Package ‘dashCoreComponents’

October 13, 2022

Title Core Interactive UI Components for ‘Dash’
Version 1.10.0
Description ‘Dash’ ships with supercharged components for interactive user interfaces. A core set of components, written and maintained by the ‘Dash’ team, is available in the ‘dashCoreComponents’ package. The source for this package is on GitHub: plotly/dash-core-components.
Depends R (>= 3.0.2)
Imports
Suggests dash, dashHtmlComponents, jsonlite, plotly, knitr, rmarkdown
License MIT + file LICENSE
Copyright Plotly Technologies, Inc.
URL https://github.com/plotly/dash-core-components
BugReports https://github.com/plotly/dash-core-components/issues
Encoding UTF-8
LazyData true
VignetteBuilder knitr
KeepSource true
NeedsCompilation no
Author Chris Parmer [aut],
Ryan Patrick Kyle [cre] (https://orcid.org/0000-0002-4958-2844),
Plotly Technologies, Inc. [cph]
Maintainer Ryan Patrick Kyle <ryan@plotly.com>
Repository CRAN
Date/Publication 2020-05-06 22:00:11 UTC
dashCoreComponents-package

Description

'Dash' ships with supercharged components for interactive user interfaces. A core set of components, written and maintained by the 'Dash' team, is available in the 'dashCoreComponents' package. The source for this package is on GitHub: plotly/dash-core-components.

Author(s)

Maintainer: Ryan Patrick Kyle <ryan@plotly.com>
Description

Checklist is a component that encapsulates several checkboxes. The values and labels of the checklist are specified in the 'options' property and the checked items are specified with the 'value' property. Each checkbox is rendered as an input with a surrounding label.

Usage

dccChecklist(id=NULL, options=NULL, value=NULL, className=NULL,
style=NULL, inputStyle=NULL, inputClassName=NULL,
labelStyle=NULL, labelClassName=NULL, loading_state=NULL,
persistence=NULL, persisted_props=NULL,
persistence_type=NULL)

Arguments

id
Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.

options
List of lists containing elements 'label', 'value', 'disabled'. those elements have the following types: - label (character | numeric; required): the checkbox’s label - value (character | numeric; required): the value of the checkbox. this value corresponds to the items specified in the 'value' property. - disabled (logical; optional): if true, this checkbox is disabled and can’t be clicked on.s. An array of options

value
List of character | numerics. The currently selected value

className
Character. The class of the container (div)

style
Named list. The style of the container (div)

inputStyle
Named list. The style of the <input> checkbox element

inputClassName
Character. The class of the <input> checkbox element

labelStyle
Named list. The style of the <label> that wraps the checkbox input and the option's label

labelClassName
Character. The class of the <label> that wraps the checkbox input and the option's label

loading_state
Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer
persistence

Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persisted' is truthy and hasn’t changed from its previous value, a 'value' that the user has changed while using the app will keep that change, as long as the new 'value' also matches what was given originally. Used in conjunction with 'persistence_type'.

persisted_props

List of a value equal to: 'value's. Properties whose user interactions will persist after refreshing the component or the page. Since only 'value' is allowed this prop can normally be ignored.

persistence_type

A value equal to: 'local', 'session', 'memory'. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

Value
	named list of JSON elements corresponding to React.js properties and their values

Examples

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashHtmlComponents)
  library(dashCoreComponents)

  app <- Dash$new()

  app$layout(
    dccChecklist(
      id = "checklist-input",
      options=list(
        list("label" = "New York City", "value" = "NYC"),
        list("label" = "Montreal", "value" = "MTL"),
        list("label" = "San Francisco", "value" = "SF")
      ),
      value=list("MTL", "SF")
    )
  )

  app$run_server()
}
```

dccConfirmDialog

ConfirmDialog component
**Description**

ConfirmDialog is used to display the browser's native "confirm" modal, with an optional message and two buttons ("OK" and "Cancel"). This ConfirmDialog can be used in conjunction with buttons when the user is performing an action that should require an extra step of verification.

**Usage**

dccConfirmDialog(id=NULL, message=NULL, submit_n_clicks=NULL, submit_n_clicks_timestamp=NULL, cancel_n_clicks=NULL, cancel_n_clicks_timestamp=NULL, displayed=NULL)

**Arguments**

- **id** Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
- **message** Character. Message to show in the popup.
- **submit_n_clicks** Numeric. Number of times the submit button was clicked.
- **submit_n_clicks_timestamp** Numeric. Last time the submit button was clicked.
- **cancel_n_clicks** Numeric. Number of times the popup was canceled.
- **cancel_n_clicks_timestamp** Numeric. Last time the cancel button was clicked.
- **displayed** Logical. Set to true to send the ConfirmDialog.

**Value**

named list of JSON elements corresponding to React.js properties and their values

**Examples**

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(dashHtmlComponents)

  app <- Dash$new()

  app$layout(
    htmlDiv(
      list(
        dccConfirmDialog(
          id='confirm',
          message='Danger danger! Are you sure you want to continue?'),
        dccDropdown(
          options=lapply(list('Safe', 'Danger!!'),function(x){list('label'= x, 'value'= x)}),
          id='dropdown'
        ),
      )
    )
  )
}
```
A wrapper component that will display a confirmation dialog when its child component has been clicked on. For example: 

```
dcc.ConfirmDialogProvider(html.Button('click me', id='btn'), message='Danger - Are you sure you want to continue.' id='confirm')
```

### Arguments

- **children**: Logical | numeric | character | named list | unnamed list. The children to hijack clicks from and display the popup.
- **id**: Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
- **message**: Character. Message to show in the popup.
- **submit_n_clicks**: Numeric. Number of times the submit was clicked
- **submit_n_clicks_timestamp**: Numeric. Last time the submit button was clicked.
- **cancel_n_clicks**: Numeric. Number of times the popup was canceled.

---

**dccConfirmDialogProvider**

**ConfirmDialogProvider component**

**Description**

A wrapper component that will display a confirmation dialog when its child component has been clicked on. For example: **dcc.ConfirmDialogProvider(html.Button('click me', id='btn'), message='Danger - Are you sure you want to continue.' id='confirm')**

**Usage**

```
dccConfirmDialogProvider(children=NULL, id=NULL, message=NULL, submit_n_clicks=NULL, submit_n_clicks_timestamp=NULL, cancel_n_clicks=NULL, cancel_n_clicks_timestamp=NULL, displayed=NULL, loading_state=NULL)
```

**Arguments**

- **children**: Logical | numeric | character | named list | unnamed list. The children to hijack clicks from and display the popup.
- **id**: Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
- **message**: Character. Message to show in the popup.
- **submit_n_clicks**: Numeric. Number of times the submit was clicked
- **submit_n_clicks_timestamp**: Numeric. Last time the submit button was clicked.
- **cancel_n_clicks**: Numeric. Number of times the popup was canceled.
dccConfirmDialogProvider

cancel_n_clicks_timestamp
    Numeric. Last time the cancel button was clicked.

displayed
    Logical. Is the modal currently displayed.

loading_state
    Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

Value
    named list of JSON elements corresponding to React.js properties and their values

Examples
    if (interactive() && require(dash)) {
        library(dash)
        library(dashCoreComponents)
        library(dashHtmlComponents)

        app <- Dash$new()

        app$layout(htmlDiv(list(
            dccConfirmDialogProvider(
                children=htmlButton(
                    'Click Me',
                    n_clicks = 0
                ),
                id='danger-danger-provider',
                message='Danger danger! Are you sure you want to continue?',
                submit_n_clicks=NULL
            ),
            htmlDiv(id='output-provider',
                children='Click the button to submit')
            )))

        app$callback(
            output = list(id = 'output-provider', property = 'children'),
            params=list(input(id = 'danger-danger-provider', property = 'submit_n_clicks')),
            function(submit_n_clicks) {
                if (is.null(unlist(submit_n_clicks))) {
                    return('
                } else {
                    paste0('That was a dangerous choice! Submitted ', submit_n_clicks, ' times."
                }
            }

        )

        app$run_server()
    }
dccDatePickerRange  DatePickerRange component

Description

DatePickerRange is a tailor made component designed for selecting timespan across multiple days off of a calendar. The DatePicker integrates well with the Python datetime module with the startDate and endDate being returned in a string format suitable for creating datetime objects. This component is based off of Airbnb’s react-dates react component which can be found here: https://github.com/airbnb/react-dates

Usage

dccDatePickerRange(id=NULL, start_date=NULL, start_date_id=NULL, end_date_id=NULL, end_date=NULL, min_date_allowed=NULL, max_date_allowed=NULL, initial_visible_month=NULL, start_date_placeholder_text=NULL, end_date_placeholder_text=NULL, day_size=NULL, calendar_orientation=NULL, is_RTL=NULL, reopen_calendar_on_clear=NULL, number_of_months_shown=NULL, with_portal=NULL, with_full_screen_portal=NULL, first_day_of_week=NULL, minimum_nights=NULL, stay_open_on_select=NULL, show_outside_days=NULL, month_format=NULL, display_format=NULL, disabled=NULL, clearable=NULL, style=NULL, className=NULL, updatemode=NULL, loading_state=NULL, persistence=NULL, persisted_props=NULL, persistence_type=NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.</td>
</tr>
<tr>
<td>start_date</td>
<td>Character. Specifies the starting date for the component. Accepts datetime.datetime objects or strings in the format 'YYYY-MM-DD'</td>
</tr>
<tr>
<td>start_date_id</td>
<td>Character. The HTML element ID of the start date input field. Not used by Dash, only by CSS.</td>
</tr>
<tr>
<td>end_date_id</td>
<td>Character. The HTML element ID of the end date input field. Not used by Dash, only by CSS.</td>
</tr>
<tr>
<td>end_date</td>
<td>Character. Specifies the ending date for the component. Accepts datetime.datetime objects or strings in the format 'YYYY-MM-DD'</td>
</tr>
<tr>
<td>min_date_allowed</td>
<td>Character. Specifies the lowest selectable date for the component. Accepts datetime.datetime objects or strings in the format 'YYYY-MM-DD'</td>
</tr>
<tr>
<td>max_date_allowed</td>
<td>Character. Specifies the highest selectable date for the component. Accepts datetime.datetime objects or strings in the format 'YYYY-MM-DD'</td>
</tr>
</tbody>
</table>
**dccDatePickerRange**

