Paper 254-2012

Write Once, Run Anywhere!
How to Make Your SAS® Applications Speak Many Languages

Mickaël Bouedo, SAS Institute Inc., Cary, NC

ABSTRACT

Do you have SAS users worldwide? Do you want your SAS application to be useable in many languages? SAS® 9.3 internationalization features will get you there efficiently.

If you want to offer your SAS application in 2, 4, or 10 languages, SAS internationalization features help you write your code once so it can run in many languages. Internationalization is the step which generalizes your product to be language independent. Localization is the second step adapts the product to meet the needs of different languages and cultures.

This paper describes how to successfully internationalize and localize your SAS programs and make them ready for the world.

INTRODUCTION

Your customers and employees today likely speak many different languages. Why should your SAS applications targeted to those customers and employees only speak one? Preparing your applications for a global audience requires the customization of your software for each target market. Typically, users want the user interface to appear in a language they understand. It should match the user's cultural characteristics, taking into account the complexity of their scripts as well as local conventions to format numbers, currency, dates, and times. Fonts, images, colors, addresses, page layout, and sorting are among other cultural differences to keep in mind.

How do you create an application for multiple languages that you might not even speak?
The answer is through globalization technologies.

Software globalization, also abbreviated as “G11N”, is the combined work of internationalization and localization. It means designing software for global markets and then adapting that internationalized software to a specific locale.

Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. The term *internationalization* is often abbreviated as “I18N”, because there are 18 letters between the first "i" and the last "n."

While the internationalization process prepares your application for the world, the localization process adapts it to the world. Localization is the process of modifying products or services to account for differences in distinct markets. With regards to software, it means adapting the software for a particular geographical region or for a set of users with common language and locale-specific customs. The term *localization* is often abbreviated as “L10N”, because there are 10 letters between the "l" and the "n."

One of the primary tasks of localization is translating the user interface elements that have been externalized in the internationalization process. Localization involves changing not only the language interaction but also other relevant changes such as the display of numbers, dates, currency, and so on. Other types of data, such as sounds and images, may require localization if they are culturally sensitive.

According to Microsoft’s documentation, an internationalized program has the two following characteristics:

- World readiness: The ability for the application to support the customer’s data, format and sort according to the customer’s preferences.
  The first four sections of this paper are related to this category. The first section describes two essential SAS options to the I18N process. The second section introduces DATA step functions to support international data. The next two cover international formatting and linguistic sorting.

- Localizability: The act of preparing the software so that localization is possible. This includes externalizing text strings from source code and avoiding truncation of strings in the UI. The last section of this paper covers this category because it introduces a new DATA step function to help with string externalization.

Without a proper and well-designed internationalization process, localization is not possible without re-writing the application for each language/culture. The better internationalized an application is, the easier it is to localize it for a particular language and character encoding scheme.

INTERNATIONALIZATION CONCEPTS WITH THE SAS SYSTEM

In order to write applications for international users, it becomes critical to design applications that interact with users in their native language using local conventions.

Two key components in the internationalization design are the locale and the encoding. The locale is the link between the user and your application. It allows the application to know the user's cultural preferences. The encoding defines the character sets that SAS uses to process syntax and user data.

Let's have a closer look at these two key concepts.

WHAT IS A LOCALE?

A *locale* is a set of conventions for handling written language text (for example, word delimitation, quoting, or sorting) and various units (for example, numeric value, date and time formatting conventions, or currency conventions).

A locale identifies a specific user preference—or a group of users who have similar cultural and linguistic expectations for human-computer interaction (and the type of data they process). A locale's identifier is a label for a given group of settings. For example, "en" (representing the English language) and "fr" (representing the French language) are identifiers for a cultural locale.

A locale represents a language in a geographical region. More than one locale can be associated with a particular language, which allows for regional differences. For example, an English-speaking user in the United States will use "en_US" (English language spoken in the United States) as the locale identifier, while an English-speaking user in Great Britain will use "en_GB" (English spoken in Great Britain).

In SAS, the LOCALE= system option must be used to specify the locale of the SAS session. The option can be set at SAS invocation and changed during the session. SAS supports two notations, a long one in the form Language_territory and a short one in the form ll_RR (where ll represents the 2-character language code and RR represents the 2-character region code).

LOCALE option settings – short name	LOCALE option settings – long name

`Option locale = en_US ;`
`Option locale = fr_CA ;`
`Option locale = ja_JP ;`
`Option locale = zh_CN ;`
	`Option locale = English_UnitedStates ;`
	`Option locale = French_Canada ;`
	`Option locale = Japanese_Japan ;`
	`Option locale = Chinese_China ;`

Table 1. Example of LOCALE option values

Refer to "Values for the LOCALE= System Option SAS Session" in SAS® 9.3 National Language Support (NLS): Reference Guide for a complete list of locale values supported by SAS 9.3.

The LOCALE option is an important option in the I18N design of SAS applications. Some components or processes such as formatting, sorting, or string externalization rely on this option to provide locale-sensitive information. The LOCALE option is the link between your application and the user regarding his or her cultural preferences.

When the LOCALE option is set, other options such as the ENCODING, PAPERSIZE, and DATESTYLE options are also set to an appropriate value that matches the user's culture unless an explicit value has been set for these options. Table 2 shows how these options are set based on the LOCALE option. The LOCALE option can change the ENCODING option only at start up. The encoding value is platform-dependent.

