Title

stata.com

inbase( ) — Base conversion

Description

`inbase(`*base*`,` *x*`)` returns a string matrix containing the values of *x* in base *base*.

`inbase(`*base*`,` *x*`,` *fdigits*`)` does the same; *fdigits* specifies the maximum number of digits to the right of the base point to appear in the returned result when *x* has a fractional part. `inbase(`*base*`,` *x*`)` is equivalent to `inbase(`*base*`,` *x*`, 8)`.

`inbase(`*base*`,` *x*`,` *fdigits*`,` *err*`)` is the same as `inbase(`*base*`,` *x*`,` *fdigits*`)`, except that it returns in *err* the difference between *x* and the converted result.

x = `frombase(`*base*`,` *s*`)` is the inverse of *s* = `inbase(`*base*`,` *x*`)`. It returns base *base* number *s* as a number. We are tempted to say, "as a number in base 10", but that is not exactly true. It returns the result as a *real*, that is, as an IEEE base-2 double-precision float that, when you display it, is displayed in base 10.

Syntax

string matrix `inbase(`*real scalar base*`,` *real matrix x* [`,` *real scalar fdigits* [`,` *err*]]`)`

real matrix `frombase(`*real scalar base*`,` *string matrix s*`)`

Remarks and examples

stata.com

Remarks are presented under the following headings:

Positive integers

`inbase(2, 1691)` is 11010011011; that is, 1691 base 10 equals 11010011011 base 2. `frombase(2, "11010011011")` is 1691.

`inbase(3, 1691)` is 2022122; that is, 1691 base 10 equals 2022122 base 3. `frombase(3, "2022122")` is 1691.

`inbase(16, 1691)` is `69b`; that is, 1691 base 10 equals 1691 base 16. `frombase(16, "69b")` is 1691. (The base-16 digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f.)

`inbase(62, 1691)` is `rh`; that is, 1691 base 10 equals rh base 62. `frombase(62, "rh")` is 1691. (The base-62 digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, . . . , z, A, B, . . . , Z.)

There is a one-to-one correspondence between the integers in different bases. The error of the conversion is always zero.

Negative integers

Negative integers are no different from positive integers. For instance, `inbase(2, -1691)` is `-11010011011`; that is, −1691 base 10 equals −11010011011 base 2. `frombase(2, "-11010011011")` is −1691.

The error of the conversion is always zero.

Numbers with nonzero fractional parts

`inbase(2, 3.5)` is `11.1`; that is, 3.5 base 10 equals 11.1 base 2. `frombase(2, "11.1")` is 3.5.

`inbase(3, 3.5)` is `10.11111111`.

`inbase(3, 3.5, 20)` is `10.11111111111111111111`.

`inbase(3, 3.5, 30)` is `10.111111111111111111111111111111`.

Therefore, 3.5 base 10 equals 1.1111... in base 3. There is no exact representation of one-half in base 3. The errors of the above three conversions are .0000762079, 1.433399e–10, and 2.45650e–15. Those are the values that would be returned in *err* if `inbase(3, 3.5,` *fdigits*`,` *err*`)` were coded.

`frombase(3, "10.11111111")` is `3.499923792`.

`frombase(3, "10.11111111111111111111")` is `3.4999999998566`.

`frombase(3, "10.111111111111111111111111111111")` is `3.49999999999999734`.

`inbase(16, 3.5)` is `3.8`; that is, 3.5 base 10 equals 3.8 base 16. The error is zero. `frombase(16, "3.8")` is 3.5.

`inbase(62, 3.5)` is `3.v`; that is, 3.5 base 10 equals 3.v base 62. `frombase(62, "3.v")` is 3.5. The error is zero.

In `inbase(`*base*`,` *x*`,` *fdigits*`)`, *fdigits* specifies the maximum number of digits to appear to the right of the base point. *fdigits* is required to be greater than or equal to 1. `inbase(16, 3.5,` *fdigits*`)` will be `3.8` regardless of the value of *fdigits* because more digits are unnecessary.

The error that is returned in `inbase(`*base*`,` *x*`,` *fdigits*`,` *err*`)` can be an understatement. For instance, `inbase(16, .1, 14,` *err*`)` is `0.1999999999999a` and returned in *err* is 0 even though there is no finite-digit representation of 0.1 base 10 in base 16. That is because the `.1` you specified in the call was not actually 0.1 base 10. The computer that you are using is binary, and it converted the `.1` you typed to

0.0001100110011001100110011001100110011001100110011001100110011010 base 2

before `inbase()` was ever called. 0.1999999999999a base 16 is an exact representation of that number.

Use of the functions

These functions are used mainly for teaching, especially on the sources and avoidance of roundoff error; see Gould (2006).

The functions can have a use in data processing, however, when used with integer arguments. You have a dataset with 10-digit identification numbers. You wish to record the 10-digit number, but more densely. You could convert the number to base 62. The largest 10-digit ID number possible is 9999999999, or aUKYOz base 62. You can record the ID numbers in a six-character string by using `inbase()`. If you needed the original numbers back, you could use `frombase()`.

In a similar way, Stata internally uses base 36 for naming temporary files, and that was important when filenames were limited to eight characters. Base 36 allows Stata to generate up to 2,821,109,907,455 filenames before wrapping of filenames occurs.

Conformability

`inbase(`*base*`,` *x*`,` *fdigits*`,` *err*`)`:

input:			
	base:	1×1	
	x:	$r \times c$	
	fdigits:	1×1	(optional)
output:			
	err:	$r \times c$	(optional)
	result:	$r \times c$	

`frombase(`*base*`,` *s*`)`:

base:	1×1	
s:	$r \times c$	
result:	$r \times c$	

Diagnostics

The digits used by `inbase()`/`frombase()` to encode/decode results are

0 0	*10* a	*20* k	*30* u	*40* E	*50* O	*60* Y
1 1	*11* b	*21* l	*31* v	*41* F	*51* P	*61* Z
2 2	*12* c	*22* m	*32* w	*42* G	*52* Q	
3 3	*13* d	*23* n	*33* x	*43* H	*53* R	
4 4	*14* e	*24* o	*34* y	*44* I	*54* S	
5 5	*15* f	*25* p	*35* z	*45* J	*55* T	
6 6	*16* g	*26* q	*36* A	*46* K	*56* U	
7 7	*17* h	*27* r	*37* B	*47* L	*57* V	
8 8	*18* i	*28* s	*38* C	*48* M	*58* W	
9 9	*19* j	*29* t	*39* D	*49* N	*59* X	

When *base* ≤ 36, `frombase()` treats `A`, `B`, `C`, ..., as if they were `a`, `b`, `c`, ...

`inbase(`*base*`,` *x*`,` *fdigits*`,` *err*`)` returns `.` (missing) if *base* < 2, *base* > 62, *base* is not an integer, or *x* is missing. If *fdigits* is less than 1 or *fdigits* is missing, results are as if *fdigits* $= 8$ were specified.

`frombase(`*base*`,` *s*`)` returns `.` (missing) if *base* < 2, *base* > 62, *base* is not an integer, or *s* is missing; if *s* is not a valid base *base* number; or if the converted value of *s* is greater than 8.988e+307 in absolute value.

Reference

Gould, W. W. 2006. Mata Matters: Precision. *Stata Journal* 6: 550–560.

Also see

[M-4] **Mathematical** — Important mathematical functions