- **initial_visible_month**
  - Character. Specifies the month that is initially presented when the user opens the calendar. Accepts datetime.datetime objects or strings in the format 'YYYY-MM-DD'

- **start_date_placeholder_text**
  - Character. Text that will be displayed in the first input box of the date picker when no date is selected. Default value is 'Start Date'

- **end_date_placeholder_text**
  - Character. Text that will be displayed in the second input box of the date picker when no date is selected. Default value is 'End Date'

- **day_size**
  - Numeric. Size of rendered calendar days, higher number means bigger day size and larger calendar overall

- **calendar_orientation**
  - A value equal to: 'vertical', 'horizontal'. Orientation of calendar, either vertical or horizontal. Valid options are 'vertical' or 'horizontal'.

- **is_RTL**
  - Logical. Determines whether the calendar and days operate from left to right or from right to left

- **reopen_calendar_on_clear**
  - Logical. If True, the calendar will automatically open when cleared

- **number_of_months_shown**
  - Numeric. Number of calendar months that are shown when calendar is opened

- **with_portal**
  - Logical. If True, calendar will open in a screen overlay portal, not supported on vertical calendar

- **with_full_screen_portal**
  - Logical. If True, calendar will open in a full screen overlay portal, will take precedent over 'withPortal' if both are set to true, not supported on vertical calendar

- **first_day_of_week**
  - A value equal to: 0, 1, 2, 3, 4, 5, 6. Specifies what day is the first day of the week, values must be from [0, ..., 6] with 0 denoting Sunday and 6 denoting Saturday

- **minimum_nights**
  - Numeric. Specifies a minimum number of nights that must be selected between the startDate and the endDate

- **stay_open_on_select**
  - Logical. If True the calendar will not close when the user has selected a value and will wait until the user clicks off the calendar

- **show_outside_days**
  - Logical. If True the calendar will display days that rollover into the next month

- **month_format**
  - Character. Specifies the format that the month will be displayed in the calendar, valid formats are variations of "MM YY". For example: "MM YY" renders as '05 97' for May 1997 "MMMM, YYYY" renders as 'May, 1997' for May 1997 "MMM, YY" renders as 'Sep, 97' for September 1997

- **display_format**
  - Character. Specifies the format that the selected dates will be displayed valid formats are variations of "MM YY DD". For example: "MM YY DD" renders as '05 10 97' for May 10th 1997 "MMMM, YY" renders as 'May, 1997' for May 10th 1997 "M, D, YYYY" renders as '07, 10, 1997' for September 10th 1997 "MMMM" renders as 'May' for May 10 1997
disabled  Logical. If True, no dates can be selected.
clearable Logical. Whether or not the dropdown is "clearable", that is, whether or not a small "x" appears on the right of the dropdown that removes the selected value.
style Named list. CSS styles appended to wrapper div
className Character. Appends a CSS class to the wrapper div component.
updatemode A value equal to: 'singledate', 'bothdates'. Determines when the component should update its value. If 'bothdates', then the DatePicker will only trigger its value when the user has finished picking both dates. If 'singledate', then the DatePicker will update its value as one date is picked.
loading_state Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer
persistence Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persisted' is truthy and hasn’t changed from its previous value, any 'persisted_props' that the user has changed while using the app will keep those changes, as long as the new prop value also matches what was given originally. Used in conjunction with 'persistence_type' and 'persisted_props'.
persisted_props List of a value equal to: 'start_date', 'end_date'. Properties whose user interactions will persist after refreshing the component or the page.
persistence_type A value equal to: 'local', 'session', 'memory'. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

Value

named list of JSON elements corresponding to React.js properties and their values

Examples

if (interactive() & require(dash)) {
  library(dash)
  library(dashCoreComponents)
  app <- Dash$new()
  app$layout(
    dccDatePickerRange(
      id = "date-picker-range",
      start_date = as.Date("1997/5/10"),
      end_date_placeholder_text="Select a date!"
    )
  )
}
Description

DatePickerSingle is a tailor made component designed for selecting a single day off of a calendar. The DatePicker integrates well with the Python datetime module with the startDate and endDate being returned in a string format suitable for creating datetime objects. This component is based off of Airbnb’s react-dates react component which can be found here: https://github.com/airbnb/react-dates

Usage

dccDatePickerSingle(id=NULL, date=NULL, min_date_allowed=NULL, max_date_allowed=NULL, initial_visible_month=NULL, day_size=NULL, calendar_orientation=NULL, is_RTL=NULL, placeholder=NULL, reopen_calendar_on_clear=NULL, number_of_months_shown=NULL, with_portal=NULL, with_full_screen_portal=NULL, first_day_of_week=NULL, stay_open_on_select=NULL, show_outside_days=NULL, month_format=NULL, display_format=NULL, disabled=NULL, clearable=NULL, style=NULL, className=NULL, loading_state=NULL, persistence=NULL, persisted_props=NULL, persistence_type=NULL)

Arguments

id Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
date Character. Specifies the starting date for the component, best practice is to pass value via datetime object
min_date_allowed Character. Specifies the lowest selectable date for the component. Accepts datetime.datetime objects or strings in the format ‘YYYY-MM-DD’
max_date_allowed Character. Specifies the highest selectable date for the component. Accepts datetime.datetime objects or strings in the format ‘YYYY-MM-DD’
initial_visible_month Character. Specifies the month that is initially presented when the user opens the calendar. Accepts datetime.datetime objects or strings in the format ‘YYYY-MM-DD’
dccDatePickerSingle

day_size  Numeric. Size of rendered calendar days, higher number means bigger day size and larger calendar overall

calendar_orientation  A value equal to: 'vertical', 'horizontal'. Orientation of calendar, either vertical or horizontal. Valid options are 'vertical' or 'horizontal'.

is_RTL  Logical. Determines whether the calendar and days operate from left to right or from right to left

placeholder  Character. Text that will be displayed in the input box of the date picker when no date is selected. Default value is 'Start Date'

reopen_calendar_on_clear  Logical. If True, the calendar will automatically open when cleared

number_of_months_shown  Numeric. Number of calendar months that are shown when calendar is opened

with_portal  Logical. If True, calendar will open in a screen overlay portal, not supported on vertical calendar

with_full_screen_portal  Logical. If True, calendar will open in a full screen overlay portal, will take precedent over 'withPortal' if both are set to True, not supported on vertical calendar

first_day_of_week  A value equal to: 0, 1, 2, 3, 4, 5, 6. Specifies what day is the first day of the week, values must be from [0, ..., 6] with 0 denoting Sunday and 6 denoting Saturday

stay_open_on_select  Logical. If True the calendar will not close when the user has selected a value and will wait until the user clicks off the calendar

show_outside_days  Logical. If True the calendar will display days that rollover into the next month

month_format  Character. Specifies the format that the month will be displayed in the calendar, valid formats are variations of "MM YY". For example: "MM YY" renders as '05 97' for May 1997 "MMMM, YYYY" renders as 'May, 1997' for May 1997 "MMM, YY" renders as 'Sep, 97' for September 1997

display_format  Character. Specifies the format that the selected dates will be displayed valid formats are variations of "MM YY DD". For example: "MM YY DD" renders as '05 10 97' for May 10th 1997 "MMMM, YY" renders as 'May, 1997' for May 10th 1997 "M, D, YYYY" renders as '07, 10, 1997' for September 10th 1997 "MMMM" renders as 'May' for May 10 1997

disabled  Logical. If True, no dates can be selected.

clearable  Logical. Whether or not the dropdown is "clearable", that is, whether or not a small "x" appears on the right of the dropdown that removes the selected value.

style  Named list. CSS styles appended to wrapper div

className  Character. Appends a CSS class to the wrapper div component.

loading_state  Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which
property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

**persistence**
Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persisted' is truthy and hasn’t changed from its previous value, a ‘date’ that the user has changed while using the app will keep that change, as long as the new ‘date’ also matches what was given originally. Used in conjunction with 'persistence_type'.

**persisted_props**
List of a value equal to: 'date's. Properties whose user interactions will persist after refreshing the component or the page. Since only ‘date’ is allowed this prop can normally be ignored.