When LOCALE is set to	en_US	ro_RO	ja_JP
ENCODING is set to	WLATIN1	WLATIN2	SHIFT-JIS
PAPERSIZE is set to	LETTER	A4	A4
DATESTYLE is set to	MDY	DMY	YMD

Table 2. Example of ENCODING, PAPERSIZE, and DATESTYLE option settings on Windows based on the LOCALE option value

Tips to retrieve the LOCALE option value.

In SAS programming, you may need to retrieve the LOCALE option value. In addition to the OPTIONS procedure, which allows you to visualize the value in the log, here are other methods to get the LOCALE option value:

- **The SQL procedure and a macro variable**

```
proc sql ;
  select distinct setting into:ploc
  from dictionary.options
  where optname="LOCALE" ;
quit;
%put my locale is &ploc  ;
```

```
/* result */


my locale is ENGLISH_UNITEDSTATES
```

- **The GetOption() function**

```
%LET PLOC=%SYSFUNC(getOption(LOCALE));
%PUT &PLOC ;
```

```
/* result */
ENGLISH_UNITEDSTATES
```

- **The GetpxLocale(), GetpxLanguage(), or GetpxRegion() functions**

The GETPXLOCALE function returns the five-letter POSIX locale value for the current SAS session.
The GETPXLANGUAGE function returns the two-letter language.
The GETPXREGION function returns the two-letter region code.

```
data _null_ ;
  locale=getpxLocale() ;
  lang=getpxLanguage() ;
  region=getpxRegion() ;
  put locale=  /lang=  /region=;
run ;
```

```
/* result */

locale=en_US
lang=en
region=US
```

CONCEPT OF SESSION ENCODING

An *encoding* is a set of abstract characters (letters, East Asian logograms, digits, punctuation marks, symbols, and control characters) that have been mapped to hexadecimal values (called *code points*) that can be used by computers. For example, IBM-1047 is an encoding in the EBCDIC family used mainly on the mainframe, Windows Cyrillic is an encoding in the Windows family, and Latin1 is an encoding in the ISO 8859 family.

Encodings have many properties. One possible categorization would be as follow:

- Single-Byte Character Set (SBCS) encodings, which represent each character in a single byte. Each SBCS encoding supports 256 code points, which limits the number of characters available in the encoding. Some code points are reserved to interpret the control characters from the system. This number is more than enough to support languages based on Latin script like most European languages. However, this

is definitely not enough for other languages with a more complex script, such as the Chinese language or the Japanese language.

- Multi-Byte Character Set (MBCS) encodings require one or more bytes to represent each character. Typically, a set of code points is reserved to represent the first byte of the set and is meaningful only if it is immediately followed by a defined second, third, or fourth byte. One of the most common MBCS encodings is UTF-8, where a character can be represented with up to 4 bytes. The Double-Byte Character Set (DBCS) is a form of MBCS.

MBCS encoded data can offer challenges for SAS programmers. When programming and manipulating MBCS data, you have to write code that would treat a set of one or more code points as one character. The next section, "Text String and Character Manipulation", explains how SAS can help you to process data in such encodings.

The SAS System option ENCODING= is used to specify what encoding to use to process data internally. The option value is also referred to as the *SAS Session encoding*. This option can be set at SAS invocation only.

The languages you plan to support in your application determine the encoding value to select. SAS supports different encodings for different scripts. The following table gives a basic overview of the encodings that are associated with each writing system.

Scripts	**SAS Encoding Name**	**Description -** also known as
Many	UTF-8	Unicode family encodings – Supports all languages
Arabic	warabic, arabic, open_ed-425	Windows 1256, ISO-8859-6 ibm-425 (ebcdic)
Simplified Chinese	euc-cn, open_ed-935	GBK ibm-935 (ebcdic)
Traditional Chinese	euc-tw, ms-950, open_ed-937	Windows 950 ibm-937 (ebcdic)
Cyrillic	wcyrillic, cyrillic, open_ed-1025	Windows1251, ISO-8859-5
English, Western and Eastern European, or other Latin script	wlatin1, latin1, latin9, open_ed-1047, 1140-1149	Windows1252, ISO-8859-1, ISO-8859-15
	wlatin2, latin2 open_ed-870, open_ed-1153	Windows1250, ISO-8859-2 (Eastern Europe)
	wlatin5, latin5, open_ed-1026	ISO-8859-9 (Turkish language)
	wlatin6, latin6	ISO-8859-4 (Baltic languages)
Greek	wgreek, greek, open_ed-875	Windows 1253, ISO-8859-7
Hebrew	whebrew, Hebrew, open_ed-424	Windows 1255, ISO-8859-8
Japanese	shift-jis, euc-jp open_ed-939	Windows 932 ibm-939 (ebcdic)
Korean	euc-kr, open_ed-933	Windows 949 ibm-933 (ebcdic)
Thai	pcoem874 open_ed-1160	Windows 874 ibm-1160 (ebcdic)
Vietnamese	wvietnamese, open_ed-1164	Windows 1258 ibm-1164 (ebcdic)

Table 3. List of writing systems and associated encodings. Refer to "Encoding Values for a SAS Session" in SAS® 9.3 National Language Support (NLS): Reference Guide for a complete list of encoding values.

UTF-8: One encoding for many languages

If one of the requirements for your SAS application is to create multilingual documents, then UTF-8 is most likely the encoding to choose. UTF-8 is a Unicode multi-byte encoding that allows you to process characters from various scripts in a single SAS session.

For more information about the concept of the SAS Unicode Server, see the technical Paper, "Processing Multilingual Data with the SAS® 9.2 Unicode Server."

