

Quick Start

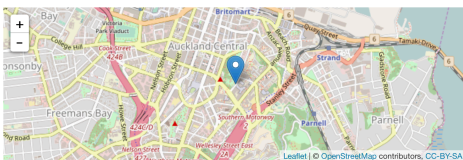
Installation

Use `install.packages("leaflet")` to install the package or directly from Github `devtools::install_github("rstudio/leaflet")`.

First Map

```
m <- leaflet() %>% # leaflet works with the pipe operator
  addTiles() %>% # setup the default OpenStreetMap map tiles
  addMarkers(lng = 174.768, lat = -36.852, popup = "The birthplace of R")
# add a single point layer
```

m



Map Widget

Initialization

```
m <- leaflet(options = leafletOptions(...))
  center # Initial geographic center of the map
  zoom # Initial map zoom level
  minZoom # Minimum zoom level of the map
  maxZoom # Maximum zoom level of the map
```

Map Methods

```
m %>% setView(lng, lat, zoom, options = list())
# Set the view of the map (center and zoom level)
m %>% fitBounds(lng1, lat1, lng2, lat2)
# Fit the view into the rectangle [lng1, lat1] - [lng2, lat2]
m %>% clearBounds()
# Clear the bound, automatically determine from the map elements
```

Data Object

Both `leaflet()` and the `map` layers have an optional data parameter that is designed to receive spatial data with the following formats:

Base R *The arguments of all layers take normal R objects:*
`df <- data.frame(lat = ..., lng = ...)`
`leaflet(df) %>% addTiles() %>% addCircles()`

sp package
`library(sp)` *Useful functions:*
`SpatialPoints`, `SpatialLines`, `SpatialPolygons`, ...
`library(maps)` *Build a map of states with colors:*
`mapStates <- map("state", fill = TRUE, plot = FALSE)`
`leaflet(mapStates) %>% addTiles() %>%`
`addPolygons(fillColor = topo.colors(10), alpha =`
`NULL, stroke = FALSE)`

Markers

Use markers to call out points, express locations with latitude/longitude coordinates, appear as icons or as circles.
 Data come from vectors or assigned data frame, or `sp` package objects.

Icon Markers

Regular Icons: default and simple

`addMarkers(lng, lat, popup, label)` *add basic icon markers*
`makeIcon/icons` (iconUrl, iconWidth, iconHeight, iconAnchorX, iconAnchorY, shadowUrl, shadowWidth, shadowHeight, ...) *customize marker icons*
`iconList()` *create a list of icons*

Awesome Icons: customizable with colors and icons

`addAwesomeMarkers`, `makeAwesomeIcon`, `awesomeIcons`, `awesomeIconList`
Marker Clusters: option of addMarkers()
`clusterOptions = markerClusterOptions()`
`freezeAtZoom` *Freeze the cluster at assigned zoom level*

Circle Markers

`addCircleMarkers(color, radius, stroke, opacity, ...)`
Customize their color, radius, stroke, opacity

Popups and Labels

`addPopups(lng, lat, ...content..., options)` *Add standalone popups*
`options = popupOptions(closeButton=FALSE)`
`addMarkers(..., popup, ...)` *Show popups with markers or shapes*
`addMarkers(..., label, labelOptions...)` *Show labels with markers or shapes*
`labelOptions = labelOptions(noHide, textSize, direction, style)`
`addLabelOnlyMarkers()` *Add labels without markers*

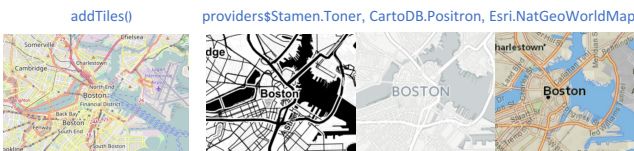
Lines and Shapes

Polygons and Polylines

`addPolygons(color, weight=1, smoothFactor=0.5, opacity=1.0, fillOpacity=0.5, fillColor = ~colorQuantile("YlOrRd", ALAND)(ALAND), highlightOptions, ...)`
`highlightOptions(color, weight=2, bringToFront=TRUE)` *highlight shapes*
 Use `rmapshaper::ms_simplify` to simplify complex shapes
Circles `addCircles(lng, lat, weight=1, radius, ...)`
Rectangles `addRectangles(lng1, lat1, lng2, lat2, fillColor="transparent", ...)`

Basemaps

`addTiles()` `providers$Stamen.Toner`, `CartoDB.Positron`, `Esri.NatGeoWorldMap`



Default Tiles
 Use `addTiles()` to add a custom map tile URL template, use `addWMSTiles()` to add WMS (Web Map Service) tiles

Third-Party Tiles `addProviderTiles()`
 Use `addTiles()` to add a custom map tile URL template, use `addWMSTiles()` to add WMS (Web Map Service) tiles

GeoJSON and TopoJSON

There are two options to use the GeoJSON/TopoJSON data.

* To read into `sp` objects with the `geojsonio` or `rgdal` package:
`geojsonio::geojson_read(..., what="sp")` `rgdal::readOGR(..., "OGRGeoJSON")`
 * Or to use the `addGeoJSON()` and `addTopoJSON()` functions:
`addTopoJSON/addGeoJSON(... weight, color, fill, opacity, fillOpacity...)`
 Styles can also be tuned separately with a `style: {...}` object.

Other packages including `RUSONIO` and `jsonlite` can help fast parse or generate the data needed.

Shiny Integration

To integrate a Leaflet map into an app:

* In the UI, call `leafletOutput("name")`
 * On the server side, assign a `renderLeaflet(...)` call to the output
 * Inside the `renderLeaflet` expression, return a Leaflet map object

Modification

To modify an existing map or add incremental changes to the map, you can use `leafletProxy()`. This should be performed in an observer on the server side.

Other useful functions to edit your map:

`fitBounds(0, 0, 11, 11)` *similar to setView*
fit the view to within these bounds
`addCircles(1:10, 1:10, layerId = LETTERS[1:10])`
create circles with layerIds of "A", "B", "C"...
`removeShape(c("B", "F"))` *remove some of the circles*
`clearShapes()` *clear all circles (and other shapes)*

Inputs/Events

Object Events

Object event names generally use this pattern:

`inputs$MAPID_OBJECTCATEGORY_EVENTNAME`.

Trigger an event changes the value of the Shiny input at this variable.

Valid values for `OBJECTCATEGORY` are *marker*, *shape*, *geojson* and *topojson*.

Valid values for `EVENTNAME` are *click*, *mouseover* and *mouseout*.

All of these events are set to either `NULL` if the event has never happened, or a `list()` that includes:

* `lat` The latitude of the object, if available; otherwise, the mouse cursor
 * `lng` The longitude of the object, if available; otherwise, the mouse cursor
 * `id` The `layerId`, if any

GeoJSON events also include additional properties:

* `featureId` The feature ID, if any
 * `properties` The feature properties

Map Events

`inputs$MAPID_click` *when the map background or basemap is clicked value -- a list with lat and lng*
`inputs$MAPID_bounds` *provide the lat/lng bounds of the visible map area value -- a list with north, east, south and west*
`inputs$MAPID_zoom` *an integer indicates the zoom level*