**persistence_type**
A value equal to: 'local', 'session', 'memory'. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

**Value**

named list of JSON elements corresponding to React.js properties and their values

**Examples**

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)

  app <- Dash$new()

  app$layout(
    dccDatePickerSingle(
      id = "date-picker-single",
      date = as.Date("1997/5/10")
    )
  )

  app$run_server()
}
```

dccDropdown

**Dropdown component**

**Description**

Dropdown is an interactive dropdown element for selecting one or more items. The values and labels of the dropdown items are specified in the ‘options’ property and the selected item(s) are specified with the ‘value’ property. Use a dropdown when you have many options (more than 5) or
when you are constrained for space. Otherwise, you can use RadioItems or a Checklist, which have the benefit of showing the users all of the items at once.

**Usage**

```r
dccDropdown(id=NULL, options=NULL, value=NULL, optionHeight=NULL, className=NULL, clearable=NULL, disabled=NULL, multi=NULL, placeholder=NULL, searchable=NULL, search_value=NULL, style=NULL, loading_state=NULL, persistence=NULL, persisted_props=NULL, persistence_type=NULL)
```

**Arguments**

- **id**
  - Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.

- **options**
  - List of lists containing elements 'label', 'value', 'disabled', 'title'. those elements have the following types:  - label (character | numeric; required): the dropdown's label - value (character | numeric; required): the value of the dropdown. this value corresponds to the items specified in the 'value' property. - disabled (logical; optional): if true, this option is disabled and cannot be selected. - title (character; optional): the html 'title' attribute for the option. allows for information on hover. for more information on this attribute, see https://developer.mozilla.org/en-us/docs/web/html/global_attributes/titles. An array of options label: [stringnumber], value: [stringnumber], an optional disabled field can be used for each option

- **value**
  - Character | numeric | list of character | numerics. The value of the input. If 'multi' is false (the default) then value is just a string that corresponds to the values provided in the 'options' property. If 'multi' is true, then multiple values can be selected at once, and 'value' is an array of items with values corresponding to those in the 'options' prop.

- **optionHeight**
  - Numeric. height of each option. Can be increased when label lengths would wrap around

- **className**
  - Character. className of the dropdown element

- **clearable**
  - Logical. Whether or not the dropdown is "clearable", that is, whether or not a small "x" appears on the right of the dropdown that removes the selected value.

- **disabled**
  - Logical. If true, this dropdown is disabled and the selection cannot be changed.

- **multi**
  - Logical. If true, the user can select multiple values

- **placeholder**
  - Character. The grey, default text shown when no option is selected

- **searchable**
  - Logical. Whether to enable the searching feature or not

- **search_value**
  - Character. The value typed in the DropDown for searching.

- **style**
  - Named list. Defines CSS styles which will override styles previously set.

- **loading_state**
  - Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer
persistence
Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persisted' is truthy and hasn’t changed from its previous value, a 'value' that the user has changed while using the app will keep that change, as long as the new 'value' also matches what was given originally. Used in conjunction with 'persistence_type'.

persisted_props
List of a value equal to: 'value'. Properties whose user interactions will persist after refreshing the component or the page. Since only 'value' is allowed this prop can normally be ignored.

persistence_type
A value equal to: 'local', 'session', 'memory'. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

Value
named list of JSON elements corresponding to React.js properties and their values

Examples
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)

  app <- Dash$new()

  app$layout(
    htmlDiv(
      dccDropdown(
        options=list(
          list(label = "New York City", value = "NYC"),
          list(label = "Montreal", value = "MTL"),
          list(label = "San Francisco", value = "SF"),
        ),
        value="MTL"
      ),
      value="MTL"
    )
  )

  app$run_server()
}
Description

Graph can be used to render any plotly.js-powered data visualization. You can define callbacks based on user interaction with Graphs such as hovering, clicking or selecting.

Usage

dccGraph(id=NULL, responsive=NULL, clickData=NULL, clickAnnotationData=NULL, hoverData=NULL, clear_on_unhover=NULL, selectedData=NULL, relayoutData=NULL, extendData=NULL, restyleData=NULL, figure=NULL, style=NULL, className=NULL, animate=NULL, animation_options=NULL, config=NULL, loading_state=NULL)

Arguments

id
Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.

responsive
A value equal to: true, false, 'auto'. If True, the Plotly.js plot will be fully responsive to window resize and parent element resize event. This is achieved by overriding ‘config.responsive’ to True, ‘figure.layout.autosize’ to True and unsetting ‘figure.layout.height’ and ‘figure.layout.width’. If False, the Plotly.js plot not be responsive to window resize and parent element resize event. This is achieved by overriding ‘config.responsive’ to False and ‘figure.layout.autosize’ to False. If ‘auto’ (default), the Graph will determine if the Plotly.js plot can be made fully responsive (True) or not (False) based on the values in ‘config.responsive’, ‘figure.layout.autosize’, ‘figure.layout.height’, ‘figure.layout.width’. This is the legacy behavior of the Graph component. Needs to be combined with appropriate dimension / styling through the ‘style’ prop to fully take effect.

clickData
Named list. Data from latest click event. Read-only.

clickAnnotationData
Named list. Data from latest click annotation event. Read-only.

hoverData
Named list. Data from latest hover event. Read-only.

clear_on_unhover
Logical. If True, ‘clear_on_unhover’ will clear the ‘hoverData’ property when the user "unhovers" from a point. If False, then the ‘hoverData’ property will be equal to the data from the last point that was hovered over.

selectedData
Named list. Data from latest select event. Read-only.

relayoutData
Named list. Data from latest relayout event which occurs when the user zooms or pans on the plot or other layout-level edits. Has the form '<attr string>: <value>' describing the changes made. Read-only.

extendData
Unnamed list | named list. Data that should be appended to existing traces. Has the form ‘[updateData, traceIndices, maxPoints]’, where ‘updateData’ is an object containing the data to extend, ‘traceIndices’ (optional) is an array of trace indices that should be extended, and ‘maxPoints’ (optional) is either an integer defining the maximum number of points allowed or an object with key:value
pairs matching 'updateData' Reference the Plotly.extendTraces API for full usage: https://plotly.com/javascript/plotlyjs-function-reference/#plotlyextendtraces

restyleData
Unused list. Data from latest restyle event which occurs when the user toggles a legend item, changes parcoords selections, or other trace-level edits. Has the form `{[edits, indices]}`, where `edits` is an object `<attr string>: <value>` describing the changes made, and `indices` is an array of trace indices that were edited. Read-only.

figure
Lists containing elements 'data', 'layout', 'frames'. those elements have the following types: - data (list of named lists; optional) - layout (named list; optional) - frames (list of named lists; optional). Plotly 'figure' object. See schema: https://plotly.com/javascript/reference

style
Named list. Generic style overrides on the plot div

className
Character. className of the parent div

animate
Logical. Beta: If true, animate between updates using plotly.js's `animate` function

animation_options
Named list. Beta: Object containing animation settings. Only applies if 'animate' is 'true'

config
List of containing elements 'staticplot', 'plotlyserverurl', 'editable', 'edits', 'autosizable', 'responsive', 'queueLength', 'fillFrame', 'framemargins', 'scrollzoom', 'doubleclick', 'doubleclickdelay', 'showTips', 'showAxisDragHandles', 'showAxisRangeEntryBoxes', 'showlink', 'sendData', 'linktext', 'displayModeBar', 'showSendToCloud', 'showEditInChartStudio', 'modeBarButtons', 'toImageButtonOptions', 'displayLogo', 'watermark', 'plotGlPixelRatio', 'topojsonurl', 'mapboxAccessToken', 'locale', 'locales'. those elements have the following types: - staticplot (logical; optional): no interactivity, for export or image generation - plotlyserverurl (character; optional): base url for a plotly cloud instance, if 'showSendToCloud' is enabled - editable (logical; optional): we can edit titles, move annotations, etc - sets all pieces of 'edits' unless a separate 'edits' config item overrides individual parts - edits (optional): a set of editable properties. edits has the following type: lists containing elements 'annotationposition', 'annotationTail', 'annotationText', 'axistitleText', 'colorbarPosition', 'colorbarTitleText', 'legendPosition', 'legendText', 'shapePosition', 'titleText'. those elements have the following types: - annotationposition (logical; optional): the main anchor of the annotation, which is the text (if no arrow) or the arrow (which drags the whole thing leaving the arrow length & direction unchanged) - annotationTail (logical; optional): just for annotations with arrows, change the length and direction of the arrow - annotationText (logical; optional): the global 'layout.title' - autosizable (logical; optional): do autosize once regardless of layout.autosize (use default width or height values otherwise) - responsive (logical; optional): whether to change layout size when the window size changes - queueLength (numeric; optional): set the length of the undo/redo queue - fillFrame (logical;
optional): if we do autosize, do we fill the container or the screen? - framemargins (numeric; optional): if we do autosize, set the frame margins in percents of plot size - scrollzoom (logical; optional): mousewheel or two-finger scroll zooms the plot - doubleclick (a value equal to: false, 'reset', 'autosize', 'reset+autosize'; optional): double click interaction (false, 'reset', 'autosize' or 'reset+autosize') - doubleclickdelay (numeric; optional): delay for registering a double-click event in ms. the minimum value is 100 and the maximum value is 1000. by default this is 300. - showtips (logical; optional): new users see some hints about interactivity - showaxisdraggable (logical; optional): enable axis drag handles - showaxisrangeentryboxes (logical; optional): enable direct range entry at the pan/zoom drag points (drag handles must be enabled above) - showlink (logical; optional): link to open this plot in plotly - senddata (logical; optional): if we show a link, does it contain data or just link to a plotly file? - linktext (character; optional): text appearing in the senddata link - displaymodebar (a value equal to: true, false, 'hover'; optional): display the mode bar (true, false, or 'hover') - showsendtocloud (logical; optional): should we include a modebar button to send this data to a plotly cloud instance, linked by 'plotlyserverurl'. by default this is false. - showeditinchartstudio (logical; optional): should we show a modebar button to send this data to a plotly chart studio plot. if both this and showsendtocloud are selected, only showeditinchartstudio will be honored. by default this is false. - modebarbuttonstoremove (unnamed list; optional): remove mode bar button by name. all modebar button names at https://github.com/plotly/plotly.js/blob/master/src/components/modebar/buttons.js common names include: senddatatocloud; (2d) zoom2d, pan2d, select2d, lasso2d, zoomin2d, zoomout2d, autoscale2d, resetscale2d; (cartesian) hoverclosestcartesian, hovercomparecartesian; (3d) zoom3d, pan3d, orbitrotation, tablerotation, handledrag3d, resetcameradefault3d, resetcamerasticky3d, hoverclosest3d; (geo) zoominggeo, zoomoutgeo, resetgeo, hoverclosestgeo, hoverclosestgl2d, hoverclosestpie, togglehover, resetviews. - modebarbuttonstoaddd (unnamed list; optional): add mode bar button using config objects - modebarbuttons (logical | numeric | character | named list | unnamed list; optional): fully custom mode bar buttons as nested array, where the outer arrays represents button groups, and the inner arrays have buttons config objects or names of default buttons - toimagebuttonoptions (optional): modifications to how the toimage modebar button works. toimagebuttonoptions has the following type: lists containing elements 'format', 'filename', 'width', 'height', 'scale'. those elements have the following types: - format (a value equal to: 'jpeg', 'png', 'webp', 'svg'; optional): the file format to create - filename (character; optional): the name given to the downloaded file - width (numeric; optional): width of the downloaded file, in px - height (numeric; optional): height of the downloaded file, in px - scale (numeric; optional): extra resolution to give the file after rendering it with the given width and height - displaylogo (logical; optional): add the plotly logo on the end of the mode bar - watermark (logical; optional): add the plotly logo even with no modebar - plotglpixelratio (numeric; optional): increase the pixel ratio for gl plot images - topojsonurl (character; optional): url to topojson files used in geo charts - mapboxaccesstoken (logical | numeric | character | named list | unnamed list; optional): mapbox access token (required to plot mapbox trace types) if using an mapbox atlas server, set this
option to ", so that plotly.js won’t attempt to authenticate to the public map-
box server.
- locale (character; optional): the locale to use. locales may be
provided with the plot (‘locales’ below) or by loading them on the page, see:
https://github.com/plotly/plotly.js/blob/master/dist/readme.md#to-include-localization
- locales (named list; optional): localization definitions, if you choose to provide
them with the plot rather than registering them globally. Plotly.js config options.
See https://plotly.com/javascript/configuration-options/ for more info.

loading_state Lists containing elements ‘is_loading’, ’prop_name’, ’component_name’. those
elements have the following types:
- is_loading (logical; optional): determines if
the component is loading or not
- prop_name (character; optional): holds which
property is loading
- component_name (character; optional): holds the name of
the component that is loading. Object that holds the loading state object coming
from dash-renderer

Value

named list of JSON elements corresponding to React.js properties and their values

Examples

if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(plotly)
  app <- Dash$new()


  worldwide <- c(219, 146, 112, 127, 180, 236, 207, 236, 263,
                  350, 430, 474, 526, 488, 537, 500, 439)

  china <- c(16, 13, 10, 11, 28, 37, 43, 55, 56, 88, 105, 156, 270,
            299, 340, 403, 549, 499)

  data <- data.frame(year, worldwide, china)

  app$layout(
    htmlDiv(
      dccGraph(
        figure = layout(
          add_trace(
            plot_ly(data,
              x = ~year,
              y = ~worldwide,
              type = "bar",
              name = "Worldwide",
              marker = list(color = "rgb(55, 83, 109)"))
            ,
              y = ~china,
              name = "China",
            plot_ly(data,
            x = ~year,
            y = ~worldwide,
            type = "bar",
            name = "Worldwide",
            marker = list(color = "rgb(55, 83, 109)"))
```
```r
marker = list(color = "rgb(26, 118, 255)")
),
yaxis = list(title = "Count"),
xaxis = list(title = "Year"),
barmode = "group",
title="US Export of Plastic Scrap"),
style = list("height" = 300),
id = "my_graph"
)
)
app$run_server()
}
```

