String functions

Contents Functions References Also see

Contents

abbrev(s,n)	name s, abbreviated to a length of n
char(n)	the character corresponding to ASCII or extended ASCII code n; "" if n is not in the domain
collatorlocale(*loc*,*type*)	the most closely related locale supported by ICU from *loc* if *type* is 1; the actual locale where the collation data comes from if *type* is 2
collatorversion(*loc*)	the version string of a collator based on locale *loc*
indexnot(s_1,s_2)	the position in ASCII string s_1 of the first character of s_1 not found in ASCII string s_2, or 0 if all characters of s_1 are found in s_2
plural(n,s)	the plural of s if $n \neq \pm 1$
plural(n,s_1,s_2)	the plural of s_1, as modified by or replaced with s_2, if $n \neq \pm 1$
real(s)	s converted to numeric or *missing*
regexcapture(n)	subexpression n from a previous regexm() or regexmatch() match
regexcapturenamed(*grp*)	subexpression corresponding to matching group named *grp* in regular expression from a previous regexm() or regexmatch() match
regexm(s,*re*)	a match of a regular expression, which evaluates to 1 if regular expression *re* is satisfied by the ASCII string s; otherwise, 0
regexmatch(s,*re*[,*noc*[,*std*[,*nlalt*]]])	a match of a regular expression, which evaluates to 1 if regular expression *re* is satisfied by the ASCII string s; otherwise, 0
regexr(s_1,*re*,s_2)	replaces the first substring within ASCII string s_1 that matches *re* with ASCII string s_2 and returns the resulting string
regexreplace(s_1,*re*,s_2[,*noc*[,*fmt*[,*std*[,*nlalt*]]]])	replaces the first substring within ASCII string s_1 that matches *re* with ASCII string s_2 and returns the resulting string
regexreplaceall(s_1,*re*,s_2[,*noc*[,*fmt*[,*std*[,*nlalt*]]]])	replaces all substrings within ASCII string s_1 that match *re* with ASCII string s_2 and returns the resulting string
regexs(n)	subexpression n from a previous regexm() or regexmatch() match, where $0 \leq n < 10$
soundex(s)	the soundex code for a string, s
soundex_nara(s)	the US Census soundex code for a string, s
strcat(s_1,s_2)	there is no strcat() function; instead the addition operator is used to concatenate strings
strdup(s_1,n)	there is no strdup() function; instead the multiplication operator is used to create multiple copies of strings
string(n)	a synonym for strofreal(n)
string(n,s)	a synonym for strofreal(n,s)
stritrim(s)	s with multiple, consecutive internal blanks (ASCII space character char(32)) collapsed to one blank

strlen(*s*)	the number of characters in ASCII *s* or length in bytes
strlower(*s*)	lowercase ASCII characters in string *s*
strltrim(*s*)	*s* without leading blanks (ASCII space character char(32))
strmatch(s_1,s_2)	1 if s_1 matches the pattern s_2; otherwise, 0
strofreal(*n*)	*n* converted to a string
strofreal(*n*,*s*)	*n* converted to a string using the specified display format
strpos(s_1,s_2)	the position in s_1 at which s_2 is first found, 0 if s_2 does not occur, and 1 if s_2 is empty
strproper(*s*)	a string with the first ASCII letter and any other letters immediately following characters that are not letters capitalized; all other ASCII letters converted to lowercase
strreverse(*s*)	the reverse of ASCII string *s*
strrpos(s_1,s_2)	the position in s_1 at which s_2 is last found, 0 if s_2 does not occur, and 1 if s_2 is empty
strrtrim(*s*)	*s* without trailing blanks (ASCII space character char(32))
strtoname(*s*[,*p*])	*s* translated into a Stata 13 compatible name
strtrim(*s*)	*s* without leading and trailing blanks (ASCII space character char(32)); equivalent to strltrim(strrtrim(*s*))
strupper(*s*)	uppercase ASCII characters in string *s*
subinstr(s_1,s_2,s_3,*n*)	s_1, where the first *n* occurrences in s_1 of s_2 have been replaced with s_3
subinword(s_1,s_2,s_3,*n*)	s_1, where the first *n* occurrences in s_1 of s_2 as a word have been replaced with s_3
substr(*s*,n_1,n_2)	the substring of *s*, starting at n_1, for a length of n_2
tobytes(*s*[,*n*])	escaped decimal or hex digit strings of up to 200 bytes of *s*
uchar(*n*)	the Unicode character corresponding to Unicode code point *n* or an empty string if *n* is beyond the Unicode code-point range
udstrlen(*s*)	the number of display columns needed to display the Unicode string *s* in the Stata Results window
udsubstr(*s*,n_1,n_2)	the Unicode substring of *s*, starting at character n_1, for n_2 display columns
uisdigit(*s*)	1 if the first Unicode character in *s* is a Unicode decimal digit; otherwise, 0
uisletter(*s*)	1 if the first Unicode character in *s* is a Unicode letter; otherwise, 0
ustrcompare(s_1,s_2[,*loc*])	compares two Unicode strings
ustrcompareex(s_1,s_2,*loc*,*st*,*case*,*cslv*,*norm*,*num*,*alt*,*fr*)	compares two Unicode strings
ustrfix(*s*[,*rep*])	replaces each invalid UTF-8 sequence with a Unicode character
ustrfrom(*s*,*enc*,*mode*)	converts the string *s* in encoding *enc* to a UTF-8 encoded Unicode string
ustrinvalidcnt(*s*)	the number of invalid UTF-8 sequences in *s*
ustrleft(*s*,*n*)	the first *n* Unicode characters of the Unicode string *s*
ustrlen(*s*)	the number of characters in the Unicode string *s*
ustrlower(*s*[,*loc*])	lowercase all characters of Unicode string *s* under the given locale *loc*

`ustrltrim(`s`)`	removes the leading Unicode whitespace characters and blanks from the Unicode string s
`ustrnormalize(`s`,`$norm$`)`	normalizes Unicode string s to one of the five normalization forms specified by $norm$
`ustrpos(`s_1`,`s_2[`,`n]`)`	the position in s_1 at which s_2 is first found; otherwise, `0`
`ustrregexm(`s`,`re[`,`noc]`)`	performs a match of a regular expression and evaluates to `1` if regular expression re is satisfied by the Unicode string s; otherwise, `0`
`ustrregexra(`s_1`,`re`,`s_2[`,`noc]`)`	replaces all substrings within the Unicode string s_1 that match re with s_2 and returns the resulting string
`ustrregexrf(`s_1`,`re`,`s_2[`,`noc]`)`	replaces the first substring within the Unicode string s_1 that matches re with s_2 and returns the resulting string
`ustrregexs(`n`)`	subexpression n from a previous `ustrregexm()` match
`ustrreverse(`s`)`	the reverse of Unicode string s
`ustrright(`s`,`n`)`	the last n Unicode characters of the Unicode string s
`ustrrpos(`s_1`,`s_2[`,`n]`)`	the position in s_1 at which s_2 is last found; otherwise, `0`
`ustrrtrim(`s`)`	remove trailing Unicode whitespace characters and blanks from the Unicode string s
`ustrsortkey(`s[`,`loc]`)`	generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`
`ustrsortkeyex(`s`,`loc`,`st`,`$case$`,`$cslv$`,`$norm$`,`num`,`alt`,`fr`)`	generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`
`ustrtitle(`s[`,`loc]`)`	a string with the first characters of Unicode words titlecased and other characters lowercased
`ustrto(`s`,`enc`,`$mode$`)`	converts the Unicode string s in UTF-8 encoding to a string in encoding enc
`ustrtohex(`s[`,`n]`)`	escaped hex digit string of s up to 200 Unicode characters
`ustrtoname(`s[`,`p]`)`	string s translated into a Stata name
`ustrtrim(`s`)`	removes leading and trailing Unicode whitespace characters and blanks from the Unicode string s
`ustrunescape(`s`)`	the Unicode string corresponding to the escaped sequences of s
`ustrupper(`s[`,`loc]`)`	uppercase all characters in string s under the given locale loc
`ustrword(`s`,`n[`,`loc]`)`	the nth Unicode word in the Unicode string s
`ustrwordcount(`s[`,`loc]`)`	the number of nonempty Unicode words in the Unicode string s
`usubinstr(`s_1`,`s_2`,`s_3`,`n`)`	replaces the first n occurrences of the Unicode string s_2 with the Unicode string s_3 in s_1
`usubstr(`s`,`n_1`,`n_2`)`	the Unicode substring of s, starting at n_1, for a length of n_2
`word(`s`,`n`)`	the nth word in s; *missing* (`""`) if n is missing

`wordbreaklocale(`*loc*`,`*type*`)`	the most closely related locale supported by ICU from *loc* if *type* is 1, the actual locale where the word-boundary analysis data come from if *type* is 2; or an empty string is returned for any other *type*
`wordcount(`*s*`)`	the number of words in *s*

Functions

In the display below, *s* indicates a string subexpression (a string literal, a string variable, or another string expression) and *n* indicates a numeric subexpression (a number, a numeric variable, or another numeric expression).

