with a length of zero.

Category: Character Manipulation

Restriction: This function is assigned an I18N Level 2 status and designed for use with SBCS, DBCS, and MBCS (UTF8).

See: “SUBSTRN Function” in *SAS Functions and CALL Routines: Reference* and “Internationalization Compatibility for SAS String Functions” on page 555 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. A numeric argument is not supported in PROC SQL.

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.

```
data _null_;
   ksubstr_result = "*" || ksubstr(1234.5678,2,6) || "*";
   put ksubstr_result=;
   ksubstrn_result = "*" || ksubstrn(1234.5678,2,6) || "*";
   put ksubstrn_result=;
run;
```

The preceding statements produce these results:

```
ksubstr_result=* 1234*
ksubstrn_result=*234.56*
```

KTRANSLATE Function

Replaces specified characters in a character expression.

Categories: CAS

Character Manipulation

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 555

Tip: Non-DBCS equivalent function is TRANSLATE in *SAS 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 555 for restrictions and more information.

You can use KTRANSLATE to translate a single-byte character expression to a double-byte character expression or to translate a double-byte character expression to a single-byte character expression.

The maximum number of *to* and *from* character pairs that KTRANSLATE accepts should not exceed 256. There is no functional difference between using many pairs of shorter arguments and using fewer pairs of longer arguments. These two examples have different argument and character combination pairs, but they have the same output.

The first example: `x=ktranslate('AABBAABABB','12','AB','3','C');` has two argument pairs and three character pairs.

The second example: `y=ktranslate('AABBAABABB','123','ABC');` has one argument pair and three character pairs.

Note: CAS supports the second example only: one argument pair and up to 256 character pairs. Extra argument pairs are ignored.

SAS Viya platform supports unlimited argument pairs and up to 256 character pairs.

```
data test;
x=ktranslate('AABBAABABB','12','AB','3','C');  /* Two to/from argument pairs, three
from/to char pairs. */
y=ktranslate('AABBAABABB','123','ABC');  /* One to/from argument pair, three from/to
char pairs. */
put x=;
put y=;
run;
```

```
x=1122112122
y=1122112122
```

If the KTRANSLATE function returns a value to a variable that has not yet been assigned a length, the variable is assigned a default length of 200.

Examples

Example 1

The following examples use UTF-8 encoding.

```
data _null_;
  result=ktranslate('Renée Dupont','ɪ,','é ');
  put result =;
run;
```

The preceding statements produce these results:

```
result=Renɪe,Dupont
```

Example 2

This example uses Chinese characters.

```
data _null_;
  result=ktranslate('仲等','中忠','仲仲');
  put result =;
run;
```

The preceding statements produce these results:

```
result=中等
```

KTRIM Function

Removes trailing DBCS blanks and SO/SI from character expressions.

Categories: CAS
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 555.

Tip: Non-DBCS equivalent function is "TRIM Function" in *SAS 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 555 for restrictions and more information.

KTRIM copies a character argument, removes all trailing blanks, and returns the trimmed argument as a result. If the argument is blank, KTRIM returns one blank. 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 whitespace with a length of 0 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 13.180 on page 655 for a list of the Unicode characters.

Example

This example uses a UTF-8 encoding.

This example uses Chinese characters.

```
data _null_;
  part1='漢字の　';  /*1*/
  part2='テスト';
  hasblank=part1 || part2;
  noblank=ktrim(part1) || part2;
  put hasblank;
  put noblank;
run;
```

1 The character string ends with a DBCS space.

The preceding statements produce these results:

```
漢字の　テスト
漢字のテスト
```

See Also

Functions:

- “KCOMPRESS Function” on page 621
- “KLEFT Function” on page 655
- “KRIGHT Function” on page 675

KTRUNCATE Function

Truncates a string to a specified length in byte unit without breaking multibyte characters.

Category: Character Manipulation

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 555.

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 555 for restrictions and more information.

The KTRUNCATE function truncates a string to a specified length using 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 a UTF-8 encoding.

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses Chinese characters and emojis.

```
data _null_;
  x1=ktruncate('Renée Dupont',4);    /*1*/
  x2=ktruncate('Renée Dupont',5);
  x3=ktruncate('Renée Dupont',6);
  y1=ktruncate('漢字のテスト',3,6);    /*2*/
  y2=ktruncate('漢字のテスト',4,6);
  y3=ktruncate('漢字のテスト',5,6);
  put x1/x2/x3/y1/y2/y3;
run;
```

1 KTRUNCATE truncates characters beginning with the 4th byte.

2 KTRUNCATE truncates 6 bytes beginning with the 3rd byte.

The preceding statements produce these results:

```
Ren
René
Renée
字
字の
の
```

KUPCASE Function

Converts the lowercase alphabetic letters to uppercase letters.

Categories: CAS
Character Manipulation

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 555.

Tip: See "UPCASE Function" in *SAS Functions and CALL Routines: Reference*.

Syntax

KUPCASE(*argument*)

Required Argument

argument
specifies any SAS character expression.

Details

The KUPCASE function copies a character argument, converts the lowercase alphabetic letters to uppercase letters, and returns the altered value.

The definitions of the alphabetic letters differ based on the SAS session encoding. Here are typical SAS sessions and how KUPCASE processes alphabetic letters for the different sessions.

UTF-8 (SAS Unicode server)
When the SAS session encoding is UTF-8, the Unicode character attributes apply. If a character is classified as a letter and has an uppercase equivalent, KUPCASE returns the uppercase letter. Otherwise, the same letter is returned.

DBCS (For example, Chinese, Japanese, and Korean languages)
KUPCASE converts lowercase letters in the English alphabet to uppercase in a DBCS SAS session.

SBCS (For example, English or European languages)
When the SAS session encoding is SBCS, the TRANTAB option is updated to use the mapping tables for uppercase letters and the lowercase equivalents. The mapping tables for uppercase letters use the naming pattern of *nnnn*_ucs, where *nnnn* matches the short encoding name that is documented in *SAS National Language Support (NLS): Reference Guide*. The KUPCASE function uses these tables to convert the lowercase letters to the uppercase letters that are supported by the session encoding.

Examples

Example 1

In this example, the text contains a mix of English alphabetic lowercase letters plus the lowercase ñ. The SAS session encoding is Windows code page 1252, which is WLATIN1. Notice that KUPCASE returns all the letters in uppercase, including the Ñ. This example demonstrates results from KUPCASE when the SAS encoding is SBCS and supports Western European characters. The same result is returned for UTF-8 text when the SAS encoding is UTF-8.

```
options locale=en_US;
data one;
   x='niño';
   y=kupcase(x);

   put x=;
   put y=;
run;
```

The preceding statements produce these results:

```
x=niño
y=NIÑO
```

Example 2

This example uses Chinese characters.

```
data _null_;
  result=kupcase('abc あいうえお');
  put result=;
run;
```

The preceding statements produce these results:

```
result=ABC あいうえお
```

KUPDATE Function

Replaces character value contents.

Categories: CAS
Character Manipulation

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 555.

Syntax

KUPDATE(*argument,position,n*<, *characters-to-replace*>)

KUPDATE(*argument,position,*<,n>, *characters-to-replace*)

Required Arguments

argument
specifies a character variable or string.

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 555 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.

In the SQL procedure, or in a WHERE clause in any procedure, the maximum length of a word that is returned by the KUPDATE function is 200 characters.

Comparisons

The KUPDATES function updates the target string *argument* based on the third argument *n*.The KUPDATE function updates the target string *argument* using the length of the last argument *characters-to-replace*, regardless of the third argument *n* value.

Here is an example of KUPDATE and KUPDATES.

```
data _null_;
   x = 'abcde';
   y = kupdate (x, 1, 4, '12');
   z = kupdates(x, 1, 4, '12');
   put y=;
   put z=;
run;
```

KUPDATE replaces abcd with 12. KUPDATES replaces abcd with 12 and blank padding.

```
   y=12e
   z=12  e
```

Examples

Example 1

The following examples use a UTF-8 encoding.

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses Japanese characters and emojis.

```
   data _null_;
     x1=kupdate('Renée Dupont',1,2,'Yè');  /*1*/
     x2=kupdate(x1,1,2,'abcde');
```

```
  x3=kupdate(x1,1,3);
  x4=kupdate(x1,3,'π!€');
  y1=kupdate('感じのテスト',1,2,'漢字');
  y2=kupdate(y1,1,2,'kanji');
  y3=kupdate(y1,1,3);
  y4=kupdate(y1,3,'かんじ');
  put x1/x2/x3/x4/y1/y2/y3/y4;
run;
```

1 KUPDATE replaces the first two characters with Yè, so Re is replaced with Yè.

The preceding statements produce these results:

```
Yènée Dupont
abcdenée Dupont
ée Dupont
Yèπ!€
漢字のテスト
kanji のテスト
テスト
漢字かんじ
```

Example 2

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses Japanese characters and emojis.

This example demonstrates the difference between KUPDATE and KUPDATES.

```
data _null_;
  x1='Renée Dupont';
  x2=kupdate(x1,1,2,'abcde');
  x3=kupdates(x1,1,2,'abcde');
  y1='感じのテスト';
  y2=kupdate(y1,1,2,'kanji');
  y3=kupdates(y1,1,2,'ka');
  put x2/x3/y2/y3;
run;
```

The preceding statements produce these results:

```
abcdenée Dupont
abnée Dupont
kanji のテスト
ka のテスト
```

See Also

Functions:

- “KUPDATES Function” on page 706

■ "KUPDATEB Function" on page 704

KUPDATEB Function

Replaces the contents of the character value according to the byte position of the character value in the argument.

Category: Character Manipulation

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 555.

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 555 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.

In the SQL procedure, or in a WHERE clause in any procedure, the maximum length of a word that is returned by the KUPDATEB function is 200 characters.

Example

This example uses a UTF-8 encoding.

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses Chinese characters and emojis.

```
data _null_;
  x1=kupdateb('Renée Dupont',1,2,'Yr');  /*1*/
  x2=kupdateb(x1,1,4,'abcde');
  x3=kupdateb(x1,1,5);
  x4=kupdateb(x1,3,'π!€');
  y1=kupdateb('感じのテスト',1,6,'漢字');
  y2=kupdateb(y1,1,6,'kanji');
  y3=kupdateb(y1,1,6);
  y4=kupdateb(y1,6,'かんじ');
  put x1/x2/x3/x4/y1/y2/y3/y4;
run;
```

1 KUPDATEB replaces the first two bytes in the character string. Re is replaced with Yr.

The preceding statements produce these results:

```
Yrnée Dupont
abcd e Dupont
e Dupont
Yrπ!€
漢字のテスト
kanji のテスト
のテスト
漢  かんじ
```

See Also

Function:

■ "KUPDATE Function" on page 701

KUPDATES Function

Replaces character value contents.

Categories: CAS
Character Manipulation

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 555.

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 555 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.

In the SQL procedure, or in a WHERE clause in any procedure, the maximum length of a word that is returned by the KUPDATES function is 200 characters.

Comparisons

The KUPDATES function updates the target string *argument* based on the third argument *n*. The KUPDATE function updates the target string *argument* using the length of the last argument *characters-to-replace*, regardless of the third argument *n* value.

Here is an example of KUPDATE and KUPDATES.

```
data _null_;
   x = 'abcde';
   y = kupdate (x, 1, 4, '12');
   z = kupdates(x, 1, 2, '12345');
   put y=;
   put z=;
run;
```

KUPDATE replaces abcd with 12. KUPDATES replaces abcd with 12 and blank padding.

```
   y=12e
   y=12  e
```

Examples

Example 1

These examples use a UTF-8 encoding.

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses Chinese characters and emojis.

```
   data _null_;
     x1=kupdates('Renée Dupont',1,2,'Yè');  /* 1 */
```

```
  x2=kupdates(x1,1,2,'abcde');
  x3=kupdates(x1,1,3);
  x4=kupdates(x1,3,'π!€');
  y1=kupdates('感じのテスト',1,2,'漢字');
  y2=kupdates(y1,1,2,'kanji');
  y3=kupdates(y1,1,3);
  y4=kupdates(y1,3,'かんじ');
  put x1/x2/x3/x4/y1/y2/y3/y4;
run;
```

1 KUPDATES replaces the first two characters in the character string. RE is replaced with Yè.

The preceding statements produce these results:

```
Yènée Dupont
abnée Dupont
ée Dupont
Yèπ!€
漢字のテスト
ka のテスト
テスト
漢字かんじ
```

Example 2

Here is an example of how support surrogate pair chars are supported, it is displayed for view in the HTML format. This example uses Chinese characters and emojis.

The following example shows the difference between KUPDATE and KUPDATES

```
data _null_;
  x1='Renée Dupont';
  x2=kupdate(x1,1,2,'abcde');
  x3=kupdates(x1,1,2,'abcde');
  y1='感じのテスト';
  y2=kupdate(y1,1,2,'kanji');
  y3=kupdates(y1,1,2,'kanji');
  put x2/x3/y2/y3;
run;
```

The preceding statements produce these results:

```
abcdenée Dupont
abnée Dupont
kanji のテスト
ka のテスト
```

See Also

Functions:

- “KUPDATE Function” on page 701

KVERIFY Function

Returns the position of the first character (character-based value) that is unique to an expression.

Categories: CAS
Character Manipulation

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 555.

Tip: See “VERIFY Function” in *SAS 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 555 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.

Examples

Example 1

The following examples use a UTF-8 encoding.

```
data _null_;
  check='»é';
  result=kverify('ééé»ééABC', check);  /*1*/
  put result=;
run;
```

1 KVERIFY specifies the position of character A. The *excerpt* value is »é, so KVERIFY searches for the first character that is not »é.

The preceding statements produce these results:

```
result=7
```

Example 2

This example uses Chinese characters.

```
data _null_;
  input grade :$3. @@;
  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
  ;
```

The preceding statements produce these results:

```
Invalid grade value: 可
```

KVERIFYB Function

Returns the position of the first character (byte-based value) that is unique to an expression.

Category: Character Manipulation

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 555.

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

KVERIFYB 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 a byte-based value. KVERIFY returns a character-based value. When processing an SBCS string (for example, in the wlatin1 encoding), these functions return identical results. But, in a DBCS session (for example, EUC-JP or SHIFT-JIS encoding), most CJK characters are 2-byte width, which makes these functions return different results.

Example

The following example uses a UTF-8 encoding.

```
data _null_;
  check='»é';
  result=kverifyb('ééé»ééABC', check);  /* 1 */
  put result=;
run;
```

1 KVERIFYB specifies the byte position of character A. The *excerpt* value is »é, so KVERIFYB searches for the first character that is not »é

The preceding statements produce these results:

```
result=13
```

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 555.

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`

`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:

- “NLDATEw. Format” on page 191

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 555.

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----+`

`options locale=English;`
`time_ampm=nldatm('24Feb2003:12:39:43'dt,'%I%p');`
`put time_ampm;`
	`12PM`

`options locale=German;`	`12nachm`

Statements	Results
`time_ampm=nldatm('24Feb2003:12:39:43'dt,'%I%p');` `put time_ampm;`	

See Also

Format:

- “NLDATMw. Format” on page 223

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 555.

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;`
	`12PM`

`options locale=German;`
`time_ampm=nltime('12:39:43't,'%i%p');`
`put time_ampm;`
	`12nachm`

See Also

Format:

- “NLTIMEw. Format” on page 500

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 555.

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 555.

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 '\u*xxxx*, 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", "n"));`	`The Access Control key was successfully saved.`
`sasmsgl("nls.mymsg", "IN_APW_SAVE_OK", "es_ES", "n"));`	`La clave de control de acceso se ha guardado.`
`sasmsgl("nls.mymsg", "IN_APW_SAVE_OK", "fr", "n"));`	`La clé de contrôle d'accès a bien été enregist`

Example 2

The following example demonstrates the open macro feature:

SAS Statements	Results
`%PUT %SYSFUNC(SASMSGL(NLS.MYDS, IN_ASD_LABEL, en_US));`	`"Edit"`
`%PUT %SYSFUNC(SASMSGL(NLS.MYDS, IN_ASD_LABEL, es_ES));`	`"Editar"`
`%PUT %SYSFUNC(SASMSGL(NLS.MYDS, IN_ASD_LABEL, fr));`	`"Modifier"`

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 555.

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	
SWISS	32	0	
ITALIC	32	0	
DATE_FORMAT	512	0	LC_TIME

Locale Element Key	Max Length	Type	Category
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
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

Locale Element Key	Max Length	Type	Category
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
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

Locale Element Key	Max Length	Type	Category
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
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

Locale Element Key	Max Length	Type	Category
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
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

Locale Element Key	Max Length	Type	Category
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¥

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 555.

Data type: Binary

Note: SORTKEY is not supported in these environments: CAS, CMP, DS2, or FedSQL.

Syntax

sortKey(*string*, <*locale*, *strength*, *case_order*, *numeric_order*, *collation order*>)

Required Argument

string
character expression

Optional Arguments

locale
specifies the locale name in the form of a POSIX name (ja_JP). See Table 21.1 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.

Example

Here is an example of the SORTKEY function:

```
data mydata;
input name & $15. ;
datalines;
côte
ôte
coté
45 Main St.
côté
zoo
```

```
cote
8 Main St.
;
run ;


data list ;
set mydata;
format key_fr $hex40.;
key_fr = sortkey(name,"fr_CA",,,'N');
run ;

PROC SQL ;
create table my_fr_CA_sort as
SELECT * FROM list
ORDER BY key_fr;
QUIT ;
```

TRANTAB Function

Transcodes data by using the specified translation table.

Category: Character

Restriction: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility on page 555.

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

See Also

Procedures:

- Chapter 20, “TRANTAB Procedure,” on page 1079

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 555.

This function is not supported in a DATA step that runs in CAS.

Syntax

TZONEID(<*time-zone-id*>)

Optional Argument

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 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=ETC/GMT-12`

`data null;` `  name_valid=tzoneid('asia/tokyo');`	`name_valid=ASIA/TOKYO` `name_invalid=`

Statements	Results
name_invalid=tzoneid('Milky Way'); put name_valid =; put name_invalid=; run;	

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 555.

This function is not supported in a DATA step that runs in CAS.

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 1153.

datetime

specifies a SAS datetime value.

Details

The TZONENAME function returns a blank value if the TIMEZONE= option and the time-zone-id argument are 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 time-zone-id 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='America/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 555.

This function is not supported in a DATA step that runs in CAS.

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 1153.

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 name returns the time zone offset for the specified time zone name. 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`

Statements	Results
`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=14400`
	`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=14400`
	`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 555.

This function is not supported in a DATA step that runs in CAS.

Syntax

TZONES2U(*datetime*,<*time-zone-id*>)

Required Argument

datetime
specifies a SAS datetime value.

Optional Argument

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 1153

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. If time-zone-id is not specified, then the timezone ID that is specified with the TIMEZONE= option is used.

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 555.

This function is not supported in a DATA step that runs in CAS.

Syntax

TZONEDSTNAME()

TZONEDSTNAME(<*time-zone-id*>)

Optional Argument

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 the time zone name and daylight savings time rules.

Details

If the time-zone-id is not specified, then the TIMEZONE ID that is specified by the TIMEZONE= option is used. If the daylight savings time does not exist for the specified time zone, 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 555.

This function is not supported in a DATA step that runs in CAS.

Syntax

TZONEDSTOFF()

TZONEDSTOFF(<*time-zone-id*>)

Optional Argument

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 the time zone name and daylight savings time rules.

Details

If the time-zone-id is not specified, then the TIMEZONE ID that is specified by the TIMEZONE= option is used. If the daylight savings time does not exist for the specified time zone, 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 555.

This function is not supported in a DATA step that runs in CAS.

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 555.

This function is not supported in a DATA step that runs in CAS.

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 time-zone-id 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 555.

This function is not supported in a DATA step that runs in CAS.

Syntax

TZONEU2S(*UTC-date-time value*<, *time-zone-id*>)

Required Argument

UTC-date time value
specifies a Coordinated Universal Time (UTC) datetime value

Optional Argument

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 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.

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=02SEP12:04:34:56`
	`sdt3=02SEP12:03:34:56`

UNICODE Function

Converts Unicode characters to the current SAS session encoding.

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 555.

Syntax

STR=**UNICODE**(<*instr*> (,<Unicode type>))

Required Argument

str
Data string that has been converted to the current SAS session encoding.

Optional Arguments

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 and UNICODEC functions can accept Unicode SMP (Supplementary Multilingual Plane) characters in UTF-8, UTF16, or UTF32 encoding, but cannot accept Unicode SMP characters in surrogate pair form such as \uD84C or \uDF8F. See “Example 2:”

Examples

Example 1

This example demonstrates the functionality of the UNICODE function. This code can be run in 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;
```

SAS writes the following output to the log:

```
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
```

Example 2

This example shows the functionality of the UNICODE and UNICODEC functions when processing SMP characters:

```
data _null_;
   length x x1 x2 $4 y $10 y1 $9 y2 $11 u16 $4 u32 $4 z z1 z2 $4;
   x = '𩸽';  /* SMP character */
   u16 = UNICODEC(x, "utf16b");
   x1 = UNICODE(u16, "utf16b");
   u32 = UNICODEC(x, "utf32b");
   x2 = UNICODE(u32, "utf32b");
   y = UNICODEC(x, "esc");
   z = UNICODE(y, "esc");
   y1 = UNICODEC(x, "ncr");
   z1 = UNICODE(y1, "ncr");
   y2 = UNICODEC(x, "paren");
   z2 = UNICODE(y2, "paren");
   put x= x=hex.;
   put x1= x1=hex.;
   put x2= x2=hex.;
   put u16=hex. /u32=hex.;
   put y= y=hex.;
   put y1= y1=hex.;
   put y2= y2=hex.;
   put z= z1= z2=;
run;
```

SAS writes the following output to the log:

```
x=𩸽 x=F0A38E8F
x1=𩸽 x1=F0A38E8F
x2=𩸽 x2=F0A38E8F
u16=D84CDF8F
u32=0002338F
y=\u0002338F y=5C7530303033333846
y1=&#144271; y1=2623313443237313B20
y2=<u0002338F> y2=3C753030303233333846 3E
z=𩸽 z1=𩸽 z2=𩸽
```

UNICODEC Function

Converts characters in the current SAS session encoding to Unicode 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 555.

Syntax

STR=**UNICODEC**(<*instr*> (,<Unicode type>))

Required Argument

str
data string that has been converted to Unicode encoding.

Optional Arguments

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.

The UNICODE and UNICODEC functions can accept Unicode SMP (Supplementary Multilingual Plane) characters in UTF-8, UTF16, or UTF32 encoding, but cannot accept Unicode SMP characters in surrogate pair form such as \uD84C or \uDF8F. See “Example 2:”

Examples

Example 1

This example demonstrates the functionality of the UNICODEC function:

```
data _null_;
  str1=unicodec("ABC", 'utf8');
  put str1= $hex12.;
  str2=unicodec("ABCé", 'utf8');
  put str2= $hex12.;
run;
```

SAS writes the following output to the log:

```
str1=414243202020
str2=414243C3A920
```

Example 2

The following example demonstrates the functionality of the UNICODE and UNICODEC functions when processing SMP characters:

```
data _null_;
   length x x1 x2 $4 y $10 y1 $9 y2 $11 u16 $4 u32 $4 z z1 z2 $4;
   x = '𠮷';  /* SMP character */
   u16 = UNICODEC(x, "utf16b");
   x1 = UNICODE(u16, "utf16b");
   u32 = UNICODEC(x, "utf32b");
   x2 = UNICODE(u32, "utf32b");
   y = UNICODEC(x, "esc");
   z = UNICODE(y, "esc");
   y1 = UNICODEC(x, "ncr");
   z1 = UNICODE(y1, "ncr");
   y2 = UNICODEC(x, "paren");
   z2 = UNICODE(y2, "paren");
   put x= x=hex.;
   put x1= x1=hex.;
   put x2= x2=hex.;
   put u16=hex. /u32=hex.;
   put y= y=hex.;
   put y1= y1=hex.;
   put y2= y2=hex.;
   put z= z1= z2=;
run;
```

SAS writes the following output to the log:

```
x=酳 x=F0A38E8F
x1=酳 x1=F0A38E8F
x2=酳 x2=F0A38E8F
u16=D84CDF8F
u32=0002338F
y=\u0002338F y=5C75303030233333846
y1=&#144271; y1=2623313434323731 3B20
y2=<u0002338F> y2=3C753030303233333846 3E
z=酳 z1=酳 z2=酳
```

UNICODELEN Function

Specifies the length of the character unit for the Unicode data.

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 555.

Syntax

UNICODELEN()

Details

The UNICODELEN function specifies the length of the character unit for the UNICODE data.

Example

This example uses the Japanese Shift_JIS session encoding, which is supported under the LINUX operating system.

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 756

UNICODEWIDTH Function

Specifies the length of a display unit for the Unicode data.

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 555.

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 the Japanese Shift_JIS session encoding, which is supported under the LINUX operating system.

Statements	Results
`len1=unicodewidth("abc ");`	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 755

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 31 in *SAS National Language Support (NLS): Reference Guide*. For information about encoding values and transcoding data, see SBCS, DBCS, and Unicode Encoding Values When Transcoding SAS Data on page 1127 in *SAS National Language Support (NLS): 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 Functions and CALL Routines: Reference*
- "OPEN Function" in *SAS Functions and CALL Routines: Reference*
- "VARNUM Function" in *SAS Functions and CALL Routines: Reference*
- "VTRANSCODE Function" on page 759

- “VTRANSCODEX Function” on page 760

VTRANSCODE Function

Returns a value that indicates whether transcoding is enabled for the specified character variable.

Categories: CAS
Variable Information

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 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 760

Statements:

- ATTRIB in

VTRANSCODEX Function

Returns a value that indicates whether transcoding is enabled for the specified argument.

Categories: CAS
Variable Information

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 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 759

Statements:

- ATTRIB

PART 6

Informats for NLS

Chapter 14
Dictionary of Informats for NLS **765**

14

Dictionary of Informats for NLS

Informats by Category 767

Dictionary 781
$CPTDWw. Informat 781
$CPTWDw. Informat 782
EUROw.d Informat 784
EUROXw.d Informat 786
JNENGOw. Informat 788
$KANJIw. Informat 789
$KANJIXw. Informat 790
$LOGVSw. Informat 791
$LOGVSRw. Informat 793
MINGUOw. Informat 794
NENGOw. Informat 796
NLDATEw. Informat 798
NLDATEWw. Informat 799
NLDATMw. Informat 800
NLDATMAP Informat 801
NLDATMWw. Informat 802
NLMNIAEDw.d Informat 803
NLMNIAUDw.d Informat 805
NLMNIBGNw.d Informat 806
NLMNIBRLw.d Informat 807
NLMNICADw.d Informat 809
NLMNICHFw.d Informat 810
NLMNICNYw.d Informat 811
NLMNICZKw.d Informat 812
NLMNIDKKw.d Informat 814
NLMNIEEKw.d Informat 815
NLMNIEGPw.d Informat 816
NLMNIEURw.d Informat 818
NLMNIGBPw.d Informat 819
NLMNIHKDw.d Informat 820
NLMNIHRKw.d Informat 821
NLMNIHUFw.d Informat 823
NLMNIIDRw.d Informat 824
NLMNIILSw.d Informat 825

NLMNIINRw.d Informat 827
NLMNIJPYw.d Informat 828
NLMNIKRWw.d Informat 829
NLMNILTLw.d Informat 830
NLMNILVLw.d Informat 832
NLMNIMOPw.d Informat 833
NLMNIMXNw.d Informat 834
NLMNIMYRw.d Informat 836
NLMNINOKw.d Informat 837
NLMNINZDw.d Informat 838
NLMNIPLNw.d Informat 839
NLMNIRUBw.d Informat 841
NLMNISEKw.d Informat 842
NLMNISGDw.d Informat 843
NLMNITHBw.d Informat 845
NLMNITRYw.d Informat 846
NLMNITWDw.d Informat 847
NLMNIUSDw.d Informat 848
NLMNIZARw.d Informat 850
NLMNLAEDw.d Informat 851
NLMNLAUDw.d Informat 852
NLMNLBGNw.d Informat 854
NLMNLBRLw.d Informat 855
NLMNLCADw.d Informat 856
NLMNLCHFw.d Informat 857
NLMNLCNYw.d Informat 859
NLMNLCZKw.d Informat 860
NLMNLDKKw.d Informat 861
NLMNLEEKw.d Informat 863
NLMNLEGPw.d Informat 864
NLMNLEURw.d Informat 865
NLMNLGBPw.d Informat 867
NLMNLHKDw.d Informat 868
NLMNLHRKw.d Informat 869
NLMNLHUFw.d Informat 870
NLMNLIDRw.d Informat 872
NLMNLILSw.d Informat 873
NLMNLINRw.d Informat 874
NLMNLJPYw.d Informat 876
NLMNLKRWw.d Informat 877
NLMNLLTLw.d Informat 878
NLMNLLVLw.d Informat 879
NLMNLMOPw.d Informat 881
NLMNLMXNw.d Informat 882
NLMNLMYRw.d Informat 883
NLMNLNOKw.d Informat 885
NLMNLNZDw.d Informat 886
NLMNLPLNw.d Informat 887
NLMNLRUBw.d Informat 888
NLMNLSEKw.d Informat 890
NLMNLSGDw.d Informat 891
NLMNLTHBw.d Informat 892
NLMNLTRYw.d Informat 894
NLMNLTWDw.d Informat 895

NLMNLUSDw.d Informat 896
NLMNLZARw.d Informat 897
NLMNYw.d Informat 899
NLMNYIw.d Informat 901
NLNUMw.d Informat 902
NLNUMIw.d Informat 904
NLPCTw.d Informat 906
NLPCTIw.d Informat 907
NLSTRMON Informat 909
NLTIMAPw. Informat 911
NLTIMEw. Informat 912
$REVERJw. Informat 913
$REVERSw. Informat 914
$UCS2Bw. Informat 915
$UCS2BEw. Informat 917
$UCS2Lw. Informat 918
$UCS2LEw. Informat 919
$UCS2Xw. Informat 921
$UCS2XEw. Informat 922
$UCS4Bw. Informat 923
$UCS4Lw. Informat 924
$UCS4Xw. Informat 926
$UCS4XEw. Informat 927
$UESCw. Informat 928
$UESCEw. Informat 930
$UNCRw. Informat 931
$UNCREw. Informat 933
$UPARENw. Informat 934
$UPARENEw. Informat 936
$UPARENPw. Informat 937
$UTF8Xw. Informat 938
$VSLOGw. Informat 940
$VSLOGRw. Informat 941
YENw.d Informat 943

Informats by Category

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

These categories for SAS informats support NLS:

Category	Description
BIDI Text Handling	Instructs SAS to read bidirectional data values from data variables. See BIDI Text Handling for a list of informats.
CAS	Instructs SAS that these informats run on the CAS server. See CAS for a list of informats.
Character	Instructs SAS to read character data values into character variables. See Character for a list of informats.
DBCS	Instructs SAS to manage various Asian languages. See DBCS for a list of informats.
Date and Time	Instructs SAS to read data values into variables that represent dates, times, and datetimes. See Date and Time for a list of informats.
Hebrew Text Handling	Instructs SAS to read Hebrew data from data variables. See Hebrew Text Handling for a list of informats.
Numeric	Instructs SAS to read numeric data values into numeric variables. See Numeric for a list of informats.

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
BIDI Text Handling	$LOGVSw. Informat (p. 791)	Reads a character string that is in left-to-right logical order, and then converts the character string to visual order.
	$LOGVSRw. Informat (p. 793)	Reads a character string that is in right-to-left logical order, and then converts the character string to visual order.
	$VSLOGw. Informat (p. 940)	Reads a character string that is in visual order, and then converts the character string to left-to-right logical order.
	$VSLOGRw. Informat (p. 941)	Reads a character string that is in visual order, and then converts the character string to right-to-left logical order.
CAS	EUROw.d Informat (p. 784)	Reads numeric values, removes embedded characters in European currency, and reverses the comma and decimal point.
	EUROXw.d Informat (p. 786)	Reads numeric values and removes embedded characters in European currency.

Category	Language Elements	Description
	$LOGVSw. Informat (p. 791)	Reads a character string that is in left-to-right logical order, and then converts the character string to visual order.
	$LOGVSRw. Informat (p. 793)	Reads a character string that is in right-to-left logical order, and then converts the character string to visual order.
	NENGOw. Informat (p. 796)	Reads Japanese date values in the form eyymmdd.
	NLDATEw. Informat (p. 798)	Reads the date value as the local SAS date value that is appropriate for the current SAS locale.
	NLDATEWw. Informat (p. 799)	Reads the date value as a local SAS date and the day of the week that is appropriate for the current SAS locale.
	NLDATMw. Informat (p. 800)	Reads a datetime value that is appropriate for the current SAS locale.
	NLDATMAP Informat (p. 801)	Reads the date value as a local SAS datetime with either a.m. or p.m. that is appropriate for the current SAS locale.
	NLDATMWw. Informat (p. 802)	Reads the date value as a local SAS day of the week and datetime that is appropriate for the current SAS locale.
	NLMNIAEDw.d Informat (p. 803)	Reads the monetary format of the international expression for the United Arab Emirates.
	NLMNIAUDw.d Informat (p. 805)	Reads the monetary format of the international expression for Australia.
	NLMNIBGNw.d Informat (p. 806)	Reads the monetary format of the international expression for Bulgaria.
	NLMNIBRLw.d Informat (p. 807)	Reads the monetary format of the international expression for Brazil.
	NLMNICADw.d Informat (p. 809)	Reads the monetary format of the international expression for Canada.
	NLMNICHFw.d Informat (p. 810)	Reads the monetary format of the international expression for Liechtenstein and Switzerland.
	NLMNICNYw.d Informat (p. 811)	Reads the monetary format of the international expression for China.
	NLMNICZKw.d Informat (p. 812)	Reads the monetary format of the international expression for Czechia.
	NLMNIDKKw.d Informat (p. 814)	Reads the monetary format of the international expression for Denmark, Faroe Island, and Greenland.
	NLMNIEEKw.d Informat (p. 815)	Reads the monetary format of the international expression for Estonia.

Category	Language Elements	Description
	NLMNIEGPw.d Informat (p. 816)	Reads the monetary format of the international expression for Egypt.
	NLMNIEURw.d Informat (p. 818)	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.
	NLMNIGBPw.d Informat (p. 819)	Reads the monetary format of the international expression for the United Kingdom.
	NLMNIHKDw.d Informat (p. 820)	Reads the monetary format of the international expression for Hong Kong.
	NLMNIHRKw.d Informat (p. 821)	Reads the monetary format of the international expression for Croatia.
	NLMNIHUFw.d Informat (p. 823)	Reads the monetary format of the international expression for Hungary.
	NLMNIIDRw.d Informat (p. 824)	Reads the monetary format of the international expression for Indonesia.
	NLMNIILSw.d Informat (p. 825)	Reads the monetary format of the international expression for Israel.
	NLMNIINRw.d Informat (p. 827)	Reads the monetary format of the international expression for India.
	NLMNIJPYw.d Informat (p. 828)	Reads the monetary format of the international expression for Japan.
	NLMNIKRWw.d Informat (p. 829)	Reads the monetary format of the international expression for South Korea.
	NLMNILTLw.d Informat (p. 830)	Reads the monetary format of the international expression for Lithuania.
	NLMNILVLw.d Informat (p. 832)	Reads the monetary format of the international expression for Latvia.
	NLMNIMOPw.d Informat (p. 833)	Reads the monetary format of the international expression for Macau.
	NLMNIMXNw.d Informat (p. 834)	Reads the monetary format of the international expression for Mexico.
	NLMNIMYRw.d Informat (p. 836)	Reads the monetary format of the international expression for Malaysia.
	NLMNINOKw.d Informat (p. 837)	Reads the monetary format of the international expression for Norway.

Category	Language Elements	Description
	NLMNINZDw.d Informat (p. 838)	Reads the monetary format of the international expression for New Zealand.
	NLMNIPLNw.d Informat (p. 839)	Reads the monetary format of the international expression for Poland.
	NLMNIRUBw.d Informat (p. 841)	Reads the monetary format of the international expression for Russia.
	NLMNISEKw.d Informat (p. 842)	Reads the monetary format of the international expression for Sweden.
	NLMNISGDw.d Informat (p. 843)	Reads the monetary format of the international expression for Singapore.
	NLMNITHBw.d Informat (p. 845)	Reads the monetary format of the international expression for Thailand.
	NLMNITRYw.d Informat (p. 846)	Reads the monetary format of the international expression for Turkey.
	NLMNITWDw.d Informat (p. 847)	Reads the monetary format of the international expression for Taiwan.
	NLMNIUSDw.d Informat (p. 848)	Reads the monetary format of the international expression for Puerto Rico and the United States.
	NLMNIZARw.d Informat (p. 850)	Reads the monetary format of the international expression for South Africa.
	NLMNLAEDw.d Informat (p. 851)	Reads the monetary format of the local expression for the United Arab Emirates.
	NLMNLAUDw.d Informat (p. 852)	Reads the monetary format of the local expression for Australia.
	NLMNLBGNw.d Informat (p. 854)	Reads the monetary format of the local expression for Bulgaria.
	NLMNLBRLw.d Informat (p. 855)	Reads the monetary format of the local expression for Brazil.
	NLMNLCADw.d Informat (p. 856)	Reads the monetary format of the local expression for Canada.
	NLMNLCHFw.d Informat (p. 857)	Reads the monetary format of the local expression for Liechtenstein and Switzerland.
	NLMNLCNYw.d Informat (p. 859)	Reads the monetary format of the local expression for China.
	NLMNLCZKw.d Informat (p. 860)	Reads the monetary format of the local expression for Czechia.

Category	Language Elements	Description
	NLMNLDKKw.d Informat (p. 861)	Reads the monetary format of the local expression for Denmark, the Faroe Island, and Greenland.
	NLMNLEEKw.d Informat (p. 863)	Reads the monetary format of the local expression for Estonia.
	NLMNLEGPw.d Informat (p. 864)	Reads the monetary format of the local expression for Egypt.
	NLMNLEURw.d Informat (p. 865)	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.
	NLMNLGBPw.d Informat (p. 867)	Reads the monetary format of the local expression for the United Kingdom.
	NLMNLHKDw.d Informat (p. 868)	Reads the monetary format of the local expression for Hong Kong.
	NLMNLHRKw.d Informat (p. 869)	Reads the monetary format of the local expression for Croatia.
	NLMNLHUFw.d Informat (p. 870)	Reads the monetary format of the local expression for Hungary.
	NLMNLIDRw.d Informat (p. 872)	Reads the monetary format of the local expression for Indonesia.
	NLMNLILSw.d Informat (p. 873)	Reads the monetary format of the local expression for Israel.
	NLMNLINRw.d Informat (p. 874)	Reads the monetary format of the local expression for India.
	NLMNLJPYw.d Informat (p. 876)	Reads the monetary format of the local expression for Japan.
	NLMNLKRWw.d Informat (p. 877)	Reads the monetary format of the local expression for South Korea.
	NLMNLLTLw.d Informat (p. 878)	Reads the monetary format of the local expression for Lithuania.
	NLMNLLVLw.d Informat (p. 879)	Reads the monetary format of the local expression for Latvia.
	NLMNLMOPw.d Informat (p. 881)	Reads the monetary format of the local expression for Macau.
	NLMNLMXNw.d Informat (p. 882)	Reads the monetary format of the local expression for Mexico.

Category	Language Elements	Description
	NLMNLMYRw.d Informat (p. 883)	Reads the monetary format of the local expression for Malaysia.
	NLMNLNOKw.d Informat (p. 885)	Reads the monetary format of the local expression for Norway.
	NLMNLNZDw.d Informat (p. 886)	Reads the monetary format of the local expression for New Zealand.
	NLMNLPLNw.d Informat (p. 887)	Reads the monetary format of the local expression for Poland.
	NLMNLRUBw.d Informat (p. 888)	Reads the monetary format of the local expression for Russia.
	NLMNLSEKw.d Informat (p. 890)	Reads the monetary format of the local expression for Sweden.
	NLMNLSGDw.d Informat (p. 891)	Reads the monetary format of the local expression for Singapore.
	NLMNLTHBw.d Informat (p. 892)	Reads the monetary format of the local expression for Thailand.
	NLMNLTRYw.d Informat (p. 894)	Reads the monetary format of the local expression for Turkey.
	NLMNLTWDw.d Informat (p. 895)	Reads the monetary format of the local expression for Taiwan.
	NLMNLUSDw.d Informat (p. 896)	Reads the monetary format of the local expression for Puerto Rico, and the United States.
	NLMNLZARw.d Informat (p. 897)	Reads the monetary format of the local expression for South Africa.
	NLMNYw.d Informat (p. 899)	Reads monetary data for the local expression as a numeric value that is appropriate for the current SAS locale.
	NLMNYIw.d Informat (p. 901)	Reads monetary data as a numeric value for the international expression that is appropriate for the current SAS locale.
	NLNUMw.d Informat (p. 902)	Reads numeric data for local expressions as a numeric value that is appropriate for the current SAS locale.
	NLNUMIw.d Informat (p. 904)	Reads numeric data for international expressions as a numeric value that is appropriate for the current SAS locale.
	NLPCTw.d Informat (p. 906)	Reads percentage data for local expressions as a numeric value that is appropriate for the current SAS locale.
	NLPCTIw.d Informat (p. 907)	Reads percentage data for international expressions as a numeric value that is appropriate for the current SAS locale.

Category	Language Elements	Description
	NLSTRMON Informat (p. 909)	Reads the month name as a numeric value that is appropriate for the current SAS locale.
	NLTIMAPw. Informat (p. 911)	Reads the time value and uses a.m. and p.m. as the local SAS time value that is appropriate for the current SAS locale.
	NLTIMEw. Informat (p. 912)	Reads the time value as the local SAS time value that is appropriate for the current SAS locale.
	$REVERJw. Informat (p. 913)	Reads character data from right to left and preserves blanks.
	$REVERSw. Informat (p. 914)	Reads character data from right to left, and then left aligns the text.
	$VSLOGw. Informat (p. 940)	Reads a character string that is in visual order, and then converts the character string to left-to-right logical order.
	$VSLOGRw. Informat (p. 941)	Reads a character string that is in visual order, and then converts the character string to right-to-left logical order.
	YENw.d Informat (p. 943)	Removes embedded yen signs, commas, and decimal points.
Character	$REVERJw. Informat (p. 913)	Reads character data from right to left and preserves blanks.
	$REVERSw. Informat (p. 914)	Reads character data from right to left, and then left aligns the text.
	$UCS2Bw. Informat (p. 915)	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.
	$UCS2BEw. Informat (p. 917)	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.
	$UCS2Lw. Informat (p. 918)	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.
	$UCS2LEw. Informat (p. 919)	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.
	$UCS2Xw. Informat (p. 921)	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.
	$UCS2XEw. Informat (p. 922)	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	Language Elements	Description
	$UCS4Bw. Informat (p. 923)	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.
	$UCS4Lw. Informat (p. 924)	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.
	$UCS4Xw. Informat (p. 926)	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.
	$UCS4XEw. Informat (p. 927)	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.
	$UESCw. Informat (p. 928)	Reads a character string that is encoded in UESC representation, and then converts the character string to the encoding of the current SAS session.
	$UESCEw. Informat (p. 930)	Reads a character string that is in the encoding of the current SAS session, and then converts the character string to UESC representation.
	$UNCRw. Informat (p. 931)	Reads an NCR character string, and then converts the character string to the encoding of the current SAS session.
	$UNCREw. Informat (p. 933)	Reads a character string in the encoding of the current SAS session, and then converts the character string to NCR.
	$UPARENw. Informat (p. 934)	Reads a character string that is encoded in UPAREN representation, and then converts the character string to the encoding of the current SAS session.
	$UPARENEw. Informat (p. 936)	Reads a character string that is in the encoding of the current SAS session, and then converts the character string to UPAREN representation.
	$UPARENPw. Informat (p. 937)	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.
	$UTF8Xw. Informat (p. 938)	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	JNENGOw. Informat (p. 788)	Reads Japanese kanji date values in the form yymmdd.
	MINGUOw. Informat (p. 794)	Reads dates in Taiwanese format.
	NENGOw. Informat (p. 796)	Reads Japanese date values in the form eyymmdd.

Category	Language Elements	Description
	NLDATEw. Informat (p. 798)	Reads the date value as the local SAS date value that is appropriate for the current SAS locale.
	NLDATEWw. Informat (p. 799)	Reads the date value as a local SAS date and the day of the week that is appropriate for the current SAS locale.
	NLDATMw. Informat (p. 800)	Reads a datetime value that is appropriate for the current SAS locale.
	NLDATMAP Informat (p. 801)	Reads the date value as a local SAS datetime with either a.m. or p.m. that is appropriate for the current SAS locale.
	NLDATMWw. Informat (p. 802)	Reads the date value as a local SAS day of the week and datetime that is appropriate for the current SAS locale.
	NLTIMAPw. Informat (p. 911)	Reads the time value and uses a.m. and p.m. as the local SAS time value that is appropriate for the current SAS locale.
	NLTIMEw. Informat (p. 912)	Reads the time value as the local SAS time value that is appropriate for the current SAS locale.
DBCS	$KANJIw. Informat (p. 789)	Removes shift code data from DBCS data.
	$KANJIXw. Informat (p. 790)	Adds shift-code data to DBCS data.
Hebrew Text Handling	$CPTDWw. Informat (p. 781)	Reads a character string that is in Hebrew DOS (cp862) encoding, and then converts the character string to Windows (cp1255) encoding.
	$CPTWDw. Informat (p. 782)	Reads a character string that is in Windows (cp1255) encoding, and then converts the character string to Hebrew DOS (cp862) encoding.
Numeric	EUROw.d Informat (p. 784)	Reads numeric values, removes embedded characters in European currency, and reverses the comma and decimal point.
	EUROXw.d Informat (p. 786)	Reads numeric values and removes embedded characters in European currency.
	NLMNIAEDw.d Informat (p. 803)	Reads the monetary format of the international expression for the United Arab Emirates.
	NLMNIAUDw.d Informat (p. 805)	Reads the monetary format of the international expression for Australia.
	NLMNIBGNw.d Informat (p. 806)	Reads the monetary format of the international expression for Bulgaria.
	NLMNIBRLw.d Informat (p. 807)	Reads the monetary format of the international expression for Brazil.

Category	Language Elements	Description
	NLMNICADw.d Informat (p. 809)	Reads the monetary format of the international expression for Canada.
	NLMNICHFw.d Informat (p. 810)	Reads the monetary format of the international expression for Liechtenstein and Switzerland.
	NLMNICNYw.d Informat (p. 811)	Reads the monetary format of the international expression for China.
	NLMNICZKw.d Informat (p. 812)	Reads the monetary format of the international expression for Czechia.
	NLMNIDKKw.d Informat (p. 814)	Reads the monetary format of the international expression for Denmark, Faroe Island, and Greenland.
	NLMNIEEKw.d Informat (p. 815)	Reads the monetary format of the international expression for Estonia.
	NLMNIEGPw.d Informat (p. 816)	Reads the monetary format of the international expression for Egypt.
	NLMNIEURw.d Informat (p. 818)	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.
	NLMNIGBPw.d Informat (p. 819)	Reads the monetary format of the international expression for the United Kingdom.
	NLMNIHKDw.d Informat (p. 820)	Reads the monetary format of the international expression for Hong Kong.
	NLMNIHRKw.d Informat (p. 821)	Reads the monetary format of the international expression for Croatia.
	NLMNIHUFw.d Informat (p. 823)	Reads the monetary format of the international expression for Hungary.
	NLMNIIDRw.d Informat (p. 824)	Reads the monetary format of the international expression for Indonesia.
	NLMNIILSw.d Informat (p. 825)	Reads the monetary format of the international expression for Israel.
	NLMNIINRw.d Informat (p. 827)	Reads the monetary format of the international expression for India.
	NLMNIJPYw.d Informat (p. 828)	Reads the monetary format of the international expression for Japan.
	NLMNIKRWw.d Informat (p. 829)	Reads the monetary format of the international expression for South Korea.

Category	Language Elements	Description
	NLMNILTLw.d Informat (p. 830)	Reads the monetary format of the international expression for Lithuania.
	NLMNILVLw.d Informat (p. 832)	Reads the monetary format of the international expression for Latvia.
	NLMNIMOPw.d Informat (p. 833)	Reads the monetary format of the international expression for Macau.
	NLMNIMXNw.d Informat (p. 834)	Reads the monetary format of the international expression for Mexico.
	NLMNIMYRw.d Informat (p. 836)	Reads the monetary format of the international expression for Malaysia.
	NLMNINOKw.d Informat (p. 837)	Reads the monetary format of the international expression for Norway.
	NLMNINZDw.d Informat (p. 838)	Reads the monetary format of the international expression for New Zealand.
	NLMNIPLNw.d Informat (p. 839)	Reads the monetary format of the international expression for Poland.
	NLMNIRUBw.d Informat (p. 841)	Reads the monetary format of the international expression for Russia.
	NLMNISEKw.d Informat (p. 842)	Reads the monetary format of the international expression for Sweden.
	NLMNISGDw.d Informat (p. 843)	Reads the monetary format of the international expression for Singapore.
	NLMNITHBw.d Informat (p. 845)	Reads the monetary format of the international expression for Thailand.
	NLMNITRYw.d Informat (p. 846)	Reads the monetary format of the international expression for Turkey.
	NLMNITWDw.d Informat (p. 847)	Reads the monetary format of the international expression for Taiwan.
	NLMNIUSDw.d Informat (p. 848)	Reads the monetary format of the international expression for Puerto Rico and the United States.
	NLMNIZARw.d Informat (p. 850)	Reads the monetary format of the international expression for South Africa.
	NLMNLAEDw.d Informat (p. 851)	Reads the monetary format of the local expression for the United Arab Emirates.
	NLMNLAUDw.d Informat (p. 852)	Reads the monetary format of the local expression for Australia.

Category	Language Elements	Description
	NLMNLBGNw.d Informat (p. 854)	Reads the monetary format of the local expression for Bulgaria.
	NLMNLBRLw.d Informat (p. 855)	Reads the monetary format of the local expression for Brazil.
	NLMNLCADw.d Informat (p. 856)	Reads the monetary format of the local expression for Canada.
	NLMNLCHFw.d Informat (p. 857)	Reads the monetary format of the local expression for Liechtenstein and Switzerland.
	NLMNLCNYw.d Informat (p. 859)	Reads the monetary format of the local expression for China.
	NLMNLCZKw.d Informat (p. 860)	Reads the monetary format of the local expression for Czechia.
	NLMNLDKKw.d Informat (p. 861)	Reads the monetary format of the local expression for Denmark, the Faroe Island, and Greenland.
	NLMNLEEKw.d Informat (p. 863)	Reads the monetary format of the local expression for Estonia.
	NLMNLEGPw.d Informat (p. 864)	Reads the monetary format of the local expression for Egypt.
	NLMNLEURw.d Informat (p. 865)	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.
	NLMNLGBPw.d Informat (p. 867)	Reads the monetary format of the local expression for the United Kingdom.
	NLMNLHKDw.d Informat (p. 868)	Reads the monetary format of the local expression for Hong Kong.
	NLMNLHRKw.d Informat (p. 869)	Reads the monetary format of the local expression for Croatia.
	NLMNLHUFw.d Informat (p. 870)	Reads the monetary format of the local expression for Hungary.
	NLMNLIDRw.d Informat (p. 872)	Reads the monetary format of the local expression for Indonesia.
	NLMNLILSw.d Informat (p. 873)	Reads the monetary format of the local expression for Israel.
	NLMNLINRw.d Informat (p. 874)	Reads the monetary format of the local expression for India.

Category	Language Elements	Description
	NLMNLJPYw.d Informat (p. 876)	Reads the monetary format of the local expression for Japan.
	NLMNLKRWw.d Informat (p. 877)	Reads the monetary format of the local expression for South Korea.
	NLMNLLTLw.d Informat (p. 878)	Reads the monetary format of the local expression for Lithuania.
	NLMNLLVLw.d Informat (p. 879)	Reads the monetary format of the local expression for Latvia.
	NLMNLMOPw.d Informat (p. 881)	Reads the monetary format of the local expression for Macau.
	NLMNLMXNw.d Informat (p. 882)	Reads the monetary format of the local expression for Mexico.
	NLMNLMYRw.d Informat (p. 883)	Reads the monetary format of the local expression for Malaysia.
	NLMNLNOKw.d Informat (p. 885)	Reads the monetary format of the local expression for Norway.
	NLMNLNZDw.d Informat (p. 886)	Reads the monetary format of the local expression for New Zealand.
	NLMNLPLNw.d Informat (p. 887)	Reads the monetary format of the local expression for Poland.
	NLMNLRUBw.d Informat (p. 888)	Reads the monetary format of the local expression for Russia.
	NLMNLSEKw.d Informat (p. 890)	Reads the monetary format of the local expression for Sweden.
	NLMNLSGDw.d Informat (p. 891)	Reads the monetary format of the local expression for Singapore.
	NLMNLTHBw.d Informat (p. 892)	Reads the monetary format of the local expression for Thailand.
	NLMNLTRYw.d Informat (p. 894)	Reads the monetary format of the local expression for Turkey.
	NLMNLTWDw.d Informat (p. 895)	Reads the monetary format of the local expression for Taiwan.
	NLMNLUSDw.d Informat (p. 896)	Reads the monetary format of the local expression for Puerto Rico, and the United States.
	NLMNLZARw.d Informat (p. 897)	Reads the monetary format of the local expression for South Africa.

Category	Language Elements	Description
	NLMNYw.d Informat (p. 899)	Reads monetary data for the local expression as a numeric value that is appropriate for the current SAS locale.
	NLMNYIw.d Informat (p. 901)	Reads monetary data as a numeric value for the international expression that is appropriate for the current SAS locale.
	NLNUMw.d Informat (p. 902)	Reads numeric data for local expressions as a numeric value that is appropriate for the current SAS locale.
	NLNUMIw.d Informat (p. 904)	Reads numeric data for international expressions as a numeric value that is appropriate for the current SAS locale.
	NLPCTw.d Informat (p. 906)	Reads percentage data for local expressions as a numeric value that is appropriate for the current SAS locale.
	NLPCTIw.d Informat (p. 907)	Reads percentage data for international expressions as a numeric value that is appropriate for the current SAS locale.
	NLSTRMON Informat (p. 909)	Reads the month name as a numeric value that is appropriate for the current SAS locale.
	YENw.d Informat (p. 943)	Removes embedded yen signs, commas, and decimal points.

Dictionary

$CPTDWw. 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

Restriction: This informat is not supported in a DATA step that runs in CAS.

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:

- "$CPTDWw. Format" on page 153
- "$CPTWDw. Format" on page 154

Informat:

- "$CPTWDw. Informat" on page 782

$CPTWDw. 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

Restriction: This informat is not supported in a DATA step that runs in CAS.

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:

- “$CPTWDw. Format” on page 154
- “$CPTDWw. Format” on page 153

Informat:

- “$CPTDWw. Informat” on page 781

EUROw.d Informat

Reads numeric values, removes embedded characters in European currency, and reverses the comma and decimal point.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “EUROw.d Format” on page 164
- “EUROXw.d Format” on page 167

Informat:

- “EUROXw.d Informat” on page 786

EUROXw.d Informat

Reads numeric values and removes embedded characters in European currency.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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

Values	Statements	Results
1.23	input x eurox10.; put x;	123
1,234.56	input x eurox10.; put x;	1.23456

See Also

Formats:

- “EUROw.d Format” on page 164
- “EUROXw.d Format” on page 167

Informat:

- “EUROw.d Informat” on page 784

JNENGOw. Informat

Reads Japanese kanji date values in the form *yymmdd*.

Category: Date and Time

Alignment: Left

Restriction: This informat is not supported in a DATA step that runs in CAS.

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

JNENGO*w.*

Syntax Description

w
specifies the width of the input field.

Default 16

Range 16–32

Details

The JNENGOw. informat reads Japanese kanji values in the form *yymmdd*. You can separate the year, month, and day values by blanks or special characters. The width of the input field should include space for blanks and special characters.

Note: SAS interprets a two-digit year as belonging to the 100-year span that is defined by the YEARCUTOFF= system option.

Example

The following example shows how to use the JNENGO informat to convert kanji values to SAS date values. The date values in the data lines are separated by special characters.

```
data _null_;
   input x jnengo.;
   datalines;
明治1年4月6日
明治45年7月29日
大正1年7月30日
大正15年12月24日
昭和1年12月25日
昭和64年 1月 7日
平成1年1月8日
平成10年12月 8日
;
```

$KANJIw. Informat

Removes shift code data from DBCS data.

Category: DBCS

Restriction: This informat is not supported in a DATA step that runs in CAS.

Syntax

$KANJI*w.*

Syntax Description

w
specifies the width of the input field.

Range The minimum width for the informat is 2.

Restriction The width must be an even number. If it is an odd number, it is truncated. The width must be equal to or greater than the length of the shift-code data.

Details

The $KANJI informat removes shift-code data from DBCS data. The $KANJI informat processes host-mainframe data. $KANJI can be used on other platforms. If you use the $KANJI informat on non-EBCDIC (non-modal encoding) hosts, the data does not change.

The data must start with SO and end with SI, unless single-byte blank data are returned. The input data length must be `2 + (SO/SI length)*2`.

See Also

Formats:

- “$KANJIw. Format” on page 178
- “$KANJIXw. Format” on page 179

Informat:

- “$KANJIXw. Informat” on page 790

$KANJIXw. Informat

Adds shift-code data to DBCS data.

Category: DBCS

Restriction: This informat is not supported in a DATA step that runs in CAS.

Syntax

$KANJIX*w.*

Syntax Description

w
specifies the width of the input field.

Range The minimum width for the informat is *2 + (length of shift code used on the current DBCSTYPE= setting)*2.*

Restriction The width must be an even number. If it is an odd number, it is truncated. The width must be equal to or greater than the length of the shift-code data.

Details

The $KANJIX informat adds shift-code data to DBCS data that does not have shift-code data. If the input data is blank, shift-code data is not added. The $KANJIX informat processes host-mainframe data, but $KANJIX can be used on other platforms. If you use the $KANJIX informat on non-EBCDIC (non-modal encoding) hosts, the data does not change.

See Also

Formats:

- "$KANJIw. Format" on page 178
- "$KANJIXw. Format" on page 179

Informat:

- "$KANJIw. Informat" on page 789

$LOGVSw. Informat

Reads a character string that is in left-to-right logical order, and then converts the character string to visual order.

Categories: BIDI Text Handling
CAS

Syntax

$LOGVS*w*.

Syntax Description

w
specifies the width of the input field.

Default 200

Range 1–32767

Comparisons

The $LOGVS*w*. informat performs processing that is opposite to the LOGVSR*w*. informat.

Example

The following example uses the Hebrew input value of "טיסה flight."

Statements	Result
	`----+----1----+`
x=input (' טיסה flight',$logvs12.); `put x;`	טיסה flight

The following example uses the Arabic input value of " كمبيوتر computer."

Statements	Result
	`----+----1----+`
x=input (' كمبيوتر computer',$logvs12.); `put x;`	كمبيوتر computer

See Also

Formats:

- "$LOGVSRw. Format" on page 182
- "$LOGVSw. Format" on page 180

Informat:

- "$LOGVSRw. Informat" on page 793

$LOGVSRw. Informat

Reads a character string that is in right-to-left logical order, and then converts the character string to visual order.

Categories: BIDI Text Handling
CAS

Syntax

$LOGVSR*w*.

Syntax Description

w
specifies the width of the input field.

Default 200

Range 1–32767

Comparisons

The $LOGVSR*w*. informat performs processing that is opposite to the $LOGVS*w*. informat.

Example

The following example uses the Hebrew input value of "טיסה flight."

Statements	Results
	----+----1----+
x=input (' טיסה flight',$logvsr12.); put x;	flight טיסה

The following example uses the Arabic input value of " كمبيوتر computer."

Statements	Results
	----+----1----+
x=input (' كمبيوتر computer',$logvsr12.); put x;	كمبيوتر computer

See Also

Formats:

- "$LOGVSw. Format" on page 180
- "$LOGVSRw. Format" on page 182

Informat:

- "$LOGVSw. Informat" on page 791

MINGUOw. Informat

Reads dates in Taiwanese format.

Category: Date and Time

Restriction: This informat is not supported in a DATA step that runs in CAS.

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “MINGUOw. Format” on page 183

Informat:

- “YYMMDDw. Informat” in *SAS Formats and Informats: Reference*

NENGOw. Informat

Reads Japanese date values in the form *eyymmdd*.

Categories: CAS
Date and Time

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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, Heisei, or Reiwa).

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 example uses different input values.

```
data _null_;
input x nengo.;
put x= date9.;
datalines;
h11108
h.11108
11/10/08
;
run;
```

```
x=08OCT1999
x=08OCT1999
x=08OCT1999
```

See Also

Formats:

- “NENGOw. Format” on page 185

NLDATEw. Informat

Reads the date value as the local SAS date value that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alias: NLDATEW

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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----+`

`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.);`
	`15760`

Statements	Results
`put y=;`	

See Also

Format:

- “NLDATEw. Format” on page 191

NLDATEWw. Informat

Reads the date value as a local SAS date and the day of the week that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATEW*w*.

Syntax Description

w
specifies the width of the input 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----+

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

NLDATMw. Informat

Reads a datetime value that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alias: NLDATMW, NLDATMAP

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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`

`options locale=German_Germany;`
`y=input('24.Februar 2003 12.39 Uhr', nldatm.);`
`put y=;`
	`1330171200`

See Also

Format:

- “NLDATMw. Format” on page 223

NLDATMAP Informat

Reads the date value as a local SAS datetime with either a.m. or p.m. that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATMAP*w*.

Syntax Description

w
specifies the width of the input 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

`options locale=Spanish_Mexico;`
`data;`
`  dy='24/02/2003 12:39:43 PM';`
`  y=input(dy,nldatmap200.);`
`  put y=;`
`run;`
	1708864783

NLDATMWw. Informat

Reads the date value as a local SAS day of the week and datetime that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLDATMW*w*

Syntax Description

w

specifies the width of the input 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

NLMNIAEDw.d Informat

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIAED*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLAEDw.d Informat” on page 851

NLMNIAUDw.d Informat

Reads the monetary format of the international expression for Australia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIAUD*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLAUDw.d Informat” on page 852

NLMNIBGNw.d Informat

Reads the monetary format of the international expression for Bulgaria.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIBGN*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLBGNw.d Informat” on page 854

NLMNIBRLw.d Informat

Reads the monetary format of the international expression for Brazil.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIBRL*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLBRLw.d Informat” on page 855

NLMNICADw.d Informat

Reads the monetary format of the international expression for Canada.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNICAD*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNICADw.d Format” on page 376

NLMNICHFw.d Informat

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNICHF*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNICHFw.d Format” on page 378

NLMNICNYw.d Informat

Reads the monetary format of the international expression for China.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNICNY*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNICNYw.d Format” on page 379

NLMNICZKw.d Informat

Reads the monetary format of the international expression for Czechia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNICZK*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLCZKw.d Informat” on page 860

NLMNIDKKw.d Informat

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIDKK*w.d*

Syntax Description

w

specifies the width of the input 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)',nlmndkk32.2);
y=input('($12,345.67)',dollar32.2);
```

Statements	Results
	----+----1----+
put x=;	-12345.67
put y=;	-12345.67

See Also

Format:

- “NLMNIDKKw.d Format” on page 382

NLMNIEEKw.d Informat

Reads the monetary format of the international expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIEEK*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLEEKw.d Informat” on page 863

NLMNIEGPw.d Informat

Reads the monetary format of the international expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding

or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIEGP*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLEGPw.d Informat” on page 864

NLMNIEURw.d 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.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIEUR*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIEURw.d Format” on page 386

NLMNIGBPw.d Informat

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIGBP*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIGBPw.d Format” on page 388

NLMNIHKDw.d Informat

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIHKD*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIHKDw.d Format” on page 389

NLMNIHRKw.d Informat

Reads the monetary format of the international expression for Croatia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIHRK*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLHRKw.d Informat” on page 869

NLMNIHUFw.d Informat

Reads the monetary format of the international expression for Hungary.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIHUF*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLHUFw.d Informat” on page 870

NLMNIIDRw.d Informat

Reads the monetary format of the international expression for Indonesia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIIDR*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLIDRw.d Informat” on page 872

NLMNIILSw.d Informat

Reads the monetary format of the international expression for Israel.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIILS*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIILSw.d Format” on page 395

NLMNIINRw.d Informat

Reads the monetary format of the international expression for India.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIINR*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLINRw.d Informat” on page 874

NLMNIJPYw.d Informat

Reads the monetary format of the international expression for Japan.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIJPY*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIJPYw.d Format” on page 397

NLMNIKRWw.d Informat

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

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIKRW*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLKRWw.d Informat” on page 877

NLMNILTLw.d Informat

Reads the monetary format of the international expression for Lithuania.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNILTL*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLLTLw.d Informat” on page 878

NLMNILVLw.d Informat

Reads the monetary format of the international expression for Latvia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNILVL*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLLVLw.d Informat” on page 879

NLMNIMOPw.d Informat

Reads the monetary format of the international expression for Macau.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIMOP*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLMOPw.d Informat” on page 881

NLMNIMXNw.d Informat

Reads the monetary format of the international expression for Mexico.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIMXN*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLMXNw.d Informat” on page 882

NLMNIMYRw.d Informat

Reads the monetary format of the international expression for Malaysia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIMYR*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIMYRw.d Format” on page 406

NLMNINOKw.d Informat

Reads the monetary format of the international expression for Norway.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNINOK*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNINOKw.d Format” on page 407

NLMNINZDw.d Informat

Reads the monetary format of the international expression for New Zealand.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNINZD*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNINZDw.d Format” on page 409

NLMNIPLNw.d Informat

Reads the monetary format of the international expression for Poland.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIPLN*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIPLNw.d Format” on page 410

NLMNIRUBw.d Informat

Reads the monetary format of the international expression for Russia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIRUB*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIRUBw.d Format” on page 411

NLMNISEKw.d Informat

Reads the monetary format of the international expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNISEK*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNISEKw.d Format” on page 413

NLMNISGDw.d Informat

Reads the monetary format of the international expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNISGD*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNISGDw.d Format” on page 414

NLMNITHBw.d Informat

Reads the monetary format of the international expression for Thailand.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNITHB*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLTHBw.d Informat” on page 892

NLMNITRYw.d Informat

Reads the monetary format of the international expression for Turkey.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNITRY*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLTRYw.d Informat” on page 894

NLMNITWDw.d Informat

Reads the monetary format of the international expression for Taiwan.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNITWD*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNITWDw.d Format” on page 418

NLMNIUSDw.d Informat

Reads the monetary format of the international expression for Puerto Rico and the United States.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIUSD*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIUSDw.d Format” on page 420

NLMNIZARw.d Informat

Reads the monetary format of the international expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNIZAR*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIZARw.d Format” on page 421

NLMNLAEDw.d Informat

Reads the monetary format of the local expression for the United Arab Emirates.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLAED*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIAEDw.d Informat” on page 803

NLMNLAUDw.d Informat

Reads the monetary format of the local expression for Australia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLAUD*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLAUDw.d Format” on page 424

NLMNLBGNw.d Informat

Reads the monetary format of the local expression for Bulgaria.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLBGN*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIBGNw.d Informat” on page 806

NLMNLBRLw.d Informat

Reads the monetary format of the local expression for Brazil.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLBRL*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIBRLw.d Informat” on page 807

NLMNLCADw.d Informat

Reads the monetary format of the local expression for Canada.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLCAD*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLCADw.d Format” on page 428

NLMNLCHFw.d Informat

Reads the monetary format of the local expression for Liechtenstein and Switzerland.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLCHF*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLCHFw.d Format” on page 430

NLMNLCNYw.d Informat

Reads the monetary format of the local expression for China.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLCNY*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLCNYw.d Format” on page 431

NLMNLCZKw.d Informat

Reads the monetary format of the local expression for Czechia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLCZK*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNICZKw.d Informat” on page 812

NLMNLDKKw.d Informat

Reads the monetary format of the local expression for Denmark, the Faroe Island, and Greenland.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLDKK*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLDKKw.d Format” on page 434

NLMNLEEKw.d Informat

Reads the monetary format of the local expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLEEK*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIEEKw.d Informat” on page 815

NLMNLEGPw.d Informat

Reads the monetary format of the local expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLEGP*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIEGPw.d Informat” on page 816

NLMNLEURw.d 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.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLEUR*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLEURw.d Format” on page 438

NLMNLGBPw.d Informat

Reads the monetary format of the local expression for the United Kingdom.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLGBP*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLGBPw.d Format” on page 440

NLMNLHKDw.d Informat

Reads the monetary format of the local expression for Hong Kong.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLHKD*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLHKDw.d Format” on page 441

NLMNLHRKw.d Informat

Reads the monetary format of the local expression for Croatia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLHRK*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIHRKw.d Informat” on page 821

NLMNLHUFw.d Informat

Reads the monetary format of the local expression for Hungary.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLHUF*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIHUFw.d Informat” on page 823

NLMNLIDRw.d Informat

Reads the monetary format of the local expression for Indonesia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLIDR*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIIDRw.d Informat” on page 824

NLMNLILSw.d Informat

Reads the monetary format of the local expression for Israel.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLILS*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLILSw.d Format” on page 447

NLMNLINRw.d Informat

Reads the monetary format of the local expression for India.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLINR*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNIINRw.d Informat” on page 827

NLMNLJPYw.d Informat

Reads the monetary format of the local expression for Japan.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLJPY*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLJPYw.d Format” on page 450

NLMNLKRWw.d Informat

Reads the monetary format of the local expression for South Korea.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLKRW*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIKRWw.d Informat” on page 829

NLMNLLTLw.d Informat

Reads the monetary format of the local expression for Lithuania.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLLTL*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNILTLw.d Informat” on page 830

NLMNLLVLw.d Informat

Reads the monetary format of the local expression for Latvia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLLVL*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNILVLw.d Informat” on page 832

NLMNLMOPw.d Informat

Reads the monetary format of the local expression for Macau.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLMOP*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIMOPw.d Informat” on page 833

NLMNLMXNw.d Informat

Reads the monetary format of the local expression for Mexico.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLMXN*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNIMXNw.d Informat” on page 834

NLMNLMYRw.d Informat

Reads the monetary format of the local expression for Malaysia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLMYR*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLMYRw.d Format” on page 458

NLMNLNOKw.d Informat

Reads the monetary format of the local expression for Norway.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLNOK*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLNOKw.d Format” on page 459

NLMNLNZDw.d Informat

Reads the monetary format of the local expression for New Zealand.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLNZD*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLNZDw.d Format” on page 461

NLMNLPLNw.d Informat

Reads the monetary format of the local expression for Poland.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLPLN*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLPLNw.d Format” on page 462

NLMNLRUBw.d Informat

Reads the monetary format of the local expression for Russia.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLRUB*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLRUBw.d Format” on page 464

NLMNLSEKw.d Informat

Reads the monetary format of the local expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLSEK*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLSEKw.d Format” on page 465

NLMNLSGDw.d Informat

Reads the monetary format of the local expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLSGD*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLSGDw.d Format” on page 467

NLMNLTHBw.d Informat

Reads the monetary format of the local expression for Thailand.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLTHB*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNITHBw.d Informat” on page 845

NLMNLTRYw.d Informat

Reads the monetary format of the local expression for Turkey.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLTRY*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNITRYw.d Informat” on page 846

NLMNLTWDw.d Informat

Reads the monetary format of the local expression for Taiwan.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLTWD*w.d*

Syntax Description

w
specifies the width of the input 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:

- “NLMNLTWDw.d Format” on page 471

NLMNLUSDw.d Informat

Reads the monetary format of the local expression for Puerto Rico, and the United States.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLUSD*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLUSDw.d Format” on page 472

NLMNLZARw.d Informat

Reads the monetary format of the local expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLMNLZAR*w.d*

Syntax Description

w

specifies the width of the input 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:

- “NLMNLZARw.d Format” on page 474

NLMNYw.d Informat

Reads monetary data for the local expression as a numeric value that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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 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);`
`y=input('($12,345.67)',dollar32);`
`put x=;`
`put y=;`
	`-12345.67`
	`-12345.67`

See Also

Formats:

- “NLMNYw.d Format” on page 475
- “NLMNYIw.d Format” on page 479

Informat:

- “NLMNYIw.d Informat” on page 901

NLMNYIw.d Informat

Reads monetary data as a numeric value for the international expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.

Example

The following example use the input value of 12,345.67.

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

`options LOCALE=English_UnitedStates;`
`x=input('(USD12,345.67)',nlmnyi32.2);`
`y=input('$-12,345.67)',dollar32.2);`
`put x=;`
`put y=;`
	`-12345.67`
	`-12345.67`

See Also

Formats:

- “NLMNYw.d Format” on page 475
- “NLMNYIw.d Format” on page 479

Informat:

- “NLMNYw.d Informat” on page 899

NLNUMw.d Informat

Reads numeric data for local expressions as a numeric value that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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. If a decimal symbol is not found, the data is regarded as an integer.