### dccInput

**Input component**

**Description**

A basic HTML input control for entering text, numbers, or passwords. Note that checkbox and radio types are supported through the Checklist and RadioItems component. Dates, times, and file uploads are also supported through separate components.

**Usage**

```r
dccInput(id=NULL, value=NULL, style=NULL, className=NULL, debounce=NULL, type=NULL, autoComplete=NULL, autoFocus=NULL, disabled=NULL, inputMode=NULL, list=NULL, max=NULL, maxLength=NULL, min=NULL, minLength=NULL, multiple=NULL, name=NULL, pattern=NULL, placeholder=NULL, readOnly=NULL, required=NULL, selectionDirection=NULL, selectionEnd=NULL, selectionStart=NULL, size=NULL, spellCheck=NULL, step=NULL, n_submit=NULL, n_submit_timestamp=NULL, n_blur=NULL, n_blur_timestamp=NULL, loading_state=NULL, persistence=NULL, persisted_props=NULL, persistence_type=NULL)
```

**Arguments**

- **id** | Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
- **value** | Character | numeric. The value of the input
- **style** | Named list. The input’s inline styles
- **className** | Character. The class of the input element
- **debounce** | Logical. If true, changes to input will be sent back to the Dash server only on enter or when losing focus. If it’s false, it will sent the value back on every change.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>A value equal to: &quot;text&quot;, 'number', 'password', 'email', 'range', 'search', 'tel', 'url', 'hidden'. The type of control to render.</td>
</tr>
<tr>
<td><strong>autoComplete</strong></td>
<td>Character. This attribute indicates whether the value of the control can be automatically completed by the browser.</td>
</tr>
<tr>
<td><strong>autoFocus</strong></td>
<td>A value equal to: 'autofocus', 'autofocus', 'autofocus'</td>
</tr>
<tr>
<td><strong>disabled</strong></td>
<td>A value equal to: 'disabled', 'disabled'</td>
</tr>
<tr>
<td><strong>inputMode</strong></td>
<td>A value equal to: &quot;verbatim&quot;, &quot;latin&quot;, &quot;latin-name&quot;, &quot;latin-prose&quot;, &quot;full-width-latin&quot;, &quot;kana&quot;, &quot;katakana&quot;, &quot;numeric&quot;, &quot;tel&quot;, &quot;email&quot;, &quot;url&quot;. Provides a hint to the browser as to the type of data that might be entered by the user while editing the element or its contents.</td>
</tr>
<tr>
<td><strong>list</strong></td>
<td>Character. Identifies a list of pre-defined options to suggest to the user. The value must be the id of a &lt;datalist&gt; element in the same document. The browser displays only options that are valid values for this input element. This attribute is ignored when the type attribute’s value is hidden, checkbox, radio, file, or a button type.</td>
</tr>
<tr>
<td><strong>max</strong></td>
<td>Character</td>
</tr>
<tr>
<td><strong>maxLength</strong></td>
<td>Character</td>
</tr>
<tr>
<td><strong>min</strong></td>
<td>Character</td>
</tr>
<tr>
<td><strong>minLength</strong></td>
<td>Character</td>
</tr>
<tr>
<td><strong>multiple</strong></td>
<td>Logical. This Boolean attribute indicates whether the user can enter more than one value. This attribute applies when the type attribute is set to email or file, otherwise it is ignored.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Character. The name of the control, which is submitted with the form data.</td>
</tr>
<tr>
<td><strong>pattern</strong></td>
<td>Character. A regular expression that the control’s value is checked against. The pattern must match the entire value, not just some subset. Use the title attribute to describe the pattern to help the user. This attribute applies when the value of the type attribute is text, search, tel, url, email, or password, otherwise it is ignored. The regular expression language is the same as JavaScript RegExp</td>
</tr>
</tbody>
</table>
algorithm, with the 'u' parameter that makes it treat the pattern as a sequence of unicode code points. The pattern is not surrounded by forward slashes.

placeholder Character | numeric. A hint to the user of what can be entered in the control. The placeholder text must not contain carriage returns or line-feeds. Note: Do not use the placeholder attribute instead of a <label> element, their purposes are different. The <label> attribute describes the role of the form element (i.e. it indicates what kind of information is expected), and the placeholder attribute is a hint about the format that the content should take. There are cases in which the placeholder attribute is never displayed to the user, so the form must be understandable without it.

readOnly Logical | a value equal to: 'readonly', 'readonly', 'readonly'. This attribute indicates that the user cannot modify the value of the control. The value of the attribute is irrelevant. If you need read-write access to the input value, do not add the "readonly" attribute. It is ignored if the value of the type attribute is hidden, range, color, checkbox, radio, file, or a button type (such as button or submit). readOnly is an HTML boolean attribute - it is enabled by a boolean or 'readOnly'. Alternative capitalizations 'readonly' & 'READONLY' are also accepted.

required A value equal to: 'required', 'required' | logical. This attribute specifies that the user must fill in a value before submitting a form. It cannot be used when the type attribute is hidden, image, or a button type (submit, reset, or button). The :optional and :required CSS pseudo-classes will be applied to the field as appropriate. required is an HTML boolean attribute - it is enabled by a boolean or 'required'. Alternative capitalizations 'REQUIRED' are also accepted.

selectionDirection Character. The direction in which selection occurred. This is "forward" if the selection was made from left-to-right in an LTR locale or right-to-left in an RTL locale, or "backward" if the selection was made in the opposite direction. On platforms on which it's possible this value isn't known, the value can be "none"; for example, on macOS, the default direction is "none", then as the user begins to modify the selection using the keyboard, this will change to reflect the direction in which the selection is expanding.

selectionEnd Character. The offset into the element's text content of the last selected character. If there's no selection, this value indicates the offset to the character following the current text input cursor position (that is, the position the next character typed would occupy).

selectionStart Character. The offset into the element’s text content of the first selected character. If there’s no selection, this value indicates the offset to the character following the current text input cursor position (that is, the position the next character typed would occupy).

size Character. The initial size of the control. This value is in pixels unless the value of the type attribute is text or password, in which case it is an integer number of characters. Starting in, this attribute applies only when the type attribute is set to text, search, tel, url, email, or password, otherwise it is ignored. In addition, the size must be greater than zero. If you do not specify a size, a default value of 20 is used. simply states "the user agent should ensure that at least that many characters are visible", but different characters can have different widths.
in certain fonts. In some browsers, a certain string with x characters will not be entirely visible even if size is defined to at least x.

