an open-source JavaScript library
for mobile-friendly interactive maps

Leaflet API reference

This reference reflects **Leaflet** . Check this list if you are using a different version of Leaflet.

Map

Usage example
Creation
Options
Events

Map Methods

Modifying map state
Getting map state
Layers and controls
Conversion methods
Other methods

Map Misc

Properties
Panes

UI Layers

Marker
DivOverlay
Popup
Tooltip

Raster Layers

TileLayer
TileLayer.WMS
ImageOverlay
VideoOverlay

Vector Layers

Path
Polyline
Polygon
Rectangle
Circle
CircleMarker
SVGOverlay
SVG
Canvas

Other Layers

LayerGroup
FeatureGroup
GeoJSON
GridLayer

Basic Types

LatLng
LatLngBounds
Point
Bounds
Icon
DivIcon

Controls

Zoom
Attribution
Layers
Scale

Utility

Browser
Util
Transformation
LineUtil
PolyUtil

DOM Utility

DomEvent
DomUtil
PosAnimation
Draggable

Base Classes

Class
Evented
Layer
Interactive layer
Control
Handler
Projection
CRS
Renderer

Misc

Event objects
global switches
noConflict
version

The central class of the API — it is used to create a map on a page and manipulate it.

Usage example

```
// initialize the map on the "map" div with a given center and zoom
var map = L.map('map', {
    center: [51.505, -0.09],
    zoom: 13
});
```

Creation

Factory	Description
`L.map(<String> id, <Map options> options?)`	Instantiates a map object given the DOM ID of a `<div>` element and optionally an object literal with `Map options`.
`L.map(<HTMLElement> el, <Map options> options?)`	Instantiates a map object given an instance of a `<div>` HTML element and optionally an object literal with `Map options`.

Options

Option	Type	Default	Description
`preferCanvas`	`Boolean`	`false`	Whether `Path`s should be rendered on a `Canvas` renderer. By default, all `Path`s are rendered in a `SVG` renderer.

Option	Type	Default	Description
`attributionControl`	`Boolean`	`true`	Whether a attribution control is added to the map by default.

Control options

Option	Type	Default	Description
`zoomControl`	`Boolean`	`true`	by default.

Option	Type	Default	Description
`closePopupOnClick`	`Boolean`	`true`	Set it to `false` if you don't want popups to close when user clicks the

Option	Type	Default	Description
			map.
`boxZoom`	`Boolean`	`true`	Whether the map can be zoomed to a rectangular area specified by dragging the mouse while pressing the shift key.
`doubleClickZoom`	`Boolean|String`	`true`	Whether the map can be zoomed in by double clicking on it and zoomed out by double clicking while holding shift. If passed `'center'`, double-click zoom will zoom to the center of the view regardless of where the mouse was.
`dragging`	`Boolean`	`true`	Whether the map is draggable with mouse/touch or not.
			Forces the map's zoom level to always be a multiple of this, particularly right

Map

`zoomSnap`	`Number`	`1`	

Interaction Options

			means the zoom level will not be snapped after `fitBounds` or a pinch-zoom.
`zoomDelta`	`Number`	`1`	Controls how much the map's zoom level will change after a `zoomIn()`, `zoomOut()`, pressing + or - on the keyboard, or using the zoom controls. Values smaller than `1` (e.g. `0.5`) allow for greater granularity.
`trackResize`	`Boolean`	`true`	Whether the map automatically handles browser window resize to update itself.

Panning Inertia Options

Option	Type	Default	Description
`inertia`	`Boolean`	`*`	If enabled, panning of the map will have an inertia effect where the map builds momentum while dragging and continues moving in the same direction for some time. Feels especially nice on touch devices. Enabled by default.
`inertiaDeceleration`	`Number`	`3000`	The rate with which the inertial movement slows down, in pixels/second^2.

Option	Type	Default	Description
`inertiaMaxSpeed`	`Number`	`Infinity`	Max speed of the inertial movement, in pixels/second.
`easeLinearity`	`Number`	`0.2`	
`worldCopyJump`	`Boolean`	`false`	With this option enabled, the map tracks when you pan to another "copy" of the world and seamlessly jumps to the original one so that all overlays like markers and vector layers are still visible.
`maxBoundsViscosity`	`Number`	`0.0`	If `maxBounds` is set, this option will control how solid the bounds are when dragging the map around. The default value of `0.0` allows the user to drag outside the bounds at normal speed, higher values will slow down map dragging outside bounds, and `1.0` makes the bounds fully solid, preventing the user from dragging outside the bounds.

Keyboard Navigation Options

Option	Type	Default	Description
`keyboard`	`Boolean`	`true`	Makes the map focusable and allows users to navigate the map with keyboard arrows and +/- keys.
`keyboardPanDelta`	`Number`	`80`	Amount of pixels to pan when pressing an arrow key.

