

Using DT in Shiny

You can render a table widget in shiny, and obtain some information about its current state, such as the indices of the rows on the current page.

1. A Minimal Example

Here is a minimal example of (client-side) DataTables in Shiny using the convenience functions `DTOutput()` and `renderDT()` :

```
library(shiny)
library(DT)
shinyApp(
  ui = fluidPage(DTOutput('tbl')),
  server = function(input, output) {
    output$tbl = renderDT(
      iris, options = list(lengthChange = FALSE)
    )
  }
)
```

Note that in **DT**, `DTOutput()` is an alias of `dataTableOutput()`, and `renderDT()` is an alias of `renderDataTable()`. You are recommended to use `DTOutput()` and `renderDT()` to avoid possible collisions with functions of the same names in **shiny** (`shiny::dataTableOutput()` and `shiny::renderDataTable()`).

If you have used the `renderDataTable()` function in **shiny** before, please note you should use `JS()` instead of `I()` to write literal JavaScript code in options.

```
library(shiny)
renderDataTable(iris, options = list(
  pageLength = 5,
  initComplete = I('function(setting, json) { alert("done"); }')
))
```

```
library(DT)
renderDT(
  iris, options = list(
    pageLength = 5,
    initComplete = JS('function(setting, json) { alert("done"); }')
  )
)
```

The `renderDataTable()` function in **shiny** uses server-side processing (`server.html`) and it has no client-side support. **DT** supports both ways; the default is server-side processing, but you can switch to client-side by calling `DT::renderDT()` with a `server = FALSE` argument. When the data object is relatively large, do not use `server = FALSE`, otherwise it will be too slow to render the table in the web browser, and the table will not be very responsive, either.

The first argument of `DT::renderDT()` can be either a data object or a table widget returned by `datatable()`. The latter form can be useful when you want to manipulate the widget before rendering it in Shiny, e.g. you may apply a formatting function to a table widget:

```
DT::renderDT({
  datatable(iris) %>% formatStyle(
    'Sepal.Width',
    backgroundColor = styleInterval(3.4, c('gray', 'yellow'))
  )
})
```

2. Interaction with Shiny

There are some information exposed to Shiny from the table widget as you interact with the table in Shiny. In the following sections, we use `tableId` to denote the output id of the table (i.e. the `outputId` in `DTOutput()`). You need to replace `tableId` with the actual id of the table in your own app.

2.1 Selecting Rows/Columns/Cells

You may select rows, columns, or cells in the table, and obtain the indices of the selected objects. See this Shiny app (<https://yihui.shinyapps.io/DT-selection>) for a comprehensive example (you can find its source code under `system.file('examples', 'DT-selection', package = 'DT')`).

2.1.1 Row Selection

The feature of row selection is enabled automatically when a table is embedded in a Shiny app. You can click on a row to toggle its selection status, and the indices of the selected rows are available through `input$tableId_rows_selected`. See a live example here (<https://yihui.shinyapps.io/DT-rows/>). You can disable row selection by `datatable(..., selection = 'none')`, or use the single selection mode by `selection = 'single'`. The default selection mode is multiple selections (try the table below).