If your strings contain Unicode characters or you are writing programs that will be used by others who might use Unicode strings, read **[U] 12.4.2 Handling Unicode strings**.

`abbrev(`*s*`,`*n*`)`

Description: name *s*, abbreviated to a length of *n*

Length is measured in the number of display columns, not in the number of characters. For most users, the number of display columns equals the number of characters. For a detailed discussion of display columns, see **[U] 12.4.2.2 Displaying Unicode characters**.

If any of the characters of *s* are a period, "`.`", and $n < 8$, then the value of *n* defaults to a value of 8. Otherwise, if $n < 5$, then *n* defaults to a value of 5. If *n* is *missing*, `abbrev()` will return the entire string *s*. `abbrev()` is typically used with variable names and variable names with factor-variable or time-series operators (the period case).

`abbrev("displacement",8)` is `displa~t`.

Domain *s*: strings
Domain *n*: integers 5 to 32
Range: strings

`char(`*n*`)`

Description: the character corresponding to ASCII or extended ASCII code *n*; `""` if *n* is not in the domain

Note: ASCII codes are from 0 to 127; extended ASCII codes are from 128 to 255. Prior to Stata 14, the display of extended ASCII characters was encoding dependent. For example, `char(128)` on Microsoft Windows using Windows-1252 encoding displayed the Euro symbol, but on Linux using ISO-Latin-1 encoding, `char(128)` displayed an invalid character symbol. Beginning with Stata 14, Stata's display encoding is UTF-8 on all platforms. The `char(128)` function is an invalid UTF-8 sequence and thus will display a question mark. There are two Unicode functions corresponding to `char()`: `uchar()` and `ustrunescape()`. You can use `uchar(8364)` or `ustrunescape("\u20AC")` to display a Euro sign on all platforms.

Domain *n*: integers 0 to 255
Range: ASCII characters

`uchar(`n`)`

Description:	the Unicode character corresponding to Unicode code point n or an empty string if n is beyond the Unicode code-point range Note that `uchar()` takes the decimal value of the Unicode code point. `ustrunescape()` takes an escaped hex digit string of the Unicode code point. For example, both `uchar(8364)` and `ustrunescape("\u20ac")` produce the Euro sign.
Domain n:	integers ≥ 0
Range:	Unicode characters

`collatorlocale(`*loc*`,`*type*`)`

Description:	the most closely related locale supported by ICU from *loc* if *type* is `1`; the actual locale where the collation data comes from if *type* is `2`
	For any other *type*, *loc* is returned in a canonicalized form.
	`collatorlocale("en_us_texas", 0)` = `en_US_TEXAS`
	`collatorlocale("en_us_texas", 1)` = `en_US`
	`collatorlocale("en_us_texas", 2)` = `root`

Domain *loc*:	strings of locale name
Domain *type*:	integers
Range:	strings

`collatorversion(`*loc*`)`

Description:	the version string of a collator based on locale *loc* The Unicode standard is constantly adding more characters and the sort key format may change as well. This can cause `ustrsortkey()` and `ustrsortkeyex()` to produce incompatible sort keys between different versions of International Components for Unicode. The version string can be used for versioning the sort keys to indicate when saved sort keys must be regenerated.
Range:	strings

`indexnot(`s_1`,`s_2`)`

Description:	the position in ASCII string s_1 of the first character of s_1 not found in ASCII string s_2, or `0` if all characters of s_1 are found in s_2 `indexnot()` is intended for use only with plain ASCII strings. For Unicode characters beyond the plain ASCII range, the position and character are given in bytes, not characters.
Domain s_1:	ASCII strings (to be searched)
Domain s_2:	ASCII strings (to search for)
Range:	integers ≥ 0

`plural(`n`,`s`)`

Description: the plural of s if $n \neq \pm 1$

The plural is formed by adding “s” to s.

`plural(1, "horse")` = `"horse"`
`plural(2, "horse")` = `"horses"`

Domain n: real numbers
Domain s: strings
Range: strings

`plural(`n`,`s_1`,`s_2`)`

Description: the plural of s_1, as modified by or replaced with s_2, if $n \neq \pm 1$

If s_2 begins with the character “+”, the plural is formed by adding the remainder of s_2 to s_1. If s_2 begins with the character “-”, the plural is formed by subtracting the remainder of s_2 from s_1. If s_2 begins with neither “+” nor “-”, then the plural is formed by returning s_2.

`plural(2, "glass", "+es")` = `"glasses"`
`plural(1, "mouse", "mice")` = `"mouse"`
`plural(2, "mouse", "mice")` = `"mice"`
`plural(2, "abcdefg", "-efg")` = `"abcd"`

Domain n: real numbers
Domain s_1: strings
Domain s_2: strings
Range: strings

`real(`s`)`

Description: s converted to numeric or *missing*

Also see `strofreal()`.

`real("5.2")+1` = `6.2`
`real("hello")` = `.`

Domain s: strings
Range: −8e+307 to 8e+307 or *missing*

`regexcapture(`n`)`

Description: subexpression n from a previous `regexm()` or `regexmatch()` match

`regexcapture(0)` returns the entire string that satisfied the regular expression.

Domain n: integers
Range: ASCII strings or *missing*

`regexcapturenamed(`grp`)`

Description: subexpression corresponding to matching group named grp in regular expression from a previous `regexm()` or `regexmatch()` match

Domain grp: ASCII strings
Range: ASCII strings or *missing*

`regexm(`*s*`,`*re*`)`

Description:	a match of a regular expression, which evaluates to `1` if regular expression *re* is satisfied by the ASCII string *s*; otherwise, `0` Regular expression syntax is based on Henry Spencer's NFA algorithm, and this is nearly identical to the POSIX.2 standard. *s* and *re* may not contain binary 0 (`\0`). `regexm()` is intended for use only with plain ASCII characters. For Unicode characters beyond the plain ASCII range, the match is based on bytes. For a character-based match, see `ustrregexm()`. For more advanced regular expression matching, see `regexmatch()`.
Domain *s*:	ASCII strings
Domain *re*:	regular expressions
Range:	0, 1, or *missing*

`regexmatch(`*s*`,`*re*[`,`*noc*[`,`*std*[`,`*nlalt*]]]`)`

Description:	a match of a regular expression, which evaluates to `1` if regular expression *re* is satisfied by the ASCII string *s*; otherwise, `0` `regexmatch()` is intended for use only with plain ASCII characters. For Unicode characters beyond the plain ASCII range, the match is based on bytes. For a character-based match, see `ustrregexm()`. If *noc* is specified and is not 0, a case-insensitive match is performed; otherwise, a case-sensitive match is performed. *std* specifies the regular expression standard: 1 for POSIX Extended Regular, 2 for POSIX Basic Regular, 3 for Emacs, 4 for AWK, 5 for grep, 6 for egrep, or any other number for Perl, the default. If *nlalt* is specified and is 0, the newline character, `char(10)`, is not treated like alternation operator `|`; otherwise, newline has the same effect as `|`. *s* and *re* may not contain binary 0 (`\0`).
Domain *s*:	ASCII strings
Domain *re*:	regular expression
Domain *noc*:	integers
Domain *std*:	integers
Domain *nlalt*:	integers
Range:	0, 1, or *missing*

`regexr(`s_1`,`re`,`s_2`)`

Description:	replaces the first substring within ASCII string s_1 that matches re with ASCII string s_2 and returns the resulting string If s_1 contains no substring that matches re, the unaltered s_1 is returned. s_1 and the result of `regexr()` may be at most 1,100,000 characters long. s_1, re, and s_2 may not contain binary 0 (`\0`). `regexr()` is intended for use only with plain ASCII characters. For Unicode characters beyond the plain ASCII range, the match is based on bytes, and the result is restricted to 1,100,000 bytes. For a character-based match, see `ustrregexrf()` or `ustrregexra()`. For more advanced regular expression replacement, see `regexreplace()` and `regexreplaceall()`.
Domain s_1:	ASCII strings
Domain re:	regular expressions
Domain s_2:	ASCII strings
Range:	ASCII strings