TEXT STRING AND CHARACTER MANIPULATION

SAS provides many string functions and call routines that can be used to manipulate characters and strings. The original SAS string-handling functions assume the size of a character is always one byte, which is true with a single-byte character set (SBCS). However, multi-byte character sets (DBCS or MBCS) require one byte or more to represent the characters in the set. Using the original SAS string-handling functions with multi-byte data can lead to unexpected behavior, such as data truncation.

To resolve issues that these string functions can cause for multi-byte data, SAS provides a set of string functions, called *K functions*, which do not make assumptions about the size of a character in a string. To use K functions, you need to understand the difference between byte-based offset and character-based offset.

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, since one character is always 1 byte in length, you can assume that the second character in the string begins in byte 2 of the string. However, if the data in the string is multi-byte data, the data in byte 2 may 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, or
- 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. K functions can be used for processing SBCS, DBCS, and MBCS (UTF-8) data in SAS.

A typical problem

Consider the following string, which is represented by its hexadecimal value:

```
STR='8361836262'x
```

In a double-byte encoding such as Shift-JIS, this string has 5 bytes and represents three characters. In a single-byte encoding such as LATIN1, this same string represents something entirely different. It represents five characters. as shown in the Figure 1.

Shift-JIS Characters	ガ		ッ		**b**
Hexadecimal Representation	83	61	83	62	62
Latin1 Characters	**ƒ**	**a**	**ƒ**	**b**	**b**

Figure 1. String representation in Latin1 and Shift-JIS encoding

The following example uses this same hexadecimal string in the INDEX and SCAN functions.
The INDEX function returns a value of 4 for both the SBCS and the multi-byte environments. However, this is not the expected result for the multi-byte environment because the sequence of characters **'bb'** does not exist in the string in a Shift-JIS encoding. In a multi-byte environment, the result should be 0.
With the SCAN function, the expected returned string would be the first two Japanese characters. However, the function returns only the first one followed by another character that does not even seem to be in the string.

```
/* SAS program to submit in a MBCS SAS session */
data _null_ ;
  str= "ヂッb" ;
  i=index(str, "bb");
  s=scan(str,1,'b' ) ;
  l=length(str) ;
  put str= $hex10. /i= /s= l=;
run ;
```

```
/* Results */

str=8361836262
i=4
s=ヂ•
l=5
```

Code box 1. Code example using regular functions

Solution
To avoid such issues, you can substitute the INDEX and SCAN functions, which only work on single-byte data, with the KINDEX and KSCAN functions, which work on single- and multi-byte data. Direct substitutions can also be made for other string-handling functions, such as changing the LENGTHN() or SUBSTR() functions to use the KLENGTH() or KSUBSTR() functions instead. Table 4 provides a list of available K functions.

```
/* SAS program to submit in a MBCS SAS session */
data _null_ ;
  str= "ヂッb" ;
  i=kindex(str, "bb");
  s=kscan(str,1,'b' ) ;
  l=klength(str) ;
  put str= $hex10. /i=  /s= /l=;
run ;
```

```
/* Results */

str=8361836262
i=0
s=ヂッ
l=3
```

Code box 2. Code example using K functions

Because the two sets of functions read the data differently, you get different results when running this code in a multi-byte environment.
The INDEX function is byte-oriented, so it reads the individual bytes of data. As such, the INDEX function finds the hexadecimal byte sequence ‘6262’x, which it incorrectly interprets as the sequence **'bb'** and returns a value of 4.
The KINDEX function, on the other hand, returns a value that is character-based and interprets the first '62'x as the second byte of the DBCS character rather than as an ASCII '**b**'. Therefore, KINDEX returns 0 instead of 4.
Like the INDEX function, the SCAN function is byte-oriented. The function truncates the string at the first ‘62’x byte found in the string and returns a truncated value, whereas the KSCAN function detects the double-byte character and returns the expected result.

K Function Compatibility

The following table lists the original SAS string-handling functions and the equivalent K functions. Note that a few of the original string-handling functions support multi-byte data correctly and can continue to be used in a multi-byte environment. For example, UPCASE and LOWCASE operate correctly on multi-byte data.

There are some compatibility issues to consider when you use K functions. For example, while the original TRIM function correctly processes multi-byte characters, it does not remove double-byte blanks, so there are some behavior differences. Table 4 only notes major differences, such as differences in the arguments. For details about these differences, refer to “Internationalization Compatibility for SAS String Functions” in the SAS® 9.3 National Language Support (NLS): Reference Guide.

