Package ‘shinylight’

October 18, 2022

Title  Web Interface to 'R' Functions
Version  1.0.0
Date  2022-10-18
Description  Web front end for your 'R' functions producing plots or tables.
If you have a function or set of related functions, you can make them available over the internet through a web browser. This is the same motivation as the 'shiny' package, but note that the development of 'shinylight' is not in any way linked to that of 'shiny' (beyond the use of the 'httpuv' package). You might prefer 'shinylight' to 'shiny' if you want a lighter weight deployment with easier horizontal scaling, or if you want to develop your front end yourself in JavaScript and HTML just using a lightweight remote procedure call interface to your R code on the server.

Author  Pieter Vermeesch [aut],
        Tim Band [aut, cre]
Maintainer  Tim Band <t.band@ucl.ac.uk>
Depends  R (>= 3.0.0)
Imports  grDevices (>= 3.6.2), httpuv (>= 1.5.4), jsonlite (>= 1.6.1),
         later (>= 1.0)
Suggests  websocket (>= 1.4.1)
License  GPL-3
RoxygenNote  7.2.1
Encoding  UTF-8
NeedsCompilation  no
Repository  CRAN
Date/Publication  2022-10-18 14:10:02 UTC

R topics documented:

  browseTo .................................................. 3
  downloadCsv ............................................. 3

1
**browseTo**

*Opens a browser to look at the server*

**Description**

Opens a browser to look at the server

**Usage**

`browseTo(server)`

**Arguments**

- **server**: The server to browse to

**Value**

No return value

---

**downloadCsv**

*Encodes a data frame as a CSV file to be downloaded*

**Description**

Encodes a data frame as a CSV file to be downloaded

**Usage**

`downloadCsv(results)`

**Arguments**

- **results**: Data frame to be returned
encodePlot

Description
Renders a plot as a base64-encoded image

Usage
encodePlot(device, mimeType, width, height, plotFn)

Arguments
- device: Graphics device function, such as `grDevices::png` or `grDevices::pdf`
- mimeType: Mime type for the data produced by device
- width: Width of the plot in units applicable to device
- height: Height of the plot in units applicable to device
- plotFn: Function to call to perform the plot

Value
list with two keys, whose values can each be NULL: 'plot' is a plot in HTML img src form and 'data' is a data frame or other non-plot result.

Examples
```
pdf <- encodePlot(grDevices::png, "image/png", 200, 300, function() {
  barplot(c(1, 2, 3, 4))
})
grDevices::png() # workaround; you do not have to do this
```
encodePlotAs

Renders a plot as a base64-encoded PNG

Description
The result can be set as the src attribute of an <img> element in HTML.

Usage
encodePlotAs(format, plotFn)

Arguments
format An object specifying the output, with the following members: format$type is "png", "pdf" or "csv", and format$width and format$height are the dimensions of the PDF (in inches) or PNG (in pixels) if appropriate.
plotFn Function to call to perform the plot

Details
You will not need to call this function unless you want to return more than one plot per call, as the last plot produced will be returned in the plot property of the result from shinylight.call anyway.

Value
list with two keys, whose values can each be NULL: 'plot' is a plot in HTML img src form and 'data' is a data frame or other non-plot result.
A list with an element named plot containing the plot encoded as required either for an HTML image element's src attribute, or a element's href attribute. If the function returns a matrix or data frame, this will be returned in the list's data element.

See Also
rrpcServer

Examples
pdf <- encodePlotAs(list(type="pdf", width=7, height=8), function() {
  barplot(c(1, 2, 3, 4))
})
grDevices::png() # workaround; you do not have to do this
**JavaScript function:** Starts the Shinylight Framework, if you want to use it.

**Description**

The Shinylight Framework allows you to declare all your functions in R and have a nice-looking web front end for your code without having to write any JavaScript.

You should never need to call this function yourself; if you do not provide your own index.html, the default Shinylight one will be used that will call this function on page load.

Using the Shinylight Framework entails calling the `slServer` function with the `interface` argument set to `list(getSchema=schema)`, where `schema` is defined in the following section.

**Arguments**

- `options` (optional) An optional object containing options to modify the behaviour of the framework.
- `options.createFileInput` (optional) A function to create an element that uploads a file, as required for `toolkit.loadFileButton`.

**The Schema**

It is a list with the following members:

- `functions` a list of functions (keyed by their names), each of which is a list with the following members:
  - `params` a list of the main parameters the function accepts. The keys are the parameter names and the values are keys into the schema’s `params` list.
  - `optiongroups` a vector of keys into the schema’s `optiongroups` list giving other parameters to this function.
- `functiongroups` optional: the menu structure for the functions menu. Each item in the list is either a function name (a string referencing a key in the `functions` list) or a list representing a submenu. Submenu keys are the name to be displayed in the list, which can be overridden in the app.json file’s `functions` object, just like providing localized names for functions.
- `params` a list of the parameters the functions take, each of which is a list with the following members:
  - `type` either a key into the schema’s `types` list, giving the type of this parameter or the values it can take, or one of a set of standard types:
    - `'b'` Boolean
    - `'f'` Floating point
    - `'u8'` 8-bit unsigned integer
    - `'color'` Colour
'subheader' Vector of settings the user can choose for each column using selectors in the subheader row. This is usually used to select units (for example percent-by-weight versus parts-per-million) for the columns.