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;`	`-1234356.78`

Statements	Results
`x=input('-1,234,356.78',nlnum32.2);` `put x=;`	

See Also

Formats:

- “NLNUMw.d Format” on page 483
- “NLMNYIw.d Format” on page 479

Informat:

- “NLNUMIw.d Informat” on page 904

NLNUMIw.d Informat

Reads numeric data for international expressions as a numeric value that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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----+`

`options locale=English_UnitedStates;`
`x=input('-1,234,356.78', nlnumi32.2);`
`put x=;`
	`-1234356.78`

See Also

Formats:

- “NLNUMw.d Format” on page 483
- “NLNUMIw.d Format” on page 485
- “NLNUMw.d Informat” on page 902

NLPCTw.d Informat

Reads percentage data for local expressions as a numeric value that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “NLPCTw.d Format” on page 487
- “NLPCTIw.d Format” on page 489
- “NLPCTIw.d Informat” on page 907

NLPCTIw.d Informat

Reads percentage data for international expressions as a numeric value that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “NLPCTw.d Format” on page 487
- “NLPCTIw.d Format” on page 489

Informat:

- “NLPCTw.d Informat” on page 906

NLSTRMON Informat

Reads the month name as a numeric value that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

NLSTRMON

Details

The NLSTRMON informat reads the name of the month in the specified locale and converts it to a numeric value. These examples use the English_United States locale:.

- The first month (January) = 1
- The second month (February) = 2
- The third month (March) = 3
- The fourth month (April) = 4
- The fifth month (May) = 5
- The sixth month (June) = 6
- The seventh month (July) = 7
- The eight month (August) = 8
- The ninth month (September) = 9
- The 10th month (October) = 10
- The 11th month (November) = 11
- The 12th month (December) = 12

Example

This example uses the French_France (fr_fr) locale:

Statements	Results

`options locale = fr_fr;`
`data test;`
`x = input('janvier',nlstrmon.);`
`put x= ;`
`put x= nlstrmon.;`
`x = input('mai',nlstrmon.);`
`put x=;`
`put x= nlstrmon.;`
`run;`
	`x=1`
	`x=janvier`
	`x=5`
	`x=mai`

`option locale =en_us;`
`data test;`
`x = input("JANUARY",nlstrmon.);`
`put x = ;`
`x = input("FEB",nlstrmon.);`
`put x = ;`
`x = input("march",nlstrmon.);`
`put x = ;`
`x = input("apr",nlstrmon.);`
`put x = ;`
`run;`
	`x=1;`
	`x=2;`
	`x=3;`
	`x=4;`

NLTIMAPw. Informat

Reads the time value and uses a.m. and p.m. as the local SAS time value that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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:

- “NLTIMAPw. Format” on page 499

NLTIMEw. Informat

Reads the time value as the local SAS time value that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alias: NLTIMAP

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

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.);`	`16:24:43`

Statements	Results
`put y time.;`	

`options locale=German_Germany;`
`y=input('16.24 Uhr',nltime.);`
`put y time.;`
	`16:24:00`

See Also

Format:

- “NLTIMEw. Format” on page 500

$REVERJw. Informat

Reads character data from right to left and preserves blanks.

Categories: CAS
Character

Syntax

$REVERJ*w.*

Syntax Description

w
specifies the width of the input field.

Default 1 if *w* is not specified

Range 1–32767

Comparisons

The $REVERJ*w*. informat is similar to the $REVERS*w*. informat except that $REVERS*w*. informat left aligns the result by removing all leading blanks.

Example

The following example uses ABCD as the input value.

```
input @1 name $reverj7.;
```

Values	Results
	----+----1
ABCD	###DCBA
ABCD	DCBA### 1

1 The character # represents a blank space.

See Also

Informat:

- "$REVERSw. Informat" on page 914

$REVERSw. Informat

Reads character data from right to left, and then left aligns the text.

Categories: CAS
Character

Syntax

$REVERS*w.*

Syntax Description

w
specifies the width of the input field.

Default 1 if *w* is not specified

Range 1–32767

Comparisons

The $REVERS*w*. informat is similar to the $REVERJ*w*. informat except that $REVERJ*w*. informat preserves all leading and trailing blanks.

Example

The following example uses ABCD as the input value.

```
input @1 name $revers7.;
```

Values	Results
	----+----1
ABCD	DCBA###
ABCD	DCBA### 1

1 The # character represents a blank space.

See Also

Informat:

- "$REVERJw. Informat" on page 913

$UCS2Bw. 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: This informat is not supported in a DATA step that runs in CAS.

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 Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+`
`x=input('5927'x,$ucs2b.);` `put x=$hex4.;`	`x=91e5`

See Also

Formats:

- “$UCS2Bw. Format” on page 507
- “$UCS2Lw. Format” on page 510
- “$UCS2Xw. Format” on page 513
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Lw. Informat” on page 918
- “$UCS2Xw. Informat” on page 921
- “$UTF8Xw. Informat” on page 938

$UCS2BEw. 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: This informat is not supported in a DATA step that runs in CAS.

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 1–32767

Comparisons

The $UCS2BE*w*. informat performs processing that is opposite of the $UCS2B*w*. informat.

Example

This example uses the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+`
`ucs2str=input ('木', $ucs2be2.);` `put ucs2str=$hex4.;`	`ucs2str=2020`

See Also

Formats:

- “$UCS2Bw. Format” on page 507
- “$UCS2BEw. Format” on page 508

Informat:

- “$UCS2Bw. Informat” on page 915

$UCS2Lw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+`
`x=input('2759'x,$ucs2l.);` `put x=$hex4.;`	`x=91e5`

See Also

Formats:

- “$UCS2Bw. Format” on page 507
- “$UCS2Lw. Format” on page 510
- “$UCS2Xw. Format” on page 513
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Xw. Informat” on page 921
- “$UTF8Xw. Informat” on page 938

$UCS2LEw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+`
`ucs2str=input ('大', $ ucs2le2.);` `put ucs2str=$hex4;`	`ucs2str=2759`

See Also

Formats:

- “$UCS2Lw. Format” on page 510
- “$UCS2LEw. Format” on page 511

Informat:

- “$UCS2Lw. Informat” on page 918

$UCS2Xw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment. This example uses little-endian formatting.

Statements	Result
	`----+----1----+`
`x=input('5927'x,$ucs2x.);` `put x=$hex4.;`	`x=91e5`

See Also

Formats:

- “$UCS2Bw. Format” on page 507
- “$UCS2Lw. Format” on page 510
- “$UCS2Xw. Format” on page 513
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Lw. Informat” on page 918
- “$UTF8Xw. Informat” on page 938

$UCS2XEw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+`
`ucs2str=input (' 木', $ ucs2xe2.);` `put ucs2str=$hex6;`	`ucs2str=5927`

See Also

Formats:

- "$UCS2Xw. Format" on page 513
- "$UCS2XEw. Format" on page 514

Informat:

- "$UCS2Xw. Informat" on page 921

$UCS4Bw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+`
`ucs4str=input(' 大', $ucs4be8.);`	`ucs4str=0000002000005927`

See Also

Format:

- “$UCS4Bw. Format” on page 516

Informats:

- “$UCS4Lw. Informat” on page 924
- “$UCS4Xw. Informat” on page 926

$UCS4Lw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+----2----+----3----+`
`z=put('.com',$UCS4L16.);` `put z $hex32.;`	`2E000000630000006F0000006D000000`

See Also

Format:

- "$UCS4Lw. Format" on page 519

Informats:

- "$UCS4Bw. Informat" on page 923
- "$UCS4Xw. Informat" on page 926

$UCS4Xw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment. This example uses little-endian formatting.

Statements	Results
	`----+----1----+`
`ucs4=put('91e5'x,$ucs4x.);`	`ucs4=27590000`

Statements	Results

`sjis=input(ucs4,$ucs4x.);`
`put ucs4=$hex8. sjis=$hex8.;`
`run;`
	`sjis=91E52020`

See Also

Formats:

- “$UCS2Xw. Format” on page 513
- “$UCS2Bw. Format” on page 507
- “$UCS2Lw. Format” on page 510
- “$UCS4Xw. Format” on page 522
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Lw. Informat” on page 918
- “$UTF8Xw. Informat” on page 938

$UCS4XEw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+`
`ucs4str=input (' 大', $ucs4xe2.);` `put ucs4str=$hex8;`	`ucs4str=00005927`

See Also

Formats:

- “$UCS4Xw. Format” on page 522
- “$UCS4XEw. Format” on page 524

Informat:

- “$UCS4Xw. Informat” on page 926

$UESCw. 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: This informat is not supported in a DATA step that runs in CAS.

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 Japanese Shift_JIS 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:

- “$UESCw. Format” on page 525
- “$UESCEw. Format” on page 527

Informat:

- “$UESCEw. Informat” on page 930

$UESCEw. 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: This informat is not supported in a DATA step that runs in CAS.

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', $uesc10.);`
`y=input('¥u5927',$uesc10.);`
`z=input('¥uu5927',$uesc10.);`
`put x y z;`
	`¥u5927`
	`¥uu5927`
	`¥uuu5927`

See Also

Formats:

- "$UESCw. Format" on page 525
- "$UESCEw. Format" on page 527

Informat:

- "$UESCw. Informat" on page 928

$UNCRw. Informat

Reads an NCR character string, and then converts the character string to the encoding of the current SAS session.

Category: Character

Restriction: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating system.

Statements	Result
	`----+----1----+`

`x=input ('大', $uncr10.);`
`y=input('abc', $uncr10.);`
`put X;`
`put Y;`
	大
	abc

See Also

Formats:

- “$UNCRw. Format” on page 528
- “$UNCREw. Format” on page 529

Informat:

- “$UNCREw. Informat” on page 933

$UNCREw. Informat

Reads a character string in the encoding of the current SAS session, and then converts the character string to NCR.

Category: Character

Restriction: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating system.

Statements	Result
	`----+----1----+`

Statements	Result
`x=input ('丈abc', $uncre12.);` `put x;`	`大abc`

See Also

Formats:

- “$UNCRw. Format” on page 528
- “$UNCREw. Format” on page 529

Informat:

- “$UNCRw. Informat” on page 931

$UPARENw. 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: This informat is not supported in a DATA step that runs in CAS.

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 $UPARENw. informat performs processing that is opposite of the $UPARENEw. informat.

Example

These examples use the Japanese Shift_JIS encoding, which is supported under the LINUX operating system.

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:

- “$UPARENw. Format” on page 531
- “$UPARENEw. Format” on page 532

Informats:

- “$UPARENEw. Informat” on page 936
- “$UPARENPw. Informat” on page 937

$UPARENEw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating system.

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

`v=input('a',$uparen10.);`
`w=input('b',$uparene10.);`
`x=input('c',$uparene10.);`
`y=input('3',$uparene10.);`
`z=input('木',$uparen10.);`
`put v;`
`put w;`
	`<u0061>`
	`<u0062>`
	`<u0063>`
	`<u0033>`
	`<u5927>`

Statements	Results
put x; put y; put z;	

See Also

Formats:

- “$UPARENw. Format” on page 531
- “$UPARENEw. Format” on page 532

Informats:

- “$UPARENw. Informat” on page 934
- “$UPARENPw. Informat” on page 937

$UPARENPw. 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: This informat is not supported in a DATA step that runs in CAS.

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 the Japanese Shift_JIS encoding, which is supported under the LINUX operating system.

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

`v=input('<u0061>',$uparen10.);`
`w=input('<u0062>',$uparenp10.);`
`x=input('<u0063>',$uparenp10.);`
`y=input('<u0033>',$uparenp10.);`
`z=input('<u5927>',$uparepn10.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	`a`
	`b`
	`c`
	`3`
	`<u5927>`

See Also

Formats:

- “$UPARENw. Format” on page 531
- “$UPARENEw. Format” on page 532

Informats:

- “$UPARENw. Informat” on page 934
- “$UPARENEw. Informat” on page 936

$UTF8Xw. 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

Restriction: This informat is not supported in a DATA step that runs in CAS.

Syntax

$UTF8X*w.*

Syntax Description

w

specifies the width of the input field.

Default 8

Range 1–32767

Comparisons

This example uses the Japanese Shift_JIS encoding, which is supported under the LINUX operating environment.

Statements	Result
	`----+----1----+`
`x=input (' e5a4a7' x, $utf8x3.);` `put x;`	大

See Also

Formats:

- “$UCS2Bw. Format” on page 507
- “$UCS2Lw. Format” on page 510
- “$UCS2Xw. Format” on page 513
- “$UTF8Xw. Format” on page 533

Informats:

- “$UCS2Bw. Informat” on page 915
- “$UCS2Lw. Informat” on page 918
- “$UCS2Xw. Informat” on page 921

$VSLOGw. Informat

Reads a character string that is in visual order, and then converts the character string to left-to-right logical order.

Categories: BIDI Text Handling
CAS

Syntax

$VSLOG*w*.

Syntax Description

w
specifies the width of the input field.

Default 200

Range 1–32000

Comparisons

The $VSLOG*w*. informat performs processing that is opposite of the $VSLOGR*w*. informat.

Example

The following example uses the Hebrew input value of "טיסה flight".

Statements	Result
	`----+----1----+`
x=input ('טיסה flight',$vslog12.); `put x;`	טיסה `flight`

The following example uses the Arabic input value of "كمبيوتر computer."

Statements	Result
	----+----1----+
x=input (' ذاتcomputer',$vslog12.); put x;	ذاتcomputer

See Also

Formats:

- "$VSLOGRw. Format" on page 537
- "$VSLOGw. Format" on page 536

Informat:

- "$VSLOGRw. Informat" on page 941

$VSLOGRw. Informat

Reads a character string that is in visual order, and then converts the character string to right-to-left logical order.

Categories: BIDI Text Handling
CAS

Syntax

$VSLOGR*w*.

Syntax Description

w
specifies the width of the input field.

Default 200

Range 1–32000

Comparisons

The $VSLOGR*w*. informat performs processing that is opposite of the $VSLOG*w*. informat.

Example

The following example uses the Hebrew input value of "טיסה flight."

Statements	Result
	`----+----1----+`
x=input ('טיסה',$vslogr12.); `put x;`	`flight` טיסה

The following example uses the Arabic input value of "كمبيوتر computer."

Statements	Result
	`----+----1----+`
x=input ('كمبيوتر computer',$vslogr12.); `put x;`	كمبيوتر computer

See Also

Formats:

- "$VSLOGw. Format" on page 536
- "$VSLOGRw. Format" on page 537

Informat:

- "$VSLOGw. Informat" on page 940

YENw.d Informat

Removes embedded yen signs, commas, and decimal points.

Categories: CAS
Numeric

Restriction: The YEN informat does not support UTF8 encoding. See the NLMNLJPY format

Note: Beginning in 2025.12, this informat supports the BIGINT data type when run in the DS2 language. Set BIGINTPROCESSING=YES to ensure that large integer values—stored as BIGINT—are displayed with full precision, up to 19 digits, without rounding or truncation. BIGINTPROCESSING=YES is recommended for large values that exceed 15–16 digits.

Syntax

YEN*w.d*

Syntax Description

w
specifies the width of the input field.

Default 1

Range 1–32

d
specifies the power of 10 by which to divide the value.

Requirement *d* must be 0 or 2

Tip If the *d* is 2, then YEN*w.d* reads a decimal point and two decimal digits. If *d* is 0, YEN*w.d* reads the value without a decimal part.

Details

The hexadecimal representation of the code for the yen sign character is 5B on EBCDIC systems and 5C on ASCII systems. The monetary character that these codes represent might be different in other countries.

Example

The following example uses yen as the input.

```
input value yen10.2;
```

Value	Result
	`----+----1----+`
`¥1254.71`	`1254.71`

See Also

Format:

- “YENw.d Format” on page 545

PART 7

Macro Functions for NLS

Chapter 15
Dictionary of Macro Functions for NLS **947**

15

Dictionary of Macro Functions for NLS

Macro Functions by Category **947**

Dictionary **948**
%KCMPRES Function 948
%KINDEX Macro Function 949
%KLEFT Macro Function 950
%KLENGTH Macro Function 951
%KSCAN Macro Function 952
%KSUBSTR Macro Function 954
%KUPCASE Macro Function 956
%QKCMPRES Function 957
%QKLEFT Macro Function 958
%QKSCAN Macro Function 959
%QKSUBSTR Macro Function 961
%QKUPCASE Macro Function 963
%VALIDCHS Macro Function 964

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 Function (p. 948)	Compresses multiple blanks and removes leading and trailing blanks.
	%KINDEX Macro Function (p. 949)	Returns the position of the first character of a string.

Category	Language Elements	Description
	%KLEFT Macro Function (p. 950)	Left-aligns an argument by removing leading blanks.
	%KLENGTH Macro Function (p. 951)	Returns the length of a string.
	%KSCAN Macro Function (p. 952)	Search for a word that is specified by its position in a string.
	%KSUBSTR Macro Function (p. 954)	Produce a substring of a character string.
	%KUPCASE Macro Function (p. 956)	Convert values to uppercase.
	%QKCMPRES Function (p. 957)	Compresses multiple blanks and removes leading and trailing blanks.
	%QKLEFT Macro Function (p. 958)	Left-aligns an argument by removing leading blanks.
	%QKSCAN Macro Function (p. 959)	Search for a word that is specified by its position in a string.
	%QKSUBSTR Macro Function (p. 961)	Produce a substring of a character string.
	%QKUPCASE Macro Function (p. 963)	Convert values to uppercase.

Dictionary

%KCMPRES Function

Compresses multiple blanks and removes leading and trailing blanks.

Category: DBCS

Type: NLS macro function

See: "%QKCMPRES Function" on page 957

Syntax

%KCMPRES (*text* | *text expression*)

Required Argument

text
text or 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. See "%QKCMPRES Function" on page 957

```
& % ' " ( ) + - * / < > = ^ ~ ; , # 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 Macro Function

Left-aligns an argument by removing leading blanks.

Category: DBCS

Requirement: MAUTOSOURCE system option

See: "%QKLEFT Macro Function"

Syntax

%KLEFT (*text* | *text expression*)

Details

The %KLEFT macro function left-aligns arguments by removing leading blanks.

%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 Macro Function

Search for a word that is specified by its position in a string.

Category: DBCS

Type: NLS macro function

See: "%QKSCAN Macro Function"

Syntax

%KSCAN (*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 Macro Function

Produce a substring of a character string.

Category: DBCS

Type: NLS macro function

Syntax

%KSUBSTR (*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.

Optional Argument

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 Macro Function

Convert values to uppercase.

Category: DBCS

Type: NLS macro function

See: "%QKUPCASE Macro Function"

Syntax

%KUPCASE (*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
```

%QKCMPRES Function

Compresses multiple blanks and removes leading and trailing blanks.

Category: DBCS

Type: NLS macro function

See: "%KCMPRES Function"

Syntax

%QKCMPRES (*text* | *text expression*)

Required Argument

text
text or 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. See "%QKCMPRES Function" on page 957

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

%QKLEFT Macro Function

Left-aligns an argument by removing leading blanks.

Category: DBCS

Requirement: MAUTOSOURCE system option

See: "%KLEFT Macro Function"

Syntax

%QKLEFT (*text* | *text expression*)

Details

The %QKLEFT macro function left-aligns arguments by removing leading blanks. If the argument contains a special character or mnemonic operator, listed here, use QKLEFT.

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

Comparisons

%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

%QKSCAN Macro Function

Search for a word that is specified by its position in a string.

Category: DBCS

Type: NLS macro function

See: "%KSCAN Macro Function"

Syntax

%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
```

%QKSUBSTR Macro Function

Produce a substring of a character string.

Category: DBCS

Type: NLS macro function

Syntax

%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.

Optional Argument

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
```

%QKUPCASE Macro Function

Convert values to uppercase.

Category: DBCS

Type: NLS macro function

See: "%KUPCASE Macro Function"

Syntax

%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

When these statements execute, the following is written to the SAS log:

```
KUPCASE produces: BEGIN
QKUPCASE produces: &A
```

%VALIDCHS Macro Function

Validates the character(s) encoding compatibility for data set variables.

Restriction: This macro should not be used when the encoding value equals DEFAULT.

Syntax

%VALIDCHS(*dsnm=dataset_name*,<
libnm=libref_name,><*ENCODING=encoding_name*,><*COMPATIBLE=encoding_name*>
)

Required Argument

dsnm=<*libref*.>dataset_name
name of the data set to be verified.

Optional Arguments

ENCODING=encoding_name
optional encoding name that the %VALIDCHS macro function uses to validate that the characters in the data set are valid for the specified encoding. If the ENCODING= option is not specified, the encoding of the data set is used.

libnm=libref_name
name of the SAS library to be verified.

COMPATIBLE=encoding_name
optional encoding that the %VALIDCHS macro function uses to verify the compatibility of the data in the data set with the specified encoding. It helps determine whether you can successfully transfer the data set between a server and a client where the encoding is different. Incompatible characters can cause data transfer issues.

Details

Basics

The %VALIDCHS macro function scans the data set and inspects the characters in the data set's character variables to determine whether they are valid for the specified encoding. If an ENCODING= option is specified, the %VALIDCHS macro function validates based on the ENCODING= value. If the ENCODING= option is not specified, then %VALIDCHS uses the encoding of the data set. If the COMPATIBLE= option is specified, %VALIDCHS validates the contents of the character data in the encoding of the data set (or the encoding of the ENCODING= option). Then it verifies the compatibility based on the encoding that is specified by the COMPATIBLE= option. For example, a data set with a euc-cn encoding, Chinese characters, and `COMPATIBLE=wlatin1` is specified. %VALIDCHS verifies that the characters are valid for the euc-cn encoding. Then %VALIDCHS verifies the compatibility of the characters to see whether they are supported by the wlatin1 encoding. If %VALIDCHS detects invalid character(s) in the data set, it displays a message and continues validation.

Automatic Macro Variable

_validchars_rc is an automatic macro variable that is generated by %VALIDCHS. It indicates the test status. Here is an example:

```
2:  No invalid characters are found but data will cause the truncation
        1:  No invalid characters are found
        0:  Invalid characters are found
       -1:  Invalid characters are found and also data will cause the truncation
       -2:  General error
```

Examples

Example 1: Simple Validation of the Character Data in the Data Set

%VALIDCHS uses the encoding of the input data set.

```
%VALIDCHS(dsnm=sashelp.class);
%put &_validchars_rc;
```

The encoding of the data set is us-ascii. The %VALIDCHS macro function finds that all the character data is valid in us-ascii so that it returns following message in the log:

```
NOTE: All characters in the dataset: sashelp.class (encoding:us-ascii) are valid.
0
```

Example 2: Validate the Character Data in theData Set in the Specified Encoding

%VALIDCHS uses the specified encoding to validate these characters:

```
%VALIDCHS(dsnm=sashelp.class, encoding=utf-8);
%put &_validchars_rc;
```

These messages are written to the SAS log:

```
NOTE: All characters in the dataset: sashelp.class (encoding:utf-8) are valid.
0
```

Example 3: Validate the Character Data and Check the Compatibility

```
%VALIDCHS(dsnm=sashelp.class, encoding=euc-cn, compatible=wlatin1);
%put &_validchars_rc;
```

These messages are written to the SAS log:

```
NOTE: All characters in the dataset: sashelp.class (encoding:euc-cn) are
compatible to wlatin1 encoding.
0
```

Example 4: Validate the Character Data and Check the Compatibility

The following data set has the wlatin1 encoding and a national character:

```
data mydata;
   length ch $1.;
   ch = '€';
run;

%VALIDCHS(dsnm=mydata, encoding=wlatin1, compatible=utf-8);
```

```
%put &_validchars_rc;
```

These messages are written to the SAS log:

```
WARNING: Found possible truncation in the variable: ch at observation number: 1
ERROR: Detected the issue in the dataset: mydata (encoding:wlatin1) when
validating the compatibility to ::utf-8 encoding.
2
```

PART 8

System Options for NLS

Chapter 16
Dictionary of System Options for NLS **971**

16

Dictionary of System Options for NLS

System Option Entries by Category **972**

Dictionary **973**
BOMFILE System Option 973
DATESTYLE= System Option 974
DBCS System Option 976
DBCSLANG System Option 977
DBCSTYPE System Option 978
DFLANG= System Option 980
ENCODING System Option 982
FSDBTYPE System Option 984
FSIMM System Option 985
FSIMMOPT System Option 986
LOCALE System Option 987
LOCALEDATA System Option 989
LOGLANGCHG System Option 989
LOGLANGENG System Option 991
LSWLANG System Option 992
MAPEBCDIC2ASCII= System Option 994
NLDECSEPARATOR System Option 994
ODSLANGCHG System Option 996
RSASIOTRANSERROR System Option 998
SORTSEQ= System Option 999
TIMEZONE= System Option 1000
TRANTAB= System Option 1002
VALIDMEMNAME= System Option 1004
VALIDVARNAME= System Option 1007
URLENCODING= System Option 1009

System Option Entries by Category

The language control category of SAS system options are 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
CAS	DATESTYLE= System Option (p. 974)	Specifies the sequence of month, day, and year when ANYDTDTE, ANYDTDTM, or ANYDTTME informat data is ambiguous.
Environment Control: Language Control	DATESTYLE= System Option (p. 974)	Specifies the sequence of month, day, and year when ANYDTDTE, ANYDTDTM, or ANYDTTME informat data is ambiguous.
	DBCS System Option (p. 976)	Recognizes double-byte character sets (DBCS).
	DBCSLANG System Option (p. 977)	Specifies a double-byte character set (DBCS) language.
	DBCSTYPE System Option (p. 978)	Specifies the encoding method to use for a double-byte character set (DBCS).
	DFLANG= System Option (p. 980)	Specifies the language for international date informats and formats.
	ENCODING System Option (p. 982)	Specifies the default character-set encoding for the SAS session.
	FSDBTYPE System Option (p. 984)	Specifies a full-screen double-byte character set (DBCS) encoding method.
	FSIMM System Option (p. 985)	Specifies input method modules (IMMs) for full-screen double-byte character set (DBCS).
	FSIMMOPT System Option (p. 986)	Specifies options for input method modules (IMMs) that are used with a full-screen double-byte character set (DBCS).
	LOCALE System Option (p. 987)	Specifies a set of attributes in a SAS session that reflect the language, local conventions, and culture for a geographical region.
	LOCALEDATA System Option (p. 989)	Specifies the source database for the locale information.
	LOGLANGENG System Option (p. 991)	Specifies using the English language for SAS log message text when the LOCALE option is set after start-up.

Category	Language Elements	Description
	MAPEBCDIC2ASCII= System Option (p. 994)	Specifies a translation table that SAS uses to transcode from EBCDIC to ASCII and from ASCII to EBCDIC.
	ODSLANGCHG System Option (p. 996)	Determines whether the language of the text of the ODS output can be changed
	TIMEZONE= System Option (p. 1000)	Specifies the user local time zone.
	TRANTAB= System Option (p. 1002)	Specifies the translation tables that are used by various parts of SAS.
	URLENCODING= System Option (p. 1009)	Specifies whether the argument to the URLENCODE function and to the URLDECODE function is interpreted using the SAS session encoding or UTF-8 encoding.
Files: External Files	BOMFILE System Option (p. 973)	Specifies whether to write the byte-order mark (BOM) prefix on Unicode-encoded external files.
Files: SAS Files	RSASIOTRANSERROR System Option (p. 998)	Displays a transcoding error when invalid data is read from a remote application.
	VALIDMEMNAME= System Option (p. 1004)	Specifies the rules for naming SAS data sets, SAS data views, and item stores.
	VALIDVARNAME= System Option (p. 1007)	Specifies the rules for valid SAS variable names that can be created and processed during a SAS session.
Input Control: Data Processing	DATESTYLE= System Option (p. 974)	Specifies the sequence of month, day, and year when ANYDTDTE, ANYDTDTM, or ANYDTTME informat data is ambiguous.
Language Control	LSWLANG System Option (p. 992)	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. 999)	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

Category: Files: External Files

PROC OPTIONS GROUP= EXTFILES

Important: The BOMFILE option's functionaliy applies when wriing to a physical external file.

The BOMFILE option might not be supported for files that are written as members of container formats, for example ZIP files.

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.

DATESTYLE= System Option

Specifies the sequence of month, day, and year when ANYDTDTE, ANYDTDTM, or ANYDTTME informat data is ambiguous.

Valid in: Autoexec file, OPTIONS statement,

Categories: Environment Control: Language Control
Input Control: Data Processing
CAS

PROC OPTIONS GROUP= INPUTCONTROL
LANGUAGECONTROL

Default: The default value is determined by the value of the LOCALE= system option.

Interaction: The value of the DATESTYLE= system option is used for new CAS sessions. The DATESTYLE= session option overrides the value of the DATESTYLE= system option within a CAS session. The DATESTYLE= session option is set by using the SESSOPTS= option in the CAS statement or by using the sesssionProp.setSessOpt action in PROC CAS.

Note: This option cannot be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

See:

Syntax

DATESTYLE= MDY | YMD | DMY | LOCALE

Syntax Description

MDY
specifies that SAS set the order as month, day, year.

YMD
specifies that SAS set the order as year, month, day.

DMY
specifies that SAS set the order as day, month, year.

LOCALE
specifies that SAS set the order based on the value that corresponds to the LOCALE= system option value and is one of the following: MDY | YMD | DMY.

Details

System option DATESTYLE= identifies the order of month, day, and year. The default value is LOCALE. The default LOCALE system option value is English. Therefore, the default DATESTYLE order is MDY.

To get the default settings for each locale option value, see Locale Values.

See Also

Informats:

- "ANYDTDTEw. Informat" in *SAS Formats and Informats: Reference*
- "ANYDTDTMw. Informat" in *SAS Formats and Informats: Reference*
- "ANYDTTMEw. Informat" in *SAS Formats and Informats: Reference*

System Options:

- "LOCALE System Option" on page 987

Session Option:

- "DATESTYLE= Session Option" in *SAS Cloud Analytic Services: User's Guide*

DBCS System Option

Recognizes double-byte character sets (DBCS).

Valid in:	configuration file, SAS invocation
Category:	Environment Control: Language Control
PROC OPTIONS GROUP=	LANGUAGECONTROL
Default:	NODBCS

Syntax

-DBCS

-NODBCS

Required Arguments

DBCS
recognizes double-byte character sets (DBCS) for encoding values. DBCS encodings are used to support East Asian languages.

NODBCS
does not recognize a DBCS for encoding values. Instead, a single-byte character set (SBCS) is used for encoding values. A single byte is used to represent each character in the character set.

Details

The DBCS system option is used for supporting languages from East Asian countries such as Chinese, Japanese, Korean, and Taiwanese.

See Also

Conceptual Information:

- Chapter 5, “Double-Byte Character Sets (DBCS),” on page 51
- “DBCS Values for a SAS Session” on page 1125
- Chapter 23, “Encoding Values in SAS Language Elements,” on page 1127

System Options:

- “DBCSLANG System Option” on page 977
- “DBCSTYPE System Option” on page 978

DBCSLANG System Option

Specifies a double-byte character set (DBCS) language.

Valid in:	configuration file, SAS invocation
Category:	Environment Control: Language Control
PROC OPTIONS GROUP=	LANGUAGECONTROL
Default:	none

Syntax

-DBCSLANG *language*

Required Argument

language
depends on the operating environment. The following table contains valid language values:

Table 16.1 *Supported DBCS Languages According to Operating Environment*

Language	LINUX
CHINESE (simplified)	yes
JAPANESE	yes
KOREAN	yes
TAIWANESE (traditional)	yes
NONE	no

Language	LINUX
UNKNOWN	no

Details

The proper setting for the DBCSLANG system option depends on which setting is used for the DBCSTYPE system option. Some of the settings of DBCSTYPE support all of the DBCSLANG languages. Other settings of DBCSTYPE support only Japanese.

CHINESE specifies the language used in the People's Republic of China, which is known as simplified Chinese. TAIWANESE specifies the Chinese language used in Taiwan, which is known as traditional Chinese.

See Also

- Chapter 5, "Double-Byte Character Sets (DBCS)," on page 51
- "DBCS Values for a SAS Session" on page 1125
- Chapter 23, "Encoding Values in SAS Language Elements," on page 1127

System Options:

- "DBCS System Option" on page 976
- "DBCSTYPE System Option" on page 978

DBCSTYPE System Option

Specifies the encoding method to use for a double-byte character set (DBCS).

Valid in: configuration file, SAS invocation

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Syntax

-DBCSTYPE *encoding-method*

Required Argument

encoding-method
specifies the method that is used to encode a double-byte character set (DBCS). Valid values for *encoding-method* depend on the standard that the computer hardware manufacturer applies to the operating environment.

Details

DBCS encoding methods vary according to the computer hardware manufacturer and the standards organization.

The DBCSLANG= system option specifies the language that the encoding method is applied to. You should specify DBCSTYPE= only if you also specify the DBCS and DBCSLANG= system options.

Comparisons

Table 16.2 *DBCS Encoding Methods for LINUX*

DBCSTYPE= Value	Description
DEC	DEC encoding method
EUC	Extended LINUX Code encoding method
HP15	Hewlett Packard encoding method
PCIBM	IBM PC encoding method
PCMS	Microsoft PC encoding method
SJIS	Shift-JIS encoding method for the Japanese language only
NONE	Disables DBCS processing

See Also

Conceptual Information:

- Chapter 5, “Double-Byte Character Sets (DBCS),” on page 51
- “DBCS Values for a SAS Session” on page 1125
- Chapter 23, “Encoding Values in SAS Language Elements,” on page 1127

System Options:

- “DBCS System Option” on page 976
- “DBCSLANG System Option” on page 977

DFLANG= System Option

Specifies the language for international date informats and formats.

Valid in:	SAS Environment Manager, Autoexec file, OPTIONS statement, SASV9_OPTIONS environment variable
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.

- 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. The value **`LOCALE`** is also supported.

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= ASCIIANY *encoding-value*

ENCODING= *encoding-value*

Required Arguments

ASCIIANY

Transcoding normally occurs when SAS detects that the session encoding and data set encoding are different. ASCIIANY enables you to create a data set that SAS will not transcode if the SAS session that accesses the data set has a session that has an encoding value of ASCII.

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

encoding-value

For valid values for all operating environments, see Chapter 24, "Encoding Values for a SAS Session," on page 1139.

Details

A character-set encoding is a set of characters that have been mapped to numeric values called code points.

The encoding for a SAS session is determined by the values of the ENCODING=, LOCALE=, DBCSTYPE=, and DBCSLANG= system options as follows:

- If the ENCODING option is not specified, the ENCODING value is determined by the value of LOCALE and the operating system where SAS is running. Also, if LOCALE is not set, the default LOCALE is en_US.
- If both LOCALE= and ENCODING= are specified, the session encoding is the value that is specified by the ENCODING= option.
- If LOCALE= is specified and ENCODING= is not specified, SAS determines the appropriate encoding value from the LOCALE= value.
- If the DBCS option is set, the values for the DBCSLANG= and DBCSTYPE= system options determine the ENCODING= and LOCALE= values.

See Also

Conceptual Information:

- "Overview of Locale Concepts for NLS"
- Conceptual discussion about "Overview: Encoding for NLS"
- Conceptual discussion about "Overview of Transcoding"
- Table 21.1
- Chapter 22, "SAS System Options for Processing DBCS Data"
- Chapter 23, "Encoding Values in SAS Language Elements"

FSDBTYPE System Option

Specifies a full-screen double-byte character set (DBCS) encoding method.

Valid in: configuration file, SAS invocation, SASV9_OPTIONS environment variable

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: DEFAULT

Syntax

-FSDBTYPE *encoding-method*

Details

The FSDBTYPE= system option specifies the encoding method that is appropriate for a full-screen DBCS enabling method. Full-screen DBCS encoding methods vary according to the computer hardware manufacturer and the standards organization.

Table 16.3 *Full-Screen DBCS Encoding Methods*

FSDBTYPE= Encoding Method	Description
dec	Digital Equipment Corporation encoding method
euc	Extended LINUX encoding method
hp15	HP-UX encoding method
jis7	7-bit Shift-JIS encoding method used in an X windows environment for the Japanese language only
pcibm	IBM PC encoding method
sjis	Shift-JIS encoding method for the Japanese language only

FSDBTYPE= Encoding Method	Description
default	default method that is used by the specific host

See Also

Conceptual Information:

- Chapter 5, “Double-Byte Character Sets (DBCS),” on page 51
- “DBCS Values for a SAS Session” on page 1125
- Chapter 23, “Encoding Values in SAS Language Elements,” on page 1127

FSIMM System Option

Specifies input method modules (IMMs) for full-screen double-byte character set (DBCS).

Valid in: configuration file, SAS invocation, SASV9_OPTIONS environment variable

Category: Environment Control: Language Control

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: none

Syntax

-FSIMM *fsdevice_name=IMM-name1*<, *fsdevice_name=IMM-name2*> ...

Details

You can specify the following values for *IMM-name*:

TTY | SASWUJT
: provides an interface for **`/dev/tty`**. This IMM enables you to enter DBCS strings through a terminal emulator that has DBCS input capability.

PIPE | SASWUJP
: provides a pipe interface. This interface forks the DBCS input server process. The default server name is `saswujms`, which uses the vendor-supplied MOTIF toolkit.

For example, to use the PIPE input method module for X11 drivers, you would specify this code:

```
-FSIMM X11=PIPE
```

Note: The server is specified by using the FSIMMOPT option.

See Also

Conceptual Information:

- Chapter 5, “Double-Byte Character Sets (DBCS),” on page 51

System Option:

- “FSIMMOPT System Option” on page 986

FSIMMOPT System Option

Specifies options for input method modules (IMMs) that are used with a full-screen double-byte character set (DBCS).

