Package ‘amapro’

January 28, 2026

Type Package

Title Thin Wrapper for Mapping Library 'AMap'('Gaode')

Date 2026-01-27

Version 0.1.4

Author Larry Helgason [aut, cre, cph]

Maintainer Larry Helgason <larry@helgasoft.com>

Description Build and control interactive 2D and 3D maps with 'R/Shiny'. Lean set of powerful commands wrapping native calls to 'AMap' <https://lbs.amap.com/api/jsapi-v2/summary/>. Deliver rich mapping functionality with minimal overhead.

URL https://github.com/helgasoft/amapro/,
https://helgasoft.github.io/amapro/

BugReports https://github.com/helgasoft/amapro/issues/

Depends R (>= 4.1.0)

Imports htmlwidgets, tcltk (>= 4.1.0)

Suggests shiny (>= 1.7.0), shinyjs, shinythemes, jsonlite, rmarkdown, knitr, testthat (>= 3.0.0)

License Apache License (>= 2)

Encoding UTF-8

Language en-US

VignetteBuilder knitr

RoxygenNote 7.3.3

NeedsCompilation no

Repository CRAN

Date/Publication 2026-01-28 03:20:02 UTC

Contents

`- Introduction -` *Introduction*

Description

Essential information, tips and tricks

Details

Welcoming JavaScript library AMap into the world of R. AMap is an advanced mapping library made in China and widely used there. It features 2D/3D animation, supports a multitude of layers and markers, data import, flyover playback, etc. Library *amapro* let you control AMap from R and Shiny. It uses AMap's native commands/parameters wrapped in just a few commands.

Translation

AMap's documentation is in Chinese and most links here make reference to it. If you happen *not* to know Chinese, it is convenient to set your browser to auto-translate. This will help a little or a lot depending on the website/page structure. One can also copy/paste text to Google translate.

Installation

Install **amapro** from Github with `remotes::install_github("helgasoft/amapro")` CRAN version also available but usually outdated.

Run with the following commands `library(amapro); am.init()` A pop-up dialog will ask for an **API key** (shows once, will not be repeated). API key is obtained through registration, expecting you to provide a Chinese phone number for SMS verification. How to get an API key if you reside out of China?

- ask a friend from China to help, or hire a local freelancer
- search the web for a shared key
- use a temporary Chinese phone number from sites like *sms24.me*, *turtle-sms.xyz*, etc. However most are probably blacklisted as the registration page shows them as *'already registered'*.
- select temporarily the 'demo' option, without guarantee to work in the long run

Shiny Demo

Interactive, hands-on showcase of many library features. Activate with the following command: `library(amapro); demo(am.shiny)`

API links

amapro is based on version 2.0 of AMap (JSAPI v2.0). “API” auto-translates as “Reference book” in web menus.

AMap:

The base library with optional plugins. Most important links are

- Summary
- Guide
- API CN documentation, good auto-translation
- API EN documentation in English, not recent/complete
- Examples - live demos

LOCA:

AMap extension with enhanced 3D features. In *amapro* it is invoked with a parameter - `am.init(loca=TRUE, ...)`. The documentation auto-translates well in the browser.

- Intro
- API documentation
- Examples - live demos

Commands

Controlling map and elements is done by sending AMap commands to them. Commands can be chained with the pipe operator |> or %>% and are executed sequentially in the order received. Example: `am.cmd('setAngle', 'carIcon', -90)` *amapro* uses native AMap commands and introduces these additional:

- **set** - create new element
  - with *name*: add new global JS object outside the map `am.cmd('set', 'VectorLayer', name='e$layer1')`
  - without *name*: add new element to map `am.cmd('set', 'e$marker1', position= c(116.478, 39.998))`
- **addTo** - append one existing JS object to another by name `am.cmd('addTo', 'e$layer1', 'e$marker1')`
- **var** - set a JavaScript variable `am.cmd('var', 'e$myOpacity', 0.8)`
- **code** - execute JavaScript code `am.cmd('code', 'alert("I am JS");')`

AMap commands starting with ‘**get**’ return data from the map or related objects. Put the data in a Shiny input variable by setting its name in parameter **r**. Example: `am.cmd('getCenter', 'map', r='inShiny1')` Above command will update *input$inShiny1* with the Lng/Lat coordinates of the map center.