data a key into the schema’s data list, giving initial or example data for this parameter.
types a list of types with keys refered to from the schema’s types list’s type values. The values are a list with the following members:

kind Mandatory; one of:
  'enum' Enumeration type
  'column' A column from the input grid
values A vector of permitted values (only if kind='enum')
factors Only if kind='enum' and this enum is used as the unit type for some column; a vector of factors to multiply column data by if the unit is changed by the user. Must have the same number of elements as the values vector. For every n, factors[[n]] of unit values[[n]] must be equal. For example, if values=c('mm', 'cm', 'inch') then factors could be c(25.4, 2.54, 1.0).
subtype Only if kind='column'. The type of data that can be entered into the column. Currently only 'f' works well.
unittype Optional and only if kind='column'. The name of an enum type defining the units that the data in this column can be expressed in.

data A list of initial data with which table columns and controls will be populated. Can be a single value or vector (or list) as appropriate.

optiongroups A list of option groups. Each one is a set of parameters that can be added as a block to functions that want them. Each element is a list with the following keys:
type The same as for param’s type: either a key into the schema’s types list or one of the standard types ('b', 'u8', 'f' or 'color').
initial The initial value for this option.

There is one special key in the optiongroups list; this is the framework key. This is reserved for options that apply to the framework itself, not to any of your functions. So far, the only option it has is autorefresh=list(type="b", initial=FALSE). You can set its initial value to TRUE if you prefer. If you add this option, it controls whether the GUI has a "Calculate" button (FALSE) or whether the output should refresh a second or two after the user finishes changing parameters (TRUE).

Localization

To display human-friendly text on the controls and to get tooltip help text, you need one or more localization files. These files are named inst/www/locales/XX/app.json where XX is replaced with the appropriate ISO language code.

These files are JSON files containing an object with the following keys:
title Text for the link to put in the top left
homepage Destination for the link to put in the top left
functions One pair of translations for each function in the schema.
params One pair of translations for each parameter in the schema.
getOptionGroups

Each of the option groups in the schema gets a key which maps to an object which has the following keys:

- **@title**: A translation pair for the option group itself.
- **...**: One translation pair for each option in the group.

**types**

One object for each 'enum' type in the schema. Each value is an object with one key per possible enum value. Each value in this object is that enum value's translation pair.

A "translation pair" is an object with the following keys:

- **name**: A short name
- **help**: Tooltip text

**See Also**

toolkit.loadFileButton

---

**getAddress**

Obtains the address that the server is listening on

**Description**

Obtains the address that the server is listening on

**Usage**

getAddress(server)

**Arguments**

- **server**: The server (returned by slServer or slRunRServer)

**Value**

The HTTP address as protocol://address:port

**Examples**

```r
server <- slServer(
  port = 50051,
  interface = list(
    multiply = function(x, y) { x * y }
  )
)
address <- getAddress(server)
# ...
slStop(server)
stopifnot(address == "http://127.0.0.1:50051")
```
indexWithInit

Get index.html with (potentially) the JSON data in 'text' inserted.

Description
Get index.html with (potentially) the JSON data in 'text' inserted.

Usage
indexWithInit(text, path)

Arguments
- text: The text to insert as shinylight_initial_data
- path: File system path to the index.html file

Value
The updated text

rrpcServer
Makes and starts a server for serving R calculations

Description
It will serve files from the app directories specified by appDirs. If a file is requested that is not in one of those directories, the files in Shinylight's own inst/www directory will be served. Some paths have special meanings: / returns /index.html, /lang/ is redirected to /locales/<language-code>/ depending on the language selected in the request's Accept-Language header (that is, the browser's language setting) and the availability of the file requested. A POST request to /init with a data parameter will return /index.html, except that if the file has a line containing shinylight_initial_data = then this line will be replaced with a line initializing shinylight_initial_data to the data passed. This is used in shinylight-framework to permit linking to a framework app with specific data preloaded – the text should be as is downloaded with the "Save Data" button. Of course, this is available to non-framework apps, too.

Usage
rrpcServer(
    interface, 
    host = "0.0.0.0", 
    port = NULL, 
    appDirs = NULL, 
    root = "/", 
    initialize = NULL 
)
Arguments

interface  List of functions to be served. The names of the elements are the names that the client will use to call them.
host  Interface to listen on (default is ’0.0.0.0’, that is, all interfaces)
port  Port to listen on
appDirs  List of directories in which to find static files to serve
root  Root of the app on the server (with trailing slash)
initialize  A json string or list (that will be converted to a JSON string) to be passed to the JavaScript as initial data. For non-framework apps, the index.html must contain a line containing var shinylight_initial_data=, which will be replaced with code that sets shinylight_initial_data to this supplied JSON string.

Value

The server object, can be passed to s1Stop

runR  Returns a function that runs an R command

Description

If you set this as a part of your interface, like: runR=shinylight::runR(c("+", "plot", "c", "x", "y")) then you can call it from Javascript like this:

```javascript
rrpc.call("runR", {
  Rcommand: "2+2"
}, function(x) {console.log(x);});
rrpc.call("runR", {
  Rcommand: "y<-c(2,0,1):plot(c(1,2,3),y);y",
  'rrpc.resultformat': {
    type: 'png',
    width: 200,
    height: 300,
  }
}, function(x) {img.setAttribute('src', x.plot[0])});
```

Usage

runR(symbolList)

Arguments

symbolList  A list of permitted symbols in the R command
Value

A function that can be passed as one of the elements of `slServer`'s interface argument.

Examples

```r
sender <- slServer(
  port = 50050,
  interface = list(
    run_the_users_r_code = runR(
      list("c", ":", "list", "+", "-", "/", "+", "sqrt")
    )
  )
)
# ...
slStop(sender)
```

---

**sendInfoText**

Sends informational text to the client.