`regexreplace(`s_1`,`re`,`s_2[`,`noc[`,`fmt[`,`std[`,`$nlalt$]]]]`)`

Description:	replaces the first substring within ASCII string s_1 that matches re with ASCII string s_2 and returns the resulting string If noc is specified and is not 0, a case-insensitive match is performed; otherwise, a case-sensitive match is performed. fmt specifies the format string syntax supported in s_2: 1 for literal, where s_2 is treated as a string literal (no special character substitution), 2 for sed, or any other number for Perl, the default. std specifies the regular expression standard: 1 for POSIX Extended Regular, 2 for POSIX Basic Regular, 3 for Emacs, 4 for AWK, 5 for grep, 6 for egrep, or any other number for Perl, the default. If $nlalt$ is specified and is 0, the newline character, `char(10)`, is not treated like alternation operator `|`; otherwise, newline has the same effect as `|`. If s_1 contains no substring that matches re, the unaltered s_1 is returned. s_1, s_2, and re may not contain binary 0 (`\0`).
Domain s_1:	ASCII strings
Domain re:	regular expression
Domain s_2:	ASCII strings
Domain noc:	integers
Domain fmt:	integers
Domain std:	integers
Domain $nlalt$:	integers
Range:	ASCII strings

`regexreplaceall(`s_1`,`re`,`s_2[`,`noc[`,`fmt[`,`std[`,`$nlalt$]]]]`)`

Description: replaces all substrings within ASCII string s_1 that match re with ASCII string s_2 and returns the resulting string

If noc is specified and is not 0, a case-insensitive match is performed; otherwise, a case-sensitive match is performed.

fmt specifies the format string syntax supported in s_2: 1 for literal, where s_2 is treated as a string literal (no special character substitution), 2 for sed, or any other number for Perl, the default.

std specifies the regular expression standard: 1 for POSIX Extended Regular, 2 for POSIX Basic Regular, 3 for Emacs, 4 for AWK, 5 for grep, 6 for egrep, or any other number for Perl, the default.

If $nlalt$ is specified and is 0, the newline character, `char(10)`, is not treated like alternation operator `|`; otherwise, newline has the same effect as `|`.

If s_1 contains no substring that matches re, the unaltered s_1 is returned. s_1, s_2, and re may not contain binary 0 (`\0`).

Domain s_1:	ASCII strings
Domain re:	regular expression
Domain s_2:	ASCII strings
Domain noc:	integers
Domain fmt:	integers
Domain std:	integers
Domain $nlalt$:	integers
Range:	ASCII strings

`regexs(`n`)`

Description: subexpression n from a previous `regexm()` or `regexmatch()` match, where $0 \leq n < 10$

Subexpression 0 is reserved for the entire string that satisfied the regular expression. The returned subexpression may be at most 1,100,000 characters (bytes) long.

For more options to return matching substrings, see `regexcapture()` and `regexcapturen`

Domain n:	0 to 9
Range:	ASCII strings

`ustrregexm(`s`,`re[`,`noc]`)`

Description:	performs a match of a regular expression and evaluates to `1` if regular expression re is satisfied by the Unicode string s; otherwise, `0`
	
	If noc is specified and not 0, a case-insensitive match is performed. The function may return a negative integer if an error occurs.
	
	`ustrregexm("12345", "([0-9]){5}")` = `1`
	`ustrregexm("de TRÈS près", "rès")` = `1`
	`ustrregexm("de TRÈS près", "Rès")` = `0`
	`ustrregexm("de TRÈS près", "Rès", 1)` = `1`

Domain s:	Unicode strings
Domain re:	Unicode regular expressions
Domain noc:	integers
Range:	integers

`ustrregexrf(`s_1`,`re`,`s_2[`, `noc]`)`

Description:	replaces the first substring within the Unicode string s_1 that matches re with s_2 and returns the resulting string
	
	If noc is specified and not 0, a case-insensitive match is performed. The function may return an empty string if an error occurs.
	
	`ustrregexrf("très près", "rès", "X")` = `"tX près"`
	`ustrregexrf("TRÈS près", "Rès", "X")` = `"TRÈS près"`
	`ustrregexrf("TRÈS près", "Rès", "X", 1)` = `"TX près"`

Domain s_1:	Unicode strings
Domain re:	Unicode regular expressions
Domain s_2:	Unicode strings
Domain noc:	integers
Range:	Unicode strings

`ustrregexra(`s_1`,`re`,`s_2[`, `noc]`)`

Description:	replaces all substrings within the Unicode string s_1 that match re with s_2 and returns the resulting string
	
	If noc is specified and not 0, a case-insensitive match is performed. The function may return an empty string if an error occurs.
	
	`ustrregexra("très près", "rès", "X")` = `"tX pX"`
	`ustrregexra("TRÈS près", "Rès", "X")` = `"TRÈS près"`
	`ustrregexra("TRÈS près", "Rès", "X", 1)` = `"TX pX"`

Domain s_1:	Unicode strings
Domain re:	Unicode regular expressions
Domain s_2:	Unicode strings
Domain noc:	integers
Range:	Unicode strings

`ustrregexs(`n`)`

Description:	subexpression n from a previous `ustrregexm()` match Subexpression 0 is reserved for the entire string that satisfied the regular expression. The function may return an empty string if n is larger than the maximum count of subexpressions from the previous match or if an error occurs.
Domain n:	integers $\geq$ 0
Range:	strings

`soundex(`s`)`

Description:	the soundex code for a string, s
	
	The soundex code consists of a letter followed by three numbers: the letter is the first ASCII letter of the name and the numbers encode the remaining consonants. Similar sounding consonants are encoded by the same number. Unicode characters beyond the plain ASCII range are ignored.
	
	`soundex("Ashcraft")` = `"A226"`
	`soundex("Robert")` = `"R163"`
	`soundex("Rupert")` = `"R163"`

Domain s:	strings
Range:	strings

`soundex_nara(`s`)`

Description:	the US Census soundex code for a string, s The soundex code consists of a letter followed by three numbers: the letter is the first ASCII letter of the name and the numbers encode the remaining consonants. Similar sounding consonants are encoded by the same number. Unicode characters beyond the plain ASCII range are ignored. `soundex_nara("Ashcraft")` = `"A261"`
Domain s:	strings
Range:	strings

`strcat(`s_1`,`s_2`)`

Description:	there is no `strcat()` function; instead the addition operator is used to concatenate strings
	
	`"hello " + "world" = "hello world"`
	`"a" + "b" = "ab"`
	`"Café " + "de Flore" = "Café de Flore"`

Domain s_1:	strings
Domain s_2:	strings
Range:	strings

`strdup(`s_1`,`n`)`

Description: there is no `strdup()` function; instead the multiplication operator is used to create multiple copies of strings

```
"hello" * 3 = "hellohellohello"
3 * "hello" = "hellohellohello"
0 * "hello" = ""
"hello" * 1 = "hello"
"Здравствуйте " * 2 = "Здравствуйте Здравствуйте "
```

Domain s_1: strings
Domain n: nonnegative integers 0, 1, 2, ...
Range: strings

`string(`n`)`

Description: a synonym for `strofreal(`n`)`

`string(`n`,`s`)`

Description: a synonym for `strofreal(`n`,`s`)`

`stritrim(`s`)`

Description: s with multiple, consecutive internal blanks (ASCII space character `char(32)`) collapsed to one blank

`stritrim("hello     there")` $=$ `"hello there"`

Domain s: strings
Range: strings with no multiple, consecutive internal blanks

`strlen(`s`)`

Description: the number of characters in ASCII s or length in bytes

`strlen()` is intended for use only with plain ASCII characters and for use by programmers who want to obtain the byte-length of a string. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, é takes 2 bytes.

For the number of characters in a Unicode string, see `ustrlen()`.

`strlen("ab")` $=$ 2
`strlen("é")` $=$ 2

Domain s: strings
Range: integers ≥ 0

`ustrlen(`*s*`)`

Description: the number of characters in the Unicode string *s*

An invalid UTF-8 sequence is counted as one Unicode character. An invalid UTF-8 sequence may contain one byte or multiple bytes. Note that any Unicode character beyond the plain ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, é takes 2 bytes.

