

Package ‘shinyAce’

May 9, 2026

Type Package

Title Ace Editor Bindings for Shiny

Version 0.4.4

Date 2025-2-2

Description Ace editor bindings to enable a rich text editing environment within Shiny.

License MIT + file LICENSE

Depends R (>= 3.3.0)

Imports shiny (>= 1.0.5), jsonlite, utils, tools

Suggests testthat (>= 2.0.0), dplyr (>= 0.8.3)

BugReports <https://github.com/trestletech/shinyAce/issues>

Encoding UTF-8

RoxygenNote 7.3.2

Language en-US

NeedsCompilation no

Author Vincent Nijs [aut, cre],
Forest Fang [aut],
Trestle Technology, LLC [aut],
Jeff Allen [aut],
Institut de Radioprotection et de Surete Nucleaire [cph],
Ajax.org B.V. [ctb, cph] (Ace)

Maintainer Vincent Nijs <radiant@rady.ucsd.edu>

Repository CRAN

Date/Publication 2025-02-03 00:20:02 UTC

Contents

.fname_regex	2
.tools	3
.utils	3

aceAnnotate	4
aceAutocomplete	5
aceEditor	6
aceTooltip	9
build_tooltip_fields	10
getAceModes	10
getAceThemes	11
get_arg_help	11
get_desc_help	12
get_help_file	12
get_usage_help	13
is.empty	13
meta_obj	14
meta_pkg	14
rd_2_html	14
re_capture	15
r_completions_function_call_metadata	15
r_completions_general_metadata	16
r_completions_metadata	16
shinyAce-options	16
shinyAce_debug	17
tooltip_html	17
updateAceEditor	18
Index	20

.fname_regex	<i>Regular expression for matching the function name in a completion line in the middle of a function call</i>
--------------	--

Description

Regular expression for matching the function name in a completion line in the middle of a function call

Usage

.fname_regex

Format

An object of class character of length 1.

.tools *Get namespace to get access to unexported functions, namely RdTags*

Description

Get namespace to get access to unexported functions, namely RdTags

Usage

.tools

Format

An object of class environment of length 792.

.utils *Get namespace to get access to unexported functions, namely .getHelpFile .assignLinebuffer .assignEnd .guessTokenFromLine .completeToken*

Description

Get namespace to get access to unexported functions, namely .getHelpFile .assignLinebuffer .assignEnd .guessTokenFromLine .completeToken

Usage

.utils

Format

An object of class environment of length 576.

 aceAnnotate

Enable Error Annotations for an Ace Code Input

Description

This function dynamically evaluate R for syntax errors using the [parse](#) function.

Usage

```
aceAnnotate(inputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

inputId	The id of the input object
session	The session object passed to function given to shinyServer

Details

You can implement your own code completer by observing modification events to `input$<editorId>_shinyAce_annotation` where `<editorId>` is the aceEditor id. This input is only used for triggering completion and will contain a random number. However, you can access `session$input[[inputId]]` to get the input text for parsing.

Value

An observer reference class object that is responsible for offering code annotations. See [observeEvent](#) for more details. You can use `suspend` or `destroy` to pause or stop dynamic code completion.

The observer reference object will send a custom shiny message using `session$sendCustomMessage` to the annotations endpoint containing a json list of annotation metadata objects. The json list should have a structure akin to:

```
[
  {
    row: <int: row of annotation reference>,
    col: <int: column of annotation reference>,
    type: <str: "error", "alert" or "flash">,
    html: <str: html of annotation hover div, used by default over text>,
    text: <num: text of annotation hover div>,
  }
]
```

aceAutocomplete *Enable Code Completion for an Ace Code Input*

Description

This function dynamically auto complete R code pieces using built-in functions `utils::.assignLinebuffer`, `utils::.assignEnd`, `utils::.guessTokenFromLine` and `utils::.completeToken`.

Usage

```
aceAutocomplete(inputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

<code>inputId</code>	The id of the input object
<code>session</code>	The session object passed to function given to shinyServer

Details

You can implement your own code completer by listening to `input$<editorId>_shinyAce_hint` where `<editorId>` is the `aceEditor` id. The input contains

- `linebuffer`: Code/Text at current editing line
- `cursorPosition`: Current cursor position at this line

Value

An observer reference class object that is responsible for offering code completion. See [observe](#) for more details. You can use `suspend` or `destroy` to pause or stop dynamic code completion.

