Package 'shinyFiles'

January 30, 2015

Type Package

Title A server-side file system viewer for shiny

Version 0.5.0

Date 2015-01-30

Author Thomas Lin Pedersen

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Description Provides functionality for client-side navigation of the server side file system in shiny apps. In case the app is running locally this gives the user direct access to the file system without the need to ``download'' files to a temporary location.

License GPL (>= 2)

Imports tools, shiny (>= 0.11), htmltools

Collate 'aaa.R' 'filechoose.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2015-01-30 13:15:05

R topics documented:

fileGetter 2
getVolumes 2
parseFilePaths 3
shinyFileChoose 4
shinyFilesButton 6
shinyFilesExample 7

Index **8**

`fileGetter` *Create a function that returns fileinfo according to the given restrictions*

Description

This functions returns a new function that can generate file information to be send to a shiny app based on a path relative to the given root. The function is secure in the sense that it prevents access to files outside of the given root directory as well as to subdirectories matching the ones given in restrictions. Furthermore can the output be filtered to only contain certain filetypes using the filter parameter and hidden files can be toggled with the hidden parameter.

Usage

```
fileGetter(roots, restrictions, filetypes, hidden = FALSE)
```

Arguments

`roots`	A named vector of absolute filepaths or a function returning a named vector of absolute filepaths (the latter is useful if the volumes should adapt to changes in the filesystem).
`restrictions`	A vector of directories within the root that should be filtered out of the results
`filetypes`	A character vector of file extensions (without dot in front i.e. 'txt' not '.txt') to include in the output. Use the empty string to include files with no extension. If not set all file types will be included
`hidden`	A logical value specifying whether hidden files should be returned or not

Value

A function taking a single path relative to the specified root, and returns a list of files to be passed on to shiny

`getVolumes` *Get a list of available volumes*

Description

This function is intended as an input to the roots parameter in `fileGetter` and `shinyFileChoose`. It returns a function that returns a named vector of available volumes on the system. This construction makes it dynamic so that a shinyFiles instance reflects new volumes as they get added (e.g. usb drives). The function takes a single argument giving names of volumes the developer wants removed from the return value.

Usage

```
getVolumes(exclude)
```

Arguments

exclude	A vector of volume names to be excluded from the return value

Details

The function is OS specific and looks for volumes/drives in different places depending on the system on which shiny is running.

Windows Returns all drives mapped to a letter

Mac OSX Looks in /Volumes/ and lists the directories therein

Linux Returns the system root

If the function does not recognize the system under which it is running it will throw an error

Value

A function returning a named vector of available volumes

---

parseFilePaths *Convert the output of a file choice to a data frame*

Description

This function takes the value of a shinyFiles button input variable and converts it to a data frame of the same format as that provided by `fileInput`. The only caveat is that the MIME type cannot be inferred so this will always be an empty string.

Usage

```
parseFilePaths(roots, files)
```

Arguments

roots	The path to the root as specified in the `shinyFileChoose()` call in `shinyServer()`
files	The corresponding input variable to be parsed

Details

The use of `parseFilePaths` makes it easy to substitute fileInput and shinyFiles in your code as code that relies on the values of a file selection doesn't have to change.

Value

A data frame mathcing the format of `link[shiny]{fileInput}`

See Also

Other shinyFiles: shinyFileChoose; shinyFilesButton; shinyFilesExample

Examples

```
## Not run:
ui <- shinyUI(bootstrapPage(
    shinyFilesButton('files', 'File select', 'Please select a file', FALSE)
    verbatimTextOutput('filepaths')
))
server <- shinyServer(function(input, output) {
    shinyFileChoose(input, 'files', roots=c(wd='.'),
                                    filetypes=c('', '.txt'))
    output$filepaths <- renderText({parseFilePaths('.', input$files)})
})

runApp(list(
    ui=ui,
    server=server
))

## End(Not run)
```

---

shinyFileChoose *Create a connection to the server side filesystem*

Description

NOTE: The syntax for this version has changed with version 0.4.0. Prior to that version the output of `shinyFileChoose()` should be assigned to the output object. This is no longer the case and doind so will result in an error. In newer versions the function returns an observer which can be ignored for the most part, or assigned to a variable if there needs to be interactions with it later on.

Usage

```
shinyFileChoose(input, id, updateFreq = 2000, session, ...)
```

Arguments

`input`	The input object of the `shinyServer()` call (usaully `input`)
`id`	The same ID as used in the matching call to `shinyFilesButton` or as the id attribute of the button, in case of a manually defined html. This id will also define the id of the file choice in the input variable
`updateFreq`	The time in milliseconds between file system lookups. This determines the responsiveness to changes in the filesystem (e.g. addition of files or drives)
`session`	The session object of the shinyServer call (usually `session`).
`...`	Arguments to be passed on to `fileGetter`

Details

This function sets up the required connection to the client in order for the user to navigate the filesystem. For this to work a matching button should be present in the html, either by using `shinyFilesButton()` or adding it manually. See `shinyFilesButton` for more information on this.