SAS Functions	Equivalent K Functions	Description	Is K Function Required for multi-byte characters?	Is K function a direct replacement?
CAT	KSTRCAT	Concatenates character strings without removing leading or trailing blanks.	CAT cannot handle contiguous SO/SI characters properly under an EBCDIC environment	Yes
COMPARE	KCOMPARE	Returns the result of a comparison of character strings.	Yes	arguments are different
COMPRESS	KCOMPRESS	Removes specific characters from a character string.	Yes	arguments are different
	KCOUNT	Returns the number of multi-byte characters in a string. KCOUNT is not compatible with COUNT.		
	KCVT	Converts data from on type of encoding data to another type of encoding data.		
INDEX	KINDEX	Searches a character expression for a string of characters.	Yes	Yes
INDEXC	KINDEXC	Searches a character expression for specific characters.	Yes	Yes
LEFT	KLEFT	Left aligns a character expression by removing unnecessary leading DBCS blanks.	LEFT is acceptable if DBSC blanks are not used	Yes
LENGTHN	KLENGTH	Returns the length of an argument.	Yes	Yes
LOWCASE	KLOWCASE	Converts all letters in an argument to lowercase.	No	Yes
REVERSE	KREVERSE	Reverses a character expression.	Yes	Yes
RIGHT	KRIGHT	Right aligns a character expression by trimming trailing DBCS blanks.	RIGHT is acceptable if DBCS blanks are not used	Yes
SCAN	KSCAN	Selects a specified word from a character expression.	Yes	arguments are different
SUBSTR	KSUBSTR	Extracts a substring from an argument.	Yes	Yes
	KSUBSTRB	Extracts a substring from an argument according to the byte position of the substring in the argument.		
TRANSLATE	KTRANSLATE	Replaces specific characters in a character expression.	Yes	direct replacement but does not support binary data like TRANSLATE
TRIM	KTRIM	Removes trailing DBCS blanks from character expressions.	Yes	Yes
	KTRUNCATE	Truncates a character string value to a specified length.		
UPCASE	KUPCASE	Converts all single-byte letters in an argument to uppercase.	No	Yes
	KUPDATE	Inserts, deletes, and replaces the contents of the character value according to the character-based position of the character value in the argument.		
	KUPDATEB	Inserts, deletes, and replaces the contents of the character value according to the byte position of the character value in the argument.		
VERIFY	KVERIFY	Returns the position of the first character that is unique to an expression.	Yes	Yes

Table 4. Differences between K Functions and Regular SAS Functions

Recommendations

The K functions work in a SBCS environment. However, before replacing all of the original SAS string-handling functions with K functions, examine your SAS program. If the string function processes data that will only contain single-byte characters, there is no need to use K functions. 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.

LOCALE-SENSITIVE FORMATTING

Around the world, people of all cultures encounter different ways of writing dates, times, and numbers. Though most cultures convey the same information, there are differences in the way this data is formatted.
When you display data to a user in an international context, it should be formatted according to the conventions of the user's native country, region, or culture. When users enter data, they may do so according to their own customs or preferences.
To take an example, in the United States the string "15.125" represents the decimal number fifteen and one eighth, whereas in Germany it represents fifteen thousand one hundred and twenty-five.

SAS provides support to read and write such locale-sensitive data. SAS language elements called National Language Support (NLS) formats and informats allow SAS to convert date, datetime, currency, and numeric values back and forth in a locale-sensitive manner.

Informats read notations or expressions, such as a number, a clock time, or a calendar date, which might be in a variety of lengths based on the culture, and then convert the data to a SAS value.
Formats write a value, recognized by SAS, such as a time or date value, as a calendar date or clock time in a variety of lengths and notations according to the SAS locale session.

1. DATE

Cultures around the world have different ways of representing dates. Date formats do not necessarily rely on language; in fact, even speakers using the same language from different countries may format dates differently. Writing a date requires placing the date's elements (e.g., the month, the day, and the year) in a way that is understandable to the reader.

For example, what date represents "04/02/12"? This date is interpreted differently depending on your cultural background. In the United States, it represents "02 April, 2012". In Japan, it is interpreted as "12 February, 2004". And in parts of Europe, it means "04 February, 2012".

SAS has several formats and informats listed in the table below to process date values in a locale-sensitive manner.

Name	Description	Option locale=en_US ;	Option locale=de_DE;
NLDATE	Date	`January 22, 2012`	`22. Januar 2012`
NLDATEMD	Day and name of the month	`January 22`	`22 Januar`
NLDATEMN	Name of the month	`January`	`Januar`
NLDATEW	Day and date of the week	`Sunday, January 22, 2012`	`Sonntag, 22. Januar 2012`
NLDATEWN	Day of the week	`Sunday`	`Sonntag`
NLDATEYM	Year and name of the month	`January 2012`	`Januar 2012`
NLDATEYQ	Year and quarter	`1st quarter 2012`	`1. Quartal 2012`
NLDATEYR	Year	`2012`	`2012`
NLDATEYW	Year and week	`Week 04 2012`	`Week 03 2012`

Table 5. List of the Formats and Informats Available for a Date Value

2. TIME AND DATETIME

Along with date formats, time and datetime formats differ culturally. Time formats are usually divided between 12-hour formats and 24-hour formats. In the United States, for instance, the 12-hour time notation is the most common

way to present the time. Most other countries in the world use a 24-hour time format, also known as *military time* in the United States.

Name	Description	Option locale=en_US ;	Option locale=fr_FR;
NLTIME	Time	23:52:12	23 h 52
NLTIMAP	Time value with a.m. or p.m.	11:52 PM	23 h 52
NLDATM	Date and time	22Jan12:23:52:12	22 janvier 2012 23 h 52
NLDATMAP	Date and time with a.m. or p.m.	January 22, 2012 11:52:12 PM	22 janvier 2012 23 h 52
NLDATMDT	Date	January 22, 2012	22 janvier 2012
NLDATMMD	Name and day of the month	January 22	22 janvier
NLDATMMN	Name of the month	January	Janvier
NLDATMTM	Time of the day	23:46:12	23 h 52
NLDATMW	Day of the week and datetime	Sunday, Jan 22, 2012 11:52:12 PM	dimanche 22 janvier 2012 23 h 52
NLDATMWN	Day of the week	Sunday	Dimanche
NLDATMWZ	Day, datetime, and time zone	Sun, Jan 22, 2012 11:52:12 PM -0500	Dim. 22 janvier 2012 23 h 52 -0500
NLDATMYM	Year and name of the month	January 2012	janvier 2012
NLDATMYQ	Year and quarter of the year	1st quarter 2012	1er trimestre 2012
NLDATMYR	Year	2012	2012
NLDATMYW	Year and name of the week	Week 04 2012	Week 03 2012
NLDATMZ	Time zone and datetime	22Jan12:23:52:12 -0500	22 janvier 2012 23 h 52 -0500

Table 6. List of the Formats and Informats Available for a Datetime Value

3.NUMBERS

In an international environment, formatting numbers properly and uniformly is essential to avoid miscommunication. The most common cultural differences regarding number formatting are decimal group separators and negative representation.