**spellCheck**
A value equal to: 'true', 'false' | logical. Setting the value of this attribute to true indicates that the element needs to have its spelling and grammar checked. The value default indicates that the element is to act according to a default behavior, possibly based on the parent element’s own spellcheck value. The value false indicates that the element should not be checked.

**step**
Character | numeric. Works with the min and max attributes to limit the increments at which a numeric or date-time value can be set. It can be the string any or a positive floating point number. If this attribute is not set to any, the control accepts only values at multiples of the step value greater than the minimum.

**n_submit**
Numeric. Number of times the ‘Enter’ key was pressed while the input had focus.

**n_submit_timestamp**
Numeric. Last time that ‘Enter’ was pressed.

**n_blur**
Numeric. Number of times the input lost focus.

**n_blur_timestamp**
Numeric. Last time the input lost focus.

**loading_state**
Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer.

**persistence**
Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persisted' is truthy and hasn’t changed from its previous value, a ‘value’ that the user has changed while using the app will keep that change, as long as the new ‘value’ also matches what was given originally. Used in conjunction with ‘persistence_type’.

**persisted_props**
List of a value equal to: 'value's. Properties whose user interactions will persist after refreshing the component or the page. Since only ‘value’ is allowed this prop can normally be ignored.

**persistence_type**
A value equal to: 'local', 'session', 'memory'. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

**Value**

named list of JSON elements corresponding to React.js properties and their values

**Examples**

```javascript
if (interactive() && require(dash)) {
```
library(dash)
library(dashHtmlComponents)
library(dashCoreComponents)

app <- Dash$new()

app$layout(
  htmlDiv(
    dccInput(
      placeholder = "Enter a value...",
      type = "text",
      value = ""
    )
  )
)

app$run_server()

---

dccInterval

Interval component

Description

A component that repeatedly increments a counter ‘n_intervals’ with a fixed time delay between each increment. Interval is good for triggering a component on a recurring basis. The time delay is set with the property "interval" in milliseconds.

Usage

dccInterval(id=NULL, interval=NULL, disabled=NULL, n_intervals=NULL, max_intervals=NULL)

Arguments

id Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.

interval Numeric. This component will increment the counter ‘n_intervals’ every ‘interval’ milliseconds

disabled Logical. If True, the counter will no longer update

n_intervals Numeric. Number of times the interval has passed

max_intervals Numeric. Number of times the interval will be fired. If -1, then the interval has no limit (the default) and if 0 then the interval stops running.

Value

named list of JSON elements corresponding to React.js properties and their values
Examples

if (interactive() && require(dash)) {
  library(dash)
  library(dashHtmlComponents)
  library(dashCoreComponents)
  library(plotly)

  app <- Dash$new()

  app$layout(
    htmlDiv(list(
      htmlH2('3 Second Updates'),
      dccInterval(id = '3s-interval',
                  interval= 3*1000,
                  n_intervals = 0),
      htmlDiv(list(
            dccGraph(id = 'live-graph'))
      ))
    )

  app$callback(
    output = list(
      output('live-graph', 'figure')
    ),
    params = list(
      input('3s-interval', 'n_intervals')
    ),

    update_graph <- function(n_intervals) {
      df <- data.frame(
        'time' = c(1:8),
        'value' = sample(1:8, 8),
        'value-2' = sample(1:8, 8)
      )

      bar <- animation_opts(plot_ly(
        data = df, x="time", y="value", type = "bar"),
        1000, easing = "cubic-in-out"
      )

      return(list(bar))
    }

  app$run_server()
}
**dccLink**

## Link component

### Description

Link allows you to create a clickable link within a multi-page app. For links with destinations outside the current app, `html.A` is a better component to use.

### Usage

```r
dccLink(children=NULL, id=NULL, href=NULL, refresh=NULL, className=NULL, style=NULL, title=NULL, target=NULL, loading_state=NULL)
```

### Arguments

- **children**: A list of or a singular dash component, string or number. The children of this component
- **id**: Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
- **href**: Character. The URL of a linked resource.
- **refresh**: Logical. Controls whether or not the page will refresh when the link is clicked.
- **className**: Character. Often used with CSS to style elements with common properties.
- **style**: Named list. Defines CSS styles which will override styles previously set.
- **title**: Character. Adds the title attribute to your link, which can contain supplementary information.
- **target**: Character. Specifies where to open the link reference.
- **loading_state**: Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

### Value

named list of JSON elements corresponding to React.js properties and their values

### Examples

```r
if (interactive() & require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(dashHtmlComponents)
}
app <- Dash$new()

app$layout(htmlDiv(list(
    # represents the URL bar, doesn't render anything
    dccLocation(id = 'url', refresh=FALSE),
    dccLink('Navigate to "/"', href='/
    htmlBr(),
    dccLink('Navigate to "/page-2"', href='/page-2'),
    # content will be rendered in this element
    htmlDiv(id='page-content')
    )
)
)

app$callback(output=list(id='page-content', property='children'),
params=list(input(id='url', property='pathname')),
function(pathname) {
    paste0('You are on page ', pathname)
}

app$run_server()
fullscreen Logical. Boolean that makes the spinner display full-screen

download Logical. If true, the spinner will display the component_name and prop_name while loading

className Character. Additional CSS class for the spinner root DOM node

style Named list. Additional CSS styling for the spinner root DOM node

color Character. Primary colour used for the loading spinners

loading_state Lists containing elements 'is_loading', 'prop_name', 'component_name'. Those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

Value
	named list of JSON elements corresponding to React.js properties and their values

Examples

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(dashHtmlComponents)

  app <- Dash$new()

  app$layout(htmlDiv(
    children=list(
      htmlH3("Edit text input to see loading state"),
      dccInput(id="input-1", value='Input triggers local spinner'),
      dccLoading(id="loading-1", children=list(htmlDiv(id="loading-output-1")), type="default"),
      htmlDiv(
        list(
          dccInput(id="input-2", value='Input triggers nested spinner'),
          dccLoading(
            id="loading-2",
            children=list(htmlDiv(list(htmlDiv(id="loading-output-2"))))),
            type="circle"
          )
        )
      )
    )
  ))

  app$callback(
    output = list(id='loading-output-1', property = 'children'),
    params = list(input(id = 'input-1', property = 'value')),
    function(value){
      Sys.sleep(1)
      return(value)
    }
  )
}
```


**dccLocation**

Location component

**Description**

Update and track the current window.location object through the window.history state. Use in conjunction with the `dash_core_components.Link` component to make apps with multiple pages.

**Usage**

```
dccLocation(id=NULL, pathname=NULL, search=NULL, hash=NULL, href=NULL, refresh=NULL)
```

**Arguments**

- **id** Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
- **pathname** Character. pathname in window.location - e.g., "/my/full/pathname"
- **search** Character. search in window.location - e.g., "?myargument=1"
- **hash** Character. hash in window.location - e.g., "#myhash"
- **href** Character. href in window.location - e.g., "/my/full/pathname?myargument=1#myhash"
- **refresh** Logical. Refresh the page when the location is updated?

**Value**

named list of JSON elements corresponding to React.js properties and their values
Examples

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(dashHtmlComponents)

  app <- Dash$new()

  app$layout(htmlDiv(list(
    # represents the URL bar, doesn't render anything
    dccLocation(id = 'url', refresh=FALSE),
    dccLink('Navigate to "/"', href='/'),
    htmlBr(),
    dccLink('Navigate to "/page-2"', href='page-2'),

    # content will be rendered in this element
    htmlDiv(id='page-content')
  ))
}

app$callback(output=list(id='page-content', property='children'),
params=list(
  input(id='url', property='pathname')),
function(pathname)
{
  paste0('You are on page ', pathname)
})

app$run_server()
}
```

---

dccLogoutButton  

Logout button component

Description

Logout button to submit a form post request to the `logout_url` prop. Usage is intended for dash-deployment-server authentication. DDS usage: `dcc.LogoutButton(logout_url=os.getenv('DASH_LOGOUT_URL'))`  
Custom usage: - Implement a login mechanism. - Create a flask route with a post method handler. `@app.server.route('/logout', methods=['POST'])` - The logout route should perform what’s necessary for the user to logout. - If you store the session in a cookie, clear the cookie: `rep = flask.Response(); rep.set_cookie('session', '', expires=0)`. - Create a logout button component and assign it the `logout_url` `dcc.LogoutButton(logout_url='logout')` See https://dash.plotly.com/dash-core-components/logout_button for more documentation and examples.
Usage

dccLogoutButton(id=NULL, label=NULL, logout_url=NULL, style=NULL, method=NULL, className=NULL, loading_state=NULL)

Arguments

id Character. Id of the button.
label Character. Text of the button
logout_url Character. Url to submit a post logout request.
style Named list. Style of the button
method Character. Http method to submit the logout form.
className Character. CSS class for the button.
loading_state Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

Value

named list of JSON elements corresponding to React.js properties and their values

Examples

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)

  app <- Dash$new()

  app$layout(
    dccLogoutButton(logout_url="/custom-auth/logout")
  )

  app$run_server()
}
```