Description

During a slow remote procedure call, call this to inform the client of progress.

Usage

```r
sendInfoText(text)
```

Arguments

- **text**
  
  The text to send

Value

No return value

See Also

- `sendProgress` for sending a progress completion ratio to the user.

Examples

```r
server <- slServer(
  port = 50051,
  interface = list(long_and_complicated = function(x) {
    # First part of work that takes some time
    # ...
    sendInfoText("We are about half way through")
    # Second part of work that takes some time
    # ...
  })
)
```
sendProgress

Sends a progress update to the client.

Description
During a slow remote procedure call, call this to inform the client of progress.

Usage
sendProgress(numerator, denominator = 1)

Arguments
numerator The progress, out of denominator
denominator What the progress is out of. You could use this for the number of known items to be completed so that each call increases either the numerator (for more items done) and/or the denominator (for more items discovered that need to be done). However, it is not necessary to be so precise; you can set the numerator and denominator however you like on each call as long as it makes sense to the user.

Value
No return value

See Also
sendInfoText for sending text to the user.

Examples
server <- slServer(
  port = 50051,
  interface = list(long_and_complicated = function(x) {
    sendProgress(0,3)
    # First part of work that takes some time
    # ...
    sendProgress(1,3)
    # Second part of work that takes some time
    # ...
    sendProgress(2,3)
    # Last part of work that takes some time
    # ...
    sendProgress(3,3)
  })
)
shinylight.call  

JavaScript function

Description

Calls a server function as defined in the server’s call to the slServer function.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn</td>
<td>string</td>
<td>The name of the R function to call.</td>
</tr>
<tr>
<td>data</td>
<td>object</td>
<td>An object whose keys are the arguments to the function being called.</td>
</tr>
<tr>
<td>plotElement</td>
<td>string, HTMLElement</td>
<td>If provided, the &lt;img&gt; element (or id of the element) that will receive the plot output (if any). The plot returned will be the size that this element already has, so ensure that it is styled in a way that it has the correct size even if no image (or an old image) has been set.</td>
</tr>
<tr>
<td>extra</td>
<td>object [optional]</td>
<td>An object whose keys can be: &quot;imgType&quot;: Type of image required, &quot;png&quot; (default) or &quot;svg&quot;; &quot;info&quot;: Function to be called if the R function sendInfoText is called; &quot;progress&quot;: Function to be called if the R function sendProgress is called.</td>
</tr>
</tbody>
</table>

Value

Result object that might have a plot property (giving a string that would work as the src attribute of an img element, representing graphics drawn by the command), a data property (giving the value returned by the command) and a headers property (giving the column names in the data returned if any). If the promise resolves to an error, the argument to the error function is a string representing the cause of the error.

shinylight.initialize  

JavaScript function

Description

Call this before calling any other ShinyLight function. Returns a promise that resolves (to nothing) when the connection is ready.
shinylight.makeTable  JavaScript function

Description

Turns data received from R into a form that can be set into dataentrygrid.js.

Arguments

data       object  Data as returned from R
extraColumns Array.<string>, number The extra column headers required or the number of extra columns required.

Value

Headers and rows

Examples

```r
## Not run:
t = shinylight.makeTable(data);
grid.init(t.headers, t.rows);

## End(Not run)
```

shinylight.passToOther  JavaScript function

Description

Open another tab with another (possibly remote from this one) instance of shinylight, initializing it with our own data.

Arguments

url       string  The URL of the other shinylight instance
data      any  The JSON to send. If a string is passed, this is assumed to be JSON and sent as-is. Otherwise it is stringified into JSON before being sent.
shinylight.runR

*JavaScript function: Runs an R function.*

**Description**

The R side must be running the slRunRServer function.

**Arguments**

- **rCommand** string The R text to run. It can plot a graph and/or return some R data structure (such as a data frame).
- **data** any A javascript value that will be translated to the R command as a value also called 'data'.
- **plotElement** string, HTMLElement If provided, the `<img>` element (or id of the element) that will receive the plot output (if any). The plot returned will be the size that this element already has, so ensure that it is styled in a way that it has the correct size even if no image (or an old image) has been set.
- **extra** object [optional] An object whose keys can be: "imgType": Type of image required, "png" (default) or "svg"; "info": Function to be called if the R function sendInfoText is called; "progress": Function to be called if the R function sendProgress is called.

**Value**

Result object that might have a plot property (giving a string that would work as the src attribute of an img element, representing graphics drawn by the command) and a data property (giving the value returned by the command). If the promise resolves to an error, the argument to the error function is a string representing the cause of the error.

shinylight.setElementJson

*JavaScript function*

**Description**

Sets the text content of an element (or its value as appropriate) to the JSON representation of an object.

**Arguments**

- **elementOrId** string, HTMLElement The element (or its id) that will have its text set
- **object** any The object whose JSON representation will be set as the text content of the element
shinylight.setElementPlot

*JavaScript function:* Sets an `<img>` element to display a plot returned by `runR`.

**Description**

Normally you do not need to call this because to get shinylight to produce a plot you need to set the `plotElement` argument, and doing so will cause this element to receive the plot automatically.

**Arguments**

- `elementOrId` *string, HTMLImageElement* The `<img>` element (or its id) that will receive the image.
- `result` *object* The result from `runR`.

shinylight.setElementText

*JavaScript function*

**Description**

Sets the text content of an element (or its value as appropriate).