In the United States, a comma separates the thousands and a period separates a whole number from a decimal value. In some countries, like Germany, the two punctuation marks have reversed roles. In other countries, different characters are used.

Name	Description	Option locale=en_US;	Option locale=ru_RU;
NLBEST	the best numerical notation	-123456.789	-123456,789
NLNUM	numeric format of the local expression	-123,456.79	-123 456,79
NLNUMI	numeric format of the international expression	-123,456.79	-123,456.79
NLPCT	percentage data of the local expression	56.84%	56,84%
NLPCTI	percentage data of the international expression	56.84%	56.84%
NLPCTN	percentages, using a minus sign for negative values	56.84%	56,84%
NLPCTP	locale-specific numeric values as percentages	56.84%	56,84%
NLPVALUE	p-values of the local expression	0.57	0,57

Table 7. List of the Formats and Informats Available for Numbers

4.CURRENCY

Formatting currency usually follows the same rules as formatting numbers; however, the position of the currency symbol varies widely from country to country.

Name	Description	Option locale=en_US;	Option locale=es_ES;
NLMNY	monetary format of the local expression	($127,654.30)	-127.654,30€
NLMNYI	monetary format of the international expression	(USD127,654.30)	-127.654,30EUR

Table 8. List of the Formats and Informats Available for Currency

The formats and informats listed here only change currency symbols and patterns based on the locale and simply display the number found in the data. **NO currency conversion is performed**.

To cover the case where the currency is fixed but the format is locale-sensitive, SAS supports additional NLS formats and informats. They use the following naming convention: NLMNL*xxx* or NLMNI*xxx*, where *xxx* is the international code for a given currency.
About 37 currencies are supported through these formats. The table below lists some of these formats and informats.

Name	Description	Option locale=en_US;	Option locale=es_ES;
NLMNLEUR	Writes the monetary format of the local expression for the Euro currency.	(€12,358.00)	-12.358,00€
NLMNIEUR	Writes the monetary format of the international expression for the Euro currency.	(EUR12,358.00)	-12.358,00EUR
NLMNLCNY	Writes the monetary format of the local expression with the Chinese currency.	(RMB12,358.00)	-12.358,00RMB
NLMNICNY	Writes the monetary format of the international expression with the Chinese currency.	(CNY12,358.00)	-12.358,00CNY

Table 9. Additional Formats and Informats Available for Currency

COLLATION IN SAS

To sort data in a linguistically correct order, use PROC SORT with the option SORTSEQ=LINGUISTIC. This option causes the SORT procedure to use the default collation rules consistent with the current LOCALE= setting.

```
/* Sorting based on the current SAS Locale session */
PROC SORT data=mydata SORTSEQ=LINGUISTIC;
```

Code box 3. An example to sort data based on the current locale session

Additional options can be specified to customize the collation order, such as the LOCALE option, to get data sorted according to the rules for a specific locale.

```
/* Sorting for a specific locale */
PROC SORT data=mydata SORTSEQ=LINGUISTIC (LOCALE=fr_FR);
```

Code box 4. An example to sort data for a specific locale

More information on the SORT procedure is available in the SGF paper 297-2007 "Creating Order out of Character Chaos" and in the technical paper "Linguistic Collation: Everyone Can Get What They Expect".

STRING EXTERNALIZATION

Localization of a product requires translation of the text that appears to your customers. For products written in SAS, it means strings in the SAS code must be translated.
One way to accomplish this would be to provide customized localizations of your application. A copy of each .sas file that contains hard-coded strings would be needed, and it would result in a separate version of your application for each language. This is a tedious approach that could produce results that are different from one language to another and would require high maintenance and testing costs. Every time a new feature is needed in your application, the feature must be added to each localized version.
The alternative to customized localizations is the string externalization process, which is part of the internalization approach.
This section of the paper describes this simple process, which you can use to localize your own SAS programs. The process involves moving hard-coded strings from your SAS code to a special properties file, called a .smd file. After the strings are in the .smd file, a copy of the .smd file is created for each translation of your application. When translations are complete, you run a SAS macro (%smd2ds) that puts the original and translated messages into a SAS data set. Finally, you modify your code to call the SASMSG function to retrieve the strings properly.

STEP 1 – IDENTIFY AND MOVE TRANSLATABLE STRINGS INTO A .SMD FILE

Here are some examples of text strings that are hardcoded in SAS code. As you can see, some of these strings contain macro variables that substitute additional text.

```
put "Welcome back &user!" ;
TITLE "Hello &user! Today's date is &today";
FOOTNOTE  "Report generated on &SYSDATE"  ;
%PUT  The file "&aFile" does not exist. ;
attrib room label="Number of rooms";
proc format ; value answer 1="Yes"  2="No" ; run ;
```

Code box 5. Examples of embedded strings in SAS code

All text strings in your SAS programs must be removed and placed in a file with the .smd extension. A key, or unique name, must be associated with each translatable string moved to the .smd file. Each string and its key are stored as a pair of strings, often referred to as a *key/value pair* of strings.