Events

Events could be defined for map and elements. All types of instances use **on/off methods** to bind and remove events. Events are set in attribute **on**(or **off**) as a list of lists. Each event is a separate list with event name in **e**, a JS function **f** and optionally a query **q**. Example:

```
am.init(center= c(116.475, 39.997), zoom= 17,
        on= list(list(e= 'complete',
                      f= "function() {alert('loaded!');}")) )
```

on/off events without *name* are ignored, except for the map itself (as above example). JavaScript function *Shiny.setInputValue()* can be used to send data back to Shiny.

Limitations

- only one map is created by *am.init* per session. It is a JS global called 'm$jmap'.
- AMap command addTo is overwritten by *amapro* and cannot be used.
- most **built-in** AMap tile layers (Satellite, Traffic, Roads) are limited to China only. However, with command *am.item('TileLayer')*, one can use any Leaflet provider for worldwide coverage.
- AMap built-in map layers are GCJ-02 coded and coordinates collected on them will display incorrectly in Leaflet or other WGS-84 based maps, and vice-versa. They need to be converted. Conversion is available through function convertFrom.
- the **supported AMap plugins** are: ControlBar, Scale, ToolBar, MoveAnimation, MouseTool, HeatMap, GeoJSON, ElasticMarker.
- AMap ecosystem is vast, **unsupported** features and plugins include: 'BesizerCurve', 'MarkerCluster', 'HawkEye', IndoorMap, CustomLayer, 'GLCustomLayer', 'DistrictLayer', 'LayerGroup', all editors like 'PolygonEditor','Webservice', 'Search(AMap.Autocomplete, AMap.PlaceSearch)', 'Geocoding(AMap.Geocoder)', Route planning, other services(weather, districts, etc.), positioning, utilities.
- most **Loca** elements are supported, but not all have been tested. Latest *AmbientLight*, *DirectionalLight* and *PointLight* objects are not supported, but parameters *ambLight*, *dirLight* and *pointLight* accomplish the same.
- Loca events are not supported yet.

Tips

- all named objects created in JS are global variables (*window.name*). Good practice is to use a name prefix (m$) to avoid overwriting accidentally external variables.
- API attributes could be set to a JS function instead of a value. Function is defined as a string starting with word "function".
- usually WMS/WMTS tiles come from external servers and may present a CORS problem - browser refusal to load. One can install a small extension in Chrome or Firefox to fix this problem manually inside the browser.
- AMap has several predefined Map styles. Could be set in map options with *mapStyle*.

- *amapro* silent errors are collected in the browser Console. Press key **F12** to open the dev.environment, then open tab “Console” to view them.
- *am.init* has a *debug* (boolean) parameter, results are displayed in the browser Console
- Chrome/Firefox extensions may interfere with map presentation (like ‘uBlock’)

`am.cmd` *Run a command*

Description

Execute a command on a target element

Usage

```
am.cmd(id, cmd = NULL, trgt = NULL, ...)
```

Arguments

`id`	A map widget from am.init or a proxy from am.proxy
`cmd`	AMap command name string, like ’setFitView’, ’setMapStyle’, etc.
`trgt`	A target’s name string, or ’map’ for the map itself.
`...`	command attributes from AMap API. For AMap commands starting with ’get’ there are two *amapro* attributes ’f’ and ’r’. ’f’ is an optional JS function to manipulate the data received, ’r’ is the name of the Shiny variable receiving the data

Details

am.cmd provides interaction with the map.
Commands are sent to the map itself, or to objects inside or outside it.
AMap built-in objects have predefined set of commands listed in the API. Commands can modify an object (setZoom), but also get data from it (getCenter).
amapro introduces its own commands like *set*, *addTo* or *code*, described in the Introduction.

Value

A map or a map proxy

See Also

am.init code example and Introduction

Examples

```
if (interactive()) {
  # 'position' and 'content' are InfoWindow parameters from AMap API
  am.init() |>
  am.cmd('set', 'InfoWindow', position=c(116.6, 40), content='Beijing')

  am.proxy("plot") |>
    am.cmd('getLayers', 'map',
           f= 'function(yy) { return yy.map(x => { return x.CLASS_NAME;}); }',
           r= 'result1')
}
```

am.control — *Add Control*

Description

Add a Control to a map.