`ustrlen("médiane")` = 7
`strlen("médiane")` = 8

Domain *s*: Unicode strings
Range: integers ≥ 0

`udstrlen(`*s*`)`

Description: the number of display columns needed to display the Unicode string *s* in the Stata Results window

A Unicode character in the CJK (Chinese, Japanese, and Korean) encoding usually requires two display columns; a Latin character usually requires one column. Any invalid UTF-8 sequence requires one column.

`udstrlen("中值") = 4`
`ustrlen("中值") = 2`
`strlen("中值") = 6`

Domain *s*: Unicode strings
Range: integers ≥ 0

`strlower(`*s*`)`

Description: lowercase ASCII characters in string *s*

Unicode characters beyond the plain ASCII range are ignored.

`strlower("THIS")` = `"this"`
`strlower("CAFÉ")` = `"cafÉ"`

Domain *s*: strings
Range: strings with lowercased characters

`ustrlower(`s[`,`*loc*]`)`

Description: lowercase all characters of Unicode string s under the given locale *loc*

If *loc* is not specified, the default locale is used. The same s but different *loc* may produce different results; for example, the lowercase letter of “I” is “i” in English but a dotless “i” in Turkish. The same Unicode character can be mapped to different Unicode characters based on its surrounding characters; for example, Greek capital letter sigma Σ has two lowercases: ς, if it is the final character of a word, or σ. The result can be longer or shorter than the input Unicode string in bytes.

`ustrlower("MÉDIANE","fr")` = `"médiane"`
`ustrlower("ISTANBUL","tr")` = `"ıstanbul"`
`ustrlower("ΌΔΥΣΣΕΎΣ")` = `"όδυσσεύς"`

Domain s: Unicode strings
Domain *loc*: locale name
Range: Unicode strings

`strltrim(`s`)`

Description: s without leading blanks (ASCII space character `char(32)`)

`strltrim(" this")` = `"this"`

Domain s: strings
Range: strings without leading blanks

`ustrltrim(`x`)`

Description: removes the leading Unicode whitespace characters and blanks from the Unicode string s

Note that, in addition to `char(32)`, ASCII characters `char(9)`, `char(10)`, `char(11)`, `char(12)`, and `char(13)` are whitespace characters in Unicode standard.

`ustrltrim(" this")` = `"this"`
`ustrltrim(char(9)+"this")` = `"this"`
`ustrltrim(ustrunescape("\u1680")+" this")` = `"this"`

Domain s: Unicode strings
Range: Unicode strings

`strmatch(`s_1`,`s_2`)`

Description: 1 if s_1 matches the pattern s_2; otherwise, 0

`strmatch("17.4","1??4")` returns 1. In s_2, `"?"` means that one character goes here, and `"*"` means that zero or more bytes go here. Note that a Unicode character may contain multiple bytes; thus, using `"*"` with Unicode characters can infrequently result in matches that do not occur at a character boundary.

Also see `regexm()`, `regexr()`, and `regexs()`.

`strmatch("café", "caf?")` = 1

Domain s_1:	strings
Domain s_2:	strings
Range:	integers 0 or 1

`strofreal(`n`)`

Description: n converted to a string

Also see `real()`.

```
strofreal(4)+"F" = "4F"
strofreal(1234567) = "1234567"
strofreal(12345678) = "1.23e+07"
strofreal(.) = "."
```

Domain n:	−8e+307 to 8e+307 or *missing*
Range:	strings

`strofreal(`n`,`s`)`

Description: n converted to a string using the specified display format

Also see `real()`.

```
strofreal(4,"%9.2f") = "4.00"
strofreal(123456789,"%11.0g") = "123456789"
strofreal(123456789,"%13.0gc") = "123,456,789"
strofreal(0,"%td") = "01jan1960"
strofreal(225,"%tq") = "2016q2"
strofreal(225,"not a format") = ""
```

Domain n:	−8e+307 to 8e+307 or *missing*
Domain s:	strings containing %*fmt* numeric display format
Range:	strings

`strpos(`s_1`,`s_2`)`

Description:	the position in s_1 at which s_2 is first found, `0` if s_2 does not occur, and `1` if s_2 is empty
	
	`strpos()` is intended for use only with plain ASCII characters and for use by programmers who want to obtain the byte-position of s_2. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, é takes 2 bytes.
	
	To find the character position of s_2 in a Unicode string, see `ustrpos()`.
	
	`strpos("this","is")` = `3`
	`strpos("this","it")` = `0`
	`strpos("this","")` = `1`

Domain s_1:	strings (to be searched)
Domain s_2:	strings (to search for)
Range:	integers ≥ 0

`ustrpos(`s_1`,`s_2[`,`n]`)`

Description:	the position in s_1 at which s_2 is first found; otherwise, `0`
	
	If *n* is specified and is greater than 0, the search starts at the nth Unicode character of s_1. An invalid UTF-8 sequence in either s_1 or s_2 is replaced with a Unicode replacement character `\ufffd` before the search is performed.
	
	`ustrpos("médiane", "édi")` = `2`
	`ustrpos("médiane", "édi", 3)` = `0`
	`ustrpos("médiane", "éci")` = `0`

Domain s_1:	Unicode strings (to be searched)
Domain s_2:	Unicode strings (to search for)
Domain n:	integers
Range:	integers

`strproper(`*s*`)`

Description:	a string with the first ASCII letter and any other letters immediately following characters that are not letters capitalized; all other ASCII letters converted to lowercase
	
	`strproper()` implements a form of titlecasing and is intended for use only with plain ASCII strings. Unicode characters beyond ASCII are treated as characters that are not letters. To titlecase strings with Unicode characters beyond the plain ASCII range or to implement language-sensitive rules for titlecasing, see `ustrtitle()`.
	
	`strproper("mR. joHn a. sMitH")` = `"Mr. John A. Smith"`
	`strproper("jack o'reilly")` = `"Jack O'Reilly"`
	`strproper("2-cent's worth")` = `"2-Cent'S Worth"`
	`strproper("vous êtes")` = `"Vous êTes"`

Domain *s*:	strings
Range:	strings

`ustrtitle(`*s*[`,`*loc*]`)`

Description:	a string with the first characters of Unicode words titlecased and other characters lowercased
	If *loc* is not specified, the default locale is used. Note that a Unicode word is different from a Stata word produced by function `word()`. The Stata word is a space-separated token. A Unicode word is a language unit based on either a set of word-boundary rules or dictionaries for some languages (Chinese, Japanese, and Thai). The titlecase is also locale dependent and context sensitive; for example, lowercase “ij” is considered a digraph in Dutch. Its titlecase is “IJ”.
	
	`ustrtitle("vous êtes", "fr")` = `"Vous Êtes"`
	`ustrtitle("mR. joHn a. sMitH")` = `"Mr. John A. Smith"`
	`ustrtitle("ijmuiden", "en")` = `"Ijmuiden"`
	`ustrtitle("ijmuiden", "nl")` = `"IJmuiden"`

Domain *s*:	Unicode strings
Domain *loc*:	Unicode strings
Range:	Unicode strings

`strreverse(`*s*`)`

Description:	the reverse of ASCII string *s* `strreverse()` is intended for use only with plain ASCII characters. For Unicode characters beyond ASCII range (code point greater than 127), the encoded bytes are reversed. To reverse the characters of Unicode string, see `ustrreverse()`. `strreverse("hello")` = `"olleh"`
Domain *s*:	ASCII strings
Range:	ASCII reversed strings

`ustrreverse(`s`)`

Description:	the reverse of Unicode string s The function does not take Unicode character equivalence into consideration. Hence, a Unicode character in a decomposed form will not be reversed as one unit. An invalid UTF-8 sequence is replaced with a Unicode replacement character `\ufffd`. `ustrreverse("médiane")` = `"enaidém"`
Domain s:	Unicode strings
Range:	reversed Unicode strings

`strrpos(`s_1`,`s_2`)`

Description:	the position in s_1 at which s_2 is last found, `0` if s_2 does not occur, and `1` if s_2 is empty
	`strrpos()` is intended for use only with plain ASCII characters and for use by programmers who want to obtain the last byte-position of s_2. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, `é` takes 2 bytes.
	To find the last character position of s_2 in a Unicode string, see `ustrrpos()`.
	`strrpos("this","is")` = `3`
	`strrpos("this is","is")` = `6`
	`strrpos("this is","it")` = `0`
	`strrpos("this is","")` = `1`

Domain s_1:	strings (to be searched)
Domain s_2:	strings (to search for)
Range:	integers ≥ 0