---

**dccMarkdown** | **Markdown component**

**Description**

A component that renders Markdown text as specified by the GitHub Markdown spec. These component uses [react-markdown](https://rexxars.github.io/react-markdown/) under the hood.
Usage

dccMarkdown(children=NULL, id=NULL, className=NULL,
dangerously_allow_html=NULL, dedent=NULL,
highlight_config=NULL, loading_state=NULL, style=NULL)

Arguments

children  Character | list of characters. A markdown string (or array of strings) that
adheres to the CommonMark spec

id  Character. The ID of this component, used to identify dash components in call-
backs. The ID needs to be unique across all of the components in an app.

className  Character. Class name of the container element

dangerously_allow_html  Logical. A boolean to control raw HTML escaping. Setting HTML from code is
risky because it’s easy to inadvertently expose your users to a cross-site scripting

dedent  Logical. Remove matching leading whitespace from all lines. Lines that are
empty, or contain *only* whitespace, are ignored. Both spaces and tab charac-
ters are removed, but only if they match; we will not convert tabs to spaces or
vice versa.

highlight_config  Lists containing elements 'theme'. those elements have the following types:
- theme (a value equal to: 'dark', 'light'; optional): color scheme; default 'light'.
Config options for syntax highlighting.

loading_state  Lists containing elements 'is_loading', 'prop_name', 'component_name'. those
elements have the following types: - is_loading (logical; optional): determines if
the component is loading or not - prop_name (character; optional): holds which
property is loading - component_name (character; optional): holds the name of
the component that is loading. Object that holds the loading state object coming
from dash-renderer

style  Named list. User-defined inline styles for the rendered Markdown

Value

named list of JSON elements corresponding to React.js properties and their values

Examples

if (interactive() && require(dash)) {
  library(dash)
  library(dashHtmlComponents)
  library(dashCoreComponents)

  app <- Dash$new()

  app$title("dccMarkdown Syntax Highlighting Demo")

  # dccMarkdown leverages Highlight.js, which allows
dccRadioItems

RadioItems component

Description

RadioItems is a component that encapsulates several radio item inputs. The values and labels of the RadioItems is specified in the 'options' property and the selected item is specified with the 'value' property. Each radio item is rendered as an input with a surrounding label.

Usage

dccRadioItems(id=NULL, options=NULL, value=NULL, style=NULL, className=NULL, inputStyle=NULL, inputClassName=NULL, labelStyle=NULL, labelClassName=NULL, loading_state=NULL, persistence=NULL, persisted_props=NULL, persistence_type=NULL)

Arguments

id Character. The ID of this component, used to identify dash components in call-backs. The ID needs to be unique across all of the components in an app.

options List of lists containing elements 'label', 'value', 'disabled'. those elements have the following types: - label (character | numeric; required): the radio item’s label - value (character | numeric; required): the value of the radio item. this value corresponds to the items specified in the 'value' property. - disabled (logical; optional): if true, this radio item is disabled and can’t be clicked on.s. An array of options
value  Character | numeric. The currently selected value
style  Named list. The style of the container (div)
className  Character. The class of the container (div)
inputStyle  Named list. The style of the <input> radio element
inputClassName  Character. The class of the <input> radio element
labelStyle  Named list. The style of the <label> that wraps the radio input and the option’s label
labelClassName  Character. The class of the <label> that wraps the radio input and the option’s label
loading_state  Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer
persistence  Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persisted' is truthy and hasn’t changed from its previous value, a ‘value’ that the user has changed while using the app will keep that change, as long as the new ‘value’ also matches what was given originally. Used in conjunction with 'persistence_type'.
persisted_props  List of a value equal to: 'value's. Properties whose user interactions will persist after refreshing the component or the page. Since only ‘value’ is allowed this prop can normally be ignored.
persistence_type  A value equal to: 'local', 'session', 'memory'. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

Value

named list of JSON elements corresponding to React.js properties and their values

Examples

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashHtmlComponents)
  library(dashCoreComponents)

  app <- Dash$new()

  app$layout(
    htmlDiv(
      dccRadioItems(
```

options=list(
  list("label" = "New York City", "value" = "NYC"),
  list("label" = "Montreal", "value" = "MTL"),
  list("label" = "San Francisco", "value" = "SF")
),
value = "MTL"
)

app$run_server()

---

dccRangeSlider

**RangeSlider component**

**Description**

A double slider with two handles. Used for specifying a range of numerical values.

**Usage**

dccRangeSlider(id=NULL, marks=NULL, value=NULL, allowCross=NULL,
className=NULL, count=NULL, disabled=NULL, dots=NULL,
included=NULL, min=NULL, max=NULL, pushable=NULL,
tooltip=NULL, step=NULL, vertical=NULL, verticalHeight=NULL,
updatemode=NULL, loading_state=NULL, persistence=NULL,
persisted_props=NULL, persistence_type=NULL)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.</td>
</tr>
<tr>
<td>marks</td>
<td>List with named elements and values of type character</td>
</tr>
<tr>
<td>value</td>
<td>List of numerics. The value of the input</td>
</tr>
<tr>
<td>allowCross</td>
<td>Logical. allowCross could be set as true to allow those handles to cross.</td>
</tr>
<tr>
<td>className</td>
<td>Character. Additional CSS class for the root DOM node</td>
</tr>
<tr>
<td>count</td>
<td>Numeric. Determine how many ranges to render, and multiple handles will be rendered (number + 1).</td>
</tr>
<tr>
<td>disabled</td>
<td>Logical. If true, the handles can’t be moved.</td>
</tr>
</tbody>
</table>
**dccRangeSlider**

- **dots**
  Logical. When the step value is greater than 1, you can set the dots to true if you want to render the slider with dots.

- **included**
  Logical. If the value is true, it means a continuous value is included. Otherwise, it is an independent value.

- **min**
  Numeric. Minimum allowed value of the slider

- **max**
  Numeric. Maximum allowed value of the slider

- **pushable**
  Logical | numeric. pushable could be set as true to allow pushing of surrounding handles when moving an handle. When set to a number, the number will be the minimum ensured distance between handles.

- **tooltip**
  Lists containing elements 'always_visible', 'placement'. Those elements have the following types: - always_visible (logical; optional): determines whether tooltips should always be visible (as opposed to the default, visible on hover) - placement (a value equal to: 'left', 'right', 'top', 'bottom', 'topleft', 'topright', 'bottomleft', 'bottomright'; optional): determines the placement of tooltips see https://github.com/react-component/tooltip#api top/bottom* sets the _origin_ of the tooltip, so e.g. 'topleft' will in reality appear to be on the top right of the handle. Configuration for tooltips describing the current slider values

- **step**
  Numeric. Value by which increments or decrements are made

- **vertical**
  Logical. If true, the slider will be vertical

- **verticalHeight**
  Numeric. The height, in px, of the slider if it is vertical.

- **updatemode**
  A value equal to: 'mouseup', 'drag'. Determines when the component should update its value. If 'mouseup', then the slider will only trigger its value when the user has finished dragging the slider. If 'drag', then the slider will update its value continuously as it is being dragged. Only use 'drag' if your updates are fast.

- **loading_state**
  Lists containing elements 'is_loading', 'prop_name', 'component_name'. Those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

- **persistence**
  Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persistence' is truthy and hasn’t changed from its previous value, a 'value' that the user has changed while using the app will keep that change, as long as the new 'value' also matches what was given originally. Used in conjunction with 'persistence_type'.

- **persisted_props**
  List of a value equal to: 'value's. Properties whose user interactions will persist after refreshing the component or the page. Since only 'value' is allowed this prop can normally be ignored.

- **persistence_type**
  A value equal to: 'local', 'session', 'memory'. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.
**dccSlider**  

**Slider component**

**Description**

A slider component with a single handle.

**Usage**

dccSlider(id=NULL, marks=NULL, value=NULL, className=NULL, disabled=NULL, dots=NULL, included=NULL, min=NULL, max=NULL, tooltip=NULL, step=NULL, vertical=NULL, verticalHeight=NULL, updatemode=NULL, loading_state=NULL, persistence=NULL, persisted_props=NULL, persistence_type=NULL)
Arguments

id Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.

marks List with named elements and values of type character | lists containing elements 'label', 'style', those elements have the following types: - label (character; optional) - style (named list; optional). Marks on the slider. The key determines the position (a number), and the value determines what will show. If you want to set the style of a specific mark point, the value should be an object which contains style and label properties.

value Numeric. The value of the input

className Character. Additional CSS class for the root DOM node

disabled Logical. If true, the handles can’t be moved.

dots Logical. When the step value is greater than 1, you can set the dots to true if you want to render the slider with dots.

included Logical. If the value is true, it means a continuous value is included. Otherwise, it is an independent value.

min Numeric. Minimum allowed value of the slider

max Numeric. Maximum allowed value of the slider

tooltip Lists containing elements 'always_visible', 'placement'. those elements have the following types: - always_visible (logical; optional): determines whether tooltips should always be visible (as opposed to the default, visible on hover) - placement (a value equal to: 'left', 'right', 'top', 'bottom', 'topleft', 'topright', 'bottomleft', 'bottomright'; optional): determines the placement of tooltips see https://github.com/react-component/tooltip#api top/bottom* sets the _origin_ of the tooltip, so e.g. ‘topleft’ will in reality appear to be on the top right of the handle. Configuration for tooltips describing the current slider value

step Numeric. Value by which increments or decrements are made

vertical Logical. If true, the slider will be vertical

verticalHeight Numeric. The height, in px, of the slider if it is vertical.

updatemode A value equal to: 'mouseup', 'drag'. Determines when the component should update its value. If 'mouseup', then the slider will only trigger its value when the user has finished dragging the slider. If 'drag', then the slider will update its value continuously as it is being dragged. Only use ‘drag’ if your updates are fast.

loading_state Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

persistence Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persisted' is truthy and hasn’t changed from its previous value, a ‘value’ that the
user has changed while using the app will keep that change, as long as the new
'value' also matches what was given originally. Used in conjunction with 'per-
sistence_type'.

`persisted_props`
List of a value equal to: 'value's. Properties whose user interactions will persist
after refreshing the component or the page. Since only 'value' is allowed this
prop can normally be ignored.

`persistence_type`
A value equal to: 'local', 'session', 'memory'. Where persisted user changes
will be stored: memory: only kept in memory, reset on page refresh. local: win-
dow.localStorage, data is kept after the browser quit. session: window.sessionStorage,
data is cleared once the browser quit.

Value

named list of JSON elements corresponding to React.js properties and their values

Examples

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(dashHtmlComponents)

  app <- Dash$new()

  app$layout(
    htmlDiv(
      list(
        dccSlider(
          id = "slider-input",
          min = -5,
          max = 10,
          step = 0.5,
          value = -3
        ),
        htmlDiv(
          id = "slider-output",
          children = "Make a selection on the slider to see the value appear here."
        )
      )
    )
  )

  app$callback(
    output("slider-output", "children"),
    list(input("slider-input", "value")),
    function(value) {
      return(paste0("You have chosen ", value, " on the slider above."))
    }
  )
}
```
Description