**Arguments**

- `elementOrId` *string, HTMLElement* The element (or its id) that will have its text set
- `text` *string* The text to set into the element

shinylight.setGridResult

*JavaScript function*

**Description**

Sets a `dataentrygrid` object to the result of `runR`, if appropriate.

**Arguments**

- `grid` *DataEntryGrid* Table that receives the result
- `result` *object* Return value promised by `runR`
Description

Sets a dataentrygrid object to the result of `runR`. The object will have fixed rows, with names derived from the row names in the original data frame.

Arguments

- `grid` : DataEntryGrid Table that receives the result
- `result` : object Return value promised by `runR`

slRunRServer

Start a ShinyLight server which runs R that it is sent

Description

Start a ShinyLight server which runs R that it is sent

Usage

```r
slRunRServer(
  permittedSymbols,
  appDir = NULL,
  host = "127.0.0.1",
  port = NULL,
  daemonize = FALSE,
  initialize = NULL
)
```

Arguments

- `permittedSymbols` : List of symbols that are permitted in the R commands passed. Remember to include `data`, `$` and `<-`.
- `appDir` : Directory containing files to serve (for example `system.file("www", package = "your-package")`)
- `host` : IP address to listen on, default is "127.0.0.1" (localhost). Use "0.0.0.0" to run in a docker container.
- `port` : Internet port of the virtual server. If not defined, a random free port will be chosen and the browser will be opened to show the GUI.
daemonize

If TRUE, keep serving forever without returning. This is useful when called from Rscript, to keep

initialize

A json string or list (that will be converted to a JSON string) to be passed to the JavaScript as initial data. The index.html must contain a line containing var shinylight_initial_data=, which will be replaced with code that sets shinylight_initial_data to this supplied JSON string.

Value

server object, unless daemonize is TRUE.

See Also

slServer for the more general form of this function, or slStop to stop a running server. shinylight.runR is the JavaScript function you need to call to pass R code from the browser to the server.

Examples

server <- slRunRServer(  
  permitted = list("*"),  
  port = 50053  
)

# Normally we would use shinylight.js to send the function over  
# and receive the result, not R and websocket.  
ws <- websocket::WebSocket$new("ws://127.0.0.1:50053/x")  
resultdata <- NULL  
ws$onMessage(function(event) {    
  resultdata <<- jsonlite::fromJSON(event$data)$result$data
})  
ws$onOpen(function(event) {    
  ws$send("/quotesingle.Var"  
    "method":"runR","params":{"Rcommand":"3 * 57"}})  
})  
timeout = 30  
while(is.null(resultdata) && 0 < timeout) {  
  later::run_now()  
  Sys.sleep(0.1)  
  timeout <- timeout - 1
}  
ws$close()  
slStop(server)  
stopifnot(resultdata == 171)  
# 3 * 57 == 171  
grDevices::png()  
# workaround; you do not have to do this
Usage

```r
slServer(
    interface,
    appDir = NULL,
    host = "127.0.0.1",
    port = NULL,
    daemonize = FALSE,
    initialize = NULL
)
```

Arguments

- **interface**: List of functions you want to be able to call from the browser. If you want to use the Shinylight Framework, this should have one member `getSchema`. For details of this, see the documentation for `shinylightFrameworkStart`.
- **appDir**: Directory containing files to serve (for example `system.file("www", package = "your-package")`)
- **host**: IP address to listen on, default is "127.0.0.1" (localhost). Use "0.0.0.0" to run in a docker container.
- **port**: Internet port of the virtual server. If not defined, a random free port will be chosen and the browser will be opened to show the GUI.
- **daemonize**: If TRUE, keep serving forever without returning. This is useful when called from RScript, to keep
- **initialize**: A json string or list (that will be converted to a JSON string) to be passed to the JavaScript as initial data. For non-framework apps, the index.html must contain a line containing `var shinylight_initial_data=`, which will be replaced with code that sets `shinylight_initial_data` to this supplied JSON string.

Value

server object, unless daemonize is TRUE in which case the function will not return.

See Also

- `slStop` to stop a running server, and `slRunRServer` to run a server that just accepts R code.

Examples

```r
# You can leave out port and daemonize to launch a browser
# pointing at your server
server <- slServer(
    port = 50052,
    interface = list(
        multiply = function(x, y) { x * y }  
    )
)
# Normally we would use shinylight.js to send the function over
# and receive the result, not R and websocket.
```
ws <- websocket::WebSocket$new("ws://127.0.0.1:50052/x")
resultdata <- NULL
ws$onMessage(function(event) {
  resultdata <<- jsonlite::fromJSON(event$data)$result$data
})
ws$onOpen(function(event) {
  ws$send("{ "method": "multiply", "params": { "x": 3, "y": 47 } }"
})
timeout = 30
while(is.null(resultdata) && 0 < timeout) {
  later::run_now()
  Sys.sleep(0.1)
  timeout <- timeout - 1
}
ws$close()
s1Stop(server)
stopifnot(resultdata == 141) # multiply(3, 47) == 141
grDevices::png() # workaround; you do not have to do this

---

slStop

*Stops a ShinyLight GUI*

**Description**

Stops a ShinyLight GUI

**Usage**

s1Stop(server = NULL)

**Arguments**

- **server**
  
  The server (returned by s1Server or s1RunRServer) to stop. If not supplied all servers will be stopped.

**Value**

No return value

**Examples**

server <- s1Server(
  port = 50051, # leave this out if you don't care about the port number
  interface = list(
    multiply = function(x, y) { x * y }
  )
)
# ...
s1Stop(server)
toolkit.all

JavaScript function: Finds if a predicate is true for all members of an array or object.