The observer reference object will send a custom shiny message using `session$sendCustomMessage` to the `codeCompletions` endpoint containing a json list of completion item metadata objects. The json list should have a structure akin to:

```
[
  {
    value:      <str: value to be inserted upon completion (e.g. "print()")>,
    caption:    <str: value to be displayed (e.g. "print() # prints text")>,
    score:      <num: score to pass to ace editor for sorting>,
    meta:       <str: meta text on right of completion>
    r_symbol:   <str: symbol name of completion item>,
    r_envir_name: <str: name of the environment from which the symbol is referenced>,
    r_help_type: <str: a datatype for dispatching help documentation function>,
    completer:  <str: used for dispatching default insertMatch functions>,
  }
]
```

`aceEditor`*Render Ace*

Description

Render an Ace editor on an application page.

Usage

```
aceEditor(  
  outputId,  
  value,  
  mode,  
  theme,  
  vimKeyBinding = FALSE,  
  readOnly = FALSE,  
  height = "400px",  
  fontSize = 12,  
  debounce = 1000,  
  wordWrap = FALSE,  
  showLineNumbers = TRUE,  
  highlightActiveLine = TRUE,  
  selectionId = NULL,  
  cursorId = NULL,  
  hotkeys = NULL,  
  code_hotkeys = NULL,  
  autoComplete = c("disabled", "enabled", "live"),  
  autoCompleters = c("snippet", "text", "keyword"),  
  autoCompleteList = NULL,  
  tabSize = 4,  
  useSoftTabs = TRUE,  
  showInvisibles = FALSE,  
  setBehavioursEnabled = TRUE,  
  showPrintMargin = TRUE,  
  autoScrollEditorIntoView = FALSE,  
  maxLines = NULL,  
  minLines = NULL,  
  placeholder = NULL  
)
```

Arguments

<code>outputId</code>	The ID associated with this element
<code>value</code>	The initial text to be contained in the editor.
<code>mode</code>	The Ace mode to be used by the editor. The mode in Ace is often the programming or markup language that you're using and determines things like syntax

	highlighting and code folding. Use the getAceModes function to enumerate all the modes available.
theme	The Ace theme to be used by the editor. The theme in Ace determines the styling and coloring of the editor. Use getAceThemes to enumerate all the themes available.
vimKeyBinding	If set to TRUE, Ace will enable vim-keybindings. Default value is FALSE.
readOnly	If set to TRUE, Ace will disable client-side editing. If FALSE (the default), it will enable editing.
height	A number (which will be interpreted as a number of pixels) or any valid CSS dimension (such as "50%", "200px", or "auto").
fontSize	Defines the font size (in px) used in the editor and should be an integer. The default is 12.
debounce	The number of milliseconds to debounce the input. This will cause the client to withhold update notifications until the user has stopped typing for this amount of time. If 0, the server will be notified of every keystroke as it happens.
wordWrap	If set to TRUE, Ace will enable word wrapping. Default value is FALSE.
showLineNumbers	If set to TRUE, Ace will show line numbers.
highlightActiveLine	If set to TRUE, Ace will highlight the active line.
selectionId	The ID associated with a change of selected text
cursorId	The ID associated with a cursor change.
hotkeys	A list whose names are ID names and whose elements are the shortcuts of keys. Shortcuts can either be a simple string or a list with elements 'win' and 'mac' that that specifies different shortcuts for win and mac (see example 05).
code_hotkeys	A nested list. The first element indicates the code type (e.g., "r") The second element is a list whose names are ID names and whose elements are the shortcuts of keys (see hotkeys)
autoComplete	<p>Enable/Disable auto code completion. Must be one of the following:</p> <p>"disabled" Disable Code Autocomplete</p> <p>"enabled" Enable Basic Code Autocomplete. Autocomplete can be triggered using Ctrl-Space, Ctrl-Shift-Space, or Alt-Space.</p> <p>"live" Enable Live Code Autocomplete. In addition to Basic Autocomplete, it will automatically trigger at each key stroke.</p> <p>By default, only local completer is used where all aforementioned code pieces will be considered as candidates. Use <code>autoCompleteList</code> for static completions and aceAutocomplete for dynamic R code completions.</p>
autoCompleters	Character vector of completers to enable. If set to NULL, all completers will be disabled. Select one or more of "snippet", "text", "static", "keyword", and "rlang" to control which completers to use. Default option is to use the "snippet", "text", and "keyword" autocompleters