Restrictions on the access rights of the client can be given in several ways. The root parameter specifies the starting position for the filesystem as presented to the client. This means that the client can only navigate in subdirectories of the root. Paths passed of to the `restrictions` parameter will not show up in the client view, and it is impossible to navigate into these subdirectories. The `filetypes` parameter takes a vector of of file extensions to filter the output on, so that the client is only presented with these filetypes. The `hidden` parameter toggles whether hidden files should be visible or not. Whenever a file choice is made the resulting files will be accessible in the input variable with the id given in the parameters. This value should probable be run through a call to `parseFilePaths` in order to get well formatted paths to work with.

Value

A reactive observer that takes care of the server side logic of the filesystem connection. Prior to v0.4.0 the return was a reactive expression that needed to be assigned to the output variable. As of now this is no longer allowed and will lead to an error (as reactive observers cannot be assigned to output)

See Also

Other shinyFiles: `parseFilePaths`; `shinyFilesButton`; `shinyFilesExample`

Examples

```
## Not run:
ui <- shinyUI(bootstrapPage(
    shinyFilesButton('files', 'File select', 'Please select a file', FALSE)
))
server <- shinyServer(function(input, output, session) {
    shinyFileChoose(input, 'files', session=session,
                    roots=c(wd='.'), filetypes=c('', '.txt'))
})

runApp(list(
    ui=ui,
    server=server
))

## End(Not run)
```

shinyFilesButton *Create a button to summon the file system navigator*

Description

This function adds the required html markup for the client to access the file system. The end result will be the appearance of a button on the webpage that summons the file system navigator dialog box. The last position in the file system is automatically remembered between instances, but not shared between several shinyFiles buttons. After adding a shinyFiles button the selected file(s) will be available in `input$inputId` (providing `input` is the name of the input object in the `shinyServer()` call). The file names should be parsed with `parseFilePaths` before usage though, to make them compliant with the `fileInput` function.

Usage

```
shinyFilesButton(id, label, title, multiple)
```

Arguments

`id`	The id matching the `shinyFileChoose`
`label`	The text that should appear on the button
`title`	The heading of the dialog box that appears when the button is pressed
`multiple`	A logical indicating whether or not it should be possible to select multiple files

Details

When a user selects one or several files the corresponding input variable is set to a list containing a character vector for each file. The character vectors gives the traversal route from the root to the selected file(s). The reason it does not give a path as a string is that the client has no knowledge of the file system on the server and can therefore not ensure proper formatting. As described above the input variable should be wrapped in a call to `parseFilePaths` for a more beautiful output.

For users wanting to design their html markup manually it is very easy to add a shinyFiles button. The only markup required is:

```
<button id="inputId" type="button" class="shinyFiles btn" data-title="title" data-selecttype="single"
```

where the id tag matches the inputId parameter, the data-title tag matches the title parameter, the data-selecttype is either "single" or "multiple" (the non-logical form of the multiple parameter) and the internal textnode mathces the label parameter.

Apart from this the html document should link to a script with the following path 'sF/shinyFiles.js' and a stylesheet with the following path 'sF/styles.css'.

The markup is bootstrap compliant so if the bootstrap css is used in the page the look will fit right in. There is nothing that hinders the developer from ignoring bootstrap altogether and designing the visuals themselves. The only caveat being that the glyphs used in the menu buttons are bundled with bootstrap. Use the css ::after pseudoclasses to add alternative content to these buttons. Additional filetype specific icons can be added with css using the following style:

.sF-file .sF-file-icon .yourFileExtension content: url(path/to/16x16/pixel/png);

.sF-fileList.sF-icons .sF-file .sF-file-icon .yourFileExtension content: url(path/to/32x32/pixel/png);

If no large version is specified the small version gets upscaled.

References

The file icons used in the file system navigator is taken from FatCows Farm-Fresh Web Icons (http://www.fatcow.com/free-icons)

See Also

Other shinyFiles: parseFilePaths; shinyFileChoose; shinyFilesExample

---

shinyFilesExample *Run a simple example app using the shinyFiles functionality*

Description

When the function is invoked a shiny app is started showing a very simple setup using shinyFiles. A button summons the dialog box allowing the user to navigate the R installation directory. To showcase the restrictions parameter the base package location has been hidden, and is thus inaccecible. A panel besides the button shows how the user selection is made accessible to the server after parsing with parseFilePaths.

Usage

```
shinyFilesExample()
```

See Also

Other shinyFiles: parseFilePaths; shinyFileChoose; shinyFilesButton

Index

fileGetter, *2*, 2, *4*
fileInput, *3*, *6*

getVolumes, 2

parseFilePaths, 3, *5–7*

shinyFileChoose, *2*, *4*, 4, *6*, *7*
shinyFilesButton, *4*, *5*, 6, *7*
shinyFilesExample, *4*, *5*, *7*, 7