`ustrrpos(`s_1`,`s_2[`,`n]`)`

Description:	the position in s_1 at which s_2 is last found; otherwise, `0`
	If n is specified and is greater than 0, only the part between the first Unicode character and the nth Unicode character of s_1 is searched. An invalid UTF-8 sequence in either s_1 or s_2 is replaced with a Unicode replacement character `\ufffd` before the search is performed.
	`ustrrpos("enchanté", "n")` = `6`
	`ustrrpos("enchanté", "n", 5)` = `2`
	`ustrrpos("enchanté", "n", 6)` = `6`
	`ustrrpos("enchanté", "ne")` = `0`

Domain s_1:	Unicode strings (to be searched)
Domain s_2:	Unicode strings (to search for)
Domain n:	integers
Range:	integers

`strrtrim(`s`)`

Description:	s without trailing blanks (ASCII space character `char(32)`) `strrtrim("this ")` = `"this"`
Domain s:	strings
Range:	strings without trailing blanks

`ustrrtrim(`s`)`

Description: remove trailing Unicode whitespace characters and blanks from the Unicode string s

Note that, in addition to `char(32)`, ASCII characters `char(9)`, `char(10)`, `char(11)`, `char(12)`, and `char(13)` are considered whitespace characters in the Unicode standard.

```
ustrrtrim("this ") = "this"
ustrltrim("this"+char(10)) = "this"
ustrrtrim("this "+ustrunescape("\u2000")) = "this"
```

Domain s: Unicode strings
Range: Unicode strings

`strtoname(`s[`,`p]`)`

Description: s translated into a Stata 13 compatible name

`strtoname()` results in a name that is truncated to 32 bytes. Each character in s that is not allowed in a Stata name is converted to an underscore character, _. If the first character in s is a numeric character and p is not 0, then the result is prefixed with an underscore. Stata 14 names may be 32 characters; see **[U] 11.3 Naming conventions**.

```
strtoname("name") = "name"
strtoname("a name") = "a_name"
strtoname("5",1) = "_5"
strtoname("5:30",1) = "_5_30"
strtoname("5",0) = "5"
strtoname("5:30",0) = "5_30"
```

Domain s: strings
Domain p: integers 0 or 1
Range: strings

`ustrtoname(`s[`,`p]`)`

Description: string s translated into a Stata name

`ustrtoname()` results in a name that is truncated to 32 characters. Each character in s that is not allowed in a Stata name is converted to an underscore character, _. If the first character in s is a numeric character and p is not 0, then the result is prefixed with an underscore.

```
ustrtoname("name",1) = "name"
ustrtoname("the médiane") = "the_médiane"
ustrtoname("0médiane") = "_0médiane"
ustrtoname("0médiane", 1) = "_0médiane"
ustrtoname("0médiane", 0) = "0médiane"
```

Domain s: Unicode strings
Domain p: integers 0 or 1
Range: Unicode strings

`strtrim(`*s*`)`

Description:	*s* without leading and trailing blanks (ASCII space character `char(32)`); equivalent to `strltrim(strrtrim(`*s*`))`
	`strtrim(" this ") = "this"`
Domain *s*:	strings
Range:	strings without leading or trailing blanks

`ustrtrim(`*s*`)`

Description:	removes leading and trailing Unicode whitespace characters and blanks from the Unicode string *s*
	Note that, in addition to `char(32)`, ASCII characters `char(9)`, `char(10)`, `char(11)`, `char(12)`, and `char(13)` are considered whitespace characters in the Unicode standard.

	`ustrtrim(" this ") = "this"`
	`ustrtrim(char(11)+" this ")+char(13) = "this"`
	`ustrtrim(" this "+ustrunescape("\u2000")) = "this"`

Domain *s*:	Unicode strings
Range:	Unicode strings

`strupper(`*s*`)`

Description:	uppercase ASCII characters in string *s*
	Unicode characters beyond the plain ASCII range are ignored.

	`strupper("this") = "THIS"`
	`strupper("café") = "CAFé"`

Domain *s*:	strings
Range:	strings with uppercased characters

`ustrupper(`*s*`[ ,`*loc*`])`

Description:	uppercase all characters in string *s* under the given locale *loc*
	If *loc* is not specified, the default locale is used. The same *s* but a different *loc* may produce different results; for example, the uppercase letter of “i” is “I” in English, but “I” with a dot in Turkish. The result can be longer or shorter than the input string in bytes; for example, the uppercase form of the German letter ß (code point `\u00df`) is two capital letters “SS”.

	`ustrupper("médiane","fr") = "MÉDIANE"`
	`ustrupper("Rußland", "de") = "RUSSLAND"`
	`ustrupper("istanbul", "tr") = "ISTANBUL"`

Domain *s*:	Unicode strings
Domain *loc*:	locale name
Range:	Unicode strings

`subinstr(`s_1`,`s_2`,`s_3`,`n`)`

Description: s_1, where the first n occurrences in s_1 of s_2 have been replaced with s_3

`subinstr()` is intended for use only with plain ASCII characters and for use by programmers who want to perform byte-based substitution. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, é takes 2 bytes.

To perform character-based replacement in Unicode strings, see `usubinstr()`.

If n is *missing*, all occurrences are replaced.

Also see `regexm()`, `regexr()`, and `regexs()`.

`subinstr("this is the day","is","X",1)` = `"thX is the day"`
`subinstr("this is the hour","is","X",2)` = `"thX X the hour"`
`subinstr("this is this","is","X",.)` = `"thX X thX"`

Domain s_1:	strings (to be substituted into)
Domain s_2:	strings (to be substituted from)
Domain s_3:	strings (to be substituted with)
Domain n:	integers ≥ 0 or *missing*
Range:	strings

`usubinstr(`s_1`,`s_2`,`s_3`,`n`)`

Description: replaces the first n occurrences of the Unicode string s_2 with the Unicode string s_3 in s_1

If n is *missing*, all occurrences are replaced. An invalid UTF-8 sequence in s_1, s_2, or s_3 is replaced with a Unicode replacement character `\ufffd` before replacement is performed.

`usubinstr("de très près","ès","es",1)` = `"de tres près"`
`usubinstr("de très pr‘es","ès","X",2)` = `"de trX prX"`

Domain s_1:	Unicode strings (to be substituted into)
Domain s_2:	Unicode strings (to be substituted from)
Domain s_3:	Unicode strings (to be substituted with)
Domain n:	integers ≥ 0 or *missing*
Range:	Unicode strings

`subinword(`s_1`,`s_2`,`s_3`,`n`)`

Description:	s_1, where the first n occurrences in s_1 of s_2 as a word have been replaced with s_3
	
	A word is defined as a space-separated token. A token at the beginning or end of s_1 is considered space-separated. This is different from a Unicode word, which is a language unit based on either a set of word-boundary rules or dictionaries for several languages (Chinese, Japanese, and Thai). If n is *missing*, all occurrences are replaced.
	
	Also see `regexm()`, `regexr()`, and `regexs()`.
	
	`subinword("this is the day","is","X",1)` = `"this X the day"`
	`subinword("this is the hour","is","X",.)` = `"this X the hour"`
	`subinword("this is this","th","X",.)` = `"this is this"`

Domain s_1:	strings (to be substituted for)
Domain s_2:	strings (to be substituted from)
Domain s_3:	strings (to be substituted with)
Domain n:	integers ≥ 0 or *missing*
Range:	strings

`substr(`s`,`n_1`,`n_2`)`

Description:	the substring of s, starting at n_1, for a length of n_2
	
	`substr()` is intended for use only with plain ASCII characters and for use by programmers who want to extract a subset of bytes from a string. For those with plain ASCII text, n_1 is the starting character, and n_2 is the length of the string in characters. For programmers, `substr()` is technically a byte-based function. For plain ASCII characters, the two are equivalent but you can operate on byte values beyond that range. Note that any Unicode character beyond ASCII range (code point greater than 127) takes more than 1 byte in the UTF-8 encoding; for example, é takes 2 bytes.
	
	To obtain substrings of Unicode strings, see `usubstr()`.
	
	If $n_1 < 0$, n_1 is interpreted as the distance from the end of the string; if $n_2 = .$ (*missing*), the remaining portion of the string is returned.
	