Valid in:	configuration file, SAS invocation, SASV9_OPTIONS environment variable
Category:	Environment Control: Language Control
PROC OPTIONS GROUP=	LANGUAGECONTROL
Default:	none

Syntax

-FSIMMOPT *fullscreen-IMM*:*IMM-option*

Details

The FSIMMOPT system option specifies an option for each full-screen IMM (input method module). You can specify only one FSIMMOPT option for each IMM. If you specify multiple FSIMMOPT options for the same IMM, only the last specification is used.

For option values for each IMM, see SAS Technical Report J-121, *DBCS Support Usage Guide* (in Japanese).

For example, you can use the FSIMMOPT option to specify the name of the server, MOTIF, to be used for the PIPE IMM:

```
-fsimmopt PIPE:MOTIF
```

See Also

Conceptual Information:

- Chapter 5, "Double-Byte Character Sets (DBCS)," on page 51

System Option:

- "FSIMM System Option" on page 985

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
Category:	Environment Control: Language Control
PROC OPTIONS GROUP=	LANGUAGECONTROL
Default:	English_UnitedStates

Syntax

LOCALE=*locale-name*

Required Argument

locale-name
For a complete list of locale values (SAS names and POSIX names), see Table 21.1.

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.

If the DBCS= system option is active, the values of the DBCSTYPE= and DBCSLANG= system options determine the locale and character-set encoding.

When you set a value for LOCALE=, the value of the following system options is modified unless explicit values have been specified:

ENCODING=
: The locale that you set has a common encoding value that is used most often in the operating environment where SAS runs. If the ENCODING= option is not set explicitly in a config file or on the command line, SAS uses the ENCODING that is default for the LOCALE and operating system. The LOCALE might be set explicitly or can default. When the ENCODING= system option is set, the TRANTAB= system option is also set.

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
- Table 21.1

System Options:

- “ENCODING System Option” on page 982
- “DATESTYLE= System Option” on page 974
- “DFLANG= System Option” on page 980
- “PAPERSIZE= System Option” in *SAS System Options: Reference*
- “TRANTAB= System Option” on page 1002

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 SAS log can be changed.

Valid in: configuration file, SAS invocation

PROC OPTIONS GROUP= LOGCONTROL

Default: LOGLANGCHG is set to NOLOGLANGCHG (OFF) in all servers except for the UNICODE server.

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

LOGLANGCHG is a start-up option, and it is specified in the configuration file or the SAS start-up command line. Values cannot be changed after start-up.

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 996
- “LSWLANG System Option” on page 992
- “LOGLANGENG System Option” on page 991

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.

- Once LOGLANGENG is set, LOGLANGCHG and LSWLANG=EN will be set automatically.
- 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 LOGLANGENG is set to ON, LOGLANGCHG is automatically set to ON, and English language messages are written to SAS log.

- If LOGLANGENG is set to OFF and LOGLANGCHG is set to 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.

LOGLANGCHG has no effect if LOGLANGENG is set. 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 992
- “LOGLANGCHG System Option” on page 989
- “ODSLANGCHG System Option” on page 996

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
JA	Japanese
KO	Korean
PL	Polish
RU	Russian

See Also

- “ODSLANGCHG System Option” on page 996
- “LOGLANGCHG System Option” on page 989
- “LOGLANGENG System Option” on page 991

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

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: NONLDECSEPARATOR

Restriction: Using NLDECSEPARATOR with ODS GRAPHICS enabled-procedures generates a syntax error or null debug errors. Use an appropriate NL format for the numeric variables.

Syntax

NLDECSEPARATOR

NONLDECSEPARATOR

Required Arguments

NLDECSEPARATOR

Alias NLD

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

"BESTDOTXw. Format" on page 150

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 991
- “LOGLANGCHG System Option” on page 989
- “LSWLANG System Option” on page 992

RSASIOTRANSERROR System Option

Displays a transcoding error when invalid data is read from a remote application.

Valid in:	configuration file, SAS invocation, OPTIONS statement
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, 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
Category:	Sort: Procedure Options
PROC OPTIONS GROUP=	SORT

Syntax

SORTSEQ=<*sort-table*> | <LINGUISTIC>

Syntax Description

sort-table
specifies a translation table that you created with PROC TRANTAB.

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 *Base SAS Procedures Guide* for more information.

Example

This example demonstrates the functionality of the SORTSEQ= option 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 in the Base SAS procedure, PROC SORT. 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.

In SAS Viya platform, the ICU library version incorporated by SAS and used by PROC SORT is ICU 63. This ICU version uses locale data from version 34 of the Unicode Common Locale Data Repository (CLDR).

See Also

- "Collating Sequence" on page 14

System Options:

- "TRANTAB= System Option" on page 1002

TIMEZONE= System Option

Specifies the user local time zone.

Valid in:	Autoexec file, OPTIONS statement,
Category:	Environment Control: Language Control
PROC OPTIONS GROUP=	LANGUAGECONTROL
Alias:	TZ=
Restriction:	The TIMEZONE= system option does not affect in-database processing. The SAS Embedded Process uses the time zone of the data source. For more information, see "In-Database Processing" in *SAS In-Database Products: User's Guide*.
Notes:	The alias, TZ=, cannot be used in the restricted options configuration file. TIMEZONE= must be used in the restricted options configuration file. This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS 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 Compute Server time zone and the SAS 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 1153.

<->
specifies the time offset from Greenwich, England in the form <->*hh*<:*mm*<:*ss*>>:

- specifies the time zone offset west of Greenwich, England. Time zone offsets east of Greenwich do not require signed notation.

hh specifies an hour by using one or two digits. If you specify an offset, the hour is required.

mm specifies minutes by using one or two digits. Minutes are optional.

ss specifies seconds by using one or two digits. Seconds are optional.

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 1153.

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 Standard Time, Cuba Daylight Time, and China Standard 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 57

TRANTAB= System Option

Specifies the translation tables that are used by various parts of SAS.

Valid in:	configuration file, SAS invocation, OPTIONS statement
Category:	Environment Control: Language Control
PROC OPTIONS GROUP=	LANGUAGECONTROL
Default:	SAS Viya platform: The shipped default is (lat1wlt1,wlt1lat1,lat1_ucs,lat1_lcs,lat1_ccl,,,,,)
Interaction:	The TRANTAB= system option specifies a translation table to use for the SAS session, including file transfers. 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.

Syntax

TRANTAB=(*catalog-entries*)

Syntax Description

catalog-entries
specifies SAS catalog entries that contain translation tables. If you specify *entry-name.type*, SAS searches SASUSER.PROFILE first and then SASUSER.HOST.

Details

Translation tables are specified in a list that is enclosed in parentheses and has ten positions. The position in which a table appears in the list determines the type of translation table that is specified. Individual entries in the list are separated by commas. See the list of positions and types that follows:

Position	Type of Translation Table
1st	local-to-transport-format
2nd	transport-to-local-format
3rd	lowercase-to-uppercase
4th	uppercase-to-lowercase
5th	character classification
6th	scanner translation
7th	delta characters
8th	scanner character classification
9th	not used
10th	DBCS user table

CAUTION
Do not change a translation table unless you are familiar with its purpose.
Translation tables are used internally by the SAS supervisor to implement NLS. If you are unfamiliar with the purpose of translation tables, do *not* change the specifications without proper technical advice.

To change one table, specify null entries for the other tables. For example, to change the lowercase-to-uppercase table, which is third in the list, specify uppercase as follows:

```
options trantab = ( , , new-uppercase-table);
```

The other tables remain unchanged. The output from the OPTIONS procedure reflects the last specification for the TRANTAB= option and not the composite specification. Here is an example:

```
options trantab = ( , , new-uppercase-table);
options trantab = ( , , , new-lowercase-table);
```

PROC OPTIONS shows that the value for TRANTAB= is

(, , , *new-lowercase-table*), but both the *new-uppercase* and *new-lowercase* tables are in effect.

See Also

Chapter 20, "TRANTAB Procedure," on page 1079

VALIDMEMNAME= System Option

Specifies the rules for naming SAS data sets, SAS data views, and item stores.

Valid in:	Autoexec file, OPTIONS statement,
Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Default:	The shipped default is COMPATIBLE.
Applies to:	Base SAS engine and SPD Engine
Restrictions:	The VALIDMEMNAME= option is not supported by the tape engines V9TAPE, V8TAPE, V7TAPE, and V6TAPE. 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.
Note:	This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS 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.

  Note: The SPD Engine does not allow '.' (the period) anywhere in the member name.

- 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).

  Note: The SPD Engine does not allow '$' as the first character of the member name.

- 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, the name starts with a number, 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. For more information, see "SAS Name Literals" in *SAS Programmer's Guide: Essentials*.

Operating environments

For LINUX operating environments, all Base SAS windows support the extended rules when VALIDMEMNAME=EXTEND is set.

For LINUX operating environments, 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.

See

"Extending SAS Names" in *SAS Programmer's Guide: Essentials*

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

- "SAS Words" in *SAS Programmer's Guide: Essentials*

System Options:

- "VALIDVARNAME= System Option" on page 1007

VALIDVARNAME= System Option

Specifies the rules for valid SAS variable names that can be created and processed during a SAS session.

Valid in:	Autoexec file, OPTIONS statement,
Category:	Files: SAS Files
PROC OPTIONS GROUP=	SASFILES
Default:	The shipped default is V7.
Interaction:	In the context of SAS/ACCESS, VALIDVARNAME= controls how DBMS column names are read into SAS and converted to SAS variable names. In general, VALIDVARNAME= specifies the rules that must be used when you create any SAS variable.
Note:	This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Syntax

VALIDVARNAME=V7 | UPCASE | ANY

Syntax Description

V7
specifies that variable names must follow these rules:

- The name 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 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.

Requirement If you use any characters other than the ones that are valid when the VALIDVARNAME system option is set to V7 (letters of the Latin alphabet, numerals, or underscores), then you must express the variable name as a name literal and you must set VALIDVARNAME=ANY. If the name includes 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. See "SAS Name Literals" in *SAS Programmer's Guide: Essentials* and "Avoiding Errors When Using Name Literals" in *SAS Programmer's Guide: Essentials*.

See "Extending SAS Names" in *SAS Programmer's Guide: Essentials*

Examples '% of profit'n=percent;

'items@warehouse'n=itemnum;

CAUTION **Throughout SAS, using the name literal syntax with SAS variable 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

- "Rules for Most SAS Names" in *SAS Programmer's Guide: Essentials*

System Options:

- "VALIDMEMNAME= System Option" on page 1004

URLENCODING= System Option

Specifies whether the argument to the URLENCODE function and to the URLDECODE function is interpreted using the SAS session encoding or UTF-8 encoding.

Valid in:	SAS Environment Manager, Autoexec file, OPTIONS statement, SASV9_OPTIONS environment variable
Category:	Environment Control: Language Control
PROC OPTIONS GROUP=	LANGUAGECONTROL
See:	"URLENCODING= System Option" in *SAS System Options: Reference*

Syntax

URLENCODING=SESSION | UTF8

PART 9

Options for Commands, Statements, and Procedures for NLS

Chapter 17
Dictionary of Command, Statement, and Procedure Option for NLS . . . 1013

17

Dictionary of Command, Statement, and Procedure Option for NLS

Commands, Statements, and Procedures for NLS by Category 1013

Dictionary 1015
CHARSET= Option 1015
Collating Sequence Option 1016
CORRECTENCODING= Option 1023
CVPBYTES=, CVPENGINE=, CVPFORMATWIDTH=, CVPMULTIPLIER=,CVPVARCHAR=, CVPEXCLUDE=, CVPINCLUDE= Options 1024
ENCODING= Option 1035
INENCODING= OUTENCODING= Options 1038
ODSCHARSET= Option 1040
ODSTRANTAB= Option 1041
TRANSCODE= Column Modifier on PROC SQL 1042
RENCODING= Option 1043
TRANSCODE= Option 1045
TRANTAB= Option 1047
XMLENCODING= Option 1048
TRANTAB Statement 1049

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=, CVPFORMATWIDTH=, CVPMULTIPLIER=,CVPVARCHAR=, CVPEXCLUDE=, CVPINCLUDE= Options (p. 1024)	Specifies attributes for character variables that are needed to transcode a SAS file.
	ENCODING= Option (p. 1035)	Overrides and transcodes the encoding for input or output processing of external files.
	INENCODING=OUTENCODING= Options (p. 1038)	Overrides and changes the encoding when reading or writing SAS data sets in the SAS library.
	ODSCHARSET= Option (p. 1040)	Specifies the character set to be generated in the META declaration for the output.
	ODSTRANTAB= Option (p. 1041)	Specifies the translation table to use when transcoding an XML document for an output file.
	RENCODING= Option (p. 1043)	Specifies the ASCII-based or EBCDIC-based encoding to use for transcoding data for a SAS/SHARE server session that is using an EBCDICANY or ASCIIANY session encoding.
	XMLENCODING= Option (p. 1048)	Overrides the encoding of an XML document to import or export an external document.
Information	TRANSCODE= Option (p. 1045)	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. 1015)	Specifies the character set to be generated in the META declaration for the output.
	TRANTAB= Option (p. 1047)	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: 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

Statements:

- "ODS MARKUP Statement" in *SAS Output Delivery System: User's Guide*

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 options*> ;

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.

Operating Environment Information: For information about behavior specific to your operating environment for the DANISH, FINNISH, NORWEGIAN, or SWEDISH *collating-sequence-option*, see the SAS documentation for your operating environment.

ASCII
sorts character variables using the ASCII collating sequence. You need this option only when you want to achieve an ASCII ordering on a system where EBCDIC is the native collating sequence.

DANISH NORWEGIAN
sorts characters according to the Danish and Norwegian

The Danish and Norwegian collating sequence is shown in Figure 17.6 on page 1017.

EBCDIC
sorts character variables using the EBCDIC collating sequence. You need this option only when you want to achieve an EBCDIC ordering on a system where ASCII is the native collating sequence.

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 17.6 on page 1017.

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. For an example of using PROC TRANTAB and PROC SORT with SORTSEQ=, see "Example 6: Using Different Translation Tables for Sorting" on page 1101.

The available translation tables are

- ASCII
- DANISH
- EBCDIC
- FINNISH
- ITALIAN
- NORWEGIAN
- POLISH
- REVERSE
- SPANISH
- SWEDISH

The following figure shows how the alphanumeric characters in each language sorts:

Figure 17.1 *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 1127.

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 1127.

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 only on PROC SORT and PROC SQL. Collating rules are supported for PROC SORT, not for the system option or SORTSEQ= on PROC SQL.

Note Linguistic collation can 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 999 and "SORT Procedure" in *Base SAS Procedures Guide*.

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 "Collating Sequence" on page 14 for detailed information about linguistic collation.

"SORTSEQ=sort-table | LINGUISTIC" in *SAS SQL Procedure User's Guide* for information about linguistic sorting in PROC SORT.

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.

Value	Description
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 Table 21.1 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.
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.

Value	Type of Collation	Description
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 17.1 *Options*

Task	Option
Specify the collating sequence	
Specify ASCII	ASCII on page 1016
Specify EBCDIC	EBCDIC on page 1016
Specify Danish	DANISH on page 1016
Specify Finnish	FINNISH on page 1017
Specify Norwegian	NORWEGIAN on page 1016
Specify Polish	POLISH on page 1016
Specify Swedish	SWEDISH on page 1017

Task	Option
Specify a customized sequence	NATIONAL on page 1017
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 1017

See Also

- "Collating Sequence" on page 14
- "SORT Procedure" in *Base SAS Procedures Guide*

System Options:

- "SORTSEQ= System Option" on page 999
- "TRANTAB= System Option" on page 1002

CORRECTENCODING= Option

Explicitly changes the encoding attribute of a SAS file to match the encoding of the data in the SAS file.

Valid in: MODIFY statement of the DATASETS procedure

Syntax

MODIFY *SAS file* </<CORRECTENCODING=*encoding-value*> > ;

Optional Argument

</ <CORRECTENCODING=*encoding-value*> >
enables you to change the encoding indicator, which is recorded in the file's descriptor information, in order to match the actual encoding of the file's data. You cannot use this option in parenthesis after the name of each SAS file; you must specify CORRECTENCODING= after the forward slash. For example:

```
modify mydata / correctencoding=latin2;
```

For a list of valid encoding values for transcoding, see "SBCS, DBCS, and Unicode Encoding Values for Transcoding Data" on page 1127 .

Restriction CORRECTENCODING= can be used only when the SAS file uses the default base engine, which is V9 in SAS 9.

Example: Using the CORRECTENCODING= Option to Resolve a SAS Session Encoding and a SAS File Encoding

A file's encoding indicator can be different from the data's encoding. For example, a SAS file that was created before SAS 9 has no encoding indicator stored on the file. If such a SAS file that has no recorded encoding is opened in a SAS 9 session, SAS assigns the encoding of the current session. For example, if the encoding of the data is Danish EBCDIC, but the encoding for the current session is Western Wlatin1, then the actual encoding of the file's data and the encoding indicator that is stored in the file's descriptor information do not match. When this action occurs, the data does not transcode correctly and could result in unreadable output. The following MODIFY statement would resolve the problem by explicitly assigning an EDCDIC encoding:

Note: CEDA creates a read-only copy. You need to copy the data with PROC COPY or a DATA step to transcode the data permanently.

```
proc datasets library=myfiles;
   modify olddata / correctencoding=ebcdic1142;
quit;
```

CVPBYTES=, CVPENGINE=, CVPFORMATWIDTH=,

CVPMULTIPLIER=,CVPVARCHAR=, CVPEXCLUDE=, CVPINCLUDE= Options

Specifies attributes for character variables that are needed to transcode a SAS file.

Valid in:	LIBNAME statement
Category:	Data Access
PROC OPTIONS GROUP=	LIBNAME statement under LINUX.
Restriction:	The CVPEXCLUDE= and CVPINCLUDE= options are available since SAS Viya platform 3.5.
See:	LIBNAME, SAS/ACCESS

Syntax

LIBNAME *libref 'SAS data-library'* <CVPBYTES=*bytes*> <CVPENGINE=*engine*> <CVPEXCLUDE=*variable expression*><CVPFORMATWIDTH=YES| NO><CVPINCLUDE=*variable expression*><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 32,766.

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 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 automatically choose the multiplier (CVPMULTIPLIER=0) 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 the CVPBYTES= option and the CVPMULTIPLIER= option. Specify only one of these options.

See "Avoiding Character Data Truncation By Using the CVP Engine" on page 41

CVPENGINE=*engine*

specifies the engine to use 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 processes the actual file.

Alias CVPENG

Default SAS uses the default SAS engine.

See "Avoiding Character Data Truncation By Using the CVP Engine" on page 41

CVPEXCLUDE=*variable expression*

specifies the variables that are excluded during data processing.

In this example, character variables var1, var2, and var3 are excluded from data processing: `cvpexclude="^(var1|var2|var3)$"`

CVPINCLUDE= and CVPEXCLUDE= cannot be used together. If both options are omitted, then all variables are processed.

If CVPEXCLUDE=var is specified, then all variables that contain var (such as var, var1, and invariant) are excluded. Specify CVPEXCLUDE="^var$" to exclude only the var variable.

CVPFORMATWIDTH=YES | NO

specifies whether to expand the character format width.

If CVPVARCHAR= is not specified, the new format width is determined by the CVPMULTIPILER= and CVPBYTES= options.

If CVPVARCHAR= is specified, the CVP engine automatically adjusts the format width to meet the maximum-byte length of a converted character variable. For example, in a UTF-8 session, the format width is multiplied by 4.

Alias CVPFMTW

Default YES

CVPINCLUDE=*variable expression*

specifies the variables that are included during data processing. Only the specified variables can be processed.

CVPINCLUDE= and CVPEXCLUDE= cannot be used together. If both options are omitted, then all variables are processed.

Note: The CVPEXCLUDE= and CVPINCLUDE= options are available since SAS Viya platform 3.5.

CVPMULTIPLIER=*multiplier*

specifies a multiplier value that expands 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 or you can specify 0 and then the CVP engine determines the multiplier automatically.

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 the CVPMULTIPLIER= option, SAS automatically uses the CVP engine to expand the character variable lengths according to your specification. If you explicitly specify the CVP engine but do not specify either the CVPMULTIPLIER= option or the CVPBYTES= option, then SAS uses CVPMULTIPLIER=0 (AUTO) to increase the lengths. AUTO(0) sets the value of the CVP engine based on the encoding of the SAS session and input data set.

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 the CVPMULTIPLIER= option and the CVPBYTES= option. Specify only one of these options.

See "Avoiding Character Data Truncation By Using the CVP Engine" on page 41

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 data type 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.

For more information about VARCHAR, see "VARCHAR Data Type in String Functions" on page 583.

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.

By default, the CVP engine uses a multiplier of 1.5 times the variable length. This expansion is usually sufficient to support the data for Western European languages because the text for those languages usually has a high ratio of ASCII characters to national characters. The 1.5 multiplier might also be sufficient for most Asian language data. Even though many Asian characters require 3 bytes in UTF-8, the double-byte (DBCS) encodings that support those languages already require 2 bytes for each Asian character.

However, many languages that are represented by single-byte (SBCS) encodings use a high ratio of national characters in the text, which require 2 or 3 bytes in UTF-8. For example, Russian text primarily uses characters from the Cyrillic

alphabet, which require 1 byte in the SBCS encoding WCYRILLIC (CP1251 and WINDOWS-1251). Since each Cyrillic character requires 2 bytes in UTF-8, the 1.5 multiplier might not be sufficient to prevent truncation, depending on the length of the variable.

If you need to change the expansion, you can specify either the CVPMULTIPLIER= or CVPBYTES= option in the LIBNAME statement. CVPMULTIPLIER= specifies a multiplier value that is applied to the length of the variable. CVPBYTES= adds a specific number of bytes to the length of the character columns.

Here are examples:

```
libname mylib 'path-to-SAS-files' cvpmultiplier=2.5;
libname mylib 'path-to-SAS-files' cvpbytes=5;
```

You can also use the "NCHARMULTIPLIER= LIBNAME Statement Option" in *SAS Cloud Analytic Services: User's Guide* to specify a multiplier in order to run on the CAS server.

These data tables can help you determine the additional storage requirements when saving your data as UTF-8.

Table 17.186 shows the storage size increase that you can expect as you move data from the specified encoding to UTF-8.

Table 17.2 *Saving Data as UTF-8 and Possible Storage Size Increases*

Source Encoding	Languages Supported	Storage Size Increase in UTF-8
ASCII	English, Malay, and other supported languages	0%
ISO-8859-1 (LATIN1) and Windows cp1252 (WLATIN1)	Western European	10%
ISO-8859-7, plain text	Greek	90%
ISO-8859-7, 50% markup	Greek	45%
TIS-620, plain text	Thai	190%
TIS-620, 50% markup	Thai	95%
EUC-KR, plain text	Korean	50%
EUC-KR, 50% markup	Korean	55%

Table 17.187 shows how many bytes are needed in order to represent the characters for the languages in the list.

Table 17.3 *UTF-8 Character Length by Language*

Character Length	Language
1 byte	US_ASCII characters
2 bytes	East and West European, Baltic, Greek, Turkish, Cyrillic, Hebrew, Arabic, and other supported character sets
3 bytes	Chinese, Japanese, Korean (CJK), Thai, Indic, and certain control characters
4 bytes	Emoji characters, less common CJK characters, and various historic scripts

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. MYFILES.WLATIN2 needs to be created in a SAS session with wlatin2 encoding.

```
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 17.1 *PROC CONTENTS Output for MYFILES.WLATIN2*

Data Set Name	MYFILES.WLATIN2	Observations	1
Member Type	DATA	Variables	4
Engine	V9	Indexes	0
Created	07/28/2017 12:02:40	Observation Length	4
Last Modified	07/28/2017 12:02:40	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
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	C:\SAS Data-Library\wlatin2.sas7bdat
Release Created	9.0401M5
Host Created	X64_7PRO
Owner Name	BUILTIN\Administrators
File Size	128KB
File Size (bytes)	131072

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 with the session encoding Wlatin2.

```
options msglevel=i;
libname myfiles 'SAS Data-Library';
data myfiles.utf8 (encoding="utf-8");
   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.

Example Code 17.1 *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 might require additional
CPU resources and might reduce performance.
   ERROR: Some character data was lost during transcoding in the dataset
          MYFILES.UTF8. Either the data contains characters that are not
          representable in the new encoding or truncation
          occurred during transcoding.
   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 and expands the variable lengths:

```
options msglevel=i;
libname myfiles 'SAS Data-Library';
libname expand cvp 'SAS Data-Library' cvpbytes=2;
data myfiles.utf8 (encoding="utf-8");
   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:

Example Code 17.2 *SAS Log Output for MYFILES.UTF8*

```
57     options msglevel=i;
58     libname myfiles 'C:\SAS Data-Library';

NOTE: Libref MYFILES was successfully assigned as follows:
       Engine:        V9
       Physical Name:  C:\SAS Data-Library
59     libname expand cvp 'C:\SAS Data-Library' 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: 'C:\SAS Data-Library'
60     data myfiles.utf8 (encoding="utf-8");;
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.
```

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 17.2 *PROC CONTENTS Output for MYFILES.UTF8*

Data Set Name	MYFILES.UTF8	Observations	1
Member Type	DATA	Variables	4
Engine	V9	Indexes	0
Created	07/28/2017 12:02:41	Observation Length	12
Last Modified	07/28/2017 12:02:41	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
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	5403
Obs in First Data Page	1
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\SAS Data-Library\utf8.sas7bdat
Release Created	9.0401M5
Host Created	X64_7PRO
Owner Name	BUILTIN\Administrators
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 1127.

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.

Some encodings use a Byte-Order Mark (BOM). The BOM is generated when the encoding is specified. For the UTF-8 encoding, you must specify *encoding=utf-8* on the filename and file DATA step statements in order for the BOM to be generated.

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 Wlatin1, but the external file's encoding needs to be UTF-8. 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="utf-8";
   put Make Model Year;
run;
```

When you tell SAS that the external file is to be in UTF-8 encoding, SAS then transcodes the data from Wlatin1 to the specified UTF-8 encoding.

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 UTF-8 character-set encoding, and the current SAS session is in the Wlatin1 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 UTF-8, SAS then transcodes the external file from UTF-8 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 Wlatin1.

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 Wlatin1, but the external file's encoding needs to be UTF-8.

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="utf-8";
data _null_;
   set myfiles.cars;
   file outfile;
   put Make Model Year;
run;
```

When you specify that the external file is to be in UTF-8 encoding, SAS then transcodes the data from Wlatin1 to the specified UTF-8 encoding 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 UTF-8, and the current SAS session encoding is Wlatin1. 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 UTF-8, SAS then transcodes the external file from UTF-8 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 Wlatin1.

See Also

Statements:

- "FILE Statement" in *SAS DATA Step Statements: Reference*
- "INFILE Statement" in *SAS DATA Step Statements: Reference*

INENCODING=

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 23, "Encoding Values in SAS Language Elements," on page 1127.

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. For more information about CLONE and NOCLONE, see COPY Statement.

Note: This interaction does not apply when using SAS/CONNECT or SAS/SHARE.

See Also

- "Overview: Encoding for NLS" on page 9

System Options:

- "ENCODING System Option" on page 982
- "LOCALE System Option" on page 987

Data Set Options:

- "ENCODING= Data Set Option" on page 82

ODSCHARSET= Option

Specifies the character set to be generated in the META declaration for the output.

Valid in: LIBNAME statement for the XML V2 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 XMLV2 and XML LIBNAME Engines: 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 XML V2 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

For SAS 9.2, using the ODSTRANTAB= option in the LIBNAME statement for the XML Engine is supported for backward compatibility. The preferred method for specifying an encoding is to use the LOCALE= system option.

See Also

Conceptual Information:

- "Transcoding and Translation Tables" on page 32
- Conceptual discussion of Chapter 2, "Locale for NLS," on page 5

System Options:

- "TRANTAB= System Option" on page 1002
- "LOCALE System Option" on page 987

Procedures:

- Chapter 20, "TRANTAB Procedure," on page 1079

Statements:

- *SAS XMLV2 and XML LIBNAME Engines: 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

Conceptual Information:

- Chapter 4, “Transcoding for NLS,” on page 31

RENCODING= Option

Specifies the ASCII-based or EBCDIC-based encoding to use for transcoding data for a SAS/SHARE server session that is using an EBCDICANY or ASCIIANY session encoding.

Valid in: LIBNAME statement for SAS/SHARE only

Category: Data Access

Note: The RENCODING= option in the LIBNAME statement is relevant only if using a SAS/SHARE server that has a session encoding set to EBCDICANY or ASCIIANY to preserve a mixed-encoding computing environment.

See: LIBNAME statement in *SAS/SHARE User's Guide*

Syntax

RENCODING=*ASCII-encoding-value* | *EBCDIC-encoding-value*

Syntax Description

ASCII-encoding-value
For a list of valid values for ASCII encodings for UNIX and Windows, see “SBCS, DBCS, and Unicode Encoding Values for Transcoding Data” on page 1127.

EBCDIC-encoding-value
For a list of valid values for EBCDIC encodings for z/OS, see “SBCS, DBCS, and Unicode Encoding Values for Transcoding Data” on page 1127.

Details

If you use SAS/SHARE in a mixed-encoding environment (for example, SAS/SHARE client sessions using incompatible encodings such as Latin1 and Latin2), you can set the following options:

- in the SAS/SHARE server session, set the SAS system option ENCODING=EBCDICANY or ENCODING=ASCIIANY
- in the SAS/SHARE client session, set the RENCODING= option in the LIBNAME statement(s) under these conditions:
  - a client session that uses an ASCII-based encoding accesses an EBCDICANY server
  - a client session that uses an EBCDIC- based encoding accesses an ASCIIANY server.

  The RENCODING= option enables SAS/SHARE clients to specify which encoding to assume the server's data is in when transcoding to or from the client session encoding.

For SAS 9, if you are processing data in a SAS/SHARE client/server session from more than one SBCS or DBCS encoding, you are advised to use the UTF8 encoding. For more information about Unicode servers that run the UTF8 session encoding, go to **`http://rnd.sas.com/sites/i18n/i18ndocs/i18nsupport/Pages/SAS%20Technical%20Papers.aspx`** and search for *Processing Multilingual Data with the SAS® 9.2 Unicode Server* and *Multilingual Computing with SAS® 9.4*.

Comparisons

In SAS 9, you can maintain multilingual data that contains characters from more than one traditional SBCS or DBCS encoding in a SAS data set by using a UTF8 encoding. To share Update access to that data using SAS/SHARE, you must also run the SAS/SHARE server using a session encoding of UTF8. SAS transcodes the data to the client encoding if necessary.

Prior to SAS 9, if a SAS/SHARE client and a SAS/SHARE server ran on common architectures (for example, the client and server ran on UNIX machines), there was no automatic transcoding of character data. It was possible to build applications that accessed data sets in different EBCDIC or ASCII encodings within a single SAS/SHARE server, or that accessed data sets in mixed different encodings within a single data set. This method was very uncommon and required careful programming to set up transcoding tables from clients that ran in different operating environments.

The following steps describe how you can maintain mixed encoding in SAS 9, if necessary.

- The SAS/SHARE server must run by using a session encoding of EBCDICANY for mixed-EBCDIC encodings or ASCIIANY for mixed-ASCII encodings.

  This restores the behavior of Version 8 and earlier releases and prevent the automatic character transcoding between different client and server encodings in the same EBCDIC or ASCII family. That is, no transcoding occurs under these circumstances:
  - if the client session encoding is an EBCDIC encoding and the server session encoding is EBCDICANY
  - if the client session encoding is an ASCII encoding and the server session encoding is ASCIIANY.

- A SAS/SHARE client that does not share the same encoding family as an ASCIIANY or EBCDICANY server can control the necessary transcoding by using an RENCODING= option on the first LIBNAME statement that accesses the server.

  For example, an ASCII client that runs in a Polish locale could access a z/OS EBCDICANY server and specify RENCODING=EBCDIC870 to access data that the client knows contains Polish-encoded data. Another ASCII client that runs in a German locale could access the same z/OS EBCDICANY server and specify RENCODING=EBCDIC1141 to access data that the client knows contains German data. Similarly, EBCDIC clients that access an ASCIIANY server can specify the precise ASCII encoding of the data that they are accessing by using the RENCODING= option in the LIBNAME statement.

See Also

Conceptual Information:

- "Overview of Transcoding" on page 31

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

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 DATA Step Statements: Reference* .

TRANSCODE= YES | NO

Specifies whether to transcode character variables. Use TRANSCODE=NO to suppress transcoding. For more information, see "Overview of Transcoding" on page 31 .

Default YES

Restriction The TRANSCODE=NO attribute is not supported by some SAS compute server clients. Variables with TRANSCODE=NO are not returned in Viya.

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 759
- "VTRANSCODEX Function" on page 760

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

In SAS 9.4, the TRANTAB= option in the ODS MARKUP statement is supported for backward compatibility. The ENCODING= option is preferred when specifying the encoding.

See Also

Conceptual Information:

- “Transcoding and Translation Tables” on page 32

Procedures:

- Chapter 20, “TRANTAB Procedure,” on page 1079

Statements:

- “ODS MARKUP Statement” in *SAS Output Delivery System: User’s Guide*
- “ODS RTF Statement” in *SAS Output Delivery System: User’s Guide*

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 1127.

The default for *encoding-value* is the current session encoding.

See Also

Statements:

- *SAS XMLV2 and XML LIBNAME Engines: 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(|)

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-options-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 31

System Options:

- “TRANTAB= System Option” on page 1002

Procedures:

- Chapter 20, “TRANTAB Procedure,” on page 1079
- “CPORT Procedure” in *Base SAS Procedures Guide*
- “UPLOAD Procedure” in *SAS/CONNECT User’s Guide*
- “DOWNLOAD Procedure” in *SAS/CONNECT User’s Guide*

PART 10

Procedures for NLS

Chapter 18
DBCSTAB Procedure 1055

Chapter 19
LOCALEDATA Procedure 1063

Chapter 20
TRANTAB Procedure 1079

18

DBCSTAB Procedure

Overview: DBCSTAB Procedure ***1055***
What Does the DBCSTAB Procedure Do? 1055
Syntax: DBCSTAB Procedure ***1056***
PROC DBCSTAB Statement 1056
Examples: DBCSTAB Procedure ***1057***
Example 1: Creating a Conversion Table with the DBCSTAB Procedure 1057
Example 2: Producing Japanese Conversion Tables with the DBCSTAB Procedure 1059

Overview: DBCSTAB Procedure

What Does the DBCSTAB Procedure Do?

The DBCSTAB procedure produces conversion tables for the double-byte character sets (DBCSs) that SAS supports.

Use the DBCSTAB procedure to modify an existing DBCS table when

- the DBCS encoding system that you are using is not supported by SAS
- the DBCS encoding system that you are using has a nonstandard translation table

You might be more likely to use the DBCSTAB procedure when a valid DBCSTYPE= value is not available. These values depend on the operating environment. In such cases, you can use the DBCSTAB procedure to modify a similar translation table, and then you can specify the use of the new table with the TRANTAB option.

Syntax: DBCSTAB Procedure

PROC DBCSTAB TABLE=*table-name*
<BASETYPE=*base-type*> <CATALOG=<*libref.*>*catalog-name*>
<DATA=<*libref.*>*table-name*> <DBCSLANG=*language*>
<DESC='*description*'> <FORCE> <VERIFY> <VERBOSE>;

PROC DBCSTAB Statement

Produces conversion tables for double-byte character sets.

Syntax

PROC DBCSTAB TABLE=*table-name*
<*options*>;

Required Argument

TABLE=*table-name*
specifies the name of the double-byte code table to produce. This table name becomes an entry of type DBCSTAB in the catalog that is specified with the CATALOG= option. By default, the catalog name is Sasuser.DBCS.

Aliases NAME=

N=

Optional Arguments

BASETYPE=base-type
specifies a base type for the double-byte code table conversion. If you use this option, you reduce the number of tables that are produced.

If you specify BASETYPE=, then all double-byte codes are first converted to the base code, and then they are converted to the required code. If you have *n* codes, then there are *n*(*n*-1) conversions that must be made.

Alias BTYPE=

CATALOG=<libref.>catalog-name
specifies the name of the catalog in which the table is stored. If the catalog does not exist, it is created.

Default Sasuser.DBCS

DATA=<*libref.*>*table-name*
specifies the data for producing the double-byte code table. Several double-byte character variables are required to produce the table. Use variable names that are equivalent to the value of the DBCSTYPE= system option and recognized by the KCVT function.

DBCSLANG=language
specifies the language that the double-byte code table uses. The value of this option should match the value of the DBCSLANG= system option.

Alias DBLANG=

DESC='*description*'
specifies a text string to put in the DESCRIPTION field for the entry.

FORCE
produces the conversion tables, even if errors are present.

VERIFY
checks the data range of the input table based on the code. This option checks for invalid double-byte code.

VERBOSE
prints the statistics detail when building DBCS tables.

Examples: DBCSTAB Procedure

Example 1: Creating a Conversion Table with the DBCSTAB Procedure

Features: PROC DBCSTAB statement options:
CATALOG=
DBLANG=
BASETYPE=
VERIFY

The following example creates a Japanese translation table called CUSTAB and demonstrates how the TRANTAB= option specifies the new translation table. The DBCS=, DBCSLANG=, and DBCSTYPE= options are specified at start-up.

Program

```
proc dbcstab
      /* name of the new translate table */
          name=custtab
      /* based on pcibm encoding */
          basetype=pcms
      /* data to create the new table */
          data=trantab
      /* japanese language */
          dbcslang=japanese
      /* catalog descriptor */
          desc='Modified Japanese Trantab'
      /* where the table is stored */
          catalog=sasuser.dbcs
      /* checks for invalid DBCS in the new data */
          verify;
run;
```

Creating a Conversion Table

```
invoke Japanese SAS with Shift_JIS encoding.

data trantab;

pcms='8342'x; dec='b9b3'x;
run;

proc dbcstab;

name=custtab;

basetype=pcms;

data=trantab;

dbcslang=japanese;

desc='Modified Japanese Trantab';

catalog=sasuser.dbcs;

verify;
run;

options trantab=(,,,,,,,,,custtab);
```

Program Description

Set the SAS environment.