Easily keep data on the client side with this component. The data is not inserted in the DOM. Data can be in memory, localStorage or sessionStorage. The data will be kept with the id as key.

Usage

dccStore(id=NULL, storage_type=NULL, data=NULL, clear_data=NULL, modified_timestamp=NULL)

Arguments

id Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.

storage_type A value equal to: 'local', 'session', 'memory'. The type of the web storage. memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

data Named list | unnamed list | numeric | character | logical. The stored data for the id.

clear_data Logical. Set to true to remove the data contained in 'data_key'.

modified_timestamp Numeric. The last time the storage was modified.

Value

named list of JSON elements corresponding to React.js properties and their values

Examples

```r
if (interactive() && require(dash)) {
  library(dashCoreComponents)
  library(dashHtmlComponents)
  library(dash)

  app <- Dash$new()

  app$layout(htmlDiv(list(
    # The memory store reverts to the default on every page refresh
dccStore(id='memory'),
  
```
# The local store will take the initial data
# only the first time the page is loaded
# and keep it until it is cleared.
dccStore(id='local', storage_type='local'),
# Same as the local store but will lose the data
# when the browser/tab closes.
dccStore(id='session', storage_type='session'),
htmlTable(list(
    htmlThead(list(
        htmlTr(htmlTh('Click to store in:', colSpan='3'))),
        htmlTr(list(
            htmlTh(htmlButton('memory', id='memory-button')),
            htmlTh(htmlButton('localStorage', id='local-button')),
            htmlTh(htmlButton('sessionStorage', id='session-button'))
        ))),
        htmlTr(list(
            htmlTh('Memory clicks'),
            htmlTh('Local clicks'),
            htmlTh('Session clicks')
        ))),
    htmlTbody(list(
        htmlTr(list(
            htmlTd(0, id='memory-clicks'),
            htmlTd(0, id='local-clicks'),
            htmlTd(0, id='session-clicks')
        )))
))

for (i in c('memory', 'local', 'session')) {
    app$callback(
        output(id = i, property = 'data'),
        params = list(
            input(id = paste0(i, '-button'), property = 'n_clicks'),
            state(id = i, property = 'data')
        ),
        function(n_clicks, data){
            if(is.null(n_clicks)){
                return()
            }
            if(is.null(data[[1]])){
                data = list('clicks' = 0)
            } else{
                data = data
            }
            data['clicks'] = data$clicks + 1
            return(data)
        }
    )
}
for (i in c('memory', 'local', 'session')) {
  app$callback(
    output(id = paste0(i, '-clicks'), property = 'children'),
    params = list(
      input(id = i, property = 'modified_timestamp'),
      state(id = i, property = 'data'))
  ),
  function(ts, data){
    if(is.null(ts)){
      return()
    }
    if(is.null(data[[1]])){
      data = list()
    } else {
      data = data
    }
    return(data$clicks[[1]])
  }
}

app$run_server()


dccTab  

**Tab component**

**Description**

Part of dcc.Tabs - this is the child Tab component used to render a tabbed page. Its children will be set as the content of that tab, which if clicked will become visible.

**Usage**

```
dccTab(children=NULL, id=NULL, label=NULL, value=NULL, disabled=NULL, disabled_style=NULL, disabled_className=NULL, className=NULL, selected_className=NULL, style=NULL, selected_style=NULL, loading_state=NULL)
```

**Arguments**

- **children**  
  A list of or a singular dash component, string or number. The content of the tab - will only be displayed if this tab is selected

- **id**  
  Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.

- **label**  
  Character. The tab’s label

- **value**  
  Character. Value for determining which Tab is currently selected

- **disabled**  
  Logical. Determines if tab is disabled or not - defaults to false
**dccTab**

- **disabled_style** Named list. Overrides the default (inline) styles when disabled.
- **disabled_className** Character. Appends a class to the Tab component when it is disabled.
- **className** Character. Appends a class to the Tab component.
- **selected_className** Character. Appends a class to the Tab component when it is selected.
- **style** Named list. Overrides the default (inline) styles for the Tab component.
- **selected_style** Named list. Overrides the default (inline) styles for the Tab component when it is selected.
- **loading_state** Lists containing elements 'is_loading', 'prop_name', 'component_name'. Those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

**Value**

Named list of JSON elements corresponding to React.js properties and their values

**Examples**

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(dashHtmlComponents)

  app <- Dash$new()

  app$layout(htmlDiv(list(
    dccTabs(id="tabs", value='tab-1', children=list(
      dccTab(label='Tab one', value='tab-1'),
      dccTab(label='Tab two', value='tab-2')
    ),
    htmlDiv(id='tabs-content')
  )))

  app$callback(output('tabs-content', 'children'),
              params = list(input('tabs', 'value')),
              function(tab){
                if(tab == 'tab-1'){
                  return(htmlDiv(list(
                    htmlH3('Tab content 1'))
                  )))
                }
                else if(tab == 'tab-2'){
                  return(htmlDiv(list(
                    htmlH3('Tab content 2'))
                  )))
              })

```
dccTabs

Tabs component

Description

A Dash component that lets you render pages with tabs - the Tabs component’s children can be dcc.Tab components, which can hold a label that will be displayed as a tab, and can in turn hold children components that will be that tab’s content.

Usage

dccTabs(children=NULL, id=NULL, value=NULL, className=NULL, content_className=NULL, parent_className=NULL, style=NULL, parent_style=NULL, content_style=NULL, vertical=NULL, mobile_breakpoint=NULL, colors=NULL, loading_state=NULL, persistence=NULL, persisted_props=NULL, persistence_type=NULL)

Arguments

children List of a list of or a singular dash component, string or numbers | a list of or a singular dash component, string or number. Array that holds Tab components
id Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
value Character. The value of the currently selected Tab
className Character. Appends a class to the Tabs container holding the individual Tab components.
content_className Character. Appends a class to the Tab content container holding the children of the Tab that is selected.
parent_className Character. Appends a class to the top-level parent container holding both the Tabs container and the content container.
style Named list. Appends (inline) styles to the Tabs container holding the individual Tab components.
parent_style Named list. Appends (inline) styles to the top-level parent container holding both the Tabs container and the content container.
content_style Named list. Appends (inline) styles to the tab content container holding the children of the Tab that is selected.
**vertical** Logical. Renders the tabs vertically (on the side)

**mobile_breakpoint** Numeric. Breakpoint at which tabs are rendered full width (can be 0 if you don’t want full width tabs on mobile)

**colors** Lists containing elements ‘border’, ‘primary’, ‘background’. those elements have the following types: - border (character; optional) - primary (character; optional) - background (character; optional). Holds the colors used by the Tabs and Tab components. If you set these, you should specify colors for all properties, so: colors: border: ‘#d6d6d6’, primary: ‘#1975FA’, background: ‘#f9f9f9’

**loading_state** Lists containing elements ‘is_loading’, ‘prop_name’, ‘component_name’. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

**persistence** Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If ‘persisted’ is truthy and hasn’t changed from its previous value, a ‘value’ that the user has changed while using the app will keep that change, as long as the new ‘value’ also matches what was given originally. Used in conjunction with ‘persistence_type’.

**persisted_props** List of a value equal to: ‘value’s. Properties whose user interactions will persist after refreshing the component or the page. Since only ‘value’ is allowed this prop can normally be ignored.

**persistence_type** A value equal to: ‘local’, ‘session’, ‘memory’. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

**Value**

named list of JSON elements corresponding to React.js properties and their values

**Examples**

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(dashHtmlComponents)

  app <- Dash$new()

  app$layout(htmlDiv(list(
    dccTabs(id="tabs", value='tab-1', children=list(
      dccTab(label='Tab one', value='tab-1'),
      dccTab(label='Tab two', value='tab-2')
    )))
```
dccTextarea

Description

A basic HTML textarea for entering multiline text.