	`substr("abcdef",2,3)` = `"bcd"`
	`substr("abcdef",-3,2)` = `"de"`
	`substr("abcdef",2,.)` = `"bcdef"`
	`substr("abcdef",-3,.)` = `"def"`
	`substr("abcdef",2,0)` = `""`
	`substr("abcdef",15,2)` = `""`

Domain s:	strings
Domain n_1:	integers ≥ 1 and ≤ -1
Domain n_2:	integers ≥ 1
Range:	strings

`usubstr(`s`,`n_1`,`n_2`)`

Description:	the Unicode substring of s, starting at n_1, for a length of n_2
	
	If $n_1 < 0$, n_1 is interpreted as the distance from the last character of the s; if $n_2 = .$ (*missing*), the remaining portion of the Unicode string is returned.
	
	`usubstr("médiane",2,3)` = `"édi"`
	`usubstr("médiane",-3,2)` = `"an"`
	`usubstr("médiane",2,.)` = `"édiane"`

Domain s:	Unicode strings
Domain n_1:	integers ≥ 1 and ≤ -1
Domain n_2:	integers ≥ 1
Range:	Unicode strings

`udsubstr(`s`,`n_1`,`n_2`)`

Description:	the Unicode substring of s, starting at character n_1, for n_2 display columns
	
	If $n_2 = .$ (*missing*), the remaining portion of the Unicode string is returned. If n_2 display columns from n_1 is in the middle of a Unicode character, the substring stops at the previous Unicode character.
	
	`udsubstr("médiane",2,3)` = `"édi"`
	`udsubstr("中值",1,1)` = `""`
	`udsubstr("中值",1,2)` = `"中"`

Domain s:	Unicode strings
Domain n_1:	integers $\geq$ 1
Domain n_2:	integers $\geq$ 1
Range:	Unicode strings

`tobytes(`s[`,`n]`)`

Description:	escaped decimal or hex digit strings of up to 200 bytes of s
	
	The escaped decimal digit string is in the form of `\dDDD`. The escaped hex digit string is in the form of `\xhh`. If n is not specified or is 0, the decimal form is produced. Otherwise, the hex form is produced.
	
	`tobytes("abc")` = `"\d097\d098\d099"`
	`tobytes("abc", 1)` = `"\x61\x62\x63"`
	`tobytes("café")` = `"\d099\d097\d102\d195\d169"`

Domain s:	Unicode strings
Domain n:	integers
Range:	strings

`uisdigit(`s`)`

Description:	`1` if the first Unicode character in s is a Unicode decimal digit; otherwise, `0` A Unicode decimal digit is a Unicode character with the character property `Nd` according to the Unicode standard. The function returns `-1` if the string starts with an invalid UTF-8 sequence.
Domain s:	Unicode strings
Range:	integers

`uisletter(`s`)`

Description: `1` if the first Unicode character in s is a Unicode letter; otherwise, `0`

A Unicode letter is a Unicode character with the character property `L` according to the Unicode standard. The function returns `-1` if the string starts with an invalid UTF-8 sequence.

Domain s: Unicode strings

Range: integers

`ustrcompare(`s_1`,`s_2`[,`loc`])`

Description: compares two Unicode strings

The function returns `-1`, `1`, or `0` if s_1 is less than, greater than, or equal to s_2. The function may return a negative number other than -1 if an error happens. The comparison is locale dependent. For example, z < ö in Swedish but ö < z in German. If *loc* is not specified, the default locale is used. The comparison is diacritic and case sensitive. If you need different behavior, for example, case-insensitive comparison, you should use the extended comparison function `ustrcompareex()`. Unicode string comparison compares Unicode strings in a language-sensitive manner. On the other hand, the `sort` command compares strings in code-point (binary) order. For example, uppercase “Z” (code-point value 90) comes before lowercase “a” (code-point value 97) in code-point order but comes after “a” in any English dictionary.

```
ustrcompare("z", "ö", "sv") = -1
ustrcompare("z", "ö", "de") = 1
```

Domain s_1: Unicode strings

Domain s_2: Unicode strings

Domain *loc*: Unicode strings

Range: integers

`ustrcompareex(`s_1`,`s_2`,`loc`,`st`,`$case$`,`$cslv$`,`$norm$`,`num`,`alt`,`fr`)`

Description: compares two Unicode strings

The function returns `-1`, `1`, or `0` if s_1 is less than, greater than, or equal to s_2. The function may return a negative number other than `-1` if an error occurs. The comparison is locale dependent. For example, z < ö in Swedish but ö < z in German. If *loc* is not specified, the default locale is used.

st controls the strength of the comparison. Possible values are `1` (primary), `2` (secondary), `3` (tertiary), `4` (quaternary), or `5` (identical). `-1` means to use the default value for the locale. Any other numbers are treated as tertiary. The primary difference represents base letter differences; for example, letter “a” and letter “b” have primary differences. The secondary difference represents diacritical differences on the same base letter; for example, letters “a” and “ä” have secondary differences. The tertiary difference represents case differences of the same base letter; for example, letters “a” and “A” have tertiary differences. Quaternary strength is useful to distinguish between Katakana and Hiragana for the JIS 4061 collation standard. Identical strength is essentially the code-point order of the string, hence, is rarely useful.

ustrcompareex("café","cafe","fr", 1, -1, -1, -1, -1, -1, -1) = 0
ustrcompareex("café","cafe","fr", 2, -1, -1, -1, -1, -1, -1) = 1
ustrcompareex("Café","café","fr", 3, -1, -1, -1, -1, -1, -1) = 1

case controls the uppercase and lowercase letter order. Possible values are 0 (use order specified in tertiary strength), 1 (uppercase first), or 2 (lowercase first). -1 means to use the default value for the locale. Any other values are treated as 0.

ustrcompareex("Café","café","fr", -1, 1, -1, -1, -1, -1, -1) = -1
ustrcompareex("Café","café","fr", -1, 2, -1, -1, -1, -1, -1) = 1

cslv controls whether an extra case level between the secondary level and the tertiary level is generated. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. Combining this setting to be "on" and the strength setting to be primary can achieve the effect of ignoring the diacritical differences but preserving the case differences. If the setting is "on", the result is also affected by the *case* setting.

ustrcompareex("café","Cafe","fr", 1, -1, 1, -1, -1, -1, -1) = -1
ustrcompareex("café","Cafe","fr", 1, 1, 1, -1, -1, -1, -1) = 1

norm controls whether the normalization check and normalizations are performed. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. Most languages do not require normalization for comparison. Normalization is needed in languages that use multiple combining characters such as Arabic, ancient Greek, or Hebrew.

num controls how contiguous digit substrings are sorted. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. Any other values are treated as 0. If the setting is "on", substrings consisting of digits are sorted based on the numeric value. For example, "100" is after value "20" instead of before it. Note that the digit substring is limited to 254 digits, and plus/minus signs, decimals, or exponents are not supported.

ustrcompareex("100","20","en", -1, -1, -1, -1, 0, -1, -1) = -1
ustrcompareex("100","20","en", -1, -1, -1, -1, 1, -1, -1) = 1

alt controls how spaces and punctuation characters are handled. Possible values are 0 (use primary strength) or 1 (alternative handling). Any other values are treated as 0. If the setting is 1 (alternative handling), "onsite", "on-site", and "on site" are considered equals.

```
ustrcompareex("onsite","on-site","en",
        -1, -1, -1, -1, -1, 1, -1) = 0
ustrcompareex("onsite","on site","en",
        -1, -1, -1, -1, -1, 1, -1) = 0
ustrcompareex("onsite","on-site","en",
        -1, -1, -1, -1, -1, 0, -1) = 1
```

fr controls the direction of the secondary strength. Possible values are 0 (off) or 1 (on). -1 means to use the default value for the locale. All other values are treated as "off". If the setting is "on", the diacritical letters are sorted backward. Note that the setting is "on" by default only for Canadian French (locale fr_CA).

`ustrcompareex("coté","côte","fr_CA",-1,-1,-1,-1,-1,-1,0)` = -1
`ustrcompareex("coté","côte","fr_CA",-1,-1,-1,-1,-1,-1,1)` = 1
`ustrcompareex("coté","côte","fr_CA",-1,-1,-1,-1,-1,-1,-1)` = 1
`ustrcompareex("coté","côte","fr",-1,-1,-1,-1,-1,-1,-1)` = 1

Domain s_1:	Unicode strings
Domain s_2:	Unicode strings
Domain *loc*:	Unicode strings
Domain *st*:	integers
Domain *case*:	integers
Domain *cslv*:	integers
Domain *norm*:	integers
Domain *num*:	integers
Domain *alt*:	integers
Domain *fr*:	integers
Range:	integers

`ustrfix(`*s*[`,`*rep*]`)`

Description: replaces each invalid UTF-8 sequence with a Unicode character

In the one-argument case, the Unicode replacement character `\ufffd` is used. In the two-argument case, the first Unicode character of *rep* is used. If *rep* starts with an invalid UTF-8 sequence, then Unicode replacement character `\ufffd` is used. Note that an invalid UTF-8 sequence can contain one byte or multiple bytes.