Description
Calls a function for each member of an array or object until either one of them returns false (in which case all returns false) or we run out of elements (in which case all returns true).

Arguments
- a: object Object or array to be iterated through.
- p: function Function to call with two arguments: the key of the element (or index in the case of an array) and the value; should return a boolean.

---
toolkit.any

JavaScript function: Finds if a predicate is true for any member of an array or object.

Description
Calls a function for each member of an array or object until either one of them returns true (in which case any returns true) or we run out of elements (in which case any returns false).

Arguments
- a: object Object or array to be iterated through.
- p: function Function to call with two arguments: the key of the element (or index in the case of an array) and the value; should return a boolean.

---
toolkit.banner

JavaScript function

Description
Returns a Container Element for displaying controls horizontally.

Arguments
- elements: Array.<HTMLControlElement> Initial array of elements to be added.
- className: string HTML class for the returned banner.

Value
The banner element.
**toolkit.button**  
*JavaScript function: Returns a button.*

**Description**
This button is an HTML element, but it is not an HTML button. Styling and JavaScript provide the button-like look-and-feel.

**Arguments**
- `id`  
  string The HTML id of the button will be `button-` + `id`. It is also used in the interpretation of the translations argument.
- `fn`  
  function Unary function that takes a single parameter of a nullary function. This function will be called on completion of the work (which will be used to remove the button’s ‘click’ animation). If the function want to use as a call-back does not take an argument, you can wrap it in `toolkit.withTimeout`. You might also want to use `toolkit.withTimeout` if your function returns too quickly, otherwise the user might not see the button click.
- `translations`  
  object An object with a key `id` having a value that is an object having a key `name` with value the display name of the button, and optionally a key ‘help’ with value of the tooltip text.

**Value**
The button.

---

**toolkit.deref**  
*JavaScript function: Dereferences an object or array through multiple indices.*

**Description**
`deref(o, [a,b,c], d)` is a safe way of doing `o[a][b][c]`. If that path does not exist, `d` is returned. If `d` is not supplied, null is returned. Any undefined values in path are ignored.

**Arguments**
- `object`  
  object The object to be dereferenced.
- `path`  
  Array The series of indices to be applied.
- `defaultValue`  
  `toolkit.any` The default value to be returned if the path cannot be followed to the end.

**Value**
Object dereferenced, `defaultValue`, or null.
JavaScript function: A panel with a smaller footer.

**Description**

Returns a Positioned Element consisting of a body and a footer.

**Arguments**

- `ftr` HTML Element The footer element.
- `main` toolkit.HTMLPositionedElement The body element.

**Value**

The element containing the footer and body.

---

JavaScript function

**Description**

Calls a function for each member of an array or object.

**Arguments**

- `a` object Object or array to be iterated through.
- `f` function Function to call with two arguments: the key of the element (or index in the case of an array) and the value.

---

JavaScript function: Option group title

**Description**

Adds a group title to an toolkit.optionsPage.

**Arguments**

- `container` HTML Element The container, preferably the return value from toolkit.optionsPage.
- `labelTranslations` object An object with two keys: 'name' is the display text for this title, 'help' (optional) is the tooltip text.
toolkit.header

*JavaScript function: A panel with a smaller header.*

**Description**

Returns a Positioned Element consisting of a header and a body.

**Arguments**

- `hdr` [HTMLElement] The header element.

**Value**

The element containing the header and body.

---

toolkit.HTMLContainerElement

*JavaScript class: A monkey-patched HTMLElement.*

**Description**

A Container Element is an element for displaying a set of controls and their labels.

**Properties**

- `makeSubElement` [function] Gets an element in which a control and its label can be stored. You do not need to call this unless you have made your own custom control; it will be called by functions such as `toolkit.paramText`. Pass in the ID of the control (you will need the ID for the `getData` and `setData` calls).

- `getData` [function] Returns an object mapping contained controls (or nested containers) to their current values.

- `setData` [function] Sets the values of the contained controls. `data` is a mapping from the IDs of the contained controls to the data that should be set on them.

**See Also**

- `toolkit.stack`
- `toolkit.banner`
- `toolkit.optionsPage`
toolkit.HTMLControlContainerElement

JavaScript class

Description
A container for a single control.

Properties

addElement  function Adds an element. Should be called once with a control’s label, and then again with the control itself.

See Also

toolkit.HTMLContainerElement

---

toolkit.HTMLControlElement

JavaScript class

Description
A monkey-patched HTMLElement representing a control with its label.

Properties

getData  function Returns the current displayed value.

setData  function Sets the value.

hide  function Makes the element invisible and non-interactive

show  function makes the element visible and (potentially) interactive

See Also

toolkit.paramBoolean
toolkit.paramColor
toolkit.paramFloat
toolkit.paramInteger
toolkit.paramSelector
toolkit.paramText
toolkit.HTMLPositionedElement

*JavaScript class: A monkey-patched HTMLElement with some extra methods.*

**Description**

Certain elements returned by Toolkit methods are Positioned Elements. It is necessary for elements in some places in the document to be Positioned Elements for the document resizing and formatting to work.

If you have an HTML element that is not a Positioned Element that you want to add to a place where only Positioned Elements are required, wrap it in `toolkit.scrollingWrapper` or `toolkit.nonScrollingWrapper`.