```
invoke Japanese SAS with Shift_JIS encoding.
```

Create the TRANTAB data set.

```
data trantab;
```

Define the pcms and dec parameters and run the code.

```
pcms='8342'x; dec='b9b3'x;
run;
```

Invoke the DBCSTAB procedure.

```
proc dbcstab;
```

Assign a name to a translate table.

```
name=custtab;
```

Apply pcibm encoding.

```
basetype=pcms;
```

Access data from the TranTab data set.

```
data=trantab;
```

Set the language to Japanese.

```
dbcslang=japanese;
```

Set the catalog descriptor to Modified Japanese Translation table.

```
desc='Modified Japanese Trantab';
```

Set the table's location.

```
catalog=sasuser.dbcs;
```

Check for invalid DBCS data and then process the program.

```
verify;
run;
```

Select the translate table using the TRANTAB= option. Translate tables are used for DBCS conversion with SAS/CONNECT, PROC CPORT and PROC CIMPORT, and the DATA step function KCVT. The TRANTAB= option can specify DBCS translate tables. For SAS 8.2 and earlier, the ninth argument specified the DBCS system table. However, for SAS 9 and later, instead of using the ninth argument, SAS uses a system table that is contained in a module. Japanese, Korean, Chinese, and Taiwanese are acceptable for the SYSTAB name. The 10th argument specifies the DBCS user table:

```
options trantab=(,,,,,,,,,custtab);
```

Example 2: Producing Japanese Conversion Tables with the DBCSTAB Procedure

Features:

PROC DBCSTAB statement options:

- TABLE=
- DATA=
- DBLANG=
- BASETYPE=
- VERIFY

Program

```
data ja_jpn;
length ibm jis euc pcibm $2.;
ibm='4040'x;
jis='2121'x;
euc='a1a1'x;
pcibm='8140'x;
run;
proc dbcstab;
table=japanese;
data=ja_jpn;
dblang=japanese;
basetype=jis;
verify;
run;
```

Program Description

Define the ja_jpn data set.

```
data ja_jpn;
```

Set the length of the specified encodings to 2.

```
length ibm jis euc pcibm $2.;
```

Assign the value 4040 to the ibm encoding.

```
ibm='4040'x;
```

Assign the value 2121 to the jis encoding.

```
jis='2121'x;
```

Assign the value a1a1 to the euc encoding.

```
euc='a1a1'x;
```

Assign the value 8140 to the pcibm encoding and run the code.

```
pcibm='8140'x;
run;
```

Invoke the DBCSTAB procedure.

```
proc dbcstab;
```

Define the table with the name, Japanese.

```
table=japanese;
```

Access the Ja_Jpn data set.

```
data=ja_jpn;
```

Specify Japanese for the language.

```
dblang=japanese;
```

Specify the jis encoding.

```
basetype=jis;
```

Check for invalid DBCS data and then process the program.

```
verify;
run;
```

SAS Log

```
1   proc dbcstab
2   table=ja_jpn
3   data=work.ja_jpn
4   dblang=japanese
5   basetype=jis
6   verify;
7   run;

NOTE: Base table for JIS created.
NOTE: IBM table for JIS created.
NOTE: PCIBM table for JIS created.
NOTE: EUC table for JIS created.
NOTE: Base table for IBM created.
NOTE: JIS table for IBM created.
NOTE: Base table for PCIBM created.
NOTE: JIS table for PCIBM created.
NOTE: Base table for EUC created.
NOTE: JIS table for EUC created.
NOTE: 10 DBCS tables are generated. Each table has 1 DBCS characters.
NOTE: Each table is 2 bytes in size.
NOTE: Required table memory size is 612.
NOTE: There were 1 observations read from the data set WORK.JA_JPN.
```

19

LOCALEDATA Procedure

Overview: LOCALEDATA Procedure ***1063***
What Does the LOCALEDATA Procedure Do? 1063
Syntax: LOCALEDATA Procedure ***1063***
PROC LOCALEDATA Statement 1064
LOAD Statement 1064
MODIFY Statement 1065
CONTENTS Statement 1071
SAVE Statement 1072
Example: Modifying Locale Data Using the LOCALEDATA Procedure ***1072***

Overview: LOCALEDATA Procedure

What Does the LOCALEDATA Procedure Do?

The LOCALEDATA procedure lets you customize data. You can view, open, modify, and store the customized locale data.

Syntax: LOCALEDATA Procedure

```
PROC LOCALEDATA;
   LOAD SASLOCALE | REGISTRY;
```

MODIFY key=*key-name* value=*key-value* | category=*category-name* value=*locale*;

CONTENTS _ALL_ | *key-name* | *category-name*;

SAVE REGISTRY <_ALL_ | *key-name* | *category-name syntax=SAS*>;

Statement	Task
PROC LOCALEDATA	Specifies the source of customized locale data.
LOAD	Specifies how to customize the locale data by locale element key or by locale category.
CONTENTS	Specifies what locale element's values are discarded.
SAVE	Stores the customized locale data in the SAS registry.
MODIFY	Customizes the locale data by locale element key or by locale category.

PROC LOCALEDATA Statement

Specifies the source of customized locale data.

Syntax

PROC LOCALEDATA;

LOAD Statement

Specifies which source of locale data is loaded for customization.

Default: Registry

Syntax

LOAD SASLOCALE | REGISTRY;

Required Arguments

SASLOCALE
loads the locale data from the factory setting default locale database.

REGISTRY
loads the locale data from the SAS registry. This locale data could be customized locale data that has been stored in the SAS registry.

MODIFY Statement

Customizes the locale data by locale element key or by locale category.

Syntax

MODIFY key=*key-name* value=*key-value* | category=*category-name* value=*locale*

Required Arguments

key=*key-name* value=*key-value*
customizes single locale element values.

category=*category-name* value=*locale*
customizes all locale elements in a category. You can select one of the following categories:

- LC_NUMERIC
- LC_MONETARY
- LC_TIME

The following table shows the element keys that you can modify.

The value of the locale element key must be less than the maximum length value.

The following values can be used for the type:

0 string

1 unassigned integer; 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	

Locale Element Key	Max Length	Type	Category
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	
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

Locale Element Key	Max Length	Type	Category
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
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

Locale Element Key	Max Length	Type	Category
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
MON06	512	0	LC_TIME
MON07	512	0	LC_TIME
MON08	512	0	LC_TIME
MON09	512	0	LC_TIME

Locale Element Key	Max Length	Type	Category
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
ABQTR3	512	0	LC_TIME
ABQTR4	512	0	LC_TIME

Locale Element Key	Max Length	Type	Category
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_N_CS_PRECEDES	3	1	LC_MONETARY
MON_N_SEP_BY_SPACE	3	1	LC_MONETARY
MON_P_SIGN_POSN	3	1	LC_MONETARY
MON_N_SIGN_POSN	3	1	LC_MONETARY

Locale Element Key	Max Length	Type	Category
DECIMAL_POINT	1	0	LC_NUMERIC
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
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	

CONTENTS Statement

Displays the value of the specified locale element key.

Syntax

CONTENTS_ALL_ | *locale-element-key* | *category-name*

Required Arguments

ALL
displays the value of all locale element keys.

locale-element-key
displays the value of the specified locale element key.

category-name
displays the value of locale elements for the specified category.

SAVE Statement

Stores the customized locale data in the SAS registry.

Syntax

SAVE REGISTRY < _ALL_ | *key-name* | *category-name syntax=SAS*>

Required Argument

REGISTRY
stores locale data in the Sasuser library and registers it in the SAS registry. The encoding of the stored locale data is in UNICODE escape. Here is the hierarchy of the saved locale element in the SAS registry: **SAS_REGISTRY** ⇨ **LOCALE** ⇨ **sas locale** ⇨ **locale element key locale category** ⇨ **locale element key**.

Optional Arguments

ALL
saves all locale data elements.

key-name
saves the specified locale element key name and value.

category-name syntax=SAS
saves all elements in the specified category.

Example: Modifying Locale Data Using the LOCALEDATA Procedure

Features: PROC LOCALEDATA statement
LOAD statement
MODIFY statement
SAVE statement
CONTENTS statement
DATA step
PUT statement

The following program modifies locale data. SAS is invoked twice.

Modifying Locale Definitions

```
localedata=registy;

PROC LOCALEDATA;

LOAD SASLOCALE;

MODIFY KEY=QTR1 VALUE='QTR1';
MODIFY KEY=QTR2 VALUE='QTR2';
MODIFY KEY=QTR3 VALUE='QTR3';
MODIFY KEY=QTR4 VALUE='QTR4';
MODIFY KEY=DATE_SHORT_FORMAT VALUE='%Y//%m//%d';
MODIFY KEY=DATE_YEAR_FORMAT VALUE='Year(%Y)';

SAVE REGISTRY / _ALL_ syntax=SAS;

CONTENTS _ALL_;

QUIT;

data _null_;

a = 19208;

put a nldate12. / a nldateyq. / a nldateyr.;
run;
```

Program Description

You must set the “LOCALEDATA System Option”.

Set the LOCALEDATA system option to registry.

```
localedata=registy;
```

Invoke the LOCALEDATA procedure.

```
PROC LOCALEDATA;
```

Load the locale definitions from the local locale database.

```
LOAD SASLOCALE;
```

Modify the definition values.

```
MODIFY KEY=QTR1 VALUE='QTR1';
MODIFY KEY=QTR2 VALUE='QTR2';
MODIFY KEY=QTR3 VALUE='QTR3';
MODIFY KEY=QTR4 VALUE='QTR4';
MODIFY KEY=DATE_SHORT_FORMAT VALUE='%Y//%m//%d';
MODIFY KEY=DATE_YEAR_FORMAT VALUE='Year(%Y)';
```

Save the modified definitions in the registry. This action overwrites the definitions that are in the registry so that there are no duplicate key values.

```
SAVE REGISTRY / _ALL_ syntax=SAS;
```

View the modified locale definitions from the registry.

```
CONTENTS _ALL_;
```

Invoke the QUIT command to end processing.

```
QUIT;
```

Create a DATA step for testing the modified locale definitions.

```
data _null_;
```

Assign a value of 19208 to variable a.

```
a = 19208;
```

Generate the value of variable a using the formats NLDATE., NLDATEYQ., and NLDATEYR.

```
put a nldate12. / a nldateyq. / a nldateyr.;
run;
```

SAS Log

```
1    PROC LOCALEDATA;
2
3      /* Load locale defintion from locale DB */
4      LOAD SASLOCALE;
5
6      /* Change the value of the elements */
7      MODIFY KEY=QTR1 VALUE='QTR1';
8      MODIFY KEY=QTR2 VALUE='QTR2';
9      MODIFY KEY=QTR3 VALUE='QTR3';
10     MODIFY KEY=QTR4 VALUE='QTR4';
11     MODIFY KEY=DATE_SHORT_FORMAT VALUE='%Y//%m//%d';
12     MODIFY KEY=DATE_YEAR_FORMAT VALUE='Year(%Y)' ;
13     /* Store the locale definition into registry, this will override the existing
definition,
13 ! so there is no duplicate key issue */
14     SAVE REGISTRY / _ALL_ syntax=SAS;
15
16     /* View the locale definition from registry */
17     CONTENTS _ALL_;
LOCALE="English_UnitedStates"
LANGUAGE="English"
LANGUAGE_SCRIPT="en"
TERRITORY="UnitedStates"
LANGID="en"
LOCID="US"
DATESTYLE="MDY"
PAPERSIZE="LETTER"
FTITLE=""
FTEXT=""
SIMFONT=""
SORTSEQ=""
MESSAGES=""
FORMATNAME_DATE="NLDATE16."
FORMATNAME_DATETIME="NLDATM24."
FORMATNAME_TIME="NLTIME8."
FORMATNAME_NUMERIC="BEST12."
FORMATNAME_PERCENT="PERCENT12."
FONT_SERIF="COMPLEX"
FONT_SANSSERIF="SIMPLEX"
FONT_CURSIVE="ITALIC"
FONT_FANTASY="BRUSH"
FONT_MONOSPACE="SIMPLEX"
BRUSH="BRUSH"
SIMPLEX="SIMPLEX"
COMPLEX="COMPLEX"
SWISS="SWISS"
ITALIC="ITALIC"
DATE_FORMAT="%B %d, %Y"
DATE_SHORT_FORMAT="%Y//%m//%d"
DATE_MEDIUM_FORMAT="%b %e, %Y"
DATE_LONG_FORMAT="%B %e, %Y"
DATE_FULL_FORMAT="%A, %B %e, %Y"
DATE_FULL_AB_FORMAT="%a, %b %e, %Y"
DATE_YYMM_FORMAT="%B %Y"
DATE_YYMM_SHORT_FORMAT="%b %Y"
DATE_MMDD_FORMAT="%B %d"
DATE_MMDD_SHORT_FORMAT="%b %d"
DATE_YEAR_FORMAT="Year(%Y)"
DATE_YEAR_SHORT_FORMAT="%y"
DATE_YYQQ_FORMAT="%Q %Y"
DATE_YYQQ_SHORT_FORMAT="%q %Y"
DATE_YYWW_FORMAT="Week %U %Y"
DATE_YYWW_SHORT_FORMAT="W%U %y"
```

```
DATETIME_AMPM_FORMAT="%B %e, %Y %I:%M:%S %p"
DATETIME_AMPM_AB_FORMAT="%b %e, %Y %I:%M:%S %p"
DATETIME_FORMAT="%d%b%Y:%H:%M:%S"
DATETIME_SHORT_FORMAT="%o/%e/%Y %I:%M:%S %p"
DATETIME_MEDIUM_FORMAT="%b %e, %Y %I:%M:%S %p"
DATETIME_LONG_FORMAT="%B %e, %Y %I:%M:%S %p"
DATETIME_FULL_FORMAT="%A, %B %e, %Y %I:%M:%S %p"
DATETIME_FULL_AB_FORMAT="%a, %b %e, %Y %I:%M:%S %p"
TIME_AMPM_FORMAT="%I:%M:%S %p"
TIME_FORMAT="%H:%M:%S"
TIME_SHORT_FORMAT="%I:%M %p"
TIME_MEDIUM_FORMAT="%I:%M:%S %p"
TIME_LONG_FORMAT="%I:%M:%S %p %Z"
TIME_FULL_FORMAT="%I:%M:%S %p %Z"
DATETIME_PATTERN="%x %X"
ABMON01="Jan"
ABMON02="Feb"
ABMON03="Mar"
ABMON04="Apr"
ABMON05="May"
ABMON06="Jun"
ABMON07="Jul"
ABMON08="Aug"
ABMON09="Sep"
ABMON10="Oct"
ABMON11="Nov"
ABMON12="Dec"
MON01="January"
MON02="February"
MON03="March"
MON04="April"
MON05="May"
MON06="June"
MON07="July"
MON08="August"
MON09="September"
MON10="October"
MON11="November"
MON12="December"
SA_ABMON01="Jan"
SA_ABMON02="Feb"
SA_ABMON03="Mar"
SA_ABMON04="Apr"
SA_ABMON05="May"
SA_ABMON06="Jun"
SA_ABMON07="Jul"
SA_ABMON08="Aug"
SA_ABMON09="Sep"
SA_ABMON10="Oct"
SA_ABMON11="Nov"
SA_ABMON12="Dec"
SA_MON01="January"
SA_MON02="February"
SA_MON03="March"
SA_MON04="April"
SA_MON05="May"
SA_MON06="June"
SA_MON07="July"
SA_MON08="August"
SA_MON09="September"
SA_MON10="October"
SA_MON11="November"
SA_MON12="December"
```

```
ABDAY1="Sun"
ABDAY2="Mon"
ABDAY3="Tue"
ABDAY4="Wed"
ABDAY5="Thu"
ABDAY6="Fri"
ABDAY7="Sat"
DAY1="Sunday"
DAY2="Monday"
DAY3="Tuesday"
DAY4="Wednesday"
DAY5="Thursday"
DAY6="Friday"
DAY7="Saturday"
SA_ABDAY1="Sun"
SA_ABDAY2="Mon"
SA_ABDAY3="Tue"
SA_ABDAY4="Wed"
SA_ABDAY5="Thu"
SA_ABDAY6="Fri"
SA_ABDAY7="Sat"
SA_DAY1="Sunday"
SA_DAY2="Monday"
SA_DAY3="Tuesday"
SA_DAY4="Wednesday"
SA_DAY5="Thursday"
SA_DAY6="Friday"
SA_DAY7="Saturday"
ABQTR1="Q1"
ABQTR2="Q2"
ABQTR3="Q3"
ABQTR4="Q4"
QTR1="QTR1"
QTR2="QTR2"
QTR3="QTR3"
QTR4="QTR4"
SA_ABQTR1="Q1"
SA_ABQTR2="Q2"
SA_ABQTR3="Q3"
SA_ABQTR4="Q4"
SA_QTR1="1st quarter"
SA_QTR2="2nd quarter"
SA_QTR3="3rd quarter"
SA_QTR4="4th quarter"
AM="AM"
PM="PM"
DATE_SEP="/"
FIRST_DAY_OF_WEEK="0"
INT_CURRENCY_SYMBOL="USD"
CURRENCY_SYMBOL="$"
MON_DECIMAL_POINT="."
MON_THOUSANDS_SEP=","
MON_GROUPING="3"
MON_POSITIVE_SIGN=""
MON_NEGATIVE_SIGN="-"
MON_INT_FRAC_DIGITS="2"
MON_FRAC_DIGITS="2"
MON_P_CS_PRECEDES="1"
MON_P_SEP_BY_SPACE="0"
MON_N_CS_PRECEDES="1"
MON_N_SEP_BY_SPACE="0"
MON_P_SIGN_POSN="1"
MON_N_SIGN_POSN="0"
```

```
CURR_FMT_L="#,##0.00;(#,##0.00)"
CURR_FMT_I="#,##0.#;(#,##0.#)"
NUM_DECIMAL_POINT="."
NUM_THOUSANDS_SEP=","
NUM_GROUPING="3"
NUM_POSITIVE_SIGN=""
NUM_NEGATIVE_SIGN="-"
NUM_P_CS_PRECEDES="1"
NUM_P_SEP_BY_SPACE="0"
NUM_N_CS_PRECEDES="1"
NUM_N_SEP_BY_SPACE="0"
NUM_P_SIGN_POSN="1"
NUM_N_SIGN_POSN="1"
DEC_FMT_L="#,##0.###"
DEC_FMT_I="#,##0.#"
PCT_FMT_L="#,##0%"
PCT_FMT_I="#,##0.#%"
HEIGHT="279"
WIDTH="216"
18
19   QUIT;

NOTE: PROCEDURE LOCALEDATA used (Total process time):
      real time           0.24 seconds
      cpu time            0.18 seconds


20
21   data _null_ ;
22   a = 19208 ;
23   put a nldate. / a NLDATEYQ. / a NLDATEYR. ;
24   run ;

August 03, 2012
3rd quarter 2012
2012
NOTE: DATA statement used (Total process time):
      real time           2.96 seconds
      cpu time            0.28 seconds
```

20

TRANTAB Procedure

Overview: TRANTAB Procedure ***1079***
What Does the TRANTAB Procedure Do? 1080

Concepts: TRANTAB Procedure ***1081***
Understanding Translation Tables and Character Sets for PROC TRANTAB 1081
Storing Translation Tables with PROC TRANTAB 1081
Modifying SAS Translation Tables with PROC TRANTAB 1082
Using Translation Tables Outside PROC TRANTAB 1082

Syntax: TRANTAB Procedure ***1084***
PROC TRANTAB Statement 1085
CLEAR Statement 1086
INVERSE Statement 1086
LIST Statement 1087
LOAD Statement 1087
REPLACE Statement 1088
SAVE Statement 1089
SWAP Statement 1090

Examples: TRANTAB Procedure ***1091***
Example 1: Viewing a Translation Table 1091
Example 2: Creating a Translation Table 1092
Example 3: Editing By Specifying a Decimal Value for the Starting Position 1095
Example 4: Editing By Using a Quoted Character for the Starting Position 1097
Example 5: Creating the Inverse of a Table 1100
Example 6: Using Different Translation Tables for Sorting 1101
Example 7: Editing Table 1 and Table 2 1103

Overview: TRANTAB Procedure

What Does the TRANTAB Procedure Do?

The TRANTAB procedure creates, edits, and displays customized translation tables. In addition, you can use PROC TRANTAB to view and modify translation tables that are supplied by SAS. These SAS supplied tables are stored in the Sashelp.Host catalog. Any translation table that you create or customize is stored in your Sasuser.Profile catalog. Translation tables have an entry type of TRANTAB.

Translation tables are operating environment-specific SAS catalog entries that translate the values of one (coded) character set to another. A translation table has two halves: table 1 provides a translation, such as ASCII to EBCDIC; table 2 provides the inverse (or reverse) translation, such as EBCDIC to ASCII. Each half of a translation table is an array of 256 two-digit *positions*, each of which contains a one-byte unsigned number that corresponds to a coded character.

SAS uses translation tables to

- determine the collating sequence in the SORT procedure
- facilitate data communications between the operating environment and a graphics device when you run SAS/GRAPH software in an IBM environment
- accommodate national language character sets other than U.S. English.

PROC TRANTAB produces no output. It can display translation tables and notes in the SAS log.

Note: Translation tables were introduced in SAS 6 to support the requirements of national languages. SAS 8.2 introduced the LOCALE= system option to improve the direct use of translation tables. SAS 9.2 supports the TRANTAB procedure for backward compatibility. However, using the LOCALE= system option is preferred in later SAS releases. PROC TRANTAB is an interactive procedure. Once you submit a PROC TRANTAB statement, you can continue to enter and execute statements without repeating the PROC TRANTAB statement. To terminate the procedure, submit a QUIT statement or another DATA or PROC statement.

Concepts: TRANTAB Procedure

Understanding Translation Tables and Character Sets for PROC TRANTAB

The *Kth* element in a translation table corresponds to the *Kth* element of an ordered character set. For example, position 00 (which is byte 1) in a translation table contains a coded value that corresponds to the first element of the ordered character set. To determine the position of a character in your operating environment's character set, use the RANK function. The following example shows how to use RANK:

```
data _null_;
   x=rank('a');
   put "The position of a is " x ".";
run;
```

The SAS log prints the following message: `The position of a is 97.`

Each position in a translation table contains a hexadecimal number that is within the range of 0 ('00'x) to 255 ('FF'x). Hexadecimal values always end with an x. You can represent one or more consecutive hexadecimal values within quotation marks followed by a single x. For example, a string of three consecutive hexadecimal values can be written as '08090A'x. The SAS log displays each row of a translation table as 16 hexadecimal values enclosed in quotation marks followed by an x. The SAS log also lists reference numbers in the vertical and horizontal margins that correspond to the positions in the table. "Example 1: Viewing a Translation Table" on page 1091 shows how the SAS log displays a translation table.

Storing Translation Tables with PROC TRANTAB

When you use PROC TRANTAB to create a customized translation table, the procedure automatically stores the table in your Sasuser.Profile catalog. This lets you use customized translation tables without affecting other users. When you specify the translation table in the SORT procedure or in a GOPTIONS statement, the software first looks in your Sasuser.Profile catalog to find the table. If the specified translation table is not in your Sasuser.Profile catalog, the software looks in the Sashelp.Host catalog.

To access your translation table globally, your SAS Installation Representative must copy the table from your Sasuser.Profile catalog (using the CATALOG

procedure) to the Sashelp.Host catalog. If the table is not found there, the software continues to search in Sashelp.Locale for it.

Modifying SAS Translation Tables with PROC TRANTAB

If a translation table that is provided by SAS does not meet your needs, you can use PROC TRANTAB to edit it and create a new table. That is, you can issue the PROC TRANTAB statement that specifies the SAS table, edit the table, and then save the table using the SAVE statement. The modified translation table is saved in your Sasuser.Profile catalog. If you are a SAS Installation Representative, you can modify a translation table with PROC TRANTAB and then use the CATALOG procedure to copy the modified table from your Sasuser.Profile catalog to the Sashelp.Host catalog, as shown in the following example:

```
proc catalog c=sasuser.profile;
   copy out=sashelp.host entrytype=trantab;
run;
```

You can use PROC TRANTAB to modify translation tables stored in the Sashelp.Host catalog only if you have update (or write) access to that data library and catalog.

Using Translation Tables Outside PROC TRANTAB

Using Translation Tables in the SORT Procedure

PROC SORT uses translation tables to determine the sort's collating sequence. You can specify an alternative translation table with the SORTSEQ= option of PROC SORT. For example, if your operating environment sorts with the EBCDIC sequence by default, and you want to sort with the ASCII sequence, you can issue the following statement to specify the ASCII translation table:

```
proc sort sortseq=ascii;
run;
```

You can also create a customized translation table with PROC TRANTAB and specify the new table with PROC SORT. This table is useful when you want to specify sorting sequences for languages other than U.S. English.

See "Example 6: Using Different Translation Tables for Sorting" on page 1101 for an example that uses translation tables to sort data in different ways. For information about the tables available for sorting and the SORTSEQ= option, see "SORTSEQ= System Option" on page 999.

Transcoding External Files

Translation tables are implicitly set by the LOCALE= system option. They are used only for transcoding external files.

You can list the translation tables with the OPTIONS procedure. The following example uses the z/OS environment with a LOCALE= system option set to en_US:

```
TRANTAB=(eol1wlt1,wlt1eol1,elat_ucs,elat_lcs,elat_ccl,,,elat_scc)
```

Using Translation Tables in SAS/GRAPH Software

In SAS/GRAPH software, translation tables are most commonly used in an IBM operating environment, where tables are necessary because graphics commands must leave IBM operating environments in EBCDIC representation but must reach asynchronous graphics devices in ASCII representation. Specifically, SAS/GRAPH software builds the command stream for these devices internally in ASCII representation. However, it must convert the commands to EBCDIC representation before they can be given to the communications software for transmission to the device. SAS/GRAPH software uses a translation table internally to make the initial conversion from ASCII to EBCDIC. The communications software then translates the command stream back to ASCII representation before it reaches the graphics device.

Translation tables are operating environment-specific. In most cases, you can simply use the default translation table, SASGTAB0, or one of the SAS supplied graphics translation tables. However, if these tables cannot perform all of the translation correctly, you can create your own translation table with PROC TRANTAB. The SASGTAB0 table might fail to do the translation correctly when it encounters characters from languages other than U.S. English.

To specify an alternative translation table for SAS/GRAPH software, you can either use the TRANTAB= option in a GOPTIONS statement or modify the TRANTAB device parameter in the device entry. For example, the following GOPTIONS statement specifies the GTABTCAM graphics translation table:

```
goptions trantab=gtabtcam;
```

Translation tables used in SAS/GRAPH software perform both device-to-operating environment translation and operating environment-to-device translation. Therefore, a translation table consists of 512 bytes, with the first 256 bytes used to perform device-to-operating environment translation (ASCII to EBCDIC on IBM mainframes) and the second 256 bytes used to perform operating environment-to-device translation (EBCDIC to ASCII on IBM mainframes). For PROC TRANTAB, the area of a translation table for device-to-operating environment translation is considered to be table 1, and the area for operating environment-to-device translation is considered to be table 2. See “Example 1: Viewing a Translation Table”

on page 1091 for a listing of the ASCII translation table (a SAS provided translation table), which shows both areas of the table.

For operating environments other than IBM mainframes, translation tables can be used to translate specific characters in the data stream that are created by the driver. For example, if the driver normally generates a vertical bar in the data stream, but you want another character to be generated in place of the vertical bar, you can create a translation table that translates the vertical bar to an alternate character.

SAS/GRAPH software also uses key maps and device maps to map codes generated by the keyboard to specified characters and to map character codes to codes required by the graphics output device. These maps are specific to SAS/GRAPH software. For more information, contact SAS Technical Support.

Syntax: TRANTAB Procedure

Tip: Supports RUN group processing

PROC TRANTAB TABLE=*table-name* <NLS>;
CLEAR <ONE | TWO | BOTH>;
INVERSE;
LIST <ONE | TWO | BOTH>;
LOAD TABLE=*table-name* <NLS>;
REPLACE *position value-1* < *...value-n*>;
SAVE <TABLE=*table-name*> <CAT=*libname.catalog* > <ONE | TWO | BOTH>;
SWAP;

Statement	Task
PROC TRANTAB	Creates, edits, or displays a translations table
CLEAR	Sets all positions in the translation table to 0
INVERSE	Creates an inverse of table 1
LIST	Displays a translation table in hexadecimal representation
LOAD	Loads a translation table into memory for editing
REPLACE	Replaces the characters in a translation table with specified values
SAVE	Saves the translation table in your Sasuser.Profile catalog
SWAP	Exchanges table 1 with table 2

PROC TRANTAB Statement

Creates, edits, or displays a translation table.

Tip: If there is an incorrect table name in the PROC TRANTAB statement, use the LOAD statement to load the correct table. You do not need to reinvoke PROC TRANTAB. New tables are not stored in the catalog until you issue the SAVE statement, so you do not have unwanted tables in your catalog.

Syntax

PROC TRANTAB TABLE=*table-name* <NLS>;

Required Argument

TABLE=*table-name*
specifies the translation table to create, edit, or display the table. The specified table name must be a valid one-level SAS name with no more than eight characters.

Optional Argument

NLS
specifies that the table listed in the TABLE= argument is one of five special internal translation tables provided with SAS. You must use the NLS option when you specify one of the five special tables in the TABLE= argument. NLS stands for National Language Support. This option and the associated translation tables provide a method to translate characters that exist in languages other than U.S. English. To use the modified NLS table, specify its name in the SAS system option TRANTAB= . When you load one of these special translation tables, the SAS log displays a note stating that table 2 is uninitialized. That is, table 2 is an empty table that contains all 0s. PROC TRANTAB does not use table 2 for translation in these special cases.

SASXPT
specifies the local-to-transport format translation table (used by the CPORT procedure).

SASLCL
specifies the transport-to-local format translation table (used by the CIMPORT procedure).

SASUCS
specifies the lowercase-to-uppercase translation table (used by the UPCASE function).

SASLCS
specifies the uppercase-to-lowercase translation table (used by the LOWCASE macro).

SASCCL
specifies the character classification table (used internally), which contains flag bytes that correspond to each character position. These positions indicate the class or classes to which each character belongs.

CLEAR Statement

Sets all positions in the translation table to 0; used when you create a new table.

Syntax

CLEAR <ONE | TWO | BOTH>;

Optional Argument

ONE | TWO | BOTH

ONE
clears table 1.

TWO
clears table 2.

BOTH
clears both table 1 and table 2.

Default ONE

INVERSE Statement

Creates an inverse of table 1 in a translation table. The INVERSE statement creates table 2.

Syntax

INVERSE;

Details

The INVERSE statement does not preserve multiple translations. Suppose table 1 has 2 (or more) different characters translated to the same value; for example, "A" and "B" are both translated to "1". For table 2, the INVERSE statement uses the last translated character for the value. "1" is always translated to "B" and not "A", assuming that "A" appears before "B" in the first table. Sort programs in SAS require an inverse table for proper operation.

LIST Statement

Displays a translation table in hexadecimal representation in the SAS log.

Syntax

LIST <ONE | TWO | BOTH>;

Optional Argument

ONE | TWO | BOTH

ONE
displays table 1.

TWO
displays table 2.

BOTH
displays both table 1 and table 2.

Default ONE

LOAD Statement

Loads a translation table into memory for editing.

Tips: Use the LOAD statement when you specify an incorrect table name in the PROC TRANTAB statement. You can specify the correct name without reinvoking the procedure.

Use the LOAD statement to edit multiple translation tables in a single PROC TRANTAB step. (Be sure to save the first table before you load another one.)

Syntax

LOAD TABLE=*table-name* <NLS>;

Required Argument

TABLE=*table-name*
specifies the name of an existing translation table to edit. The specified table name must be a valid one-level SAS name.

Optional Argument

NLS
specifies that the table listed in the TABLE= argument is one of five special internal translation tables that are provided with SAS. You must use the NLS option when you specify one of the five special tables in the TABLE= argument:

SASXPT
specifies the local-to-transport format translation table.

SASLCL
specifies the transport-to-local format translation table.

SASUCS
specifies the lowercase-to-uppercase translation table.

SASLCS
specifies the uppercase-to-lowercase translation table.

SASCCL
specifies the character classification table, which contains flag bytes that correspond to each character position. These positions indicate the class or classes to which each character belongs.

NLS stands for National Language Support. This option and the associated translation tables provide a method to map characters from languages other than English to programs, displays, and files. When you load one of these special translation tables, the SAS log displays a note that states that table 2 is uninitialized. That is, table 2 is an empty table that contains all 0s. PROC TRANTAB does not use table 2 for translation in these special cases.

REPLACE Statement

Replaces characters in a translation table with the specified values, starting at the specified position.

Alias: REP

Tip: To save edits, you must issue the SAVE statement.

Syntax

REPLACE *position value-1* < *...value-n*>;

Required Arguments

position
specifies the position in a translation table where the replacement begins. The editable positions in a translation table begin at decimal 0 and end at decimal 255. To specify the position, you can do either of the following:

- Use a decimal or hexadecimal value to specify an actual location. If you specify a decimal value (for example, 20), PROC TRANTAB locates position 20 in the table, which is byte 21. If you specify a hexadecimal value (for example, '14'x), PROC TRANTAB locates the decimal position that is equivalent to the specified hexadecimal value, which in this case is position 20 (or byte 21) in the table.
- Use a quoted character. PROC TRANTAB locates the quoted character in the table (that is, the quoted character's hexadecimal value) and uses that character's position as the starting position. For example, specifying the following REPLACE statement replaces the first occurrence of the hexadecimal value for "a" and the next two hexadecimal values with the hexadecimal equivalent of "ABC": `replace 'a' 'ABC';`

  This action is useful when you want to locate alphabetic and numeric characters, but you do not know their actual location. If the quoted character is not found, PROC TRANTAB displays an error message and ignores the statement.

  To edit positions 256 through 511 (table 2), follow these steps:

1. Issue the SWAP statement.
2. Issue the appropriate REPLACE statement.
3. Issue the SWAP statement again to reposition the table.

value-1 < ...value-n>
is one or more decimal, hexadecimal, or character constants that specify the actual value to place in the table, starting at *position*. You can also use a mixture of these values. That is, you can specify a decimal, a hexadecimal, and a character value in one REPLACE statement. "Example 3: Editing By Specifying a Decimal Value for the Starting Position" on page 1095 shows a mixture of all three types of values in the REPLACE statement.

SAVE Statement

Saves the translation table in your Sasuser.Profile catalog.

Syntax

SAVE <TABLE=*table-name*> <CAT=*libname.catalog*> <ONE | TWO | BOTH>;

Optional Arguments

TABLE=*table-name*
specifies the name under which the current table is saved. The name must be a valid one-level SAS name.

Default If you omit the TABLE= option, the current table is saved under the name you specify in the PROC TRANTAB statement or in the LOAD statement.

CAT=*libname.catalog*
specifies the name of the output catalog when you are saving a TRANTAB entry.

ONE | TWO | BOTH

ONE
saves table 1.

TWO
saves table 2.

BOTH
saves both table 1 and table 2.

Default BOTH

SWAP Statement

Exchanges table 1 with table 2 to enable you to edit positions 256 through 511.

Tip: After you edit the table, you must issue the SWAP statement again to reposition the table.

Syntax

SWAP;

Examples: TRANTAB Procedure

Example 1: Viewing a Translation Table

Features: LIST statement

This example uses PROC TRANTAB to display the ASCII translation table supplied by SAS. All examples were produced in the Linux environment.

Set the options and specify a translation table.

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=ascii;
```

Display both halves of the translation table. The LIST BOTH statement displays both the table that provides the translation and the table that provides the inverse translation.

```
  list both;
```

SAS Log

```
NOTE: Table specified is ASCII.
ASCII table 1:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '606162636465666768696A6B6C6D6E6F'x
     70 '707172737475767778797A7B7C7D7E7F'x
     80 '808182838485868788898A8B8C8D8E8F'x
     90 '909192939495969798999A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x


ASCII table 2:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '606162636465666768696A6B6C6D6E6F'x
     70 '707172737475767778797A7B7C7D7E7F'x
     80 '808182838485868788898A8B8C8D8E8F'x
     90 '909192939495969798999A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

Example 2: Creating a Translation Table

Features: Procedures features:
- LIST statement
- REPLACE statement
- SAVE statement

This example uses PROC TRANTAB to create a customized translation table. All examples were produced in the Linux environment.

Set the system options and specify the translation table to edit.