`ustrfix(char(200))` = `ustrunescape("\ufffd")`
`ustrfix("ab"+char(200)+"cdé", "")` = `"abcdé"`
`ustrfix("ab"+char(229)+char(174)+"cdé", "é")` = `"abécdé"`

Domain *s*:	Unicode strings
Domain *rep*:	Unicode character
Range:	Unicode strings

`ustrfrom(`*s*`,`*enc*`,`*mode*`)`

Description: converts the string *s* in encoding *enc* to a UTF-8 encoded Unicode string

mode controls how invalid byte sequences in *s* are handled. The possible values are `1`, which substitutes an invalid byte sequence with a Unicode replacement character `\ufffd`; `2`, which skips any invalid byte sequences; `3`, which stops at the first invalid byte sequence and returns an empty string; or `4`, which replaces any byte in an invalid sequence with an escaped hex digit sequence `%Xhh`. Any other values are treated as `1`. A good use of value `4` is to check what invalid bytes a Unicode string *ust* contains by examining the result of `ustrfrom(ust, "utf-8", 4)`.

Also see `ustrto()`.

```
ustrfrom("caf"+char(233), "latin1", 1) = "café"
ustrfrom("caf"+char(233), "utf-8", 1) =
        "caf"+ustrunescape("\ufffd")
ustrfrom("caf"+char(233), "utf-8", 2) = "caf"
ustrfrom("caf"+char(233), "utf-8", 3) = ""
ustrfrom("caf"+char(233), "utf-8", 4) = "caf%XE9"
```

Domain *s*:	strings in encoding *enc*
Domain *enc*:	Unicode strings
Domain *mode*:	integers
Range:	Unicode strings

`ustrinvalidcnt(`*s*`)`

Description: the number of invalid UTF-8 sequences in *s*

An invalid UTF-8 sequence may contain one byte or multiple bytes.

```
ustrinvalidcnt("médiane") = 0
ustrinvalidcnt("médiane"+char(229)) = 1
ustrinvalidcnt("médiane"+char(229)+char(174)) = 1
ustrinvalidcnt("médiane"+char(174)+char(158)) = 2
```

Domain *s*:	Unicode strings
Range:	integers

`ustrleft(`*s*`,`*n*`)`

Description: the first *n* Unicode characters of the Unicode string *s*

An invalid UTF-8 sequence is replaced with a Unicode replacement character `\ufffd`.

```
ustrleft("Экспериментальные",3) = "Экс"
ustrleft("Экспериментальные",5) = "Экспе"
```

Domain *s*:	Unicode strings
Domain *n*:	integers
Range:	Unicode strings

`ustrnormalize(`*s*`,`*norm*`)`

Description: normalizes Unicode string *s* to one of the five normalization forms specified by *norm*

The normalization forms are `nfc`, `nfd`, `nfkc`, `nfkd`, or `nfkcc`. The function returns an empty string for any other value of *norm*. Unicode normalization removes the Unicode string differences caused by Unicode character equivalence. `nfc` specifies `Normalization Form C`, which normalizes decomposed Unicode code points to a composited form. `nfd` specifies `Normalization Form D`, which normalizes composited Unicode code points to a decomposed form. `nfc` and `nfd` produce canonical equivalent form. `nfkc` and `nfkd` are similar to `nfc` and `nfd` but produce compatibility equivalent forms. `nfkcc` specifies `nfkc` with casefolding. This normalization and casefolding implement the Unicode Character Database.

In the Unicode standard, both "i" (`\u0069` followed by a diaeresis `\u0308`) and the composite character `\u00ef` represent "i" with 2 dots as in "naïve". Hence, the code-point sequence `\u0069\u0308` and the code point `\u00ef` are considered Unicode equivalent. According to the Unicode standard, they should be treated as the same single character in Unicode string operations, such as in display, comparison, and selection. However, Stata does not support multiple code-point characters; each code point is considered a separate Unicode character. Hence, `\u0069\u0308` is displayed as two characters in the Results window. `ustrnormalize()` can be used with `"nfc"` to normalize `\u0069\u0308` to the canonical equivalent composited code point `\u00ef`.

`ustrnormalize(ustrunescape("\u0069\u0308"), "nfc")` = `"ï"`

The decomposed form `nfd` can be used to removed diacritical marks from base letters. First, normalize the Unicode string to canonical decomposed form, and then call `ustrto()` with mode `skip` to skip all non-ASCII characters.

Also see `ustrfrom()`.

`ustrto(ustrnormalize("café", "nfd"), "ascii", 2)` = `"cafe"`

Domain *s*:	Unicode strings
Domain *norm*:	Unicode strings
Range:	Unicode strings

`ustrright(`*s*`,`*n*`)`

Description:	the last *n* Unicode characters of the Unicode string *s*

An invalid UTF-8 sequence is replaced with a Unicode replacement character `\ufffd`.

```
ustrright("Экспериментальные",3) = "ные"
ustrright("Экспериментальные",5) = "льные"
```

Domain *s*:	Unicode strings
Domain *n*:	integers
Range:	Unicode strings

`ustrsortkey(`*s*[`,`*loc*]`)`

Description:	generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`

The function may return an empty array if an error occurs. The result is locale dependent. If *loc* is not specified, the default locale is used. The result is also diacritic and case sensitive. If you need different behavior, for example, case-insensitive results, you should use the extended function `ustrsortkeyex()`. See **[U] 12.4.2.5 Sorting strings containing Unicode characters** for details and examples.

Domain *s*:	Unicode strings
Domain *loc*:	Unicode strings
Range:	null-terminated byte array

`ustrsortkeyex(`*s*`,`*loc*`,`*case*`,`*cslv*`,`*norm*`,`*num*`,`*alt*`,`*fr*`)`

Description: generates a null-terminated byte array that can be used by the `sort` command to produce the same order as `ustrcompare()`

The function may return an empty array if an error occurs. The result is locale dependent. If *loc* is not specified, the default locale is used. See **[U] 12.4.2.5 Sorting strings containing Unicode characters** for details and examples.

st controls the strength of the comparison. Possible values are `1` (primary), `2` (secondary), `3` (tertiary), `4` (quaternary), or `5` (identical). `-1` means to use the default value for the locale. Any other numbers are treated as tertiary. The primary difference represents base letter differences; for example, letter “a” and letter “b” have primary differences. The secondary difference represents diacritical differences on the same base letter; for example, letters “a” and “ä” have secondary differences. The tertiary difference represents case differences of the same base letters; for example, letters “a” and “A” have tertiary differences. Quaternary strength is useful to distinguish between Katakana and Hiragana for the JIS 4061 collation standard. Identical strength is essentially the code-point order of the string and, hence, is rarely useful.

case controls the uppercase and lowercase letter order. Possible values are `0` (use order specified in tertiary strength), `1` (uppercase first), or `2` (lowercase first). `-1` means to use the default value for the locale. Any other values are treated as `0`.

cslv controls if an extra case level between the secondary level and the tertiary level is generated. Possible values are `0` (off) or `1` (on). `-1` means to use the default value for the locale. Any other values are treated as `0`. Combining this setting to be “on” and the strength setting to be primary can achieve the effect of ignoring the diacritical differences but preserving the case differences. If the setting is “on”, the result is also affected by the *case* setting.

norm controls whether the normalization check and normalizations are performed. Possible values are `0` (off) or `1` (on). `-1` means to use the default value for the locale. Any other values are treated as `0`. Most languages do not require normalization for comparison. Normalization is needed in languages that use multiple combining characters such as Arabic, ancient Greek, or Hebrew.

num controls how contiguous digit substrings are sorted. Possible values are `0` (off) or `1` (on). `-1` means to use the default value for the locale. Any other values are treated as `0`. If the setting is “on”, substrings consisting of digits are sorted based on the numeric value. For example, “100” is after “20” instead of before it. Note that the digit substring is limited to 254 digits, and plus/minus signs, decimals, or exponents are not supported.

alt controls how spaces and punctuation characters are handled. Possible values are `0` (use primary strength) or `1` (alternative handling). Any other values are treated as `0`. If the setting is `1` (alternative handling), “onsite”, “on-site”, and “on site” are considered equals.

fr controls the direction of the secondary strength. Possible values are `0` (off) or `1` (on). `-1` means to use the default value for the locale. All other values are treated as “off”. If the setting is “on”, the diacritical letters are sorted backward. Note that the setting is “on” by default only for Canadian French (locale `fr_CA`).