The format to store the key/value pair uses the equal sign as a delimiter. The key name must contain 7-bits ASCII characters only and the size is limited to 60 characters.

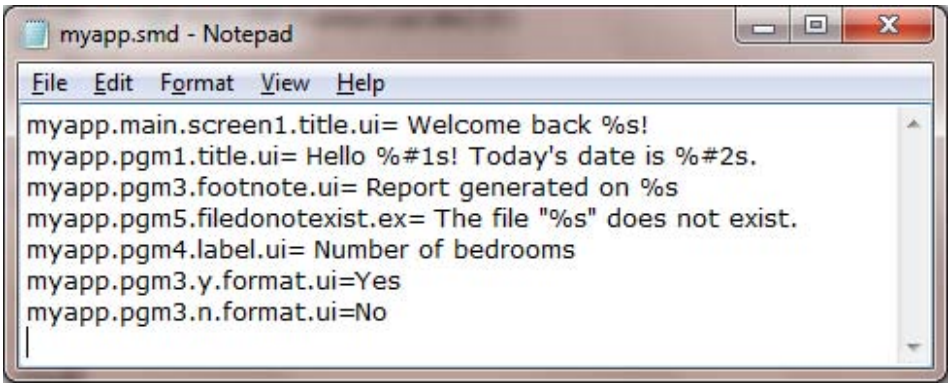


Figure 1. Example of a .smd file

Naming and Character Encoding of the .smd Files

The name of the .smd file is used as the name of a SAS data set. Any restriction that applies to a SAS data set name also applies to the name of the .smd file.

When you create a localized version of your .smd file, you use the name of the original .smd file and append an underscore (_) followed by the language or locale identifier. Therefore, do not use a name for your .smd file that ends with the underscore character _ followed by two letters because this ending is reserved for localized files.

For example, if you had created a file myapp.smd and the file is translated into Spanish for Mexico and French, you

should have the following .smd files in your directory:

myapp.smd
myapp_es_MX.smd
myapp_fr.smd

The .smd files must be created in an encoding-neutral format called 7bits-ASCII. All characters that cannot be expressed as ASCII must be represented with the Unicode escape sequence, which uses the form \u*xxxx*, where *xxxx* is the hexadecimal representation of the Unicode character. For example, the character "small letter e with acute accent" (é) would be represented as \u00E9 in the .smd file.

A tool or utility can be used to convert the file to use Unicode escapes for non-ASCII characters. The Java tool native2ascii is a handy tool build-in in the JDK, which is used to convert a file with non-Latin 1 or non-Unicode characters to Unicode-encoded characters using the Unicode escape sequence.

SAS can also perform such conversion using the KPROPDATA function or the UNICODEC function.

```
%MACRO SMD2ASCII(INF=,OUTF=,LRECL=,INENCODING=) ;
   DATA _NULL_;
     attrib tmp length=$ &LRECL ;
     INFILE "&INF" lrecl=&LRECL  ;
     INPUT ;
     FILE "&OUTF" lrecl=&LRECL ;
     tmp = kpropdata(_INFILE_,"UESC", "&INENCODING","ASCII") ;
     put tmp ;
RUN ;
%MEND SMD2ASCII ;

%SMD2ASCII( INF = U:\tmp\myprod_pl_wlt2.smd,
            OUTF = U:\smd\ascii\myprod_pl.smd,
            INENCODING=wlatin2,
            LRECL=300 ) ;
```

Code Box 6. Example of a macro to convert a .smd file from a given encoding to ASCII with Unicode escape notation

Providing context for externalized strings

One of the drawbacks to removing strings from SAS code is that the translators will not know the context in which the string was used. In order to provide useful information to help translators and the localization process, here are a few suggestions you can apply:

- Create your key names with a naming convention that gives a hint about how the text is used. As an example, you could add a suffix to the key to categorize your strings:
  - `my_key_title` where the suffix "_title" indicates the string is used as a title.
  - `my_key_footn` for a string used as a footnote statement.
  - `my_key_label` for a string used as a label for a variable.
  - `my_key_warning` for a warning message.
- Append a number to the key name to indicate a size limit:
  - `my_key_title_40`
- Separate your messages into categories.
- Add comments for the context. A comment must begin with the # character.

Substitutions in the message text

Substitutions are made into message text as text strings. As many as seven substitutions can be made into each message. A substitution is represented in the .smd file using the %s formatting specifier.

```
DI_ATTR_WARN=The attribute "%s" is already registered for the table %s.
```

Figure 2. Example of message with substitutions

In the SAS code, the substitutions are passed to the SASMSG function as character strings.

During translation of a .smd file, the order of a substitution may change in the message text depending on the order of words for the language. SAS programmers or translators may add an argument number specification (%#1s %#2s ...) to indicate the order in which substitutions are to be made from the SASMSG argument list.

STEP 2 – CREATE A SAS DATA SET FROM THE .SMD FILE

The next step in the string externalization process is to create a SAS message data set from the .smd files. This step is done once by an administrator to gather the localizations into a single SAS data set. The SAS data set is the only runtime component of the application.
Some restrictions apply for the data set:

- The data set must contain the four variables described in Table 10.
- The SAS message data set must be a 7-bit ASCII data set with an encoding of US-ASCII or ASCIIANY.
- The data set must be sorted on the following variables: *locale*, *key*, and *lineno,* with the latter in descending order.
- A composite index on the variables locale and key must be defined.

