Package 'shinyFiles'

April 30, 2019

Type Package

Title A Server-Side File System Viewer for Shiny

Version 0.7.3

Date 2019-4-29

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. Both file and folder selection as well as file saving is available.

License GPL (>= 2)

Encoding UTF-8

Imports htmltools, jsonlite, tools, shiny (>= 1.1.0), fs (>= 1.2.6), tibble (>= 1.4.2)

Collate 'aaa.R' 'filechoose.R' 'dirchoose.R' 'filesave.R' 'shinyFiles-package.R'

RoxygenNote 6.1.1

URL https://github.com/thomasp85/shinyFiles

BugReports https://github.com/thomasp85/shinyFiles/issues

NeedsCompilation no

Author Thomas Lin Pedersen [cre, aut],
Vincent Nijs [aut],
Thomas Schaffner [aut],
Eric Nantz [aut]

Repository CRAN

Date/Publication 2019-04-30 07:10:03 UTC

R topics documented:

`shinyFiles-package` *A Server-Side File System Viewer for Shiny*

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. Both file and folder selection as well as file saving is available.

Author(s)

Maintainer: Thomas Lin Pedersen <thomasp85@gmail.com>

Authors:

- Vincent Nijs
- Thomas Schaffner
- Eric Nantz

See Also

Useful links:

- https://github.com/thomasp85/shinyFiles
- Report bugs at https://github.com/thomasp85/shinyFiles/issues

dirCreator *Create a function that creates a new directory*

Description

This function returns a function that can be used to create new directories based on the information returned from the client. The returned function takes the name, path and root of the directory to create and parses it into a platform compliant format and then uses dir.create to create it. The returned function returns TRUE if the directory was created successfully and FALSE otherwise.

Usage

```
dirCreator(roots, ...)
```

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).
...	Currently unused

Value

A function that creates directories based on the information returned by the client.

dirGetter *Create a function that updates a folder tree based on the given restrictions*

Description

This functions returns a new function that will handle updating the folder tree. It is the folder equivalent of `fileGetter()` but functions slightly different as it needs to handle expanded branches of the folder hierarchy rather than just the content of a single directory at a time. The returned function takes a representation of a folder hierarchy along with the root to where it belongs and updates the tree to correspond with the current state of the file system, without altering expansions etc.

Usage

```
dirGetter(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	Currently unused
hidden	A logical value specifying whether hidden files should be returned or not

Value

A function taking a list representation of a folder hierarchy along with the name of the root where it starts. See `traverseDirs()` for a description of the format for the list representation.

---

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, pattern, 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
pattern	A regular expression used to select files to show. See `base::grepl()` for additional discussion on how to construct a regular expression (e.g., "log.*\.txt")
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

formatFiletype *Formats the value of the filetype argument*

Description

This function is intended to format the filetype argument of `shinySaveButton()` into a json string representation, so that it can be attached to the button.

Usage

```
formatFiletype(filetype)
```

Arguments

`filetype`	A named list of file extensions or NULL or NA

Value

A string describing the input value in json format

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

`shinyFiles-buttons` *Create a button to summon a shinyFiles dialog*

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 one of the shinyFiles dialog boxes. The last position in the file system is automatically remembered between instances, but not shared between several shinyFiles buttons. For a button to have any functionality it must have a matching observer on the server side. shinyFilesButton() is matched with shinyFileChoose() and shinyDirButton with shinyDirChoose(). The id argument of two matching calls must be the same. See `shinyFiles-observers()` on how to handle client input on the server side.

Usage

```
shinyFilesButton(id, label, title, multiple, buttonType = "default",
  class = NULL, icon = NULL, style = NULL)

shinyFilesLink(id, label, title, multiple, class = NULL, icon = NULL,
  style = NULL)

shinyDirButton(id, label, title, buttonType = "default", class = NULL,
  icon = NULL, style = NULL)

shinyDirLink(id, label, title, class = NULL, icon = NULL,
  style = NULL)

shinySaveButton(id, label, title, filename = "", filetype,
  buttonType = "default", class = NULL, icon = NULL, style = NULL)

shinySaveLink(id, label, title, filename = "", filetype, class = NULL,
  icon = NULL, style = NULL)
```

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
`buttonType`	The Bootstrap button markup used to colour the button. Defaults to 'default' for a neutral appearance but can be changed for another look. The value will be pasted with 'btn-' and added as class.

class	Additional classes added to the button
icon	An optional icon to appear on the button.
style	Additional styling added to the button (e.g., "margin-top: 25px;")
filename	A predefined filename to be filed in. Can be modified by the user during saving.
filetype	A named list of file extensions. The name of each element gives the name of the filetype and the content of the element the possible extensions e.g. `list(picture=c('jpg', 'jpeg'))`. The first extension will be used as default if it is not supplied by the user.

Details

Selecting files

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. The `parseFilePaths()` function can be used on the server to format the input variable into a format similar to that returned by `shiny::fileInput()`.

Selecting folders

When a folder is selected it will also be available in its respective input variable as a list giving the traversal route to the selected folder. To properly format it, feed it into `parseDirPath()` and a string with the full folder path will be returned.

Creating files (saving)

When a new filename is created it will become available in the respective input variable and can be formatted with `parseSavePath()` into a data.frame reminiscent that returned by fileInput. There is no size column and the type is only present if the filetype argument is used in `shinySaveButton`. In that case it will be the name of the chosen type (not the extension).

Manual markup

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

shinyFilesButton

```
<button id="inputId" type="button" class="shinyFiles btn btn-default" data-title="title" data-selectt
```

shinyDirButton

```
<button id="inputId" type="button" class="shinyDirectories btn-default" data-title="title">label</bu
```

shinySaveButton