<code>autoCompletelist</code>	A named list that contains static code completions candidates. This can be especially useful for Non-Standard Evaluation (NSE) functions such as those in <code>dplyr</code> and <code>ggvis</code> . Each element in list should be a character array whose words will be listed under the element key. For example, to suggest column names from <code>mtcars</code> and <code>airquality</code> , you can use <code>list(mtcars = colnames(mtcars), airquality = colnames(airquality))</code> .
<code>tabSize</code>	Set tab size. Default value is 4
<code>useSoftTabs</code>	Replace tabs by spaces. Default value is TRUE
<code>showInvisibles</code>	Show invisible characters (e.g., spaces, tabs, newline characters). Default value is FALSE
<code>setBehavioursEnabled</code>	Determines if the auto-pairing of special characters, like quotation marks, parenthesis, or brackets should be enabled. Default value is TRUE.
<code>showPrintMargin</code>	Show print margin. Default value is True
<code>autoScrollEditorIntoView</code>	If TRUE, expands the size of the editor window as new lines are added
<code>maxLines</code>	Maximum number of lines the editor window will expand to when <code>autoScrollEditorIntoView</code> is TRUE
<code>minLines</code>	Minimum number of lines in the editor window when <code>autoScrollEditorIntoView</code> is TRUE
<code>placeholder</code>	A string to use a placeholder when the editor has no content

Author(s)

Jeff Allen <jeff@trestletech.com>

Examples

```
## Not run:
aceEditor(
  outputId = "myEditor",
  value = "Initial text for editor here",
  mode = "r",
  theme = "ambiance"
)

aceEditor(
  outputId = "myCodeEditor",
  value = "# Enter code",
  mode = "r",
  hotkeys = list(
    helpKey = "F1",
    runKey = list(
      win = "Ctrl-R|Ctrl-Shift-Enter",
      mac = "CMD-ENTER|CMD-SHIFT-ENTER"
    )
  )
),
```

```

    wordWrap = TRUE, debounce = 10
  )

  aceEditor(
    outputId = "mySmartEditor",
    value = "plot(wt ~ mpg, data = mtcars)",
    mode = "r",
    autoComplete = "live",
    autoCompleteList = list(mtcars = colnames(mtcars))
  )

  ## End(Not run)

```

 aceTooltip

Enable Completion Tooltips for an Ace Code Input

Description

This function uses the completion item object to retrieve tooltip information by parsing R [help](#) documentation and rendering to html.

Usage

```
aceTooltip(inputId, session = shiny::getDefaultReactiveDomain())
```

Arguments

inputId	The id of the input object
session	The session object passed to function given to shinyServer

Details

You can implement your own tooltips by observing modification events to `input$<editorId>_shinyAce_tooltipItem` where `<editorId>` is the `aceEditor` id. This input contains the object passed to `codeCompletion` for this item. See the help for [aceAutocomplete](#) for details on the fields of the completion item object.

Value

An observer reference class object that is responsible for offering completion tooltips. See [observe](#) for more details. You can use `suspend` or `destroy` to pause or stop dynamic code completion.

The observer reference object will send a custom shiny message using `session$sendCustomMessage` to the `docTooltip` endpoint containing a json list of completion item metadata objects. The json list should have a structure akin to one of:

A text object

```
<str: text to display for tooltip>
```

An object containing a docHTML property

```
{
  docHTML: <str: html to display for tooltip div, used if available>,
}
```

An object containing a docText property

```
{
  docText: <str: text to display for tooltip div>
}
```

build_tooltip_fields *Build the fields used to make an html tooltip*

Description

Build the fields used to make an html tooltip

Usage

```
build_tooltip_fields(v)
```

Arguments

v Autocomplete metadata values used for building tooltip info

Value

a list with html-formatted character values "title" and "body"

getAceModes *Get available modes*

Description

Gets all of the available modes available in the installed version of shinyAce. Modes are often the programming or markup language which will be used in the editor and determine things like syntax highlighting and code folding.