**Properties**

- `setSize` function Sets the position of the element on the document in pixels, with parameters for left, top, width and height in that order.
- `getSize` function Returns an object with members `left`, `top`, `width` and `height` for the position of the element.
- `hide` function Makes the element invisible and non-interactive
- `show` function makes the element visible and (potentially) interactive

**toolkit.image**

*JavaScript function*

**Description**

An image element.

**Arguments**

- `updateSizeFunction` function Nullary function called when the object's size is changed.

**Value**

Image element. It has a `getSize()` method, returning an object with width and height members. This is the width and height set by `reposition()`, not the actual on-screen width and height, if that is different for some reason. In other words, it returns the width and height the image "should" have.
JavaScript function: A panel with a side bar.

Description

Returns a Positioned Element consisting of a left side bar and a body.

Arguments

- **bar**: HTMLElement The side bar element.
- **main**: toolkit.HTMLPositionedElement The body element.

Value

The Toolkit Positioned Element containing the side bar and body.

toolkit.loadFileButton

JavaScript function: Returns a button that uploads a file from the client.

Description

This button is an HTML element, but it is not an HTML button. Styling and JavaScript provide the button-like look-and-feel.

Arguments

- **id**: string The HTML id of the button will be 'button-' + id. It is also used in the interpretation of the translations argument.
- **fn**: function A binary callback function. Its two parameters are the File object uploaded and a (nullary) function that will be called when the operation completes.
- **translations**: object An object with a key id having a value that is an object having a key 'name' with value the display name of the button, and optionally a key 'help' with value of the tooltip text.
- **createFileInput**: function [optional] A function to create an element that uploads a file. By default this is a normal <input type="file"> with an extra show member function that does nothing. The function takes two parameters: uploadFn and doneFn. uploadFn must be called when a file has been chosen for upload; it takes two parameters: a File object and a callback function that is called on completion. You should either pass doneFn as this second parameter, or a function that performs some actions then calls doneFn() itself. The return value of createFileInput should be the element itself, monkey-patched to include a show() method that will be called when the Load button is clicked.
toolkit.makeLabel  
*JavaScript function: Makes a label suitable for labelling a control.*

**Description**

The label has translatable text and a help tooltip (if translated for).

**Arguments**

- `translations` object: translations[id].name is the string to use as label's text, translations[id].help is the string to use as the label's tooltip. If id is undefined or null, translations.name and translations.help are used.
- `container` toolkit.HTMLControlContainerElement [optional] Where to put the label.
- `id` string [optional] Where to look in translations for the text.
- `idFor` string [optional] The id attribute of the HTML element that this element refers to.

**Value**

The label.

---

toolkit.nonScrollingWrapper  
*JavaScript function: Returns a Positioned Element just containing one element.*

**Description**

This element does not gain scrollbars if it is too large for this returned container, and it will try to take up its full size in the layout.

**Arguments**

- `element` HTMLElement The element to be wrapped
- `verticalPadding` int The number of extra pixels above the element's height to use as the returned element's default height.
- `horizontalPadding` int The number of extra pixels above the element's width to use as the returned element's default width.

**Value**

The wrapper.
toolkit.optionsPage

JavaScript function: Returns a Container Element for displaying controls vertically.

Description

Returns an element with a makeSubElement method that adds elements vertically. This differs from toolkit.stack in that the labels will be aligned on the left and the controls will be aligned on the right. It would make a nice options page, for example.

Value

A Container Element for displaying elements vertically.

toolkit.overlay

JavaScript function: A panel with an overlay.

Description

Returns a Positioned Element consisting of two elements placed in the same position. To be able to see the lower (main) element you must either call hide() on the overlay, or make it transparent with CSS.

Arguments

- overlay HTMLElement The higher element. Any getData() or setData() call on the returned element will not be passed on to this overlay element.
- main toolkit.HTMLPositionedElement The lower element.

Value

The element containing both elements.
### toolkit.pages

*JavaScript function: Returns a Positioned Element for displaying controls in tabbed pages.*

**Description**

Only one page will be visible at a time. The returned element has `getData` and `setData` methods that take or return (respectively) an object with keys that are the IDs of the pages.

**Arguments**

- **pageElements**
  
  object dictionary of pageIds to elements (that will be added to the return value of this function). These elements each need methods `show`, `hide` and `setData` (like the ones returned by `toolkit.header`, `toolkit.scrollingWrapper`, `toolkit.nonScrollingWrapper`, `toolkit.leftSideBar`, that is to say, Positioned Elements) if they are to be output pages. Only show and hide if they are to be available permanently and not be set through the setData call.

- **labelTranslations**
  
  object dictionary of pageIds to objects with keys `name` (for the label text) and `help` (for tooltip help HTML)

- **tabIdPrefix**
  
  string If you want HTML IDs for your tab elements, set this and the ID will be set to `tabIdPrefix + pageId`.

**Value**

An element that has the tabs and the tabs that switch between them. The active tab has the "active" class. It has the following extra methods: `setData(data)`: data is a dictionary with keys matching the pageIds. The values are passed to the `setData()` functions of the corresponding elements. Pages without any data (and their corresponding radio buttons) are summarily disabled. Pages with data are enabled. `reposition()`: sets each page to the same dimensions as the container and calls each page’s `reposition()` method (if it exists).

### toolkit.paramBoolean

*JavaScript function: Returns a checkbox input Toolkit Control.*

**Description**

A control for a boolean value rendered as a checkbox.
toolkit.paramColor

Arguments

id string when getData or setData is called on the container, the value at 'id' refers to this selector. The HTML id is set to 'param-' + id.

container toolkit.HTMLContainerElement [optional] Where to put the control.

translations object Optional mapping: translations.id is the name of the control to be displayed and translations.help is help text to be displayed if the user hovers over the label

initial string Optional initial value for the control

callback function Optional function to be called whenever the input value changes

Value

Checkbox input control.