Usage

dccTextarea(id=NULL, value=NULL, autoFocus=NULL, cols=NULL, disabled=NULL, form=NULL, maxLength=NULL, minLength=NULL, name=NULL, placeholder=NULL, readOnly=NULL, required=NULL, rows=NULL, wrap=NULL, accessKey=NULL, className=NULL, contentEditable=NULL, contextMenu=NULL, dir=NULL, draggable=NULL, hidden=NULL, lang=NULL, spellCheck=NULL, style=NULL, tabIndex=NULL, title=NULL, n_blur=NULL, n_blur_timestamp=NULL, n_clicks=NULL, n_clicks_timestamp=NULL, loading_state=NULL, persistence=NULL, persisted_props=NULL, persistence_type=NULL)

Arguments

id Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Character. The value of the textarea.</td>
</tr>
<tr>
<td>autofocus</td>
<td>Character. The element should be automatically focused after the page loaded.</td>
</tr>
<tr>
<td>cols</td>
<td>Character</td>
</tr>
<tr>
<td>disabled</td>
<td>Character</td>
</tr>
<tr>
<td>form</td>
<td>Character. Indicates the form that is the owner of the element.</td>
</tr>
<tr>
<td>maxLength</td>
<td>Character</td>
</tr>
<tr>
<td>minLength</td>
<td>Character</td>
</tr>
<tr>
<td>name</td>
<td>Character. Name of the element. For example used by the server to identify the fields in form submits.</td>
</tr>
<tr>
<td>placeholder</td>
<td>Character. Provides a hint to the user of what can be entered in the field.</td>
</tr>
<tr>
<td>readOnly</td>
<td>Logical</td>
</tr>
<tr>
<td>required</td>
<td>A value equal to: 'required', 'required'</td>
</tr>
<tr>
<td>rows</td>
<td>Character</td>
</tr>
<tr>
<td>wrap</td>
<td>Character. Indicates whether the text should be wrapped.</td>
</tr>
<tr>
<td>accessKey</td>
<td>Character. Defines a keyboard shortcut to activate or add focus to the element.</td>
</tr>
<tr>
<td>className</td>
<td>Character. Often used with CSS to style elements with common properties.</td>
</tr>
<tr>
<td>contentEditable</td>
<td>Character</td>
</tr>
<tr>
<td>contextMenu</td>
<td>Character. Defines the ID of a &lt;menu&gt; element which will serve as the element’s context menu.</td>
</tr>
<tr>
<td>dir</td>
<td>Character. Defines the text direction. Allowed values are ltr (Left-To-Right) or rtl (Right-To-Left)</td>
</tr>
<tr>
<td>draggable</td>
<td>A value equal to: 'true', 'false'</td>
</tr>
<tr>
<td>hidden</td>
<td>Character. Prevents rendering of given element, while keeping child elements, e.g. script elements, active.</td>
</tr>
<tr>
<td>lang</td>
<td>Character. Defines the language used in the element.</td>
</tr>
<tr>
<td>spellCheck</td>
<td>A value equal to: 'true', 'false'</td>
</tr>
<tr>
<td>style</td>
<td>Named list. Defines CSS styles which will override styles previously set.</td>
</tr>
<tr>
<td>tabIndex</td>
<td>Character</td>
</tr>
<tr>
<td>title</td>
<td>Character. Text to be displayed in a tooltip when hovering over the element.</td>
</tr>
</tbody>
</table>
n_blur Numeric. Number of times the textarea lost focus.
n_blur_timestamp Numeric. Last time the textarea lost focus.
n_clicks Numeric. Number of times the textarea has been clicked.
n_clicks_timestamp Numeric. Last time the textarea was clicked.

loading_state Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

persistence Logical | character | numeric. Used to allow user interactions in this component to be persisted when the component - or the page - is refreshed. If 'persisted' is truthy and hasn't changed from its previous value, a 'value' that the user has changed while using the app will keep that change, as long as the new 'value' also matches what was given originally. Used in conjunction with 'persistence_type'.

persisted_props List of a value equal to: 'value's. Properties whose user interactions will persist after refreshing the component or the page. Since only 'value' is allowed this prop can normally be ignored.

persistence_type A value equal to: 'local', 'session', 'memory'. Where persisted user changes will be stored: memory: only kept in memory, reset on page refresh. local: window.localStorage, data is kept after the browser quit. session: window.sessionStorage, data is cleared once the browser quit.

Value

named list of JSON elements corresponding to React.js properties and their values

Examples

```r
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)

  app <- Dash$new()

  app$layout(
    htmlDiv(
      dccTextarea(
        placeholder = 'Enter a value...',
        value = 'This is a TextArea component'
      )
    )
  )
}
```
Description

Upload components allow your app to accept user-uploaded files via drag'n'drop

Usage

dccUpload(children=NULL, id=NULL, contents=NULL, filename=NULL,
last_modified=NULL, accept=NULL, disabled=NULL,
disable_click=NULL, max_size=NULL, min_size=NULL,
multiple=NULL, className=NULL, className_active=NULL,
className_reject=NULL, className_disabled=NULL, style=NULL,
style_active=NULL, style_reject=NULL, style_disabled=NULL,
loading_state=NULL)

Arguments

children A list of or a singular dash component, string or number | character. Contents of the upload component
id Character. The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
contents Character | list of characters. The contents of the uploaded file as a binary string
filename Character | list of characters. The name of the file(s) that was(were) uploaded. Note that this does not include the path of the file (for security reasons).
last_modified Numeric | list of numerics. The last modified date of the file that was uploaded in unix time (seconds since 1970).
accept Character. Allow specific types of files. See https://github.com/okonet/attr-accept for more information. Keep in mind that mime type determination is not reliable across platforms. CSV files, for example, are reported as text/plain under macOS but as application/vnd.ms-excel under Windows. In some cases there might not be a mime type set at all. See: https://github.com/react-dropzone/react-dropzone/issues/276
disabled Logical. Enable/disable the upload component entirely
disable_click Logical. Disallow clicking on the component to open the file dialog
max_size Numeric. Maximum file size. If ‘-1’, then infinite
min_size Numeric. Minimum file size
multiple Logical. Allow dropping multiple files
className Character. HTML class name of the component
className_active
Character. HTML class name of the component while active

className_reject
Character. HTML class name of the component if rejected

className_disabled
Character. HTML class name of the component if disabled

style
Named list. CSS styles to apply

style_active
Named list. CSS styles to apply while active

style_reject
Named list. CSS styles if rejected

style_disabled
Named list. CSS styles if disabled

loading_state
Lists containing elements 'is_loading', 'prop_name', 'component_name'. those elements have the following types: - is_loading (logical; optional): determines if the component is loading or not - prop_name (character; optional): holds which property is loading - component_name (character; optional): holds the name of the component that is loading. Object that holds the loading state object coming from dash-renderer

Value
named list of JSON elements corresponding to React.js properties and their values

Examples
if (interactive() && require(dash)) {
  library(dash)
  library(dashCoreComponents)
  library(dashHtmlComponents)
  library(jsonlite)

  app <- Dash$new()

  app$layout(htmlDiv(list(
    dccUpload(
      id='upload-image',
      children=htmlDiv(list(
        'Drag and Drop or ','
        htmlA('Select Files')
      ),
      style=list(
        'height'='60px',
        'lineHeight'='60px',
        'borderWidth'='1px',
        'borderStyle'='dashed',
        'borderRadius'='5px',
        'textAlign'='center',
        'margin'='10px'
      ),
      # Allow multiple files to be uploaded
      multiple=TRUE
    )
  )
})}
```
dccUpload

htmlDiv(id='output-image-upload')

parse_content = function(contents, filename, date) {
  return(htmlDiv(list(
    htmlH5(filename),
    htmlH6(as.POSIXct(date, origin="1970-01-01")),
    htmlImg(src=contents),
    htmlHr(),
    htmlDiv('Raw Content'),
    htmlPre(paste(substr(toJSON(contents), 1, 100), "..."), style=list(  
      whiteSpace='pre-wrap',
      wordBreak='break-all'
    ))
  )))
}

app$callback(  
  output = list(id='output-image-upload', property = 'children'),
  params = list(input(id = 'upload-image', property = 'contents'),
                 state(id = 'upload-image', property = 'filename'),
                 state(id = 'upload-image', property = 'last_modified')),
  function(list_of_contents, list_of_names, list_of_dates) {  
    if (!is.null(list_of_contents) && !is.null(list_of_names) && !is.null(list_of_dates[[1]])){
      children = lapply(1:length(list_of_contents), function(x){
        parse_content(list_of_contents[[x]], list_of_names[[x]], list_of_dates[[x]])
      })
    } else {
      children = "Upload a file to see the raw data."
    }
    return(children)
  }
)

app$run_server()
```
Index

dashCoreComponents
  (dashCoreComponents-package), 2
dashCoreComponents-package, 2
dccChecklist, 3
dccConfirmDialog, 4
dccConfirmDialogProvider, 6
dccDatePickerRange, 8
dccDatePickerSingle, 11
dccDropdown, 13
dccGraph, 15
dccInput, 20
dccInterval, 24
dccLink, 26
dccLoading, 27
dccLocation, 29
dccLogoutButton, 30
dccMarkdown, 31
dccRadioItems, 33
dccRangeSlider, 35
dccSlider, 37
dccStore, 40
dccTab, 42
dccTabs, 44
dccTextarea, 46
dccUpload, 49