```
options nodate pageno=1 linesize=80
pagesize=60;
proc trantab table=newtable;
```

Replace characters in the translation table starting at a specified position. The REPLACE statement places the values in the table starting at position 0. You can use hexadecimal values of any length in the REPLACE statement. This example uses strings of length 16 to match how translation tables appear in the SAS log.

```
replace 0
'00010203a309e57ff9ecc40b0c0d0e0f'x
'10111213a5e008e71819c6c51c1d1e1f'x
'c7fce9e2e40a171beaebe8efee050607'x
'c9e616f4f6f2fb04ffd6dca2b6a7501a'x
'20e1edf3faf1d1aababfa22e3c282b7c'x
'265facbdbca1abbb5f5f21242a293bac'x
'2d2f5fa6a6a6a62b2ba6a62c255f3e3f'x
'a62b2b2b2b2b2b2d2d603a2340273d22'x
'2b6162636465666768692d2ba6a62b2b'x
'2d6a6b6c6d6e6f7071722da62d2b2d2d'x
'2d7e737475767778787a2d2b2b2b2b2b'x
'2b2b2b5f5fa65f5f5fdf5fb65f5fb55f'x
'7b41424344454647484955f5f5f5f5f5f'x
'7d4a4b4c4d4e4f5051525f5f5fb15f5f'x
'5c83535455565758595a5f5ff75f5fb0'x
'303132333435363738393b75f6eb25f5f'x
;
```

Save the table. The SAVE statement saves the table under the name that is specified in the PROC TRANTAB statement. By default, the table is saved in your Sasuser.Profile catalog.

```
save;
```

Display both halves of the translation table in the SAS log. The LIST BOTH statement displays both the table that provides the translation and the table that provides the inverse translation.

```
list both;
```

SAS Log

```
 -->Create
and edit table 2. table 2 is
empty; that is, it
consists entirely of 0s. To create table 2, you can use the INVERSE statement.
(See: SAS(R) Statements: Reference)


.) To edit table 2, you can use the SWAP statement with the REPLACE statement.
(See: SAS(R) Statements: Reference)

.)" commented-out by old2new conversion -->

 NOTE: Table specified is NEWTABLE.
WARNING: Table NEWTABLE not found! New table is assumed.
NOTE: NEWTABLE table 1 is uninitialized.
NOTE: NEWTABLE table 2 is uninitialized.

NOTE: Saving table NEWTABLE.
NOTE: NEWTABLE table 2 will not be saved because it is uninitialized.
NEWTABLE table 1:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '00010203A309E57FF9ECC40B0C0D0E0F'x
     10 '10111213A5E008E71819C6C51C1D1E1F'x
     20 'C7FCE9E2E40A171BEAEBE8EFEE050607'x
     30 'C9E616F4F6F2FB04FFD6DCA2B6A7501A'x
     40 '20E1EDF3FAF1D1AABABFA22E3C282B7C'x
     50 '265FACBDBCA1ABBB5F5F21242A293BAC'x
     60 '2D2F5FA6A6A6A62B2BA6A62C255F3E3F'x
     70 'A62B2B2B2B2B2B2D2D603A2340273D22'x
     80 '2B6162636465666768692D2BA6A62B2B'x
     90 '2D6A6B6C6D6E6F7071722DA62D2B2D2D'x
     A0 '2D7E737475767778787A2D2B2B2B2B2B'x
     B0 '2B2B2B5F5FA65F5F5FDF5FB65F5FB55F'x
     C0 '7B41424344454647484955F5F5F5F5F5F'x
     D0 '7D4A4B4C4D4E4F5051525F5F5FB15F5F'x
     E0 '5C83535455565758595A5F5FF75F5FB0'x
     F0 '30313233343536373839B75F6EB25F5F'x



NOTE: NEWTABLE table 2 is uninitialized.
NEWTABLE table 2:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '00000000000000000000000000000000'x
     10 '00000000000000000000000000000000'x
     20 '00000000000000000000000000000000'x
     30 '00000000000000000000000000000000'x
     40 '00000000000000000000000000000000'x
     50 '00000000000000000000000000000000'x
     60 '00000000000000000000000000000000'x
     70 '00000000000000000000000000000000'x
     80 '00000000000000000000000000000000'x
     90 '00000000000000000000000000000000'x
     A0 '00000000000000000000000000000000'x
     B0 '00000000000000000000000000000000'x
     C0 '00000000000000000000000000000000'x
     D0 '00000000000000000000000000000000'x
     E0 '00000000000000000000000000000000'x
     F0 '00000000000000000000000000000000'x
```

Example 3: Editing By Specifying a Decimal Value for the Starting Position

Features:
LIST statement
REPLACE statement
SAVE statement

This example edits the translation table that was created in "Example 2: Creating a Translation Table" on page 1092. The decimal value specified in the REPLACE statement marks the starting position for the table changes.

The vertical arrow in both SAS logs marks the point at which the changes begin. All examples were produced in the Linux environment.

Program 1: Display the Original Table

```
   options nodate pageno=1 linesize=80 pagesize=60; proc trantab
table=newtable;

   list one;
```

Program Description

Set the system options and specify the translation table to edit.

```
   options nodate pageno=1 linesize=80 pagesize=60; proc trantab
table=newtable;
```

Display the original table. This LIST statement displays the original NEWTABLE translation table.

```
   list one;
```

SAS Log

Original NEWTABLE Translation Table

```
Table specified is NEWTABLE.
NOTE: NEWTABLE table 2 is uninitialized.
NEWTABLE table 1:
                              ↓
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '00010203A309E57FF9ECC40B0C0D0E0F'x
     10 '10111213A5E008E71819C6C51C1D1E1F'x
     20 'C7FCE9E2E40A171BEAEBE8EFEE050607'x
     30 'C9E616F4F6F2FB04FFD6DCA2B6A7501A'x
     40 '20E1EDF3FAF1D1AABABFA22E3C282B7C'x
     50 '265FACBDBCA1ABBB5F5F21242A293BAC'x
     60 '2D2F5FA6A6A6A62B2BA6A62C255F3E3F'x
     70 'A62B2B2B2B2B2B2D2D603A2340273D22'x
     80 '2B6162636465666768692D2BA6A62B2B'x
     90 '2D6A6B6C6D6E6F7071722DA62D2B2D2D'x
     A0 '2D7E737475767778787A2D2B2B2B2B2B'x
     B0 '2B2B2B5F5FA65F5F5FDF5FB65F5FB55F'x
     C0 '7B41424344454647484955F5F5F5F5F5F'x
     D0 '7D4A4B4C4D4E4F5051525F5F5FB15F5F'x
     E0 '5C83535455565758595A5F5FF75F5FB0'x
     F0 '30313233343536373839B75F6EB25F5F'x
```

Program 2: Edit the Table

```
replace 10
   20 10 200 'x' 'ux' '092040'x;

save;

list one;
```

Program Description

Replace characters in the translation table, starting at a specified position. The REPLACE statement starts at position decimal 10, which is byte 11 in the original table, and performs a byte-to-byte replacement with the given values.

```
replace 10
   20 10 200 'x' 'ux' '092040'x;
```

Save the changes. The SAVE statement saves the changes that you made to the NEWTABLE translation table.

```
save;
```

Display the new table. The second LIST statement displays the edited NEWTABLE translation table.

```
list one;
```

SAS Log

```
Saving table NEWTABLE.
NOTE: NEWTABLE table 2 will not be saved because it is uninitialized.
NEWTABLE table 1:
                               ↓
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '00010203A309E57FF9EC140AC8787578'x
     10 '09204013A5E008E71819C6C51C1D1E1F'x
     20 'C7FCE9E2E40A171BEAEBE8EFEE050607'x
     30 'C9E616F4F6F2FB04FFD6DCA2B6A7501A'x
     40 '20E1EDF3FAF1D1AABABFA22E3C282B7C'x
     50 '265FACBDBCA1ABBB5F5F21242A293BAC'x
     60 '2D2F5FA6A6A6A62B2BA6A62C255F3E3F'x
     70 'A62B2B2B2B2B2B2D2D603A2340273D22'x
     80 '2B6162636465666768692D2BA6A62B2B'x
     90 '2D6A6B6C6D6E6F7071722DA62D2B2D2D'x
     A0 '2D7E737475767778787A2D2B2B2B2B2B'x
     B0 '2B2B2B5F5FA65F5F5FDF5FB65F5FB55F'x
     C0 '7B414243444454647484 95F5F5F5F5F5F'x
     D0 '7D4A4B4C4D4E4F5051525F5F5FB15F5F'x
     E0 '5C83535455565758595A5F5FF75F5FB0'x
     F0 '303132333435363738 39B75F6EB25F5F'x
```

Output Details

At position 10 (which is byte 11), a vertical arrow denotes the starting point for the changes to the translation table.

At byte 11, decimal 20 (which is hexadecimal 14) replaces hexadecimal C4.

At byte 12, decimal 10 (which is hexadecimal 0A) replaces hexadecimal 0B.

At byte 13, decimal 200 (which is hexadecimal C8) replaces hexadecimal 0C.

At byte 14, character 'x' (which is hexadecimal 78) replaces hexadecimal 0D.

At bytes 15 and 16, characters 'ux' (which are hexadecimal 75 and 78, respectively) replace hexadecimal 0E and 0F.

At bytes 17, 18, and 19, hexadecimal 092040 replaces hexadecimal 101112.

Example 4: Editing By Using a Quoted Character for the Starting Position

Features: LIST statement
LOAD statement
REPLACE statement
SAVE statement

This example creates a new translation table by editing the ASCII translation table. The first occurrence of the hexadecimal equivalent of the quoted character that was specified in the REPLACE statement is the starting position for the changes to the table. This method differs from "Example 3: Editing By Specifying a Decimal Value for the Starting Position" on page 1095 because you do not need to know the exact position at which to start the changes to the table. PROC TRANTAB finds the correct position for you.

The edited table is saved under a new name. Horizontal arrows in both SAS logs denote the edited rows in the translation table. All examples were produced in the Linux environment.

Program 1: Display the Original Table

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=ascii;

list one;
```

Program Description

Set the system options and specify which translation table to edit.

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=ascii;
```

Display the translation table. The LIST statement displays the original translation table in the SAS log.

```
list one;
```

SAS Log

```
NOTE: Table specified is ASCII.
ASCII table 1:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '606162636465666768696A6B6C6D6E6F'x   ←
     70 '707172737475767778797A7B7C7D7E7F'x   ←
     80 '808182838485868788898A8B8C8D8E8F'x
     90 '909192939495969798999A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

Program 2: Edit the Table

```
replace 'a' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

save table=upper;

load table=upper;
   list one;
```

Program Description

Replace characters in the translation table, starting at a specified position. The REPLACE statement finds the first occurrence of the hexadecimal "a" (which is 61) and replaces it and the next 25 hexadecimal values with the hexadecimal values for uppercase "A" through "Z."

```
replace 'a' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
```

Save your changes. The SAVE statement saves the changes made to the ASCII translation table under the new table name UPPER. The stored contents of the ASCII translation table remain unchanged.

```
save table=upper;
```

Load and display the translation table. The LOAD statement loads the edited translation table UPPER. The LIST statement displays the translation table UPPER in the SAS log.

```
load table=upper;
   list one;
```

SAS Log

```
NOTE: Table UPPER
being loaded.
UPPER table 1:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '604142434445464748494A4B4C4D4E4F'x   ←
     70 '505152535455565758595A7B7C7D7E7F'x   ←
     80 '808182838485868788898A8B8C8D8E8F'x
     90 '909192939495969798999A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

Example 5: Creating the Inverse of a Table

Features: INVERSE statement
LIST statement
SAVE statement

This example creates the inverse of the translation table that was created in "Example 4: Editing By Using a Quoted Character for the Starting Position" on page 1097. The new translation table that is created in this example is the operating environment-to-device translation for use in data communications.

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=upper;
```

Create the inverse translation table, save the tables, and then display them. The INVERSE statement creates table 2 by inverting the original table 1 (called UPPER). The SAVE statement saves the translation tables. The LIST BOTH statement displays both the original translation table and its inverse.

```
inverse;
   save;
   list both;
```

SAS Log

The INVERSE statement lists in the SAS log all of the multiple translations that it encounters as it inverts the translation table. In "Example 4: Editing By Using a Quoted Character for the Starting Position" on page 1097, all the lowercase letters are converted to uppercase in the translation table UPPER, which means that there are two sets of uppercase letters in UPPER. When INVERSE cannot make a translation, PROC TRANTAB fills the value with 00. Note that the inverse of the translation table UPPER has numerous 00 values.

The SAS log lists all the duplicate values that it encounters as it creates the inverse of table 1. To conserve space, most of these messages are deleted in this example.

```
NOTE: This table cannot be mapped one to one.
 duplicate of '41'x found at '61'x in table one.
 duplicate of '42'x found at '62'x in table one.
 duplicate of '43'x found at '63'x in table one.
   .
   .
   .
 duplicate of '58'x found at '78'x in table one.
 duplicate of '59'x found at '79'x in table one.
 duplicate of '5A'x found at '7A'x in table one.
NOTE: Saving table UPPER.
UPPER table 1:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '604142434445464748494A4B4C4D4E4F'x
     70 '505152535455565758595A7B7C7D7E7F'x
     80 '808182838485868788898A8B8C8D8E8F'x
     90 '909192939495969798999A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x


UPPER table 2:
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '600000000000000000000000000000000'x
     70 '00000000000000000000007B7C7D7E7F'x
     80 '808182838485868788898A8B8C8D8E8F'x
     90 '909192939495969798999A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

Example 6: Using Different Translation Tables for Sorting

Features: PROC SORT statement option: SORTSEQ=
PRINT procedure

This example shows how to specify a different translation table to sort data in an order that is different from the default sort order. Characters that are written in a language other than U.S. English might require a sort order that is different from the default order.

You can use the TRABASE program in the SAS Sample Library to create translation tables for several languages. All examples were produced in the Linux environment.

Set the SAS system options.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the TESTSORT data set. The DATA step creates a SAS data set with four pairs of words, each pair different only in the case of the first letter.

```
data testsort;
   input Values $10.;
datalines;
Always
always
Forever
forever
Later
later
Yesterday
yesterday
;
```

Sort the data in an order that is different from the default sort order. PROC SORT sorts the data by using the default translation table, which sorts all lowercase words first, then all uppercase words.

```
proc sort;
   by values;
run;
```

Print the data set. PROC PRINT prints the sorted data set.

```
proc print noobs;
   title 'Default Sort Sequence';
run;
```

SAS Output

The following output is the output from sorting values with the default translation table. The default sort sequence sorts all the capitalized words in alphabetical order before it sorts any lowercase words.

```
Default Sort Sequence
1

                                  Values

                                  Always
                                  Forever
                                  Later
                                  Yesterday
                                  always
                                  forever
                                  later
                                  yesterday
```

Sort the data according to the translation table UPPER and print the new data set. The SORTSEQ= system option specifies that PROC SORT sort the data according to the customized translation table UPPER, which treats lowercase and uppercase letters alike. This method is useful for sorting without regard for case. PROC PRINT prints the sorted data set.

```
proc sort sortseq=upper;
   by values;
run;
proc print noobs;
   title 'Customized Sort Sequence';
run;
```

SAS Output

The following output is the result from sorting values with a customized translation table. The customized sort sequence sorts all the words in alphabetical order, without regard for the case of the first letters.

```
                        Customized Sort Sequence                    2

                              Values

                              Always
                              always
                              Forever
                              forever
                              Later
                              later
                              Yesterday
                              yesterday
```

Example 7: Editing Table 1 and Table 2

Features: LIST statement
REPLACE statement

SAVE statement
SWAP statement

This example shows how to edit both areas of a translation table. To edit positions 256 through 511 (table 2), you must

- Issue the SWAP statement to have table 2 change places with table 1.
- Issue an appropriate REPLACE statement to make changes to table 2.
- Issue the SWAP statement again to reposition the table.
- Arrows in the SAS logs mark the rows and columns that are changed.

Set the SAS system options and specify the translation table.

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=upper;
```

Display the original translation table. The LIST statement displays the original UPPER translation table.

```
list both;
```

SAS Log

The following output is the original UPPER translation table.

```
NOTE: Table specified is UPPER.
UPPER table 1:
          ↓
         0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x   ←
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '604142434445464748494A4B4C4D4E4F'x
     70 '505152535455565758595A7B7C7D7E7F'x
     80 '808182838485868788898A8B8C8D8E8F'x
     90 '909192939495969798999A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x

UPPER table 2:
          ↓
         0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000102030405060708090A0B0C0D0E0F'x   ←
     10 '101112131415161718191A1B1C1D1E1F'x
     20 '202122232425262728292A2B2C2D2E2F'x
     30 '303132333435363738393A3B3C3D3E3F'x
     40 '404142434445464748494A4B4C4D4E4F'x
     50 '505152535455565758595A5B5C5D5E5F'x
     60 '60000000000000000000000000000000'x
     70 '00000000000000000000007B7C7D7E7F'x
     80 '808182838485868788898A8B8C8D8E8F'x
     90 '909192939495969798999A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

Replace characters in the translation table starting at a specified position. The REPLACE statement starts at position 1 and replaces the current value of 01 with '0A'.

```
replace 1 '0A'x;
```

Prepare table 2 for editing. The first SWAP statement positions table 2 so that it can be edited. The second REPLACE statement makes the same change in table 2 that was made in table 1.

```
swap;
   replace 1 '0A'x;
```

Save and display the tables in their original positions. The second SWAP statement restores table 1 and table 2 to their original positions. The SAVE statement saves both areas of the translation table by default. The LIST statement displays both areas of the table.

```
swap;
   save;
```

```
list both;
```

SAS Log

The upper table 1 is modified. The hexadecimal value ‘0A’ replaces hexadecimal value 01 in byte 2 for both areas of the translation table. Arrows mark the rows and columns of the table in which this change is made.

```
NOTE: Table specified is UPPER.
UPPER table 1:
           [darr]
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000A02030405060708090A0B0C0D0E0F'x   <--
     10 '10111213141516171819 1A1B1C1D1E1F'x
     20 '20212223242526272829 2A2B2C2D2E2F'x
     30 '30313233343536373839 3A3B3C3D3E3F'x
     40 '40414243444546474849 4A4B4C4D4E4F'x
     50 '50515253545556575859 5A5B5C5D5E5F'x
     60 '60414243444546474849 4A4B4C4D4E4F'x
     70 '50515253545556575859 5A7B7C7D7E7F'x
     80 '80818283848586878889 8A8B8C8D8E8F'x
     90 '90919293949596979899 9A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x

UPPER table 2:
           [darr]
          0 1 2 3 4 5 6 7 8 9 A B C D E F
     00 '000A02030405060708090A0B0C0D0E0F'x   <--
     10 '10111213141516171819 1A1B1C1D1E1F'x
     20 '20212223242526272829 2A2B2C2D2E2F'x
     30 '30313233343536373839 3A3B3C3D3E3F'x
     40 '40414243444546474849 4A4B4C4D4E4F'x
     50 '50515253545556575859 5A5B5C5D5E5F'x
     60 '60000000000000000000000000000000'x
     70 '00000000000000000000007B7C7D7E7F'x
     80 '80818283848586878889 8A8B8C8D8E8F'x
     90 '90919293949596979899 9A9B9C9D9E9F'x
     A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
     B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
     C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
     D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
     E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
     F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

PART 11

Values for Locale, Encoding, and Transcoding

Chapter 21
Values for the LOCALE= System Option **1109**

Chapter 22
SAS System Options for Processing DBCS Data **1125**

Chapter 23
Encoding Values in SAS Language Elements **1127**

Chapter 24
Encoding Values for a SAS Session **1139**

21

Values for the LOCALE= System Option

Locale-to-Encoding Mapping 1109

Locale-to-Encoding Mapping

This 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. The settings for 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:

```
DATESTYLE=DMY
PAPERSIZE=A4
```

Table 21.1 lists the SAS name, the alias, the POSIX value, the LINUX encoding value, the DATESTYLE= system option, and the PAPERSIZE= system option value.

Table 21.1 *Default Values for the DATESTYLE= and PAPERSIZE= System Options Are Based on the POSIX Locale and Linux Encoding Values*

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
Afrikaans_SouthAfrica alias: Afrikaans, af	af_ZA	latin1	YMD	A4
Albanian_Albania alias: Albanian, sq	sq_AL	latin2	YMD	A4

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
Arabic_Bahrain	ar_BH	arabic	DMY	A4
Arabic_Algeria	ar_DZ	arabic	DMY	A4
Arabic_Egypt	ar_EG	arabic	DMY	A4
Arabic_India	ar_IN	arabic	DMY	A4
Arabic_Iraq	ar_IQ	arabic	DMY	A4
Arabic_Jordan	ar_JO	arabic	DMY	A4
Arabic_Kuwait	ar_KW	arabic	DMY	A4
Arabic_Lebanon	ar_LB	arabic	DMY	A4
Arabic_Libya	ar_LY	arabic	DMY	A4
Arabic_Morocco	ar_MA	arabic	DMY	A4
Arabic_Oman	ar_OM	arabic	DMY	A4
Arabic_Qatar	ar_QA	arabic	DMY	A4
Arabic_SaudiArabia	ar_SA	arabic	DMY	A4
Arabic_Sudan	ar_SD	arabic	DMY	A4
Arabic_Syria	ar_SY	arabic	DMY	A4
Arabic_Tunisia	ar_TN	arabic	DMY	A4
Arabic_UnitedArabEmirates alias: Arabic, ar	ar_AE	arabic	DMY	A4
Arabic_Yemen	ar_YE	arabic	DMY	A4
Basque_Spain alias: Basque, Euskara, eu	eu_ES	latin9	DMY	A4
Bengali_India alias: Bengali, bn	bn_IN	latin1	DMY	A4
Bosnian_BosniaHerzegovina alias: Bosnian, bs	bs_BA	latin2	YMD	A4
Bulgarian_Bulgaria alias: Bulgarian, bg	bg_BG	cyrillic	YMD	A4

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
Byelorussian_Belarus alias: Byelorussian, Belarusian, Byelorussian_Belarus, be	be_BY	cyrillic	DMY	A4
Catalan_Spain alias: Catalan, ca	ca_ES	latin1	DMY	A4
Chinese_China alias:Chinese, zh_HANS, zh_HANS_CN	zh_CN	euc-cn	YMD	A4
Chinese_HongKong alias: zh_HANT_HK	zh_HK	euc-tw, ms-950[1]	YMD	A4
Chinese_Macau alias: zh_HANT_MO	zh_MO	euc-tw, mis-950[1]	YMD	A4
Chinese_Singapore alias: zh_HANS_SG	zh_SG	euc-cn	DMY	A4
Chinese_Taiwan alias: zh_HANT_TW	zh_TW	euc-tw, ms-950[1]	YMD	A4
Cornish_UnitedKingdom alias: Cornish, kw	kw_GB	latin1	DMY	A4
Croatian_BosniaHerzegovina	hr_BA	latin2	YMD	A4
Croatian_Croatia alias: Croatian, hr	hr_HR	latin2	YMD	A4
Czechia alias: Czech_CzechRepublic, cs	cs_CZ	latin2	DMY	A4
Danish_Denmark alias: Danish, da	da_DK	latin9	DMY	A4
Dutch_Belgium	nl_BE	latin1	DMY	A4
Dutch_Netherlands alias: Dutch, nl	nl_NL	latin1	DMY	A4
English_Australia	en_AU	latin1	DMY	A4

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
English_Belgium	en_BE	latin9	DMY	A4
English_Botswana	en_BW	latin1	DMY	A4
English_Canada	en_CA	latin1	DMY	letter
English_Caribbean	en_CB	latin1	MDY	A4
English_HongKong	en_HK	latin9	DMY	A4
English_Ireland	en_IE	latin9	DMY	A4
English_India	en_IN	latin9	DMY	A4
English_Jamaica	en_JM	latin1	DMY	A4
English_Malta	en_MT	latin9	DMY	A4
English_NewZealand	en_NZ	latin1	DMY	A4
English_Philippines	en_PH	latin1	MDY	letter
English_Singapore	en_SG	latin9	DMY	A4
English_SouthAfrica	en_ZA	latin1	DMY	A4
English_UnitedStates alias: English, en	en_US	latin1	MDY	letter
English_UnitedKingdom	en_GB	latin9	DMY	A4
English_Zimbabwe	en_ZW	latin1	DMY	A4
Estonian_Estonia	et_EE	latin6	DMY	A4
Faroese_FaroeIslands alias: Faroese, fo	fo_FO	latin1	DMY	A4
Finnish_Finland alias: Finnish, fi	fi_FI	latin9	DMY	A4
French_Belgium	fr_BE	latin9	DMY	A4
French_Canada	fr_CA	latin1	DMY	letter
French_France alias: French, fr	fr_FR	latin9	DMY	A4
French_Luxembourg	fr-LU	latin9	DMY	A4

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
French_Switzerland	fr_CH	latin9	DMY	A4
German_Austria	de_AT	latin9	DMY	A4
German_Germany alias: German, de	de_DE	latin9	DMY	A4
German_Liechtenstein	de_LI	latin9	DMY	A4
German_Luxembourg	de_LU	latin9	DMY	A4
German_Switzerland	de_CH	latin9	DMY	A4
Greek_Cyprus	el_CY	greek	DMY	A4
Greek_Greece alias: Greek, el	el_GR	greek	DMY	A4
Greenlandic_Greenland alias: Greenlandic, kl	kl_GL	latin1	DMY	A4
Hebrew_Israel alias: Hebrew, he	he_IL	hebrew	DMY	A4
Hindi_India alias: Hindi, hi	hi_IN	latin1	DMY	A4
Hungarian_Hungary alias: Hungarian, hu	hu_HU	latin2	YMD	A4
Icelandic_Iceland alias: Icelandic, is	is_IS	latin1	DMY	A4
Indonesian_Indonesia alias: Indonesian, id	id_ID	latin1	DMY	A4
Irish_Ireland alias: Irish, ga	ga_IE	Latin8	DMY	A4
Italian_Italy alias: Italian, it	it_IT	latin9	DMY	A4
Italian_Switzerland	it_CH	latin9	DMY	A4
Japanese_Japan alias: Japanese, ja	ja_JP	euc-jp, shift-jis[1]	YMD	A4

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
Kazakh_Kazakhstan alias: Kazakh, kk	kk_KZ	Cyrillic	DMY	A4
Korean_Korea alias: Korean, ko, Korean_SouthKorea	ko_KR	euc-kr	YMD	A4
Latvian_Latvia alias: Latvian, lv	lv_LV	latin6	YMD	A4
Lithuanian_Lithuania alias: Lithuanian, lt	lt_LT	latin6	YMD	A4
Macedonian_Macedonia alias: Macedonian, mk	mk_MK	cyrillic	DMY	A4
Malay_Malaysia alias: Malay, ms	ms_MY	latin1	DMY	A4
Maltese_Malta	mt_MT	latin3	DMY	A4
Marathi_India alias: Marathi, mr	mr_IN	latin1	DMY	A4
NorwegianBokmal_Norway alias: NorwegianBokmal, nb	nb_NO	latin9	DMY	A4
NorwegianNynorsk_Norway alias: NorwegianNynorsk, nn	nn_NO	latin9	DMY	A4
Norwegian_Norway alias: Norwegian, no	no_NO	latin9	DMY	A4
Persian_India	fa_IN	arabic	YMD	A4
Persian_Iran alias: Persian, fa	fa_IR	arabic	YMD	A4
Polish_Poland alias: Polish, pl	pl_PL	latin2	YMD	A4
Portuguese_Brazil	pt_BR	latin1	DMY	A4
Portuguese_Portugal alias: Portuguese, pt	pt_PT	latin1	DMY	A4

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
Romanian_Romania alias: Romanian, ro	ro_RO	latin2	DMY	A4
Russian_Russia alias: Russian, ru	ru_RU	cyrillic	DMY	A4
Russian_Ukraine	ru_UA	cyrillic	DMY	A4
Serbian_BosniaHerzegovina	sr_BA	cyrillic	DMY	A4
Serbian_Montenegro	sr_ME	cyrillic	DMY	A4
Serbian_Serbia alias: sr_RS, sr_RS	sr_RS sr_CYRL_RS	cyrillic	DMY	A4
SerbianLatin_BosniaHerzegovina	sh_BA	latin2	DMY	A4
SerbianLatin_Montenegro	sh_ME	latin2	DMY	A4
SerbianLatin_Serbia alias: sh_LATN, sh_LATN_RS	sh_RS sh_RS	latin2	DMY	A4
Slovak_Slovakia alias: Slovak, Slovakian,Slovakian_Slovakia , sk	sk_SK	latin2	DMY	A4
Slovenian_Slovenia alias: Slovenian, si	sl_SI	latin2	YMD	A4
Spanish_Argentina	es_AR	latin1	DMY	A4
Spanish_Bolivia	es_BO	latin1	DMY	A4
Spanish_Chile	es_CL	latin1	DMY	letter
Spanish_Colombia	es_CO	latin1	DMY	letter
Spanish_CostaRica	es_CR	latin1	DMY	letter
Spanish_DominicanRepublic	es_DO	latin1	DMY	A4
Spanish_Ecuador	es_EC	latin1	DMY	A4
Spanish_ElSalvador	es_SV	latin1	MDY	letter
Spanish_Guatemala	es_GT	latin1	DMY	letter

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
Spanish_Honduras	es_HN	latin1	MDY	A4
Spanish_Mexico	es_MX	latin1	DMY	letter
Spanish_Nicaragua	es_NI	latin1	MDY	letter
Spanish_Panama	es_PA	latin1	MDY	letter
Spanish_Paraguay	es_PY	latin1	DMY	A4
Spanish_Peru	es_PE	latin1	DMY	A4
Spanish_PuertoRico	es_PR	latin1	MDY	letter
Spanish_Spain alias: Spanish, es	es_ES	latin9	DMY	A4
Spanish_UnitedStates	es_US	latin1	DMY	letter
Spanish_Uruguay	es_UY	latin1	DMY	A4
Spanish_Venezuela	es_VE	latin1	DMY	letter
Swedish_Sweden alias: Swedish, sv	sv_SE	latin9	YMD	A4
Tagalog_Philippines alias: Tagalog, Filipino, Filipino_Philippines, ti	ti_ph	latin1	MDY	letter
Tamil_India alias: Tamil, ta	ta_IN	latin1	DMY	A4
Telugu_India alias: Telugu, te	te_IN	latin1	DMY	A4
Thai_Thailand alias: Thai, th	th_TH	thai	DMY	A4
Turkish_Turkey alias: Turkish, tr	tr_TR	latin5	DMY	A4
Ukrainian_Ukraine alias: Ukrainian, uk	uk_UA	cyrillic	DMY	A4
Vietnamese_Vietnam	vi_VN	latin1	DMY	A4

SAS Name	POSIX Locale	LINUX Encoding	DATESTYLE=	PAPERSIZE=
alias: Vietnamese, vi				

1 depends on the platform

Table 21.2 lists the SAS name, the alias, the POSIX value, the LINUX encoding value, and the DFLANG= system option value.

Table 21.2 *Default Values for the DFLANG= System Option Are Based on the POSIX Locale and Linux Encoding Values*

SAS Name	POSIX Locale	LINUX Encoding	DFLANG=
Afrikaans_SouthAfrica alias: Afrikaans, af	af_ZA	latin1	English
Albanian_Albania alias: Albanian, sq	sq_AL	latin2	English
Arabic_Bahrain	ar_BH	arabic	English
Arabic_Algeria	ar_DZ	arabic	English
Arabic_Egypt	ar_EG	arabic	English
Arabic_India	ar_IN	arabic	English
Arabic_Iraq	ar_IQ	arabic	English
Arabic_Jordan	ar_JO	arabic	English
Arabic_Kuwait	ar_KW	arabic	English
Arabic_Lebanon	ar_LB	arabic	English
Arabic_Libya	ar_LY	arabic	English
Arabic_Morocco	ar_MA	arabic	English
Arabic_Oman	ar_OM	arabic	English
Arabic_Qatar	ar_QA	arabic	English
Arabic_SaudiArabia	ar_SA	arabic	English
Arabic_Sudan	ar_SD	arabic	English

SAS Name	POSIX Locale	LINUX Encoding	DFLANG=
Arabic_Syria	ar_SY	arabic	English
Arabic_Tunisia	ar_TN	arabic	English
Arabic_UnitedArabEmirates alias: Arabic, ar	ar_AE	arabic	English
Arabic_Yemen	ar_YE	arabic	English
Basque_Spain alias: Basque, Euskara, eu	eu_ES	latin9	English
Bengali_India alias: Bengali, bn	bn_IN	latin1	English
Bosnian_BosniaHerzegovina alias: Bosnian, bs	bs_BA	latin2	English
Bulgarian_Bulgaria alias: Bulgarian, bg	bg_BG	cyrillic	English
Byelorussian_Belarus alias: Byelorussian, Belarusian, Byelorussian_Belarus, be	be_BY	cyrillic	English
Catalan_Spain alias: Catalan, ca	ca_ES	latin1	English
Chinese_China alias:Chinese, zh_HANS, zh_HANS_CN	zh_CN	euc-cn	Locale
Chinese_HongKong alias: zh_HANT_HK	zh_HK	euc-tw, ms-950	Locale
Chinese_Macau alias: zh_HANT_MO	zh_MO	euc-tw, ms-950	Locale
Chinese_Singapore alias: zh_HANS_SG	zh_SG	euc-cn	Locale
Chinese_Taiwan alias: zh_HANT_TW	zh_TW	euc-tw, ms950	Locale

SAS Name	POSIX Locale	LINUX Encoding	DFLANG=
Cornish_UnitedKingdom alias: Cornish, kw	kw_GB	latin1	English
Croatian_BosniaHerzegovina	hr_BA	latin2	Croatian
Croatian_Croatia alias: Croatian, hr	hr_HR	latin2	Croatian
Czech_CzechRepublic alias: Czech, cs	cs_CZ	latin2	Czech
Danish_Denmark alias: Danish, da	da_DK	latin9	Danish
Dutch_Belgium	nl_BE	latin1	Dutch
Dutch_Netherlands alias: Dutch, nl	nl_NL	latin1	Dutch
English_Australia	en_AU	latin1	English
English_Belgium	en_BE	latin9	English
English_Botswana	en_BW	latin1	English
English_Canada	en_CA	latin1	English
English_Caribbean	en_CB	latin1	English
English_HongKong	en_HK	latin9	English
English_Ireland	en_IE	latin9	English
English_India	en_IN	latin9	English
English_Jamaica	en_JM	latin1	English
English_Malta	en_MT	latin9	English
English_NewZealand	en_NZ	latin1	English
English_Philippines	en_PH	latin1	English
English_Singapore	en_SG	latin9	English

SAS Name	POSIX Locale	LINUX Encoding	DFLANG=
English_SouthAfrica	en_ZA	latin1	English
English_UnitedStates alias: English, en	en_US	latin1	English
English_UnitedKingdom	en_GB	latin9	English
English_Zimbabwe	en_ZW	latin1	English
Estonian_Estonia	et_EE	latin6	English
Faroese_FaroeIslands alias: Faroese, fo	fo_FO	latin1	English
Finnish_Finland alias: Finnish, fi	fi_FI	latin9	Finnish
French_Belgium	fr_BE	latin9	French
French_Canada	fr_CA	latin1	French
French_France alias: French, fr	fr_FR	latin9	French
French_Luxembourg	fr-LU	latin9	French
French_Switzerland	fr_CH	latin9	Swiss_French
German_Austria	de_AT	latin9	German
German_Germany alias: German, de	de_DE	latin9	German
German_Liechtenstein	de_LI	latin9	German
German_Luxembourg	de_LU	latin9	German
German_Switzerland	de_CH	latin9	Swiss_ German
Greek_Cyprus	el_CY	greek	English
Greek_Greece alias: Greek, el	el_GR	greek	English

SAS Name	POSIX Locale	LINUX Encoding	DFLANG=
Greenlandic_Greenland alias: Greenlandic, kl	kl_GL	latin1	English
Hebrew_Israel alias: Hebrew, he	he_IL	hebrew	English
Hindi_India alias: Hindi, hi	hi_IN	latin1	English
Hungarian_Hungary alias: Hungarian, hu	hu_HU	latin2	Hungarian
Icelandic_Iceland alias: Icelandic, is	is_IS	latin1	English
Indonesian_Indonesia alias: Indonesian, id	id_ID	latin1	English
Irish_Ireland alias: Irish, ga	ga_IE	Latin8	English
Italian_Italy alias: Italian, it	it_IT	latin9	Italian
Italian_Switzerland	it_CH	latin9	Italian
Japanese_Japan alias: Japanese, ja	ja_JP	euc-jp, shift-jis	Japanese
Kazakh_Kazakhstan alias: Kazakh, kk	kk_KZ	Cyrillic	English
Korean_Korea alias: Korean, ko, Korean_SouthKorea	ko_KR	euc-kr	Locale
Latvian_Latvia alias: Latvian, lv	lv_LV	latin6	English
Lithuanian_Lithuania alias: Lithuanian, lt	lt_LT	latin6	English
Macedonian_Macedonia	mk_MK	cyrillic	English

SAS Name	POSIX Locale	LINUX Encoding	DFLANG=
alias: Macedonian, mk			
Malay_Malaysia alias: Malay, ms	ms_MY	latin1	English
Maltese_Malta	mt_MT	latin3	English
Marathi_India alias: Marathi, mr	mr_IN	latin1	English
NorwegianBokmal_Norway alias: NorwegianBokmal, nb	nb_NO	latin9	Norwegian
NorwegianNynorsk_Norway alias: NorwegianNynorsk, nn	nn_NO	latin9	Norwegian
Norwegian_Norway alias: Norwegian, no	no_NO	latin9	Norwegian
Persian_India	fa_IN	arabic	English
Persian_Iran alias: Persian, fa	fa_IR	arabic	English
Polish_Poland alias: Polish, pl	pl_PL	latin2	Polish
Portuguese_Brazil	pt_BR	latin1	Portuguese
Portuguese_Portugal alias: Portuguese, pt	pt_PT	latin1	Portuguese
Romanian_Romania alias: Romanian, ro	ro_RO	latin2	English
Russian_Russia alias: Russian, ru	ru_RU	cyrillic	Russian
Russian_Ukraine	ru_UA	cyrillic	Russian
Serbian_BosniaHerzegovina	sr_BA	cyrillic	English
Serbian_Montenegro	sr_ME	cyrillic	English

SAS Name	POSIX Locale	LINUX Encoding	DFLANG=
Serbian_Serbia alias: sr_RS, sr_RS	sr_RS sr_CYRL_RS	cyrillic	English
SerbianLatin_BosniaHerzegovina	sh_BA	latin2	English
SerbianLatin_Montenegro	sh_ME	latin2	English
SerbianLatin_Serbia alias: sh_LATN, sh_LATN_RS	sh_RS sh_RS	latin2	English
Slovak_Slovakia alias: Slovak, Slovakian, Slovakian_Slovakia ,sk	sk_SK	latin2	English
Slovenian_Slovenia alias: Slovenian, si	sl_SI	latin2	Slovenian
Spanish_Argentina	es_AR	latin1	Spanish
Spanish_Bolivia	es_BO	latin1	Spanish
Spanish_Chile	es_CL	latin1	Spanish
Spanish_Colombia	es_CO	latin1	Spanish
Spanish_CostaRica	es_CR	latin1	Spanish
Spanish_DominicanRepublic	es_DO	latin1	Spanish
Spanish_Ecuador	es_EC	latin1	Spanish
Spanish_ElSalvador	es_SV	latin1	Spanish
Spanish_Guatemala	es_GT	latin1	Spanish
Spanish_Honduras	es_HN	latin1	Spanish
Spanish_Mexico	es_MX	latin1	Spanish
Spanish_Nicaragua	es_NI	latin1	Spanish
Spanish_Panama	es_PA	latin1	Spanish
Spanish_Paraguay	es_PY	latin1	Spanish

SAS Name	POSIX Locale	LINUX Encoding	DFLANG=
Spanish_Peru	es_PE	latin1	Spanish
Spanish_PuertoRico	es_PR	latin1	Spanish
Spanish_Spain alias: Spanish, es	es_ES	latin9	Spanish
Spanish_UnitedStates	es_US	latin1	Spanish
Spanish_Uruguay	es_UY	latin1	Spanish
Spanish_Venezuela	es_VE	latin1	Spanish
Swedish_Sweden alias: Swedish, sv	sv_SE	latin9	Swedish
Tagalog_Philippines alias: Tagalog, Filipino, Filipino_Philippines, ti	ti_ph	latin1	English
Tamil_India alias: Tamil, ta	ta_IN	latin1	English
Telugu_India alias: Telugu, te	te_IN	latin1	English
Thai_Thailand alias: Thai, th	th_TH	thai	English
Turkish_Turkey alias: Turkish, tr	tr_TR	latin5	English
Ukrainian_Ukraine alias: Ukrainian, uk	uk_UA	cyrillic	English
Vietnamese_Vietnam alias: Vietnamese, vi	vi_VN	latin1	English

22

SAS System Options for Processing DBCS Data

Overview to System Options Used in a SAS Session for DBCS 1125
DBCS Values for a SAS Session 1125

Overview to System Options Used in a SAS Session for DBCS

You can use the DBCSLANG= and DBCSTYPE= system options to specify the DBCS encoding values for a SAS session. However, the ENCODING= system option is the recommended method in setting a SAS session for DBCS. The ENCODING= system option has priority. For more information, see "Setting the Encoding of a SAS Session" on page 20. Please see the "ENCODING System Option" on page 982 for more information.