```
<button id="inputId" type="button" class="shinySave btn-default" data-title="title" data-filetype="[
```

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 matches the label parameter. The data-filetype tag is a bit more involved as it is a json formatted array of objects with the properties 'name' and 'ext'. 'name' gives the name of the filetype as a string and 'ext' the allowed extensions as an array of strings. The non-exported `formatFiletype()` function can help convert from a named R list into the string representation. In the example above "btn-default" is used as button styling, but this can be changed to any other Bootstrap style.

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.

Client side events

If the shiny app uses custom Javascript it is possible to react to selections directly from the javascript. Once a selection has been made, the button will fire of the event 'selection' and pass the selection data along with the event. To listen for this event you simple add:

```
$(button).on('selection', function(event, path) {
  // Do something with the paths here
})
```

in the same way a 'cancel' event is fired when a user dismisses a selection box. In that case, no path is passed on.

Outside events the current selection is available as an object bound to the button and can be accessed at any time:

```
// For a shinyFilesButton
$(button).data('files')

// For a shinyDirButton
$(button).data('directory')

// For a shinySaveButton
$(button).data('file')
```

Value

This function is called for its side effects

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: shinyFiles-observers, shinyFiles-parsers, shinyFilesExample

---

shinyFiles-observers *Create a connection to the server side filesystem*

Description

These 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 one of the button generating functions or adding it manually. See shinyFiles-buttons() for more details.

Usage

```
shinyFileChoose(input, id, updateFreq = 0, session = getSession(),
  defaultRoot = NULL, defaultPath = "", ...)

shinyDirChoose(input, id, updateFreq = 0, session = getSession(),
  defaultPath = "", defaultRoot = NULL, ...)

shinyFileSave(input, id, updateFreq = 0, session = getSession(),
  defaultPath = "", defaultRoot = NULL, ...)
```

Arguments

input	The input object of the shinyServer() call (usually 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). For the default value (0) changes in the filesystem are shown only when a shinyFiles button is clicked again
session	The session object of the shinyServer call (usually session).
defaultRoot	The default root to use. For instance if roots = c('wd' = '.', 'home', '/home') then defaultRoot can be either 'wd' or 'home'.
defaultPath	The default relative path specified given the defaultRoot.
...	Arguments to be passed on to fileGetter() or dirGetter()

Details

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 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 or folder choice is made the resulting files/folder will be accessible in the input variable with the id given in the parameters. This value should probable be run through a call to one of the parser (`shinyFiles-parsers()`) 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.

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 doing 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.

See Also

Other shinyFiles: `shinyFiles-buttons`, `shinyFiles-parsers`, `shinyFilesExample`

Examples

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

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

## End(Not run)

## Not run:
# Folder selections
ui <- shinyUI(bootstrapPage(
  shinyDirButton('folder', 'Folder select', 'Please select a folder', FALSE)
))
```

```
server <- shinyServer(function(input, output) {
  shinyDirChoose(input, 'folder', roots=c(wd='.'), filetypes=c('', 'txt'))
})

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

## End(Not run)

## Not run:
# File selections
ui <- shinyUI(bootstrapPage(
  shinySaveButton('save', 'Save', 'Save as...')
))
server <- shinyServer(function(input, output) {
  shinyFileSave(input, 'save', roots=c(wd='.'))
})

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

## End(Not run)
```

`shinyFiles-parsers` *Convert the output of a selection to platform specific path(s)*

Description

This function takes the value of a shinyFiles button input variable and converts it to be easier to work with on the server side. In the case of file selections and saving the input variable is converted to a data frame (using `parseFilePaths()` or `parseSavePath() respectively`) of the same format as that provided by `shiny::fileInput()`. The only caveat here is that the MIME type cannot be inferred in file selections so this will always be an empty string and new files doesn't have a size so this is left out with file saving. In the case of folder selection the input variable is converted to a string (using `parseDirPath()`) giving the absolute path to the selected folder.

Usage

```
parseFilePaths(roots, selection)

parseDirPath(roots, selection)

parseSavePath(roots, selection)
```

Arguments

roots	The path to the root as specified in the shinyFileChoose() call in shinyServer()
selection	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 matching the format of shiny::fileInput()

See Also

Other shinyFiles: shinyFiles-buttons, shinyFiles-observers, shinyFilesExample

Examples

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

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

## End(Not run)
```

---

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 inaccessible. 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: shinyFiles-buttons, shinyFiles-observers, shinyFiles-parsers

---

traverseDirs *Traverse and update a tree representing the file system*

Description

This function takes a tree representing a part of the file system and updates it to reflect the current state of the file system as well as the settings for each node. Children (contained folders) are recursed into if the parents expanded element is set to TRUE, no matter if children are currently present.

Usage

```
traverseDirs(tree, root, restrictions, hidden)
```

Arguments

tree	A list representing the tree structure of the file system to traverse. Each element should at least contain the elements 'name' and 'expanded'. The elements 'empty' and 'children' will be created or updates if they exist.
root	A string with the location of the root folder for the tree
restrictions	A vector of directories within the root that should be filtered out of the results
hidden	A logical value specifying whether hidden folders should be returned or not

Value

A list of the same format as 'tree', but with updated values to reflect the current file system state.

`updateChildren` *Update the children element to reflect current state*

Description

This function create new entries for new folders and remove entries for no longer existing folders, while keeping the state of transient folders in the children element of the tree structure. The function does not recurse into the folders, but merely creates a shell that traverseDirs can take as input.

Usage

```
updateChildren(oldChildren, currentChildren)
```

Arguments

`oldChildren`	A list of children folders from the parent$children element of tree in `traverseDirs()`
`currentChildren`	A vector of names of the folders that are currently present in the parent of oldChildren

Value

An updated list equal in format to oldChildren

Index