Usage

```
am.control(id, ctype = NULL, ...)
```

Arguments

`id`	amapro id or widget from am.init
`ctype`	A string for name of control, like 'Scale','ControlBar','ToolBar'.
`...`	A named list of parameters for the chosen control

Details

controls are ControlBar, ToolBar and Scale.
Parameters could be position or offset.

Value

A map widget to plot, or to save and expand with more features.

See Also

am.init code example

Examples

```
if (interactive()) {
  am.init() |> am.control("Scale")
}
```

am.init *Map Initialization*

Description

First command to build a map

Usage

```
am.init(..., width = NULL, height = NULL)
```

Arguments

`...`	attributes of map, see here. Additional attribute *loca*(boolean) is to add a Loca.Container to the map.
`width, height`	A valid CSS unit (like `'100%'`)

Details

Command *am.init* creates a widget with `createWidget`, then adds features to it.
On first use, *am.init* prompts for AMap API key. There is a temporary *demo* mode when key is unavailable.

Value

A widget to plot, or to store and expand with more features

Examples

```
if (interactive()) {
  ctr <- c(22.430151, 37.073011)
  tu <- paste0('http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/',
                 'MapServer/tile/[z]/[y]/[x]')
  am.init( center= ctr, zoom= 10, pitch= 60, viewMode= '3D') |>
  am.control(ctype= 'ControlBar', position= 'RT') |>
  am.item('TileLayer', tileUrl= tu) |>
  am.item('Marker', position= ctr,
    icon= 'https://upload.wikimedia.org/wikipedia/commons/9/9d/Ancient_Greek_helmet.png'
  ) |>
  am.cmd('set', 'InfoWindow', name='iwin', content='This is Sparta') |>
  am.cmd('open', 'iwin', 'm$jmap', ctr)   # m$jmap is the map name in JavaScript
}
```

`am.inspect` *Map to JSON*

Description

Convert map elements to JSON string

Usage

```
am.inspect(wt, json = TRUE, ...)
```

Arguments

`wt`	An `amapro` widget as returned by am.init
`json`	Boolean whether to return a JSON, or a `list`, default TRUE
`...`	Additional arguments to pass to toJSON

Details

Must be invoked or chained as last command.

Value

A JSON string if `json` is `TRUE` and a `list` otherwise.

Examples

```
if (interactive()) {
  am.init(viewMode= '3D', zoom= 10, pitch= 60) |>
    am.control(ctype= 'ControlBar', position= 'RT') |>
    am.inspect()
}
```

`am.item` *Add Item*

Description

Add an item to a map

Usage

```
am.item(id, itype, ...)
```

Arguments

`id`	A valid widget from am.init
`itype`	A string for item type name, like 'Marker'
`...`	attributes of item

Details

To add an item like Marker, Text or Polyline to the map

Value

A map widget to plot, or to save and expand with more features

See Also

am.init code example

Examples

```
if (interactive()) {
  am.init() |> am.item('Marker', position=c(116.6, 40))
}
```

`am.output` *Shiny: map UI*

Description

Placeholder for a map in Shiny UI

Usage

```
am.output(outputId, width = "100%", height = "400px")
```

Arguments

`outputId`	Name of output UI element.
`width, height`	Must be a valid CSS unit (like `'100%'`, `'400px'`, *'auto'*) or a number, which will be coerced to a string and have *'px'* appended.

Value

An output or render function that enables the use of the widget within Shiny applications. See shinyWidgetOutput.

See Also

Shiny demo in demo(am.shiny)

am.proxy *Shiny: create a map proxy*

Description

Create a proxy for an existing map in Shiny. It allows to add, merge, delete elements to a map without reloading it.

Usage

```
am.proxy(id)
```

Arguments

`id`	Map id from the Shiny UI

Value

A proxy object to update the map

Examples

```
if (interactive()) {
  demo(am.shiny)
}
```

am.render *Shiny: render a map*

Description

This is the initial rendering of a map in the UI.

Usage

```
am.render(wt, env = parent.frame())
```

Arguments

`wt`	An `amapro` widget to generate the chart.
`env`	The environment in which to evaluate `expr`.

Value

An output or render function that enables the use of the widget within Shiny applications.

See Also

`am.proxy` for example, `shinyRenderWidget` for return value.

Index