DBCS Values for a SAS Session

The following table shows the supported values for the DBCSLANG= and DBCSTYPE= system options under the LINUX, operating environment.

Note: If an encoding value contains a hyphen (-), enclose the encoding value in quotation marks.

Table 22.1 *DBCS Supported Values for the DBCSLANG= and DBCSTYPE= System Options*

DBCSLANG=	LINUX DBCSTYPE=
Chinese	dec
Chinese	hp15
Chinese	euc
Chinese	pcms
Japanese	dec
Japanese	pcibm
Japanese	hp15
Japanese	sjis
Korean	pcibm
Korean	pcms
Korean	dec
Korean	euc
Korean	hp15
Taiwanese	dec
Taiwanese	pcibm
Taiwanese	hp15
Taiwanese	pcms

23

Encoding Values in SAS Language Elements

Overview to SAS Language Elements That Use Encoding Values 1127
SBCS, DBCS, and Unicode Encoding Values for Transcoding Data 1127

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 moved to an IBM mainframe that uses the German EBCDIC encoding. In order for the IBM mainframe to successfully access the file, the SAS data file must be transcoded from the Western Latin1 encoding to the German EBCDIC encoding. For information about transcoding concepts, including SAS language elements that contain options for transcoding, see Chapter 4, “Transcoding for NLS,” on page 31 .

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 23.1 *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
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

Encoding Name	Short Name	Description	Maximum bytes per Character
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
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

Encoding Name	Short Name	Description	Maximum bytes per Character
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
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

Encoding Name	Short Name	Description	Maximum bytes per Character
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	jfuj	Japanese FACOM	4
fujitsu-ko	kfuj	Korean FACOM	4
fujitsu-tw	yfuj	Traditional Chinese FACOM	4
gb18030	gbke	Simplified Chinese GB18030	4
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

Encoding Name	Short Name	Description	Maximum bytes per Character
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
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

Encoding Name	Short Name	Description	Maximum bytes per Character
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
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

Encoding Name	Short Name	Description	Maximum bytes per Character
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

Encoding Name	Short Name	Description	Maximum bytes per Character
open_ed-1156	eo56	Baltic Euro OpenEdition EBCDIC	1
open_ed-1157	eo57	Estonia Euro OpenEdition EBCDIC	1
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

Encoding Name	Short Name	Description	Maximum bytes per Character
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
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 [1]
utf-16le	u16l	Unicode (UTF-16LE)	2 [1]
utf-32be	u32b	Unicode (UTF-32BE)	4 [2]
utf-32le	u32l	Unicode (UTF-32LE)	4 [2]
warabic	wara	Arabic Windows	1

Encoding Name	Short Name	Description	Maximum bytes per Character
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

1 UTF-16BE and UTF-16LE have a fixed length of two bytes per character.
2 UTF-32BE and UTF-32LE have a fixed length of four bytes per character.

24

Encoding Values for a SAS Session

LINUX Encoding Values 1139

LINUX Encoding Values

The encodings in the following tables are valid in LINUX environments.

Note: If an encoding value contains a hyphen (-), enclose the encoding value in quotation marks.

Table 24.1 *Single-Byte Encodings for LINUX*

ENCODING= Value	Description
arabic	Arabic (ISO 8859-6)
cyrillic	Cyrillic (ISO 8859-5)
greek	Greek (ISO 8859-7)
hebrew	Hebrew (ISO 8859-8)
latin1	Western (ISO 8859-1)
latin2	Central Europe (ISO 8859-2)
latin5	Turkish (ISO 8859-9)

ENCODING= Value	Description
latin6	Baltic (ISO 8859-4)
Latin7	Baltic (ISO 8859-13)
latin8	Celtic (ISO 8859-14)
latin9	European (ISO 8859-15)
latin10	South-Eastern European (ISO 8859-16)
thai	Thai (ISO 8859-11)

Table 24.2 *Double-Byte Encodings for LINUX*

ENCODING= Value	Description
big5	Traditional Chinese (Big5)
euc-cn	Simplified Chinese (EUC)
euc-jp	Japanese (EUC)
euc-kr	Korean (EUC)
euc-tw	Traditional Chinese (EUC)
shift-jis	Japanese (SJIS)

LINUX also supports the utf-8 Unicode encoding.

PART 12

Appendixes

Appendix 1
Encoding and Alias Names **1143**

Appendix 2
Time Zone IDs and Time Zone Names **1153**

Appendix 3
TK_NLS_REPLACE_ICVEXT **1193**

Appendix 1

Encoding and Alias Names

Encodings and Their Aliases and Encoding Character Set Compatibility 1143

Encodings and Their Aliases and Encoding Character Set Compatibility

This table lists common encoding methods and their corresponding encoding and alias names.

The Encoding Character Set Compatibility table shows languages (locales) that support the same set of characters

Table A12.1 *Encoding and Alias Names*

Encoding Methods and Character Sets	Encoding Names in SAS	Alias
Unicode	UTF-8	UTF-8
SBCS	WLATIN2	CP1250, WINDOWS-1250
	WCYRILLIC	CP1251, WINDOWS-1251
	WLATIN1	CP1252, WINDOWS-1252
	WGREEK	CP1253, WINDOWS-1253
	WTURKISH	CP1254, WINDOWS-1254

Encoding Methods and Character Sets	Encoding Names in SAS	Alias
	WHEBREW	CP1255, WINDOWS-1255
	WARABIC	CP1256, WINDOWS-1256
	WBALTIC	CP1257, WINDOWS-1257
	WVIETNAMESE	CP1258, WINDOWS-1258
	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
	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	CP1122

Encoding Methods and Character Sets	Encoding Names in SAS	Alias
	EBCDIC1122	
	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, GBK, GB2312
	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, OPEN_ED-425. Encoding names used by SAS for the character sets are listed in the fourth column.

Table A12.2 *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, WINDOWS-1256
Bengali	bn	Bengali		
Persian	fa	Arabic	WARABIC, ARABIC, MSDOS720, PCOEM864, OPEN_ED-425	CP1256, ISO8859_6, CP864, WINDOWS-1256
Chinese (Simplified)	zh	Chinese (Simplified)	MS-936, EUC-CN, IBM-935	GBK, GBK, GB2312, 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, WINDOWS-1251
Bulgarian	bg	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED- 1025	CP1251, ISO8859_5, CP866, CP1025, WINDOWS-1251
Macedonian	mk	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED- 1025	CP1251, ISO8859_5, CP866, CP1025, WINDOWS-1251
Russian	ru	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED- 1025	CP1251, ISO8859_5, CP866, CP1025, WINDOWS-1251
Serbian	sr	Cyrillic	WCYRILLIC, CYRILLIC,	CP1251, ISO8859_5,

Language	Language code (ISO 639)	Script group	Encoding names (in SAS)	Encoding names (in Java)
			PCOEM866, OPEN_ED- 1025	CP866, CP1025, WINDOWS-1251
Ukrainian	uk	Cyrillic	WCYRILLIC, CYRILLIC, PCOEM866, OPEN_ED- 1025	CP1251, ISO8859_5, CP866, CP1025, WINDOWS-1251
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, WINDOWS-1253
Hebrew	he	Hebrew	WHEBREW, HEBREW, PCOEM862, OPEN_ED-424	CP1255, ISO8859_8, CP862, CP424, WINDOWS-1255
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, WINDOWS-1252
Catalan	ca	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED- 1145	CP1252, ISO8859_1, ISO8859_15, CP850, CP1145, WINDOWS-1252
Cornish	kw	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED- 1146	CP1252, ISO8859_1, ISO8859_15, CP850, CP1146, WINDOWS-1252

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, WINDOWS-1252
Dutch	nl	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED- 1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047, WINDOWS-1252
English	en	LATIN1	WLATIN1, LATIN1, PCOEM437, OPEN_ED- 1047	CP1252, ISO8859_1, ISO8859_15, CP437, CP1047, WINDOWS-1252
Faroese	fo	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED- 1142	CP1252, ISO8859_1, ISO8859_15, CP850, CP1142, WINDOWS-1252
Finnish	fi	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED- 1143	CP1252, ISO8859_1, ISO8859_15, CP850, CP1143, WINDOWS-1252
French	fr	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED- 1147	CP1252, ISO8859_1, ISO8859_15, CP850, CP1147, WINDOWS-1252
German	de	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED- 1141	CP1252, ISO8859_1, ISO8859_15, CP850, CP1141, WINDOWS-1252
Greenlandic	kl	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED- 1142	CP1252, ISO8859_1, ISO8859_15, CP850, CP1142, WINDOWS-1252

Language	Language code (ISO 639)	Script group	Encoding names (in SAS)	Encoding names (in Java)
Icelandic	is	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED- 1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047, WINDOWS-1252
Indonesian	id	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED- 1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047, WINDOWS-1252
Italian	it	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED- 1144	CP1252, ISO8859_1, ISO8859_15, CP850, CP1144, WINDOWS-1252
Malay	ms	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED- 1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047, WINDOWS-1252
Manx	gv	LATIN1	WLATIN1, LATIN8, PCOEM850, OPEN_ED- 1148	CP1252, ISO8859_14, CP850, CP1148, WINDOWS-1252
Norwegian	no	LATIN1	WLATIN1, LATIN1, PCOEM850, OPEN_ED- 1142	CP1252, ISO8859_1, ISO8859_15, CP850, CP1142, WINDOWS-1252
Portuguese	pt	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED- 1047	CP1252, ISO8859_1, ISO8859_15, CP850, CP1047, WINDOWS-1252
Spanish	es	LATIN1	WLATIN1, LATIN9, PCOEM850, OPEN_ED- 1145	CP1252, ISO8859_1, ISO8859_15, CP850, CP1145, WINDOWS-1252

Language	Language code (ISO 639)	Script group	Encoding names (in SAS)	Encoding names (in Java)
Albanian	sq	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Bosnian	bs	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Croatian	hr	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Czech	cs	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Hungarian	hu	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Polish	pl	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Romanian	ro	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Serbian (Latin)	sh	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250

Language	Language code (ISO 639)	Script group	Encoding names (in SAS)	Encoding names (in Java)
Slovak	sk	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Slovenian	sl	LATIN2	WLATIN2, LATIN2, PCOEM852, OPEN_ED- 870	CP1250, ISO8859_2, ISO8859_16, CP852, CP870, WINDOWS-1250
Maltese	mt	LATIN3	CP1254, ISO8859_9, CP857, CP1026	ISO8859_3
Turkish	tr	LATIN5	WTURKISH, TURKISH, PCOEM857, OPEN_ED-1026	CP1254, ISO8859_9, CP857, CP1026, WINDOWS-1254
Estonian	et	LATIN6	WBALTIC, LATIN6, MSDOS775, PCOEM921, OPEN_ED-1112	CP1257, ISO8859_6, CP775, CP1112, WINDOWS-1257
Latvian	lv	LATIN6	WBALTIC, LATIN6, MSDOS775, PCOEM921, OPEN_ED-1112	CP1257, ISO8859_6, CP775, CP1112, WINDOWS-1257
Lithuanian	lt	LATIN6	WBALTIC, LATIN6, MSDOS775, PCOEM921, OPEN_ED-1112	CP1257, ISO8859_6, CP775, CP1112, WINDOWS-1257
Tamil	ta	Tamil		
Telugu	te	Telegu		
Thai	th	Thai	PCOEM874, THAI	CP874, TIS620
Vietnamese	vi	Vietnamese	WVIETNAMESE, OPEN_ED-1130	CP1258, CP1130, WINDOWS-1258

Appendix 2

Time Zone IDs and Time Zone Names

Area: Africa 1153
Area: America–North, Central, and South 1156
Area: Antarctica 1166
Area: Arctic 1167
Area: Asia 1167
Area: Atlantic 1172
Area: Australia 1173
Area: Miscellaneous 1176
Area: Europe 1182
Area: Pacific 1189

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/Algiers	CET	DZ	Central European Standard Time	01:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Africa/Bamako	GMT	ML	Greenwich Mean Time	00:00
Africa/Bangui	WAT	CF	West Africa Standard Time	01:00
Africa/Banjul	GMT	GM	Greenwich Mean Time	00:00
Africa/Bissau	GMT	GW	Greenwich Mean Time	00:00
Africa/Brazzaville	WAT	CG	West Africa Time	01:00
Africa/Casablanca	+01	MA	Western European Standard Time	01:00
Africa/Casablanca	+00	MA	Western European Summer Time	00:00
Africa/Ceuta	CET	ES	Central European Standard Time	01:00
Africa/Ceuta	CEST	ES	Central European Summer Time	02:00
Africa/Conakry	GMT	GN	Greenwich Mean Time	00:00
Africa/Dakar	GMT	SN	Greenwich Mean Time	00:00
Africa/Douala	WAT	CM	West Africa Time	01:00
Africa/El_Aaiun	+01	EH	Western European Standard Time	01:00
Africa/El_Aaiun	+00	EH	Western European Summer Time	00:00
Africa/Freetown	GMT	SL	Greenwich Mean Time	00:00
Africa/Lome	GMT	TG	Greenwich Mean Time	00:00
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	CI	Greenwich Mean Time	00.00
Africa/Tunis	CET	TN	Central European Standard Time	01:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Africa/Kinshasa	WAT	CD	West Africa Standard Time	01:00
Africa/Lagos	WAT	NG	West Africa Standard Time	01:00
Africa/Libreville	WAT	GA	West Africa Standard Time	01:00
Africa/Luanda	WAT	AO	West Africa Standard Time	01:00
Africa/Malabo	WAT	GQ	West Africa Standard Time	01:00
Africa/Ndjamena	WAT	TD	West Africa Standard Time	01:00
Africa/Niamey	WAT	NE	West Africa Standard Time	01:00
Africa/Porto-Novo	WAT	BJ	West Africa Standard Time	01:00
Africa/Windhoek	CAT	NA	Central Africa Time	02:00
Africa/Blantyre	CAT	MW	Central Africa Time	02:00
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	EG	Eastern European Standard Time	02:00
Africa/Tripoli	EET	LY	Eastern European Standard 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	ET	East Africa Time	3.00
Africa/Asmera	EAT	ER	East Africa Time	03:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
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	CAT	SD	Central Africa Time	02:00
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	HST	US	Hawaii-Aleutian Standard Time	-10:00
America/Adak	HDT	US	Hawaii-Aleutian Daylight Time	-09:00
America/Anchorage	AKST	US	Alaska Standard Time	-09:00
America/Anchorage	AKDT	US	Alaska Daylight Time	-08:00
America/Anguilla	AST	AI	Atlantic Standard Time	-04:00
America/Antigua	AST	AG	Atlantic Standard Time	-04:00
America/Araguaina	-03	BR	Brasilia Standard Time	-03:00
America/Argentina/Buenos_Aires	-03	AR	Argentina Standard Time	-03:00
America/Argentina/Catamarca	-03	AR	Argentina Standard Time	-03:00
America/Argentina/ComodRivadavia	-03	AR	Argentina Standard Time	-03:00
America/Argentina/Cordoba	-03	AR	Argentina Time	-03:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Argentina/Jujuy	-03	AR	Argentina Standard Standard Time	-03:00
America/Argentina/La_Rioja	-03	AR	Argentina Standard Time	-03:00
America/Argentina/Mendoza	-03	AR	Argentina Standard Time	-03:00
America/Argentina/Rio_Gallegos	-03	AR	Argentina Standard Time	-03:00
America/Argentina/Salta	-03	AR	Argentina Standard Time	-03:00
America/Argentina/San_Juan	-03	AR	Argentina Standdard Time	-03:00
America/Argentina/San_Luis	-03	AR	Western Argentina Standard Time	-03:00
America/Argentina/Tucuman	-03	AR	Argentina Standard Time	-03:00
America/Argentina/Ushuaia	-03	AR	Argentina Standard Time	-03:00
America/Aruba	AST	AW	Atlantic Standard Time	-04:00
America/Asuncion	-04	PY	Paraguay Standard Time	-04:00
America/Asuncion	-03	PY	Paraguay Summer Time	-03:00
America/Atikokan	EST	CA	Eastern Standard Time	-05:00
America/Atka	HST	US	Hawaii-Aleutian Standard Time	-10:00
America/Atka	HDT	US	Hawaii-Aleutian Daylight Time	-09:00
America/Bahia	-03	BR	Brasilia Standard Time	-03:00
America/Bahia_Banderas	CST	MX	Central Standard Time	-06:00
America/Bahia_Banderas	CDT	MX	Central Daylight Time	-05:00
America/Barbados	AST	BB	Atlantic Standard Time	-04:00
America/Belem	-03	BR	Brasilia Standard Time	-03:00
America/Belize	CST	BZ	Central Standard Time	-06:00
America/Blanc-Sablon	AST	CA	Atlantic Standard Time	-04:00
America/Boa_Vista	-04	BR	Amazon Standard Time	-04:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Bogota	-05	CO	Colombia Standard Time	-05:00
America/Boise	MST	US	Mountain Standard Time	-07:00
America/Boise	MDT	US	Mountain Daylight Time	-06:00
America/Buenos_Aires	-03	AR	Argentina Standard Time	-03:00
America/Cambridge_Bay	MST	CA	Mountain Standard Time	-07:00
America/Cambridge_Bay	MDT	CA	Mountain Daylight Time	-06:00
America/Campo_Grande	-04	BR	Amazon Standard Time	-04:00
America/Campo_Grande	-03	BR	Amazon Summer Time	-03:00
America/Cancun	EST	MX	Eastern Standard Time	-05:00
America/Caracas	-04	VE	Venezuela Time	-04:00
America/Catamarca	-03	AR	Argentina Standard Time	-03:00
America/Cayenne	-03	GF	French Guiana Time	-03:00
America/Cayman	EST	KY	Eastern Standard Time	-05:00
America/Chicago	CST	US	Central Standard Time	-06:00
America/Chicago	CDT	US	Central Daylight Time	-05:00
America/Chihuahua	MST	MX	Mexican Pacific Standard Time	-07:00
America/Chihuahua	MDT	MX	Mexican Pacific Daylight Time	-06:00
America/Coral_Harbour	EST	CA	Eastern Standard Time	-05:00
America/Cordoba	-03	AR	Argentina Standard Time	-03:00
America/Costa_Rica	CST	CR	Central Standard Time	-06:00
America/Creston	MST	CA	Mountain Standard Time	-07:00
America/Cuiaba	-04	BR	Amazon Standard Time	-04:00
America/Cuiaba	-03	BR	Amazon Summer Time	-03:00
America/Curacao	AST	CW	Atlantic Standard Time	-04:00
America/Danmarkshavn	GMT	GL	Greenwich Mean Time	00:00
America/Dawson	PST	CA	Pacific Standard Time	-08:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Dawson	PDT	CA	Pacific Daylight Time	-07:00
America/Dawson_Creek	MST	CA	Mountain Standard Time	-07:00
America/Denver	MST	US	Mountain Standard Time	-07:00
America/Denver	MDT	US	Mountain Daylight Time	-06:00
America/Detroit	EST	US	Eastern Standard Time	-05:00
America/Detroit	EDT	US	Eastern Daylight Time	-04:00
America/Dominica	AST	DM	Atlantic Standard Time	-04:00
America/Edmonton	MST	CA	Mountain Standard Time	-07:00
America/Edmonton	MDT	CA	Mountain Daylight Time	-06:00
America/Eirunepe	-05	BR	Acre Standard Time	-05:00
America/El_Salvador	CST	SV	Central Standard Time	-06:00
America/Ensenada	PST	MX	Pacific Time	-08:00
America/Ensenada	PDT	MX	Pacific Daylight Time	-07:00
America/Fort_Nelson	MST	CA	Mountain Standard Time	-07:00
America/Fort_Wayne	EST	US	Eastern Standard Time	-05:00
America/Fort_Wayne	EDT	US	Eastern Daylight Time	-04:00
America/Fortaleza	-03	BR	Brasilia Standard Time	-03:00
America/Glace_Bay	AST	CA	Atlantic Standard Time	-04:00
America/Glace_Bay	ADT	CA	Atlantic Daylight Time	-03:00
America/Godthab	-03	GL	West Greenland Standard Time	-03:00
America/Godthab	-02	GL	West Greenland Summer Time	-02:00
America/Goose_Bay	AST	CA	Atlantic Standard Time	-04:00
America/Goose_Bay	ADT	CA	Atlantic Daylight Time	-03:00
America/Grand_Turk	EDT	TC	Eastern Daylight Time	-04:00
America/Grand_Turk	EST	TC	Eastern Standard Time	-05:00
America/Grenada	AST	GD	Atlantic Standard Time	-04:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Guadeloupe	AST	GP	Atlantic Standard Time	-04:00
America/Guatemala	CST	GT	Central Standard Time	-06:00
America/Guayaquil	-05	EC	Ecuador Time	-05:00
America/Guyana	-04	GY	Guyana Time	-04:00
America/Halifax	AST	CA	Atlantic Standard Time	-04:00
America/Halifax	ADT	CA	Atlantic Daylight Time	-03:00
America/Havana	CST	CU	Cuba Standard Time	-05:00
America/Havana	CDT	CU	Cuba Daylight Time	-04:00
America/Hermosillo	MST	MX	Mexican Pacific Standard Time	-07:00
America/Indiana/ Indianapolis	EST	US	Eastern Standard Time	-05:00
America/Indiana/ Indianapolis	EDT	US	Eastern Daylight Time	-04:00
America/Indiana/Knox	CST	US	Central Standard Time	-06:00
America/Indiana/Knox	CDT	US	Central Daylight Time	-05:00
America/Indiana/Marengo	EST	US	Eastern Standard Time	-05:00
America/Indiana/Marengo	EDT	US	Eastern Daylight Time	-04:00
America/Indiana/ Petersburg	EST	US	Eastern Standard Time	-05:00
America/Indiana/ Petersburg	EDT	US	Eastern Daylight Time	-04:00
America/Indiana/Tell_City	CST	US	Central Standard Time	-06:00
America/Indiana/Tell_City	CDT	US	Central Daylight Time	-05:00
America/Indiana/Vevay	EST	US	Eastern Standard Time	-05:00
America/Indiana/Vevay	EDT	US	Eastern Daylight Time	-04:00
America/Indiana/Vincennes	EST	US	Eastern Standard Time	-05:00
America/Indiana/Vincennes	EDT	US	Eastern Daylight Time	-04:00
America/Indiana/Winamac	EST	US	Eastern Standard Time	-05:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Indiana/Winamac	EDT	US	Eastern Daylight Time	-04:00
America/Indianapolis	EST	US	Eastern Standard Time	-05:00
America/Indianapolis	EDT	US	Eastern Daylight Time	-04:00
America/Inuvik	MST	CA	Mountain Standard Time	-07:00
America/Inuvik	MDT	CA	Mountain Daylight Time	-06:00
America/Iqaluit	EST	CA	Eastern Standard Time	-05:00
America/Iqaluit	EDT	CA	Eastern Daylight Time	-04:00
America/Jamaica	EST	JM	Eastern Standard Time	-05:00
America/Jujuy	-03	AR	Argentina Standard Time	-03:00
America/Juneau	AKST	US	Alaska Standard Time	-09:00
America/Juneau	AKDT	US	Alaska Daylight Time	-08:00
America/Kentucky/Louisville	EST	US	Eastern Standard Time	-05:00
America/Kentucky/Louisville	EDT	US	Eastern Daylight Time	-04:00
America/Kentucky/Monticello	EST	US	Eastern Standard Time	-05:00
America/Kentucky/Monticello	EDT	US	Eastern Daylight Time	-04:00
America/Knox_IN	CST	US	Central Standard Time	-06:00
America/Knox_IN	CDT	US	Central Daylight Time	-05:00
America/Kralendijk	AST	BQ	Atlantic Standard Time	-04:00
America/La_Paz	-04	BO	Bolivia Time	-04:00
America/Lima	-05	PE	Peru Standard Time	-05:00
America/Los_Angeles	PST	US	Pacific Standard Time	-08:00
America/Los_Angeles	PDT	US	Pacific Daylight Time	-07:00
America/Louisville	EST	US	Eastern Standard Time	-05:00
America/Louisville	EDT	US	Eastern Daylight Time	-04:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Lower_Princes	AST	SX	Atlantic Standard Time	-04:00
America/Maceio	-03	BR	Brasilia Standard Time	-03:00
America/Managua	CST	NI	Central Standard Time	-06:00
America/Manaus	-04	BR	Amazon Standard Time	-04:00
America/Marigot	AST	MF	Atlantic Standard Time	-04:00
America/Martinique	AST	MQ	Atlantic Standard Time	-04:00
America/Matamoros	CST	MX	Central Standard Time	-06:00
America/Matamoros	CDT	MX	Central Daylight Time	-05:00
America/Mazatlan	MST	MX	Mexican Pacific Standard Time	-07:00
America/Mazatlan	MDT	MX	Mexican Pacific Daylight Time	-06:00
America/Mendoza	-03	AR	Argentina Standard Time	-03:00
America/Menominee	CST	US	Central Standard Time	-06:00
America/Menominee	CDT	US	Central Daylight Time	-05:00
America/Merida	CST	MX	Central Standard Time	-06:00
America/Merida	CDT	MX	Central Daylight Time	-05:00
America/Metlakatla	AKST	US	Alaska Standard Time	-09:00
America/Metlakatla	AKDT	US	Alaska Daylight Time	-08:00
America/Mexico_City	CST	MX	Central Standard Time	-06:00
America/Mexico_City	CDT	MX	Central Daylight Time	-05:00
America/Miquelon	-03	PM	St Pierre and Miquelon Standard Time	-03:00
America/Miquelon	-02	PM	St Pierre and Miquelon Daylight Time	-02:00
America/Moncton	AST	CA	Atlantic Standard Time	-04:00
America/Moncton	ADT	CA	Atlantic Daylight Time	-03:00
America/Monterrey	CST	MX	Central Standard Time	-06:00
America/Monterrey	CDT	MX	Central Daylight Time	-05:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Montevideo	-03	UY	Uruguay Standard Time	-03:00
America/Montreal	EST	CA	Eastern Standard Time	-05:00
America/Montreal	EDT	CA	Eastern Daylight Time	-04:00
America/Montserrat	AST	MS	Atlantic Standard Time	-04:00
America/Nassau	EST	BS	Eastern Standard Time	-05:00
America/Nassau	EDT	BS	Eastern Daylight Time	-04:00
America/New_York	EST	US	Eastern Standard Time	-05:00
America/New_York	EDT	US	Eastern Daylight Time	-04:00
America/Nipigon	EST	CA	Eastern Standard Time	-05:00
America/Nipigon	EDT	CA	Eastern Daylight Time	-04:00
America/Nome	AKST	US	Alaska Standard Time	-09:00
America/Nome	AKDT	US	Alaska Daylight Time	-08:00
America/Noronha	-02	BR	Fernando de Noronha Standard Time	-02:00
America/North_Dakota/Beulah	CST	US	Central Standard Time	-06:00
America/North_Dakota/Beulah	CDT	US	Central Daylight Time	-05:00
America/North_Dakota/Center	CST	US	Central Standard Time	-06:00
America/North_Dakota/Center	CDT	US	Central Daylight Time	-05:00
America/North_Dakota/New_Salem	CST	US	Central Standard Time	-06:00
America/North_Dakota/New_Salem	CDT	US	Central Daylight Time	-05:00
America/Ojinaga	MST	MX	Mountain Standard Time	-07:00
America/Ojinaga	MDT	MX	Mountain Daylight Time	-06:00
America/Panama	EST	PA	Eastern Standard Time	-05:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Pangnirtung	EST	CA	Eastern Standard Time	-05:00
America/Pangnirtung	EDT	CA	Eastern Daylight Time	-04:00
America/Paramaribo	-03	SR	Suriname Time	-03:00
America/Phoenix	MST	US	Mountain Standard Time	-07:00
America/Port-au-Prince	EST	HT	Eastern Standard Time	-05:00
America/Port-au-Prince	EDT	HT	Eastern Daylight Time	-04:00
America/Port_of_Spain	AST	TT	Atlantic Standard Time	-04:00
America/Porto_Acre	-05	BR	Acre Standard Time	-05:00
America/Porto_Velho	-04	BR	Amazon Standard Time	-04:00
America/Puerto_Rico	AST	PR	Atlantic Standard Time	-04:00
America/Punta_Arenas	-03	CL	Chile Standard Time	-03:00
America/Rainy_River	CST	CA	Central Standard Time	-06:00
America/Rainy_River	CDT	CA	Central Daylight Time	-05:00
America/Rankin_Inlet	CST	CA	Central Standard Time	-06:00
America/Rankin_Inlet	CDT	CA	Central Daylight Time	-05:00
America/Recife	-03	BR	Brasilia Standard Time	-03:00
America/Regina	CST	CA	Central Standard Time	-06:00
America/Resolute	CST	CA	Central Standard Time	-06:00
America/Resolute	CDT	CA	Central Daylight Time	-05:00
America/Rio_Branco	-05	BR	Acre Standard Time	-05:00
America/Rosario	-03	AR	Argentina Standard Time	-03:00
America/Santa_Isabel	PST	MX	Pacific Standard Time	-08:00
America/Santa_Isabel	PDT	MX	Pacific Daylight Time	-07:00
America/Santarem	-03	BR	Brasilia Standard Time	-03:00
America/Santiago	-04	CL	Chile Standard Time	-04:00
America/Santiago	-03	CL	Chile Summer Time	-03:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Santo_Domingo	AST	DO	Atlantic Standard Time	-04:00
America/Sao_Paulo	-03	BR	Brasilia Standard Time	-03:00
America/Sao_Paulo	-02	BR	Brasilia Summer Time	-02:00
America/Scoresbysund	-01	GL	East Greenland Standard Time	-01:00
America/Scoresbysund	+00	GL	East Greenland Summer Time	00:00
America/Shiprock	MST	US	Mountain Standard Time	-07:00
America/Shiprock	MDT	US	Mountain Daylight Time	-06:00
America/Sitka	AKST	US	Alaska Standard Time	-09:00
America/Sitka	AKDT	US	Alaska Daylight Time	-08:00
America/St_Barthelemy	AST	BL	Atlantic Standard Time	-04:00
America/St_Johns	NST	CA	Newfoundland Standard Time	-03:30
America/St_Johns	NDT	CA	Newfoundland Daylight Time	-02:30
America/St_Kitts	AST	KN	Atlantic Standard Time	-04:00
America/St_Lucia	AST	LC	Atlantic Standard Time	-04:00
America/St_Thomas	AST	VI	Atlantic Standard Time	-04:00
America/St_Vincent	AST	VC	Atlantic Standard Time	-04:00
America/Swift_Current	CST	CA	Central Standard Time	-06:00
America/Tegucigalpa	CST	HN	Central Standard Time	-06:00
America/Thule	AST	GL	Atlantic Standard Time	-04:00
America/Thule	ADT	GL	Atlantic Daylight Time	-03:00
America/Thunder_Bay	EST	CA	Eastern Standard Time	-05:00
America/Thunder_Bay	EDT	CA	Eastern Daylight Time	-04:00
America/Tijuana	PST	MX	Pacific Standard Time	-08:00
America/Tijuana	PDT	MX	Pacific Daylight Time	-07:00
America/Toronto	EST	CA	Eastern Standard Time	-05:00
America/Toronto	EDT	CA	Eastern Daylight Time	-04:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
America/Tortola	AST	VG	Atlantic Standard Time	-04:00
America/Vancouver	PST	CA	Pacific Standard Time	-08:00
America/Vancouver	PDT	CA	Pacific Daylight Time	-07:00
America/Virgin	AST	AI	Atlantic Standard Time	-04:00
America/Whitehorse	PST	CA	Pacific Standard Time	-08:00
America/Whitehorse	PDT	CA	Pacific Daylight Time	-07:00
America/Winnipeg	CST	CA	Central Standard Time	-06:00
America/Winnipeg	CDT	CA	Central Daylight Time	-05:00
America/Yakutat	AKST	US	Alaska Standard Time	-09:00
America/Yakutat	AKDT	US	Alaska Daylight Time	-08:00
America/Yellowknife	MST	CA	Mountain Standard Time	-07:00
America/Yellowknife	MDT	CA	Mountain Daylight Time	-06:00