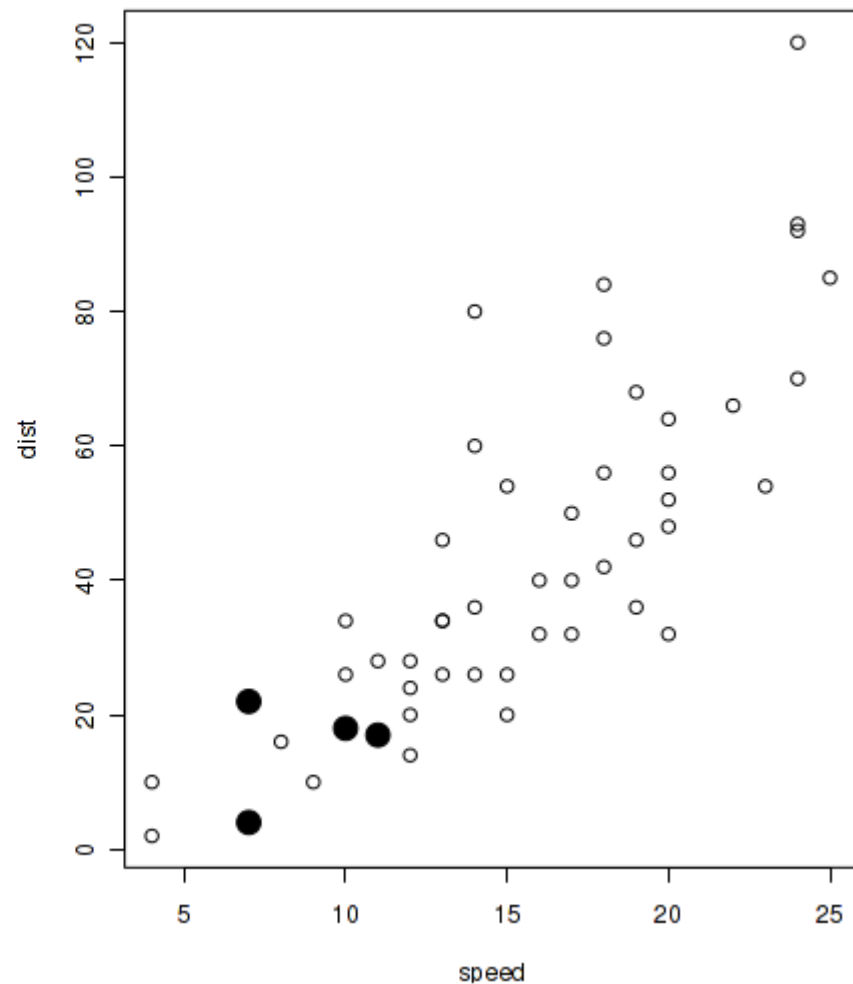
	Sepal.Length ▴ ▾	Sepal.Width ▴ ▾	Petal.Length ▴ ▾	Petal.Width ▴ ▾	Species ▴ ▾
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
# you can find the example locally in the DT package
app = system.file('examples', 'DT-rows', package = 'DT')
shiny::runApp(app)
```

Search:

	speed 	dist 
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10
7	10	18
8	10	26
9	10	34
10	11	17

Previous 1 2 3 4 5 Next



(<https://yihui.shinyapps.io/DT-rows/>)

You can also obtain the index of the last clicked row via `input$tableId_row_last_clicked` (it may be `NULL` initially and when the table is redrawn).

2.1.2 Column Selection

Row selection is the default mode in **DT**. You can turn on column selection using `datatable(..., selection = list(target = 'column'))`. In this case, you can click on any cell to select a column, and the (numeric) indices of the selected columns will be available in `input$tableId_columns_selected`.

You may also select rows and columns simultaneously using `target = 'row+column'`. In this case, column selection is achieved by clicking on the table footer. Clicking on the table body will select/deselect rows.

2.1.3 Cell Selection

Cell selection is enabled by `target = 'cell'` in the `selection` argument. The indices of selected cells are available through `input$tableId_cells_selected` as a matrix of two columns: each row of the matrix contains the row index and column index of a selected cell. When no cells are selected, the matrix has zero rows.

2.1.4 Pre-selection

The `selection` argument of `datatable()` can also include a component `selected` to specify which rows/columns/cells to be pre-selected when the table is initialized. When `target = 'row'` or `'column'`, `selected` is a vector of row or column indices. For the case of `target = 'row+column'`, `selected` should be a list of two components `rows` and `cols`, e.g. `list(rows = c(1, 2, 4, 9), cols = c(1, 3))`. For `target = 'cell'`, it should be a matrix of two columns: the first column is the row indices of selected cells, and the second column is the column indices.

2.2 DataTables Information



As you interact with the table (e.g. sort columns, search the table, or navigate through pages), **DT** will expose some information about the current state of the table to Shiny. At the moment, these information are available in the `input` object of the Shiny server function (suppose the table output id is `tableId`):

- `input$tableId_cell_clicked`: information about the cell being clicked of the form `list(row = row_index, col = column_index, value = cell_value)` (example (<https://yihui.shinyapps.io/DT-click/>))
- `input$tableId_rows_current`: the indices of rows on the current page
- `input$tableId_rows_all`: the indices of rows on all pages (after the table is filtered by the search strings)
- `input$tableId_search`: the global search string
- `input$tableId_search_columns`: the vector of column search strings when column filters are enabled
- `input$tableId_state`: the state information of the table (a list containing the search string, ordering and paging information; it is available only if the option `stateSave = TRUE` is applied to the table)

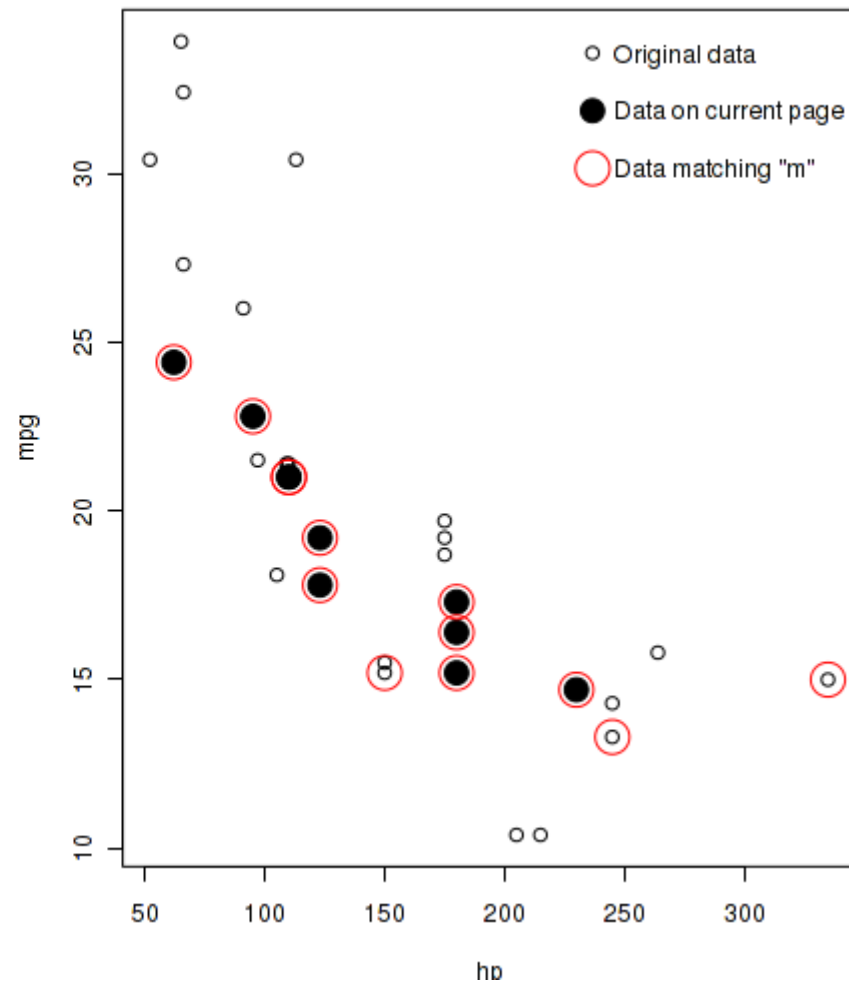
This example (<https://yihui.shinyapps.io/DT-info/>) shows how you can make use of these information to highlight points on a scatterplot and download the filtered data.