#	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	char	1200	1,200 text of the message

Table 10. Description of a SAS message data set

Such data sets can be created with the %SMD2DS() macro function. The macro %SMD2DS is available in the SAS AUTOCALL library:

```
%SMD2DS(DIR=,BASENAME=,LOCALE=,LIB=)
```

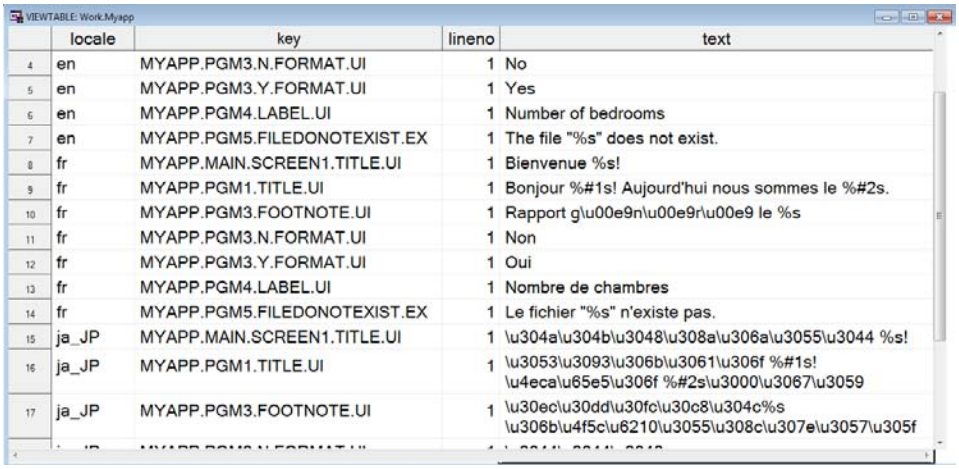
where

`DIR=` is the directory where the .smd files are located.
`BASENAME=` is the basename of the file to process.
`LOCALE=` <optional> is the list of locales to include separated by a blank. The default file to process is *basename.smd* if the parameter is not specified.
`LIB=` <optional> is the library where you create the data set. The default library is WORK if the parameter is not specified.

```
%SMD2DS(DIR=c:\myplaypen\myproduct\smd, BASENAME=myapp, LOCALE=fr  ja_JP) ;
```

In the example here, the macro %SMD2DS looks for a file called myapp.smd in the directory specified within the DIR parameter. If the file does not exist, the macro looks for the file myapp_en.smd. If that file does not exist either, the macro stops. In addition to the default files, the macro looks for files that match the LOCALE= parameter. Here, the files are myapp_fr.smd and myapp_de_DE.smd. The data set is created in the WORK library.

The following example shows what a sasmsg data set looks like. In this data set, the messages have been translated into French and Japanese and converted to 7-bit ASCII characters.



VIEWTABLE: Work.Myapp

	locale	key	lineno	text
4	en	MYAPP.PGM3.N.FORMAT.UI	1	No
5	en	MYAPP.PGM3.Y.FORMAT.UI	1	Yes
6	en	MYAPP.PGM4.LABEL.UI	1	Number of bedrooms
7	en	MYAPP.PGM5.FILEDONOTEXIST.EX	1	The file "%s" does not exist.
8	fr	MYAPP.MAIN.SCREEN1.TITLE.UI	1	Bienvenue %s!
9	fr	MYAPP.PGM1.TITLE.UI	1	Bonjour %#1s! Aujourd'hui nous sommes le %#2s.
10	fr	MYAPP.PGM3.FOOTNOTE.UI	1	Rapport g\u00e9n\u00e9r\u00e9 le %s
11	fr	MYAPP.PGM3.N.FORMAT.UI	1	Non
12	fr	MYAPP.PGM3.Y.FORMAT.UI	1	Oui
13	fr	MYAPP.PGM4.LABEL.UI	1	Nombre de chambres
14	fr	MYAPP.PGM5.FILEDONOTEXIST.EX	1	Le fichier "%s" n'existe pas.
15	ja_JP	MYAPP.MAIN.SCREEN1.TITLE.UI	1	\u304a\u304b\u3048\u308a\u306a\u3055\u3044 %s!
16	ja_JP	MYAPP.PGM1.TITLE.UI	1	\u3053\u3093\u306b\u3061\u306f %#1s! \u4eca\u65e5\u306f %#2s\u3000\u3067\u3059
17	ja_JP	MYAPP.PGM3.FOOTNOTE.UI	1	\u30ec\u30dd\u30fc\u30c8\u304c%s \u306b\u4f5c\u6210\u3055\u308c\u307e\u3057\u305f

Figure 3 - Example of a SAS message data set

STEP 3 – RETRIEVING MESSAGES WITHIN SAS CODE

Use the function SASMSG to access and retrieve the message for display within your application. The SASMSG function retrieves a message from the data set created in step 2 based on the current locale and a specified key.

Here is the syntax to retrieve messages within the macro language or within a DATA step or SCL code:

```
%SYSFUNC(SASMSG(BASENAME, KEY, <<QUOTE|Q|DQUOTE|D|NOQUOTE|N> <, substitute1, substitute2, ...>>)

SASMSG("BASENAME", "KEY", <<"QUOTE"|"Q"|"DQUOTE"|"D"|"NOQUOTE"|"N"> <, "subs1", "subs2", ...>>)
```

BASENAME is the name of the data set created from the %SMD2DS macro; it contains the messages.
KEY is the message key. If a key name is specified for a key that does not exist, the key name is returned.