Domain *s*:	Unicode strings
Domain *loc*:	Unicode strings
Domain *st*:	integers
Domain *case*:	integers
Domain *cslv*:	integers
Domain *norm*:	integers
Domain *num*:	integers
Domain *alt*:	integers
Domain *fr*:	integers
Range:	null-terminated byte array

`ustrto(`*s*`,`*enc*`,`*mode*`)`

Description: converts the Unicode string *s* in UTF-8 encoding to a string in encoding *enc*

See [D] **unicode encoding** for details on available encodings. Any invalid sequence in *s* is replaced with a Unicode replacement character `\ufffd`. *mode* controls how unsupported Unicode characters in the encoding *enc* are handled. The possible values are `1`, which substitutes any unsupported characters with the *enc*’s substitution strings (the substitution character for both `ascii` and `latin1` is `char(26)`); `2`, which skips any unsupported characters; `3`, which stops at the first unsupported character and returns an empty string; or `4`, which replaces any unsupported character with an escaped hex digit sequence `\uhhhh` or `\Uhhhhhhhh`. The hex digit sequence contains either 4 or 8 hex digits, depending if the Unicode character’s code-point value is less than or greater than `\uffff`. Any other values are treated as `1`.

```
ustrto("café", "ascii", 1) = "caf"+char(26)
ustrto("café", "ascii", 2) = "caf"
ustrto("café", "ascii", 3) = ""
ustrto("café", "ascii", 4) = "caf\u00E9"
```

`ustrto()` can be used to removed diacritical marks from base letters. First, normalize the Unicode string to `NFD` form using `ustrnormalize()`, and then call `ustrto()` with value `2` to skip all non-ASCII characters.

Also see `ustrfrom()`.

```
ustrto(ustrnormalize("café", "nfd"), "ascii", 2) = "cafe"
```

Domain *s*:	Unicode strings
Domain *enc*:	Unicode strings
Domain *mode*:	integers
Range:	strings in encoding *enc*

`ustrtohex(`s`[,`n`])`

Description: escaped hex digit string of s up to 200 Unicode characters

The escaped hex digit string is in the form of `\uhhhh` for code points less than `\uffff` or `\Uhhhhhhhh` for code points greater than `\uffff`. The function starts at the nth Unicode character of s if n is specified and larger than 0. Any invalid UTF-8 sequence is replaced with a Unicode replacement character `\ufffd`. Note that the null terminator `char(0)` is a valid Unicode character. Function `ustrunescape()` can be applied on the result to get back the original Unicode string s if s does not contain any invalid UTF-8 sequences.

Also see `ustrunescape()`.

```
ustrtohex("нулю") = "\u043d\u0443\u043b\u044e"
ustrtohex("нулю", 2) = "\u0443\u043b\u044e"
ustrtohex("i"+char(200)+char(0)+"s") =
        "\u0069\ufffd\u0000\u0073"
```

Domain s: Unicode strings
Domain n: integers ≥ 1
Range: strings

`ustrunescape(`s`)`

Description: the Unicode string corresponding to the escaped sequences of s

The following escape sequences are recognized: 4 hex digit form `\uhhhh`; 8 hex digit form `\Uhhhhhhhh`; 1–2 hex digit form `\xhh`; and 1–3 octal digit form `\ooo`, where `h` is `[0-9A-Fa-f]` and `o` is `[0-7]`. The standard ANSI C escapes `\a`, `\b`, `\t`, `\n`, `\v`, `\f`, `\r`, `\e`, `\"`, `\'`, `\?`, `\\` are recognized as well. The function returns an empty string if an escape sequence is badly formed. Note that the 8 hex digit form `\Uhhhhhhhh` begins with a capital letter “U”.

Also see `ustrtohex()`.

```
ustrunescape("\u043d\u0443\u043b\u044e") = "нулю"
```

Domain s: strings of escaped hex values
Range: Unicode strings

`word(`s`,`n`)`

Description: the nth word in s; *missing* (`""`) if n is missing

Positive numbers count words from the beginning of s, and negative numbers count words from the end of s. (`1` is the first word in s, and `-1` is the last word in s.) A word is a set of characters that start and terminate with spaces. This is different from a Unicode word, which is a language unit based on either a set of word-boundary rules or dictionaries for several languages (Chinese, Japanese, and Thai).

Domain s: strings
Domain n: integers
Range: strings

`ustrword(`s`,`n`[,`*loc*`])`

Description:	the nth Unicode word in the Unicode string s
	
	Positive n counts Unicode words from the beginning of s, and negative n counts Unicode words from the end of s. For examples, n equal to 1 returns the first word in s, and n equal to -1 returns the last word in s. If *loc* is not specified, the default locale is used. A Unicode word is different from a Stata word produced by the `word()` function. A Stata word is a space-separated token. A Unicode word is a language unit based on either a set of word-boundary rules or dictionaries for some languages (Chinese, Japanese, and Thai). The function returns *missing* (`""`) if n is greater than *cnt* or less than −*cnt*, where *cnt* is the number of words s contains. *cnt* can be obtained from `ustrwordcount()`. The function also returns *missing* (`""`) if an error occurs.
	
	`ustrword("Parlez-vous français", 1, "fr")` = `"Parlez"`
	`ustrword("Parlez-vous français", 2, "fr")` = `"-"`
	`ustrword("Parlez-vous français",-1, "fr")` = `"français"`
	`ustrword("Parlez-vous français",-2, "fr")` = `"vous"`

Domain s:	Unicode strings
Domain *loc*:	Unicode strings
Domain n:	integers
Range:	Unicode strings

`wordbreaklocale(`*loc*`,`*type*`)`

Description:	the most closely related locale supported by ICU from *loc* if *type* is 1, the actual locale where the word-boundary analysis data come from if *type* is 2; or an empty string is returned for any other *type*
	
	`wordbreaklocale("en_us_texas", 1)` = `en_US`
	`wordbreaklocale("en_us_texas", 2)` = `root`

Domain *loc*:	strings of locale name
Domain *type*:	integers
Range:	strings

`wordcount(`s`)`

Description:	the number of words in s A word is a set of characters that starts and terminates with spaces, starts with the beginning of the string, or terminates with the end of the string. This is different from a Unicode word, which is a language unit based on either a set of word-boundary rules or dictionaries for several languages (Chinese, Japanese, and Thai).
Domain s:	strings
Range:	nonnegative integers 0, 1, 2, ...

`ustrwordcount(`*s*[`,`*loc*]`)`

Description: the number of nonempty Unicode words in the Unicode string *s*

An empty Unicode word is a Unicode word consisting of only Unicode whitespace characters. If *loc* is not specified, the default locale is used. A Unicode word is different from a Stata word produced by the `word()` function. A Stata word is a space-separated token. A Unicode word is a language unit based on either a set of word-boundary rules or dictionaries for some languages (Chinese, Japanese, and Thai). The function may return a negative number if an error occurs.

`ustrwordcount("Parlez-vous français", "fr")` = 4

Domain *s*: Unicode strings

Domain *loc*: Unicode strings

Range: integers

References

Cox, N. J. 2004. Stata tip 6: Inserting awkward characters in the plot. *Stata Journal* 4: 95–96.

———. 2011. Stata tip 98: Counting substrings within strings. *Stata Journal* 11: 318–320.

———. 2022. Stata tip 148: Searching for words within strings. *Stata Journal* 22: 998–1003.

Jeanty, P. W. 2013. Dealing with identifier variables in data management and analysis. *Stata Journal* 13: 699–718.

Koplenig, A. 2018. Stata tip 129: Efficiently processing textual data with Stata's new Unicode features. *Stata Journal* 18: 287–289.

Schwarz, C. 2019. lsemantica: A command for text similarity based on latent semantic analysis. *Stata Journal* 19: 129–142.

Also see

[FN] **Functions by category**

[D] **egen** — Extensions to generate

[D] **generate** — Create or change contents of variable

[M-4] **String** — String manipulation functions

[U] 12.4.2 Handling Unicode strings

[U] 13.2.2 String operators

[U] 13.3 Functions