---

JavaScript function: Returns a colour input Toolkit Control.

Description

It is a standard HTML input control with type color. The value returned is a six-hex-digit string prefixed with a #.

Arguments

id string when getData or setData is called on the container, the value at 'id' refers to this selector. The HTML id is set to 'param-' + id.

container toolkit.HTMLContainerElement [optional] Where to put the control.

translations object Optional mapping: translations.id is the name of the control to be displayed and translations.help is help text to be displayed if the user hovers over the label

initial string Optional initial value for the control

callback function Optional function to be called whenever the input value changes

Value

Text input control.
### toolkit.paramFloat

*JavaScript function: Returns a floating point input Toolkit Control.*

**Description**

Values outside the permitted range will gain the "invalid" class, but there is no other effect.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string when <code>getData</code> or <code>setData</code> is called on the container, the value at 'id' refers to this selector. The HTML id is set to 'param-' + id.</td>
</tr>
<tr>
<td>container</td>
<td><code>toolkit.HTMLContainerElement</code> [optional] Where to put the control.</td>
</tr>
<tr>
<td>translations</td>
<td>object Optional mapping: translations.id is the name of the control to be displayed and translations.help is help text to be displayed if the user hovers over the label</td>
</tr>
<tr>
<td>initial</td>
<td>string Optional initial value for the control</td>
</tr>
<tr>
<td>callback</td>
<td>function Optional function to be called whenever the input value changes</td>
</tr>
<tr>
<td>min</td>
<td>float Minimum permitted value (optional).</td>
</tr>
<tr>
<td>max</td>
<td>float Maximum permitted value (optional).</td>
</tr>
</tbody>
</table>

**Value**

Text input control.

### toolkit.paramInteger

*JavaScript function: Returns an integer input Toolkit Control.*

**Description**

Values outside the permitted range will gain the "invalid" class, but there is no other effect.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string when <code>getData</code> or <code>setData</code> is called on the container, the value at 'id' refers to this selector. The HTML id is set to 'param-' + id.</td>
</tr>
<tr>
<td>container</td>
<td><code>toolkit.HTMLContainerElement</code> [optional] Where to put the control.</td>
</tr>
<tr>
<td>translations</td>
<td>object Optional mapping: translations.id is the name of the control to be displayed and translations.help is help text to be displayed if the user hovers over the label</td>
</tr>
<tr>
<td>initial</td>
<td>string Optional initial value for the control</td>
</tr>
<tr>
<td>callback</td>
<td>function Optional function to be called whenever the input value changes</td>
</tr>
<tr>
<td>min</td>
<td>int Minimum permitted value (optional).</td>
</tr>
<tr>
<td>max</td>
<td>int Maximum permitted value (optional).</td>
</tr>
</tbody>
</table>
Value

Text input control.

---

**toolkit.paramSelector**

*JavaScript function: Returns a custom selection box Toolkit Control.*

---

**Description**

This is different to a normal selection box because it allows tooltips on the items within the list.

**Arguments**

- **id** string: When `getData` or `setData` is called on the container, the value at `id` refers to this selector. The HTML id is set to `param-` + id.
- **container** `toolkit.HTMLContainerElement` (optional): Where to put the control. The container came from `optionsPage()` the new selection box will be formatted as a table row.
- **labelTranslations** object: A dictionary with two optional keys; 'name' gives the label to display and 'help' gives HTML help text. 'help' has no effect unless 'name' is also present.
- **values** `Array.<int>`: An array of the IDs of the options in the selection.
- **valueTranslations** object: A dictionary whose keys are the IDs of the options in the selection, the values are more dictionaries. These dictionaries have two optional keys; 'name' (giving the name to display for this option) and 'help' (giving tooltip HTML text).
- **initial** string: ID of the option to start selecting (optional)
- **callback** function: The (nullary) function to call when the value changes (optional)

**Value**

The selection box.
toolkit.preformattedText

JavaScript function: Returns a text input Toolkit Control.

Description
Any text is permitted unless a validate function is supplied.

Arguments

- id: string when getData or setData is called on the container, the value at 'id' refers to this selector. The HTML id is set to 'param-' + id.
- translations: object Optional mapping: translations.id is the name of the control to be displayed and translations.help is help text to be displayed if the user hovers over the label
- initial: string Optional initial value for the control
- callback: function Optional function to be called whenever the input value changes
- validate: function Optional function returning true if passed a value that this control should accept or false otherwise.

Value

Text input control.

toolkit.preformattedText

JavaScript function: A static text Toolkit Control in a preformatted style.

Description
This element is like a control in that it has a label and actual text content, but it is not interactive.

Arguments

- id: string The ID of this control within the container
- translations: object An object with keys 'name' for the label displayed by the text and 'help' for tooltip text.

Value

The static text element. The text content can be set by calling its setData() function with any plain text.
toolkit.progressBar  
*JavaScript function: Returns a Positioned Element progress bar.*

**Description**

The progress is set by calling the `setData()` method.

**Value**

The progress bar element.