The third parameter of the SASMSG() function allows you to control which type of quotes are added to the message or whether quotes are added by SASMSG. The default setting is “DQUOTE” if this parameter is omitted. The values for the quoting option are as follows:

N | NOQUOTE : No quotes added.
Q | QUOTE : Add single quotes around the string.
D | DQUOTE : Add double quotes around the string (the default).

The fourth and subsequent arguments to SASMSG are for substitutions into the message text. All substitution parameters specified in SASMSG must be character strings. As many as seven substitutions can be made for one message.

Language of the messages and locale fallback mechanism

The setting of the LOCALE system option determines the language, and possibly the regional translation, to use for the messages.
At run time, the SASMSG function gets the LOCALE value from the LOCALE= system option. A POSIX name in the form ll_RR is returned. The SASMSG function tries to load the key that matches that locale (ll_RR). If no key is found for that locale, the execution falls back to language only (ll). If the code can still not find a key, SASMSG falls back to the default language.
If the message has not been translated for the LOCALE, the default language message text is used.
If the key name specified does not exist, the key name is returned.

Example

To demonstrate the SASMSG function, let’s run the following code three times with a different setting for the LOCALE option for each execution:

```
%LET MYDS=MYAPP ;
%LET ADATE= %SYSFUNC(today(),nldate.) ;
%LET USER=Steve ;

%PUT %SYSFUNC(SASMSG(&MYDS, myapp.pgm1.title.ui, D, &user, &adate)) ;
```

Code box 7. SASMSG function usage examples

When the LOCALE option is set to en_US, this program returns the following English message:

```
Hello Steve! Today's date is February 17, 2012.
```

The same code returns the French translation when the LOCALE option is set to fr_FR:

```
Bonjour Steve! Aujourd'hui nous sommes le 17 février 2012.
```

And it returns a Japanese message when the option is set to ja_JP:

こんにちは Steve! 今日は 2012年02月17日 です

Macro language statements with special characters in a message

The SASMSG function can be used in macro language with the %SYSFUNC macro function. Please note that arguments passed to a function called by the %SYSFUNC macro must not be in quotes while arguments passed to the SASMSG function outside of %SYSFUNC must be quoted.

When the SASMSG function is used with the %SYSFUNC macro function, the macro function %NRBQUOTE() is added internally to the returned message. This macro function masks the quote character (‘) and the double quote character (“) when they are not matched:

Ampersand character: &

Messages can contain the ampersand character (&) followed by a valid character other than a blank to reference a macro variable name. The macro variable is resolved with its value, for example:

```
pgm3_osname_ui = Operating System: &osname.
```

At execution time, there are no warnings when the following code is executed:

```
%let osname=%put %sysfunc(sysget(OS) ) ;
%put %sysfunc(sasmsg(&dataset, pgm3_osname_ui));
```

On a Windows NT machine, the generated message is: “Operating System: Windows_NT.”

Percent sign character: %

The percent sign character (%) is reserved for format modifiers and could cause unpredictable results if not followed by an appropriate character. The following code is interpreted as format modifiers: %s or %# %1 %2 %3.

To display correctly this character in a message, it must be followed with another % character, for example:

```
msg_note = Increase: %s%%
```

The execution of the following code produces the following message: “Increase: 97%”

```
%put %sysfunc(sasmsg(&dataset,msg_note,d,97)) ;
```

Left and right parenthesis: ()

Because the %NRBQUOTE function is added to the returned message, the parentheses must match. For an open parenthesis, the message must have a close parenthesis.

The SASMSGL function

Like the SASMSG function, the SASMSGL function retrieves a message from a data set based on a locale value and a key. However, the SASMSL function does not rely on the current session locale. The locale value must be specified as a parameter.

The syntax to retrieve a message with this function is as follow:

```
%SYSFUNC(SASMSGL(BASENAME, KEY, LOCALE, < <Q|D|N> <, substitute1, substitute2, ...>>)

SASMSGL("BASENAME", "KEY", LOCALE, <<"Q"|"D"|"N"> <, "subs1", "subs2", ...>>)
```

Example:

```
%LET MYDS=MYAPP ;
%LET ADATE= %SYSFUNC(today(),nldate.) ;
%LET USER=STEVE ;

data _null_ ;
   my_msg = SASMSGL("&MYDS", "myapp.pgm1.title.ui", "fr_FR", 'N', "&USER", "&ADATE") ;
   put my_msg= ;
run ;
```

Code box 8. SASMSGL function usage example

CONCLUSION

This paper is a brief introduction to the vast and complex field of internationalization. When developing applications for a global market, internationalization must be part of your initial requirements. SAS provides support to develop and execute such applications.

By applying the techniques described in this paper--use of a locale, appropriate string manipulation functions, and string externalization, and with the addition of localized data--the same source code of your SAS program can target a global audience.

REFERENCES

Creating Order out of Character Chaos: Collation Capabilities of the SAS System. SGF paper 297-2007

SAS Technical paper: Linguistic Collation: Everyone Can Get What They Expect

SAS 9.3 National Language Support (NLS): Reference Guide.

Microsoft Globalization Step-by-Step: Understanding Internationalization.

SAS Technical paper: Processing Multilingual Data with the SAS® 9.2 Unicode Server

ACKNOWLEDGMENTS

I would like to express my gratitude to the people who helped me to create and review this document, especially Elizabeth Bales, Steve Beatrous, Manfred Kiefer, Shin Kayano, and Junichi Arai.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Mickaël Bouedo
SAS Campus Drive
SAS Institute
E-mail: Mickael.Bouedo@sas.com

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.