Usage

```
getAceModes()
```

Author(s)

Jeff Allen <jeff@trestletech.com>

getAceThemes	<i>Get available themes</i>
--------------	-----------------------------

Description

Gets all of the available themes available in the installed version of shinyAce. Themes determine the styling and colors used in the editor.

Usage

```
getAceThemes()
```

Author(s)

Jeff Allen <jeff@trestletech.com>

get_arg_help	<i>Retrieve argument documentation from help document</i>
--------------	---

Description

Retrieve argument documentation from help document

Usage

```
get_arg_help(..., args = character())
```

Arguments

...	arguments passed to get_help_file
args	function arguments names to get documentation for

Value

A character vector of help

Examples

```
shinyAce::get_arg_help("match", package = "base", args = c("table", "nomatch"))
```

get_desc_help	<i>Retrieve description section from help document</i>
---------------	--

Description

Retrieve description section from help document

Usage

```
get_desc_help(...)
```

Arguments

... arguments passed to get_help_file

Value

a character value representing the description section of a help document, rendered as HTML

Examples

```
shinyAce::get_desc_help("match", package = "base")
```

get_help_file	<i>Retrieve an Rd object of a help query</i>
---------------	--

Description

Safely return NULL if an error is encountered.

Usage

```
get_help_file(...)
```

Arguments

... arguments passed to utils::help

Value

the Rd object returned from utils::getHelpFile

get_usage_help	<i>Retrieve usage section from help document</i>
----------------	--

Description

Retrieve usage section from help document

Usage

```
get_usage_help(...)
```

Arguments

... arguments passed to get_help_file

Value

a character value representing the usage section of a help document, rendered as HTML

Examples

```
shinyAce::get_usage_help("match", package = "base")
```

is.empty	<i>Check if vector is empty</i>
----------	---------------------------------

Description

Check if vector is empty

Usage

```
is.empty(x)
```

Arguments

x vector

Examples

```
is.empty(NULL)
is.empty(NA)
is.empty(c())
is.empty("")
is.empty(" ")
is.empty(c(" ", " "))
is.empty(list())
is.empty(list(a = "", b = ""))
```

meta_obj	<i>Character value to use for object meta field</i>
----------	---

Description

Character value to use for object meta field

Usage

```
meta_obj()
```

meta_pkg	<i>Character value to use for package meta field</i>
----------	--

Description

Character value to use for package meta field

Usage

```
meta_pkg()
```

rd_2_html	<i>Convert an Rd object to HTML</i>
-----------	-------------------------------------

Description

Convert an Rd object to HTML

Usage

```
rd_2_html(...)
```

Arguments

... additional parameters to pass to [parse_Rd](#) when Rd is a filename.

Value

a character value of Rd content rendered as HTML

re_capture	<i>Retrieve regular expression named capture groups as a list</i>
------------	---

Description

Retrieve regular expression named capture groups as a list

Usage

```
re_capture(x, re, ...)
```

Arguments

x	a character string to capture from
re	the regular expression to use
...	additional arguments passed to regexpr

Value

a named list of matches

Examples

```
shinyAce:::re_capture("ak09j b", "(?<num>\\d+)(?<alpha>[a-zA-Z]+)", perl = TRUE)
```

r_completions_function_call_metadata

R completions when cursor is within a function call

Description

R completions when cursor is within a function call

Usage

```
r_completions_function_call_metadata(fname, completions)
```

Arguments

fname	the function name for which the function call specific completion metadata should be constructed
completions	a character vector of completions. These will serve as the foundation for building added R-specific metadata

r_completions_general_metadata
R completions for general case

Description

R completions for general case

Usage

```
r_completions_general_metadata(completions)
```

Arguments

completions a character vector of completions. These will serve as the foundation for building added R-specific metadata

r_completions_metadata
Return completions for a given line of text

Description

Return completions for a given line of text

Usage

```
r_completions_metadata(line)
```

Arguments

line the text up until the cursor in the line for autocompletion

shinyAce-options *Options available for shinyAce*

Description

shinyAce.debug Logical value to enable or disable debugging messages being printed to console. default behavior equivalent to FALSE.