Area: Antarctica

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Antarctica/Casey	+08	AQ	Australian Western Standard Time	08:00
Antarctica/Davis	+07	AQ	Davis Time	07:00
Antarctica/DumontDUrville	+10	AQ	Dumont-d'Urville Time	10:00
Antarctica/Macquarie	+11	AU	Macquarie Island Time	11:00
Antarctica/Mawson	+05	AQ	Mawson Time	05:00
Antarctica/McMurdo	NZST	AQ	New Zealand Standard Time	12:00
Antarctica/McMurdo	NZDT	AQ	New Zealand Daylight Time	13:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Antarctica/Palmer	-03	AQ	Chile Standard Time	-04:00
Antarctica/Palmer	CLST	AQ	Chile Summer Time	-03:00
Antarctica/Rothera	-03	AQ	Rothera Time	-03:00
Antarctica/South_Pole	NZST	AQ	New Zealand Standard Time	12:00
Antarctica/South_Pole	NZDT	AQ	New Zealand Daylight Time	13:00
Antarctica/Syowa	+03	AQ	Syowa Time	03:00
Antarctica/Troll	+00	AQ	Greenwich Mean Time	00:00
Antarctica/Troll	+02	AQ	Greenwich Mean Time	02:00
Antarctica/Vostok	+06	AQ	Vostok Time	06:00

Area: Arctic

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Arctic/Longyearbyen	CET	SJ	Central European Standard Time	01:00
Arctic/Longyearbyen	CEST	SJ	Central European Summer Time	02:00

Area: Asia

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Aden	+03	YE	Arabian Standard Time	03:00
Asia/Almaty	+06	KZ	East Kazakhstan Time	06:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Amman	EET	JO	Eastern European Standard Time	02:00
Asia/Amman	EEST	JO	Eastern European Summer Time	03:00
Asia/Anadyr	+12	RU	Anadyr Standard Time	12:00
Asia/Aqtau	+05	KZ	West Kazakhstan Time	05:00
Asia/Aqtobe	+05	KZ	West Kazakhstan Time	05:00
Asia/Ashgabat	+05	TM	Turkmenistan Standard Time	05:00
Asia/Ashkhabad	+05	TM	Turkmenistan Standard Time	05:00
Asia/Atyrau	+05	KZ	West Kazakhstan Time	05:00
Asia/Baghdad	+03	IQ	Arabian Standard Time	03:00
Asia/Bahrain	+03	BH	Arabian Standard Time	03:00
Asia/Baku	+04	AZ	Azerbaijan Standard Time	04:00
Asia/Baku	+05	AZ	Azerbaijan Time	05:00
Asia/Bangkok	+07	TH	Indochina Time	07:00
Asia/Barnaul	+07	RU	Krasnoyarsk Time	07:00
Asia/Beijing	CST	CN	China Time	08:00
Asia/Beirut	EET	LB	Eastern European Standard Time	02:00
Asia/Beirut	EEST	LB	Eastern European Summer Time	03:00
Asia/Bishkek	+06	KG	Kyrgyzstan Time	06:00
Asia/Brunei	+08	BN	Brunei Darussalam Time	08:00
Asia/Calcutta	IST	IN	India Standard Time	05:30
Asia/Chita	+09	RU	Yakutsk Standard Time	09:00
Asia/Choibalsan	+08	MN	Choibalsan Standard Time	08:00
Asia/Choibalsan	+09	MN	Choibalsan Summer Time	09:00
Asia/Chongqing	CST	CN	China Standard Time	08:00
Asia/Chungking	CST	CN	China Standard Time	08:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Colombo	+0530	LK	India Standard Time	05:30
Asia/Dacca	+06	BD	Bangladesh Standard Time	06:00
Asia/Damascus	EET	SY	Eastern European Standard Time	02:00
Asia/Damascus	EEST	SY	Eastern European Summer Time	03:00
Asia/Dhaka	+06	BD	Bangladesh Standard Time	06:00
Asia/Dili	+09	TL	East Timor Time	09:00
Asia/Dubai	+04	AE	Gulf Standard Time	04:00
Asia/Dushanbe	+05	TJ	Tajikistan Time	05:00
Asia/Famagusta	EET	CY	Eastern European Standard Time	02:00
Asia/Famagusta	EEST	CY	Eastern European Summer Time	03:00
Asia/Gaza	EET	PS	Eastern European Standard Time	02:00
Asia/Gaza	EEST	PS	Eastern European Summer Time	03:00
Asia/Harbin	CST	CN	China Standard Time	08:00
Asia/Hebron	EET	PS	Eastern European Standard Time	02:00
Asia/Hebron	EEST	PS	Eastern European Summer Time	03:00
Asia/Ho_Chi_Minh	+07	VN	Indochina Time	07:00
Asia/Hong_Kong	HKT	HK	Hong Kong Standard Time	08:00
Asia/Hovd	+07	MN	Hovd Standard Time	07:00
Asia/Hovd	+08	MN	Hovd Summer Time	08:00
Asia/Irkutsk	+08	RU	Irkutsk Standard Time	08:00
Asia/Istanbul	+03	TR	Eastern European Time	03:00
Asia/Jakarta	WIB	ID	Western Indonesia Time	07:00
Asia/Jayapura	WIT	ID	Eastern Indonesia Time	09:00
Asia/Jerusalem	IST	IL	Israel Standard Time	02:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Jerusalem	IDT	IL	Israel Daylight Time	03:00
Asia/Kabul	+0430	AF	Afghanistan Time	04:30
Asia/Kamchatka	+12	RU	Petropavlovsk-Kamchatski Standard Time	12:00
Asia/Karachi	PKT	PK	Pakistan Standard Time	05:00
Asia/Kashgar	+06	CN	China Time	06:00
Asia/Katmandu	+0545	NP	Nepal Time	05:45
Asia/Khandyga	+09	RU	Yakutsk Standard Time	09:00
Asia/Kolkata	IST	IN	India Standard Time	05:30
Asia/Krasnoyarsk	+07	RU	Krasnoyarsk Standard Time	08:00
Asia/Kuala_Lumpur	+08	MY	Malaysia Time	08:00
Asia/Kuching	+08	MY	Malaysia Time	08:00
Asia/Kuwait	+03	KW	Arabian Standard Time	03:00
Asia/Macao	CST	MO	China Standard Time	08:00
Asia/Macau	CST	MO	China Standard Time	08:00
Asia/Magadan	+11	RU	Magadan Standard Time	11:00
Asia/Makassar	WITA	ID	Central Indonesia Time	08:00
Asia/Manila	PST	PH	Philippine Standard Time	08:00
Asia/Muscat	+04	OM	Gulf Standard Time	04:00
Asia/Nicosia	EET	CY	Eastern European Standard Time	02:00
Asia/Nicosia	EEST	CY	Eastern European Summer Time	03:00
Asia/Novokuznetsk	+07	RU	Krasnoyarsk StandardTime	07:00
Asia/Novosibirsk	+07	RU	Novosibirsk Standard Time	07:00
Asia/Omsk	+06	RU	Omsk Standard Time	06:00
Asia/Qostanay	+06	KZ	East Kazakhstan Time	06:00
Asia/Oral	+05	KZ	West Kazakhstan Time	05:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Osaka	JST	JP	Japan Time	09:00
Asia/Phnom_Penh	+07	KH	Indochina Time	07:00
Asia/Pontianak	WIB	ID	Western Indonesia Time	07:00
Asia/Pyongyang	KST	KP	Korean Standard Time	09:00
Asia/Qatar	+03	QA	Arabian Standard Time	03:00
Asia/Qyzylorda	+05	KZ	West Kazakhstan Time	05:00
Asia/Rangoon	+0630	MM	Myanmar Time	06:30
Asia/Riyadh	+03	SA	Arabian Standard Time	03:00
Asia/Saigon	+07	VN	Indochina Time	07:00
Asia/Sakhalin	+11	RU	Sakhalin Standard Time	11:00
Asia/Samarkand	+05	UZ	Uzbekistan Standard Time	05:00
Asia/Sapporo	JST	JP	Japan Time	09:00
Asia/Seoul	KST	KR	Korean Standard Time	09:00
Asia/Shanghai	CST	CN	China Standard Time	08:00
Asia/Singapore	+08	SG	Singapore Standard Time	08:00
Asia/Srednekolymsk	+11	RU	Magadan Standard Time	11:00
Asia/Taipei	CST	TW	Taipei Standard Time	08:00
Asia/Tashkent	+05	UZ	Uzbekistan Standard Time	05:00
Asia/Tbilisi	+04	GE	Georgia Standard Time	04:00
Asia/Tehran	+0330	IR	Iran Standard Time	03:30
Asia/Tehran	+0430	IR	Iran Daylight Time	04:30
Asia/Tel_Aviv	IST	IL	Israel Standard Time	02:00
Asia/Tel_Aviv	IDT	IL	Israel Daylight Time	03:00
Asia/Thimbu	+06	BT	Bhutan Time	06:00
Asia/Thimphu	BTT	BT	Bhutan Time	06:00
Asia/Tokyo	JST	JP	Japan Standard Time	09:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Asia/Tomsk	+07	RU	Omsk Time	07:00
Asia/Ujung_Pandang	WITA	ID	Central Indonesia Time	08:00
Asia/Ulan_Bator	+08	MN	Ulaanbaatar Bator Standard Time	08:00
Asia/Urumqi	XJT	CN	China Standard Time	08:00
Asia/Ust-Nera	+10	RU	Vladivostok Standard Time	10:00
Asia/Vientiane	+07	LA	Indochina Time	07:00
Asia/Vladivostok	+10	RU	Vladivostok Standard Time	10:00
Asia/Yakutsk	+09	RU	Yakutsk Standard Time	09:00
Asia/Yangon	+0630	MM		06:30
Asia/Yekaterinburg	+05	RU	Yekaterinburg Standard Time	05:00
Asia/Yerevan	+04	AM	Armenia Standard Time	04:00
Asia/Yerevan	+05	AM	Armenia Time	05:00

Area: Atlantic

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Atlantic/Azores	AZOT	PT	Azores Time	-01:00
Atlantic/Azores	+00	PT	Azores Summer Standard Time	00:00
Atlantic/Bermuda	AST	BM	Atlantic Standard Time	-04:00
Atlantic/Bermuda	ADT	BM	Atlantic Daylight Time	-03:00
Atlantic/Canary	WET	ES	Western European Standard Time	00:00
Atlantic/Canary	WEST	ES	Western European Summer Time	01:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Atlantic/Cape_Verde	-01	CV	Cape Verde Standard Time	-01:00
Atlantic/Faeroe	WET	FO	Western European Standard Time	00:00
Atlantic/Faeroe	WEST	FO	Western European Summer Time	01:00
Atlantic/Faroe	WET	FO	Western European Time	00:00
Atlantic/Faroe	WEST	FO	Western European Summer Time	01:00
Atlantic/Jan_Mayen	CET	SJ	Central European Standard Time	01:00
Atlantic/Jan_Mayen	CEST	SJ	Central European Summer Time	02:00
Atlantic/Madeira	WET	PT	Western European Standard Time	00:00
Atlantic/Madeira	WEST	PT	Western European Summer Time	01:00
Atlantic/Reykjavik	GMT	IS	Greenwich Mean Time	00:00
Atlantic/South_Georgia	-02	GS	South Georgia Time	-02:00
Atlantic/St_Helena	GMT	SH	Greenwich Mean Time	00:00
Atlantic/Stanley	-03	FK	Falkland Islands Standard Time	-03:00

Area: Australia

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Australia/ACT	AEST	AU	Australian Eastern Standard Time	10:00
Australia/ACT	AEDT	AU	Australian Eastern Daylight Time	11:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Australia/Adelaide	ACST	AU	Australian Central Standard Time	09:30
Australia/Adelaide	ACDT	AU	Australian Central Daylight Time	10:30
Australia/Brisbane	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Broken_Hill	ACST	AU	Australian Central Standard Time	09:30
Australia/Broken_Hill	ACDT	AU	Australian Central Daylight Time	10:30
Australia/Canberra	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Canberra	AEDT	AU	Australian Eastern Daylight Time	11:00
Australia/Currie	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Currie	AEDT	AU	Australian Eastern Daylight Time	11:00
Australia/Darwin	ACST	AU	Australian Central Standard Time	09:30
Australia/Eucla	+0845	AU	Australian Central Western Standard Time	08:45
Australia/Hobart	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Hobart	AEDT	AU	Australian Eastern Daylight Time	11:00
Australia/LHI	+1030	AU	Lord Howe Standard Time	10:30
Australia/LHI	+11	AU	Lord Howe Daylight Time	11:00
Australia/Lindeman	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Lord_Howe	+1030	AU	Lord Howe Standard Time	10:30
Australia/Lord_Howe	+11	AU	Lord Howe Daylight Time	11:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Australia/Melbourne	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Melbourne	AEDT	AU	Australian Eastern Daylight Time	11:00
Australia/NSW	AEST	AU	Australian Eastern Standard Time	10:00
Australia/NSW	AEDT	AU	Australian Eastern Daylight Time	11:00
Australia/North	ACST	AU	Australian Central Standard Time	09:30
Australia/Perth	AWST	AU	Australian Western Standard Time	08:00
Australia/Queensland	AEST	AU	Australian Eastern Standard Time	10:00
Australia/South	ACST	AU	Australian Central Standard Time	09:30
Australia/South	ACDT	AU	Australian Central Daylight Time	10:30
Australia/Sydney	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Sydney	AEDT	AU	Australian Eastern Daylight Time	11:00
Australia/Tasmania	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Tasmania	AEDT	AU	Australian Eastern Daylight Time	11:00
Australia/Victoria	AEST	AU	Australian Eastern Standard Time	10:00
Australia/Victoria	AEDT	AU	Australian Eastern Daylight Time	11:00
Australia/West	AWST	AU	Australian Western Standard Time	08:00
Australia/Yancowinna	ACST	AU	Australian Central Standard Time	09:30

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Australia/Yancowinna	ACDT	AU	Australian Central Daylight Time	10:30

Area: Miscellaneous

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Brazil/Acre	-05	BR	Acre Standard Time	-05:00
Brazil/DeNoronha	-02	BR	Fernando de Noronha Standard Time	-02:00
Brazil/East	-03	BR	Brasilia Standard Time	-03:00
Brazil/East	-02	BR	Brasilia Summer Time	-02:00
Brazil/West	-04	BR	Amazon Standard Time	-04:00
CST6CDT	CST	US	Central Standard Time	-06:00
CST6CDT	CDT	US	Central Daylight Time	-05:00
Canada/Atlantic	AST	CA	Atlantic Standard Time	-04:00
Canada/Atlantic	ADT	CA	Atlantic Daylight Time	-03:00
Canada/Central	CST	CA	Central Standard Time	-06:00
Canada/Central	CDT	CA	Central Daylight Time	-05:00
Canada/Eastern	EST	CA	Eastern Standard Time	-05:00
Canada/Eastern	EDT	CA	Eastern Daylight Time	-04:00
Canada/Mountain	MST	CA	Mountain Standard Time	-07:00
Canada/Mountain	MDT	CA	Mountain Daylight Time	-06:00
Canada/Newfoundland	NST	CA	Newfoundland Standard Time	-03:30
Canada/Newfoundland	NDT	CA	Newfoundland Daylight Time	-02:30

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Canada/Pacific	PST	CA	Pacific Standard Time	-08:00
Canada/Pacific	PDT	CA	Pacific Daylight Time	-07:00
Canada/Saskatchewan	CST	CA	Central Standard Time	-06:00
Canada/Yukon	PST	CA	Pacific Standard Time	-08:00
Canada/Yukon	PDT	CA	Pacific Daylight Time	-07:00
Chile/Continental	-04	CL	Chile Standard Time	-04:00
Chile/Continental	-03	CL	Chile Summer Time	-03:00
Chile/EasterIsland	-06	CL	Easter Island Standard Time	-06:00
Chile/EasterIsland	-05	CL	Easter Island Summer Time	-05:00
Cuba	CST	CU	Cuba Standard Time	-05:00
Cuba	CDT	CU	Cuba Daylight Time	-04:00
EST5EDT	EST	US	Eastern Standard Time	-05:00
EST5EDT	EDT	US	Eastern Daylight Time	-04:00
Egypt	EET	EG	Eastern European Standard Time	02:00
Eire	GMT	IE	Greenwich Mean Time	00:00
Eire	IST	IE	Greenwich Mean Time	01:00
Etc/GMT	GMT		Greenwich Mean Time	00:00
Etc/GMT+0	GMT		Greenwich Mean Time	00:00
Etc/GMT+1	GMT-01		1 hours behind GMT	-01:00
Etc/GMT+2	GMT-02		2 hours behind GMT	-02:00
Etc/GMT+3	GMT-03		3 hours behind GMT	-03:00
Etc/GMT+4	GMT-04		4 hours behind GMT	-04:00
Etc/GMT+5	GMT-05		5 hours behind GMT	-05:00
Etc/GMT+6	GMT-06		6 hours behind GMT	-06:00
Etc/GMT+7	GMT-07		7 hours behind GMT	-07:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Etc/GMT+8	GMT-08		8 hours behind GMT	-08:00
Etc/GMT+9	GMT-09		9 hours behind GMT	-09:00
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-0	GMT		Greenwich Mean Time	00:00
Etc/GMT-1	GMT+01		1 hour ahead of GMT	01:00
Etc/GMT-2	GMT+02		2 hours ahead of GMT	02:00
Etc/GMT-3	GMT+03		3 hours ahead of GMT	03:00
Etc/GMT-4	GMT+04		4 hours ahead of GMT	04:00
Etc/GMT-5	GMT+05		5 hours ahead of GMT	05:00
Etc/GMT-6	GMT+06		6 hours ahead of GMT	06:00
Etc/GMT-7	GMT+07		7 hours ahead of GMT	07:00
Etc/GMT-8	GMT+08		8 hours ahead of GMT	08:00
Etc/GMT-9	GMT+09		9 hours ahead of GMT	09:00
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/GMT0	GMT		Greenwich Mean Time	00:00
Etc/Greenwich	GMT		Greenwich Mean Time	00:00
Etc/UCT	UCT		Greenwich Mean Time	00:00
Etc/UTC	UTC		Greenwich Mean Time	00:00
Etc/Universal	UTC		Greenwich Mean Time	00:00
Etc/Zulu	UTC		Greenwich Mean Time	00:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
GB	GMT	GB	Greenwich Mean Time	00:00
GB	BST	GB	Greenwich Mean Time	01:00
GB-Eire	GMT	GB	Greenwich Mean Time	00:00
GB-Eire	BST	GB	Greenwich Mean Time	01:00
GMT	GMT		Greenwich Mean Time	00:00
GMT+0	GMT		Greenwich Mean Time	00:00
GMT-0	GMT		Greenwich Mean Time	00:00
GMT0	GMT		Greenwich Mean Time	00:00
Greenwich	GMT		Greenwich Mean Time	00:00
Hongkong	HKT	HK	Hong Kong Standard Time	08:00
Iceland	GMT	IS	Greenwich Mean Time	00:00
Indian/Antananarivo	EAT	MG	East Africa Time	03:00
Indian/Chagos	+06	IO	Indian Ocean Time	06:00
Indian/Christmas	+07	CX	Christmas Island Time	07:00
Indian/Cocos	+0630	CC	Cocos Islands Time	06:30
Indian/Comoro	EAT	KM	East Africa Time	03:00
Indian/Kerguelen	+05	TF	French Southern and Antarctic Time	05:00
Indian/Mahe	+04	SC	Seychelles Time	04:00
Indian/Maldives	+05	MV	Maldives Time	05:00
Indian/Mauritius	+04	MU	Mauritius Standard Time	04:00
Indian/Mayotte	EAT	YT	East Africa Time	03:00
Indian/Reunion	+04	RE	Reunion Time	04:00
Iran	+0330	IR	Iran Standard Time	03:30
Iran	+0430	IR	Iran Daylight Time	04:30
Israel	IST	IL	Israel Standard Time	02:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Israel	IDT	IL	Israel Daylight Time	03:00
Jamaica	EST	JM	Eastern Standard Time	-05:00
Japan	JST	JP	Japan Standard Time	09:00
Kwajalein	+12	MH	Marshall Islands Time	12:00
Libya	EET	LY	Eastern European Standard Time	02:00
MST7MDT	MST	US	Mountain Standard Time	-07:00
MST7MDT	MDT	US	Mountain Daylight Time	-06:00
Mexico/BajaNorte	PST	MX	Mexican Pacific Standard Time	-08:00
Mexico/BajaNorte	PDT	MX	Mexican Pacific Daylight Time	-07:00
Mexico/BajaSur	MST	MX	Mexican Pacific Standard Time	-07:00
Mexico/BajaSur	MDT	MX	Mexican Pacific Daylight Time	-06:00
Mexico/General	CST	MX	Central Standard Time	-06:00
Mexico/General	CDT	MX	Central Daylight Time	-05:00
NZ	NZST	AQ	New Zealand Standard Time	12:00
NZ	NZDT	AQ	New Zealand Daylight Time	13:00
NZ-CHAT	+1245	NZ	Chatham Standard Time	12:45
NZ-CHAT	+1345	NZ	Chatham Daylight Time	13:45
Navajo	MST	US	Mountain Standard Time	-07:00
Navajo	MDT	US	Mountain Daylight Time	-06:00
PRC	CST	CN	China Standard Time	08:00
PST8PDT	PST	US	Pacific Standard Time	-08:00
PST8PDT	PDT	US	Pacific Daylight Time	-07:00
Poland	CET	PL	Central European Standard Time	01:00
Poland	CEST	PL	Central European Summer Time	02:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Portugal	WET	PT	Western European Standard Time	00:00
Portugal	WEST	PT	Western European Summer Time	01:00
ROC	CST	TW	Taipei Standard Time	08:00
ROK	KST	KR	Korean Standard Time	09:00
Singapore	+08	SG	Singapore Standard Time	08:00
Turkey	+03	TR	Eastern European Time	03:00
UCT	UCT		Greenwich Mean Time	00:00
US/Alaska	AKST	US	Alaska Standard Time	-09:00
US/Alaska	AKDT	US	Alaska Daylight Time	-08:00
US/Aleutian	HST	US	Hawaii-Aleutian Standard Time	-10:00
US/Aleutian	HDT	US	Hawaii-Aleutian Daylight Time	-09:00
US/Arizona	MST	US	Mountain Standard Time	-07:00
US/Central	CST	US	Central Standard Time	-06:00
US/Central	CDT	US	Central Daylight Time	-05:00
US/East-Indiana	EST	US	Eastern Standard Time	-05:00
US/East-Indiana	EDT	US	Eastern Daylight Time	-04:00
US/Eastern	EST	US	Eastern Standard Time	-05:00
US/Eastern	EDT	US	Eastern Daylight Time	-04:00
US/Hawaii	HST	US	Hawaii-Aleutian Standard Time	-10:00
US/Indiana-Starke	CST	US	Central Standard Time	-06:00
US/Indiana-Starke	CDT	US	Central Daylight Time	-05:00
US/Michigan	EST	US	Eastern Standard Time	-05:00
US/Michigan	EDT	US	Eastern Daylight Time	-04:00
US/Mountain	MST	US	Mountain Standard Time	-07:00
US/Mountain	MDT	US	Mountain Daylight Time	-06:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
US/Pacific	PST	US	Pacific Standard Time	-08:00
US/Pacific	PDT	US	Pacific Daylight Time	-07:00
US/Pacific-New	PST	US	Pacific Standard Time	-08:00
US/Pacific-New	PDT	US	Pacific Daylight Time	-07:00
US/Samoa	SST	UM	Samoa Standard Time	-11:00
UTC	UTC		Greenwich Mean Time	00:00
Universal	UTC		Greenwich Mean Time	00:00
W-SU	MSK	RU	Moscow Standard Time	03:00
Zulu	UTC		Greenwich Mean Time	00:00

Area: Europe

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Amsterdam	CET	NL	Central European Standard Time	01:00
Europe/Amsterdam	CEST	NL	Central European Summer Time	02:00
Europe/Andorra	CET	AD	Central European Standard Time	01:00
Europe/Andorra	CEST	AD	Central European Summer Time	02:00
Europe/Astrakhan	+04	RU	Moscow Standard Time	04:00
Europe/Athens	EET	GR	Eastern European Standard Time	02:00
Europe/Athens	EEST	GR	Eastern European Summer Time	03:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Belfast	GMT	GB	Greenwich Mean Time	00:00
Europe/Belfast	BST	GB	Greenwich Mean Time	01:00
Europe/Belgrade	CET	RS	Central European Standard Time	01:00
Europe/Belgrade	CEST	RS	Central European Summer Time	02:00
Europe/Berlin	CET	DE	Central European Standard Time	01:00
Europe/Berlin	CEST	DE	Central European Summer Time	02:00
Europe/Bratislava	CET	SK	Central European Standard Time	01:00
Europe/Bratislava	CEST	SK	Central European Summer Time	02:00
Europe/Brussels	CET	BE	Central European Standard Time	01:00
Europe/Brussels	CEST	BE	Central European Summer Time	02:00
Europe/Bucharest	EET	RO	Eastern European Standard Time	02:00
Europe/Bucharest	EEST	RO	Eastern European Summer Time	03:00
Europe/Budapest	CET	HU	Central European Standard Time	01:00
Europe/Budapest	CEST	HU	Central European Summer Time	02:00
Europe/Busingen	CET	DE	Central European Standard Time	01:00
Europe/Busingen	CEST	DE	Central European Summer Time	02:00
Europe/Chisinau	EET	MD	Eastern European Standard Time	02:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Chisinau	EEST	MD	Eastern European Summer Time	03:00
Europe/Copenhagen	CET	DK	Central European Standard Time	01:00
Europe/Copenhagen	CEST	DK	Central European Summer Time	02:00
Europe/Dublin	GMT	IE	Greenwich Mean Time	00:00
Europe/Dublin	IST	IE	Greenwich Mean Time	01:00
Europe/Gibraltar	CET	GI	Central European Standard Time	01:00
Europe/Gibraltar	CEST	GI	Central European Summer Time	02:00
Europe/Guernsey	GMT	GG	Greenwich Mean Time	00:00
Europe/Guernsey	BST	GG	Greenwich Mean Time	01:00
Europe/Helsinki	EET	FI	Eastern European Standard Time	02:00
Europe/Helsinki	EEST	FI	Eastern European Summer Time	03:00
Europe/Isle_of_Man	GMT	IM	Greenwich Mean Time	00:00
Europe/Isle_of_Man	BST	IM	Greenwich Mean Time	01:00
Europe/Istanbul	+03	TR	Eastern European Time	03:00
Europe/Jersey	GMT	JE	Greenwich Mean Time	00:00
Europe/Jersey	BST	JE	Greenwich Mean Time	01:00
Europe/Kaliningrad	EET	RU	Eastern European Standard Time	02:00
Europe/Kiev	EET	UA	Eastern European Standard Time	02:00
Europe/Kiev	EEST	UA	Eastern European Summer Time	03:00
Europe/Kirov	+03	RU	Moscow Time	03:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Lisbon	WET	PT	Western European Standard Time	00:00
Europe/Lisbon	WEST	PT	Western European Summer Time	01:00
Europe/Ljubljana	CET	SI	Central European Standard Time	01:00
Europe/Ljubljana	CEST	SI	Central European Summer Time	02:00
Europe/London	GMT	GB	Greenwich Mean Time	00:00
Europe/London	BST	GB	Greenwich Mean Time	01:00
Europe/Luxembourg	CET	LU	Central European Standard Time	1:00
Europe/Luxembourg	CEST	LU	Central European Summer Time	02:00
Europe/Madrid	CET	ES	Central European Standard Time	01:00
Europe/Madrid	CEST	ES	Central European Summer Time	02:00
Europe/Malta	CET	MT	Central European Standard Time	01:00
Europe/Malta	CEST	MT	Central European Summer Time	02:00
Europe/Mariehamn	EET	AX	Eastern European Standard Time	02:00
Europe/Mariehamn	EEST	AX	Eastern European Summer Time	03:00
Europe/Minsk	+03	BY	Moscow Standard Time	03:00
Europe/Monaco	CET	MC	Central European Standard Time	01:00
Europe/Monaco	CEST	MC	Central European Summer Time	02:00
Europe/Moscow	MSK	RU	Moscow Standard Time	03:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Nicosia	EET	CY	Eastern European Standard Time	02:00
Europe/Nicosia	EEST	CY	Eastern European Summer Time	03:00
Europe/Oslo	CET	NO	Central European Standard Time	01:00
Europe/Oslo	CEST	NO	Central European Summer Time	02:00
Europe/Paris	CET	FR	Central European Time	01:00
Europe/Paris	CEST	FR	Central European Summer Time	02:00
Europe/Podgorica	CET	ME	Central European Standard Time	01:00
Europe/Podgorica	CEST	ME	Central European Summer Time	02:00
Europe/Prague	CET	CZ	Central European Standard Time	01:00
Europe/Prague	CEST	CZ	Central European Summer Time	02:00
Europe/Riga	EET	LV	Eastern European Standard Time	02:00
Europe/Riga	EEST	LV	Eastern European Summer Time	03:00
Europe/Rome	CET	IT	Central European Standard Time	01:00
Europe/Rome	CEST	IT	Central European Summer Time	02:00
Europe/Samara	+04	RU	Samara Standard Time	04:00
Europe/San_Marino	CET	SM	Central European Standard Time	01:00
Europe/San_Marino	CEST	SM	Central European Summer Time	02:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Sarajevo	CET	BA	Central European Standard Time	01:00
Europe/Sarajevo	CEST	BA	Central European Summer Time	02:00
Europe/Saratov	+04	RU	Moscow Standard Time	04:00
Europe/Simferopol	MSK	UA	Moscow StandardTime	03:00
Europe/Skopje	CET	MK	Central European Standard Time	01:00
Europe/Skopje	CEST	MK	Central European Summer Time	02:00
Europe/Sofia	EET	BG	Eastern European Standard Time	02:00
Europe/Sofia	EEST	BG	Eastern European Summer Time	03:00
Europe/Stockholm	CET	SE	Central European Standard Time	01:00
Europe/Stockholm	CEST	SE	Central European Summer Time	02:00
Europe/Tallinn	EET	EE	Eastern European Standard Time	02:00
Europe/Tallinn	EEST	EE	Eastern European Summer Time	03:00
Europe/Tirane	CET	AL	Central European Standard Time	01:00
Europe/Tirane	CEST	AL	Central European Summer Time	02:00
Europe/Tiraspol	EET	MD	Eastern European Standard Time	02:00
Europe/Tiraspol	EEST	MD	Eastern European Summer Time	03:00
Europe/Ulyanovsk	+04	RU	Moscow Standard Time	04:00
Europe/Uzhgorod	EET	UA	Eastern European Standard Time	02:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Uzhgorod	EEST	UA	Eastern European Summer Time	03:00
Europe/Vaduz	CET	LI	Central European Standard Time	01:00
Europe/Vaduz	CEST	LI	Central European Summer Time	02:00
Europe/Vatican	CET	VA	Central European Standard Time	01:00
Europe/Vatican	CEST	VA	Central European Summer Time	02:00
Europe/Vienna	CET	AT	Central European Standard Time	01:00
Europe/Vienna	CEST	AT	Central European Summer Time	02:00
Europe/Vilnius	EET	LT	Eastern European Standard Time	02:00
Europe/Vilnius	EEST	LT	Eastern European Summer Time	03:00
Europe/Volgograd	+04	RU	Volgograd Standard Time	04:00
Europe/Warsaw	CET	PL	Central European Standard Time	01:00
Europe/Warsaw	CEST	PL	Central European Summer Time	02:00
Europe/Zagreb	CET	HR	Central European Standard Time	01:00
Europe/Zagreb	CEST	HR	Central European Summer Time	02:00
Europe/Zaporozhye	EET	UA	Eastern European Standard Time	02:00
Europe/Zaporozhye	EEST	UA	Eastern European Summer Time	03:00
Europe/Zurich	CET	CH	Central European Standard Time	01:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Europe/Zurich	CEST	CH	Central European Summer Time	02:00

Area: Pacific

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Pacific/Apia	+13	WS	Apia Standard Time	13:00
Pacific/Apia	+14	WS	Apia 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/Bougainville	+11	PG	Papua New Guinea Time	11:00
Pacific/Chatham	+1245	NZ	Chatham Standard Time	12:45
Pacific/Chatham	+1345	NZ	Chatham Daylight Time	13:45
Pacific/Chuuk	+10	FM	Chuuk Time	10:00
Pacific/Easter	-06	CL	Easter Island Standard Time	-06:00
Pacific/Easter	-05	CL	Easter Island Summer Time	-05:00
Pacific/Efate	+11	VU	Vanuatu Standard Time	11:00
Pacific/Enderbury	+13	KI	Phoenix Islands Time	13:00
Pacific/Fakaofo	+13	TK	Tokelau Time	13:00
Pacific/Fiji	+12	FJ	Fiji Standard Time	12:00
Pacific/Fiji	+13	FJ	Fiji Summer Time	13:00
Pacific/Funafuti	+12	TV	Tuvalu Time	12:00
Pacific/Galapagos	-06	EC	Galapagos Time	-06:00
Pacific/Gambier	-09	PF	Gambier Time	-09:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Pacific/Guadalcanal	+11	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	US	Hawaii-Aleutian Standard Time	-10:00
Pacific/Kiritimati	+14	KI	Line Islands Time	14:00
Pacific/Kosrae	+11	FM	Kosrae Time	11:00
Pacific/Kwajalein	+12	MH	Marshall Islands Time	12:00
Pacific/Majuro	+12	MH	Marshall Islands Time	12:00
Pacific/Marquesas	-0930	PF	Marquesas Time	-09:30
Pacific/Midway	SST	UM	Samoa Standard Time	-11:00
Pacific/Nauru	+12	NR	Nauru Time	12:00
Pacific/Niue	-11	NU	Niue Time	-11:00
Pacific/Norfolk	+11	NF	Norfolk Island Time	11:00
Pacific/Noumea	+11	NC	New Caledonia Standard Time	11:00
Pacific/Pago_Pago	SST	AS	Samoa Standard Time	-11:00
Pacific/Palau	+09	PW	Palau Time	09:00
Pacific/Pitcairn	-08	PN	Pitcairn Time	-08:00
Pacific/Pohnpei	PONT	FM	Ponape Time	11:00
Pacific/Ponape	+11	FM	Ponape Time	11:00
Pacific/Port_Moresby	+10	PG	Papua New Guinea Time	10:00
Pacific/Rarotonga	-10	CK	Cook Islands Standard Time	-10:00
Pacific/Saipan	CHST	MP	Chamorro Standard Time	10:00
Pacific/Samoa	SST	UM	Samoa Standard Time	-11:00
Pacific/Tahiti	-10	PF	Tahiti Time	-10:00
Pacific/Tarawa	+12	KI	Gilbert Islands Time	12:00
Pacific/Tongatapu	+13	TO	Tonga Standard Time	13:00

Time Zone Information	Time Zone Name	Region	Time Zone Description	Time Zone Offset
Pacific/Tongatapu	+14	TO	Tonga Summer Time	14:00
Pacific/Truk	+10	FM	Chuuk Time	10:00
Pacific/Wake	+12	UM	Wake Island Time	12:00
Pacific/Wallis	+12	WF	Wallis and Futuna Time	12:00
Pacific/Yap	+10	FM	Chuuk Time	10:00

25

TK_NLS_REPLACE_ICVEXT

Dictionary 1193
TK_NLS_REPLACE_ICVEXT Environment Variable 1193

Dictionary

TK_NLS_REPLACE_ICVEXT Environment Variable

Access a new transcoding table at startup.

Valid in: Configuration file

Restriction: You must specify the icvywinu translation table in order to correctly display the Hong Kong supplementary character set. For example:

```
export TK_NLS_REPLACE_ICVEXT=icvywinu;
```

Note: Beginning in 2025.03 this environment variable is new.

Example:

```
export TK_NLS_REPLACE_ICVEXT=translation-table  (sh/bash)

setenv TK_NLS_REPLACE_ICVEXT translation-table  (csh/tcsh)
```

Syntax

TK_NLS_REPLACE_ICVEXT=*translation-table*;

Required Argument

translation-table
specifies the new translation table.

Details

TK_NLS_REPLACE_ICVEXT specifies a new transcoding table at startup by replacing the default transcoding table. You can disable or enable the new transcoding table.The new transcoding table remains active for the duration of the SAS session.

You can enable TK_NLS_REPLACE_ICVEXT on Viya4 by using the Environment Manager to configure a compute server session.

You can restore the original transcoding table by removing the new transcoding table from the directory and restarting SAS sessions.

Users of the Hong Kong supplementary character set must specify the icvywinu translation table in order to view accurate characters

Table A14.1 *Translation Table - icvywinu*

Translation-table		Description
icvywinu	BIG5 MS-950 (YWIN) MACOS-2 (YMAC)	BIG5-HKSCS mapping table