pathjoin( ) — File path manipulation

Description Syntax Remarks and examples Conformability Diagnostics Also see

Description

`pathjoin(`*path1*`,` *path2*`)` forms, logically speaking, *path1*`/`*path2*, but does so in the appropriate style. For instance, *path1* might be a URL and *path2* a Windows *dirname*`\`*filename*, and the two paths will, even so, be joined correctly. All issues of whether *path1* ends with a directory separator, *path2* begins with one, etc., are handled automatically.

`pathsplit(`*path*`,` *path1*`,` *path2*`)` performs the inverse operation, removing the last element of the path (which is typically a filename) and storing it in *path2* and storing the rest in *path1*.

`pathbasename(`*path*`)` returns the last element of *path*.

`pathsuffix(`*path*`)` returns the file suffix, with leading dot, if there is one, and returns `""` otherwise. For instance, `pathsuffix("this\that.ado")` returns "`.ado`".

`pathrmsuffix(`*path*`)` returns *path* with the suffix removed, if there was one. For instance, `pathrmsuffix("th`
returns "`this\that`".

`pathisurl(`*path*`)` returns 1 if *path* is a URL and 0 otherwise.

`pathisabs(`*path*`)` returns 1 if *path* is absolute and 0 if relative. `c:\this` is an absolute path. `this\that` is a relative path. URLs are considered to be absolute.

`pathasciisuffix(`*path*`)` and `pathstatasuffix(`*path*`)` are more for StataCorp use than anything else. `pathasciisuffix()` returns 1 if the file is known to be text, based on its file suffix. StataCorp uses this function in Stata's `net` command to decide whether end-of-line characters, which differ across operating systems, should be modified during downloads. `pathstatasuffix()` is the function used by Stata's `net` and `update` commands to decide whether a file belongs in the official directories. `pathstatasuffix("example.ado")` is true, but `pathstatasuffix("example.do")` is false because do-files do not go in system directories.

`pathlist(`*dirlist*`)` returns a row vector, each element of which contains an element of a semicolon-separated path list *dirlist*. For instance, `pathlist("`*a*`;`*b*`;`*c*`")` returns `("`*a*`", "`*b*`", "`*c*`")`.

`pathlist()` without arguments returns `pathlist(c("adopath"))`, the broken-out elements of the official Stata ado-path.

`pathsubsysdir(`*pathlist*`)` returns *pathlist* with any elements that are Stata system directories' shorthands, such as `PLUS`, `PERSONAL`, substituted with the actual directory names. For instance, the right way to obtain the official directories over which Stata searches for files is `pathsubsysdir(pathlist())`.

`pathsearchlist(`*fn*`)` returns a row vector. The elements are full paths/filenames specifying all the locations, in order, where Stata would look for *fn* along the official Stata ado-path.

`pathresolve(`*basepath*`,` *path*`)` returns a resolved path of *path* against *basepath*. For example, `pathresolve("c:/test", "../test1")` returns `"c:/test1"`.

`pathgetparent(`*path*`)` returns the parent path of *path*. For example, `pathgetparent("c:/test/test.do")` returns `"c:/test"`. Neither `c:/test` nor `c:/test/test.do` need exist.

Syntax

string scalar	`pathjoin(`*string scalar path1*`,` *string scalar path2*`)`
void	`pathsplit(`*string scalar path*`,` *path1*`,` *path2*`)`
string scalar	`pathbasename(`*string scalar path*`)`
string scalar	`pathsuffix(`*string scalar path*`)`
string scalar	`pathrmsuffix(`*string scalar path*`)`
real scalar	`pathisurl(`*string scalar path*`)`
real scalar	`pathisabs(`*string scalar path*`)`
real scalar	`pathasciisuffix(`*string scalar path*`)`
real scalar	`pathstatasuffix(`*string scalar path*`)`
string rowvector	`pathlist(`*string scalar dirlist*`)`
string rowvector	`pathlist()`
string rowvector	`pathsubsysdir(`*string rowvector pathlist*`)`
string rowvector	`pathsearchlist(`*string scalar fn*`)`
string scalar	`pathresolve(`*string scalar basepath*`,` *path*`)`
string scalar	`pathgetparent(`*string scalar path*`)`

Remarks and examples

Using these functions, you are more likely to produce code that works correctly regardless of operating system.

`pathremove()` returns an error code if you do not have the necessary permission to remove the directory or its contents.

`pathresolve(`*basepath*`,` *path*`)` simply returns *path* if it is an absolute path.

`pathresolve(`*basepath*`,` *path*`)`'s behavior is undefined if *basepath* has no root element, for example, `../`.

Conformability

`pathjoin(`*path1*`,` *path2*`)`:

path1:	1×1	
path2:	1×1	
result:	1×1	

`pathsplit(`*path*`,` *path1*`,` *path2*`)`:

input:		
path:	1×1	
output:		
path1:	1×1	
path2:	1×1	

`pathbasename(`*path*`)`, `pathsuffix(`*path*`)`, `pathrmsuffix(`*path*`)`:

path:	1×1	
result:	1×1	

`pathisurl(`*path*`)`, `pathisabs(`*path*`)`, `pathasciisuffix(`*path*`)`, `pathstatasuffix(`*path*`)`:

path:	1×1	
result:	1×1	

`pathlist(`*dirlist*`)`:

dirlist:	1×1	(optional)
result:	$1 \times k$	

`pathsubsysdir(`*pathlist*`)`:

pathlist:	$1 \times k$	
result:	$1 \times k$	

`pathsearchlist(`*fn*`)`:

fn:	1×1	
result:	$1 \times k$	

`pathresolve(`*basepath*`,` *path*`)`:

basepath:	1×1	
path:	1×1	
result:	1×1	

`pathgetparent(`*path*`)`:

path:	1×1	
result:	1×1	

Diagnostics

All routines abort with error if the path is too long for the operating system.

Also see

[M-4] **IO** — I/O functions