```
app = system.file('examples', 'DT-info', package = 'DT')
shiny::runApp(app)
```

Search:

	hp 	mpg 
Mazda RX4	110	21
Mazda RX4 Wag	110	21
Merc 240D	62	24.4
Merc 230	95	22.8
Merc 280	123	19.2
Merc 280C	123	17.8
Merc 450SE	180	16.4
Merc 450SL	180	17.3
Merc 450SLC	180	15.2
Chrysler Imperial	230	14.7

Previous 1 2 Next



(<https://yihui.shinyapps.io/DT-info/>)

2.3 Manipulate An Existing DataTables Instance

After a table has been rendered in a Shiny app, you can use the proxy object returned from `dataTableProxy()` to manipulate it. Currently supported methods are `selectRows()`, `selectColumns()`, `selectCells()`, `selectPage()`, and `addRow()`. See this Shiny app (<https://yihui.shinyapps.io/DT-proxy>) for an example of using these methods to update a table.

Sometimes the data behind the table may change, and you do not want to regenerate the whole table, but only want to replace the data values. The state of the table (sorting, filtering, and pagination) can be preserved after you replace the data with `replaceData()`. Here is a short example:

```
app = system.file('examples', 'DT-reload', package = 'DT')
shiny::runApp(app)
```

2.4 Edit Tables

You can edit a table via the `editable` argument of `datatable()`. After you finish editing, you can obtain the row/column indices and the new values of the cells that were edited via `input$tableId_cell_info`. See this Shiny app (<https://yihui.shinyapps.io/DT-edit/>) for more concrete examples.

Table 6: server-side processing (editable = "row")

Show 5 entries

Search:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Date
1	5.1	3.5	1.4	0.2	setosa	2019-04-04T21:
2	4.9	3	1.4	0.2	setosa	2019-04-04T21:53:03Z
3	4.7	3.2	1.3	0.2	setosa	2019-04-04T21:53:04Z
4	4.6	3.1	1.5	0.2	setosa	2019-04-04T21:53:05Z
5	5	3.6	1.4	0.2	setosa	2019-04-04T21:53:06Z

Showing 1 to 5 of 150 entries

Previous

1

2

3

4

5

...

30

Next

Table 7: server-side processing (editable = "column")

Show 5 entries

Search:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Date
1	5.1	3.5	1.4	0.2	setosa	2019-04-04T21:53:02Z
2	4.9	3	1.4	0.2	setosa	2019-04-04T21:53:03Z

3	4.7	3.2	1.3	0.2	setosa	2019-04-04T21:53:04Z
4	4.6	3.1	1.5	0.2	setosa	2019-04-04T21:53:05Z
5	5	3.6	1.4	0.2	setosa	2019-04-04T21:53:06Z

Showing 1 to 5 of 150 entries

Previous 1 2 3 4 5 ... 30 Next

Table 8: server-side processing (editable = "all")

Show 5 entries

Search:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Date
1	5.1	3.5	1.4	0.2	setosa	2019-04-04T21:
2	4.9	3	1.4	0.2	setosa	2019-04-04T21:
3	4.7	3.2	1.3	0.2	setosa	2019-04-04T21:
4	4.6	3.1	1.5	0.2	setosa	2019-04-04T21:
5	5	3.6	1.4	0.2	setosa	2019-04-04T21:

(<https://yihui.shinyapps.io/DT-edit/>)

3. More Examples

Below is a list of miscellaneous examples:

1. Radio Buttons in Tables (011-radio.html)
2. Useful callbacks for DT (in Shiny) (<https://laustep.github.io/stlahblog/posts/DTcallbacks.html>)

3. Plotting the columns of a Datatable (<https://laustep.github.io/stlahblog/posts/PlotDatatableColumns.html>)