---

toolkit.rightSideBar  
*JavaScript function: A panel with a side bar.*

**Description**

Returns a Positioned Element consisting of a right side bar and a body.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar</td>
<td>HTMLElement</td>
<td>The side bar element.</td>
</tr>
<tr>
<td>main</td>
<td>toolkit.HTMLPositionedElement</td>
<td>The body element.</td>
</tr>
</tbody>
</table>

**Value**

The Toolkit Positioned Element containing the side bar and body.

---

toolkit.scrollingWrapper  
*JavaScript function: Returns a Positioned Element just containing one element.*

**Description**

This element gains scrollbars if it is too large for this returned container.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>element</td>
<td>HTMLElement</td>
<td>The element to be wrapped</td>
</tr>
<tr>
<td>verticalPadding</td>
<td>int</td>
<td>The number of extra pixels above the element’s height to use as the returned element’s default height.</td>
</tr>
<tr>
<td>horizontalPadding</td>
<td>int</td>
<td>The number of extra pixels above the element’s width to use as the returned element’s default width.</td>
</tr>
</tbody>
</table>
Value

The wrapper.

---

**toolkit.setAsBody**  
*JavaScript function: Replaces the `<main>` tag in the document with this element.*

**Description**

The element will have its resize event wired up. If `el` is a Toolkit Positioned Element, it will be resized correctly when the window is resized.

**Arguments**

`el`  
`HTMLElement` The element to set as `<main>`

---

**toolkit.stack**  
*JavaScript function: Returns a Container Element for displaying controls vertically.*

**Description**

Returns a Container Element with a `makeSubElement` method that adds elements vertically, with the labels above the controls they correspond to.

**Arguments**

`elements`  
`Array.<HTMLElement>` Initial array of elements to be added.

**Value**

A Container Element for displaying elements vertically.
**toolkit.staticText**  
*JavaScript function: A static text Toolkit Control.*

**Description**

This element is like a control in that it has a label and actual text content, but it is not interactive.

**Arguments**

- **id** string The ID of this control within the container
- **container** toolkit.HTMLContainerElement [optional] Where to put the control.
- **translations** object An object with keys 'name' for the label displayed by the text and 'help' for tooltip text.

**Value**

The static text element. The text content can be set by calling its `setData()` function. This text can include HTML entities, so you might want to replace `&` with `&` and `<` with `<` if it is plain text.

**toolkit.verticalDivide**  
*JavaScript function: Left/right panels with a draggable divider.*

**Description**

Returns a Positioned Element with a draggable vertical divider bordering two other Positioned Elements.

**Arguments**

- **container** toolkit.HTMLPositionedElement The container to divide. If null, a container will be created for you.
- **left** toolkit.HTMLPositionedElement The element to put on the left of the divider.
- **right** toolkit.HTMLPositionedElement The element to put on the right of the divider.

**Value**

The element created. If a container was provided it is this argument.
toolkit.withTimeout

JavaScript function: Adds a fake callback argument to a nullary function.

Description

Perhaps you have a nullary function that you want called when the user clicks a button, but the toolkit.button function wants a unary function that has a completion callback so that the button knows when to pop back up again. In this situation you might wrap your function with a call to toolkit.withTimeout.

Arguments

fn function Nullary function to wrap.

Value

Unary function (taking one function as an argument) that simply calls fn immediately then calls its argument again after 200ms.
Index

browseTo, 3

downloadCsv, 3

encodePlot, 4
encodePlotAs, 5

framework.shinylightFrameworkStart, 6

getAddress, 8
grDevices::pdf, 4
grDevices::png, 4

indexWithInit, 9

rrpcServer, 5, 9
runR, 10, 16, 17

sendInfoText, 11, 12, 13, 15
sendProgress, 11, 12, 13, 15
shinylight.call, 13
shinylight.initialize, 13
shinylight.makeTable, 14
shinylight.passToOther, 14
shinylight.runR, 15, 18
shinylight.setElementJson, 15
shinylight.setElementPlot, 16
shinylight.setElementText, 16
shinylight.setGridResult, 16
shinylight.setGridResultWithNamedRows, 17
slRunRServer, 8, 17, 19, 20
slServer, 6, 8, 11, 18, 18, 20
slStop, 10, 18, 19, 20

toolkit.all, 21
toolkit.any, 21, 22
toolkit.banner, 21, 24
toolkit.button, 22, 38
toolkit.deref, 22
toolkit.footer, 23

toolkit.forEach, 23
toolkit.groupTitle, 23
toolkit.header, 24, 30
toolkit.HTMLContainerElement, 24, 25, 31–34, 37
toolkit.HTMLControlContainerElement, 25, 28
toolkit.HTMLControlElement, 25
toolkit.HTMLPositionedElement, 23, 24, 26, 27, 29, 35, 37
toolkit.image, 26
toolkit.leftSideBar, 27, 30
toolkit.loadFileButton, 27
toolkit.makeLabel, 28
toolkit.nonScrollingWrapper, 26, 28, 30
toolkit.optionsPage, 23, 24, 29
toolkit.overlay, 29
toolkit.pages, 30
toolkit.paramBoolean, 25, 30
toolkit.paramColor, 25, 31
toolkit.paramFloat, 25, 32
toolkit.paramInteger, 25, 32
toolkit.paramSelector, 25, 33
toolkit.paramText, 24, 25, 34
toolkit.preformattedText, 34
toolkit.progressBar, 35
toolkit.rightSideBar, 35
toolkit.scrollingWrapper, 26, 30, 35
toolkit.setAsBody, 36
toolkit.stack, 24, 29, 36
toolkit.staticText, 37
toolkit.verticalDivide, 37
toolkit.whenQuiet, 38
toolkit.withTimeout, 22, 38, 38