shinyAce_debug	<i>Function for handling optional debugging messages</i>
----------------	--

Description

Function for handling optional debugging messages

Usage

```
shinyAce_debug(...)
```

Arguments

... zero or more objects which can be coerced to character (and which are pasted together with no separator) or (for message only) a single condition object.

tooltip_html	<i>A helper for formatting a tooltip entry</i>
--------------	--

Description

A helper for formatting a tooltip entry

Usage

```
tooltip_html(title = "", body = "")
```

Arguments

title a character value to use as the title
body an html block to embed as the body of the tooltip

 updateAceEditor

Update Ace Editor

Description

Update the styling or mode of an aceEditor component.

Usage

```
updateAceEditor(
  session,
  editorId,
  value,
  theme,
  readOnly,
  mode,
  fontSize,
  showLineNumbers,
  wordWrap,
  useSoftTabs,
  tabSize,
  showInvisibles,
  showPrintMargin,
  border = c("normal", "alert", "flash"),
  autoComplete = c("disabled", "enabled", "live"),
  autoCompleters = c("snippet", "text", "keyword", "static", "rlang"),
  autoCompleteList = NULL
)
```

Arguments

session	The Shiny session to whom the editor belongs
editorId	The ID associated with this element
value	The initial text to be contained in the editor.
theme	The Ace theme to be used by the editor. The theme in Ace determines the styling and coloring of the editor. Use getAceThemes to enumerate all the themes available.
readOnly	If set to TRUE, Ace will disable client-side editing. If FALSE (the default), it will enable editing.
mode	The Ace mode to be used by the editor. The mode in Ace is often the programming or markup language that you're using and determines things like syntax highlighting and code folding. Use the getAceModes function to enumerate all the modes available.
fontSize	If set, will update the font size (in px) used in the editor. Should be an integer.

showLineNumbers	If set to TRUE, Ace will show line numbers.
wordWrap	If set to TRUE, Ace will enable word wrapping. Default value is FALSE.
useSoftTabs	Replace tabs by spaces. Default value is TRUE
tabSize	Set tab size. Default value is 4
showInvisibles	Show invisible characters (e.g., spaces, tabs, newline characters). Default value is FALSE
showPrintMargin	Show print margin. Default value is True
border	Set the border 'normal', 'alert', or 'flash'.
autoComplete	Enable/Disable code completion. See aceEditor for details.
autoCompleters	Character vector of completers to enable. If set to NULL, all completers will be disabled.
autoCompleterList	If set to NULL, existing static completions list will be unset. See aceEditor for details.

Author(s)

Jeff Allen <jeff@trestletech.com>

Examples

```
## Not run:
shinyServer(function(input, output, session) {
  observe({
    updateAceEditor(session, "myEditor", "Updated text for editor here",
      mode = "r", theme = "ambiance")
  })
})
## End(Not run)
```

Index

- * **datasets**
 - .fname_regex, 2
 - .tools, 3
 - .utils, 3
- .fname_regex, 2
- .tools, 3
- .utils, 3
- aceAnnotate, 4
- aceAutocomplete, 5, 7, 9
- aceEditor, 6, 19
- aceTooltip, 9
- build_tooltip_fields, 10
- get_arg_help, 11
- get_desc_help, 12
- get_help_file, 12
- get_usage_help, 13
- getAceModes, 7, 10, 18
- getAceThemes, 7, 11, 18
- help, 9
- is.empty, 13
- meta_obj, 14
- meta_pkg, 14
- observe, 5, 9
- observeEvent, 4
- parse, 4
- parse_Rd, 14
- r_completions_function_call_metadata, 15
- r_completions_general_metadata, 16
- r_completions_metadata, 16
- rd_2_html, 14
- re_capture, 15
- regexpr, 15
- shinyAce-options, 16
- shinyAce_debug, 17
- tooltip_html, 17
- updateAceEditor, 18