Mouse wheel options

Option	Type	Default	Description
`scrollWheelZoom`	`Boolean|String`	`true`	Whether the map can be zoomed by using the mouse wheel. If passed `'center'`, it will zoom to the center of the view regardless of where the mouse was.
`wheelDebounceTime`	`Number`	`40`	Limits the rate at which a wheel can fire (in milliseconds). By default user can't zoom via wheel more often than once per 40 ms.
`wheelPxPerZoomLevel`	`Number`	`60`	How many scroll pixels (as reported by L.DomEvent.getWheelDelta) mean a change of one full zoom level. Smaller values will make wheel-zooming faster (and vice versa).

Touch interaction options

Option	Type	Default	Description
`tapHold`	`Boolean`		Enables simulation of `contextmenu` event, default is `true` for mobile Safari.
`tapTolerance`	`Number`	`15`	The max number of pixels a user can shift his finger during touch for it to be considered a valid tap.
`touchZoom`	`Boolean|String`	*	Whether the map can be zoomed by touch-dragging with two fingers. If passed `'center'`, it will zoom to the center of the view regardless of where the touch events (fingers) were. Enabled for touch-capable web browsers.
`bounceAtZoomLimits`	`Boolean`	`true`	Set it to false if you don't want the map to zoom beyond min/max zoom and then bounce back when pinch-zooming.

Map State Options

Option	Type	Default	Description
`crs`	CRS	`L.CRS.EPSG3857`	The Coordinate Reference System to use. Don't change this if you're not sure what it means.
`center`	LatLng	`undefined`	Initial geographic center of the map
`zoom`	`Number`	`undefined`	Initial map zoom level
`minZoom`	`Number`	*	Minimum zoom level of the map. If not specified and at least one `GridLayer` or `TileLayer` is in the map, the lowest of their `minZoom` options will be used instead.
`maxZoom`	`Number`	*	Maximum zoom level of the map. If not specified and at least one `GridLayer` or `TileLayer` is in the map, the highest of their `maxZoom` options will be used instead.
`layers`	`Layer[]`	`[]`	Array of layers that will be added to the map initially
`maxBounds`	LatLngBounds	`null`	When this option is set, the map restricts the view to the given geographical bounds, bouncing the user back if the user tries to pan outside the view. To set

Option	Type	Default	Description
			the restriction dynamically, use `setMaxBounds` method.
`renderer`	`Renderer`	*	The default method for drawing vector layers on the map. `L.SVG` or `L.Canvas` by default depending on browser support.

Animation Options

Option	Type	Default	Description
`zoomAnimation`	`Boolean`	`true`	Whether the map zoom animation is enabled. By default it's enabled in all browsers that support CSS3 Transitions except Android.
`zoomAnimationThreshold`	`Number`	`4`	Won't animate zoom if the zoom difference exceeds this value.
`fadeAnimation`	`Boolean`	`true`	Whether the tile fade animation is enabled. By default it's enabled in all browsers that support CSS3 Transitions except Android.
`markerZoomAnimation`	`Boolean`	`true`	Whether markers animate their zoom with the zoom animation, if disabled they will disappear for the length of the animation. By default it's enabled in all browsers that support CSS3 Transitions except Android.
`transform3DLimit`	`Number`	`2^23`	Defines the maximum size of a CSS translation transform. The default value should not be changed unless a web browser positions layers in the wrong place after doing a large `panBy`.

Events

Layer events

Event	Data	Description
`baselayerchange`	`LayersControlEvent`	Fired when the base layer is changed through the layers control.
`overlayadd`	`LayersControlEvent`	Fired when an overlay is selected through the layers control.
`overlayremove`	`LayersControlEvent`	Fired when an overlay is deselected through the layers control.

Event	Data	Description
`layeradd`	`LayerEvent`	Fired when a new layer is added to the map.
`layerremove`	`LayerEvent`	Fired when some layer is removed from the map



Map state change events

Event	Data	Description
`zoomlevelschange`	`Event`	Fired when the number of zoomlevels on the map is changed due to adding or removing a layer.
`resize`	`ResizeEvent`	Fired when the map is resized.
`unload`	`Event`	Fired when the map is destroyed with remove method.
`viewreset`	`Event`	Fired when the map needs to redraw its content (this usually happens on map zoom or load). Very useful for creating custom overlays.
`load`	`Event`	Fired when the map is initialized (when its center and zoom are set for the first time).
`zoomstart`	`Event`	Fired when the map zoom is about to change (e.g. before zoom animation).
`movestart`	`Event`	Fired when the view of the map starts changing (e.g. user starts dragging the map).
`zoom`	`Event`	Fired repeatedly during any change in zoom level, including zoom and fly animations.
`move`	`Event`	Fired repeatedly during any movement of the map, including pan and fly animations.
`zoomend`	`Event`	Fired when the map zoom changed, after any animations.
`moveend`	`Event`	Fired when the center of the map stops changing (e.g. user stopped dragging the map or after non-centered zoom).

Popup events

Event	Data	Description
`popupopen`	`PopupEvent`	Fired when a popup is opened in the map
`popupclose`	`PopupEvent`	Fired when a popup in the map is closed
`autopanstart`	`Event`	Fired when the map starts autopanning when opening a popup.

Tooltip events

Event	Data	Description
`tooltipopen`	`TooltipEvent`	Fired when a tooltip is opened in the map.
`tooltipclose`	`TooltipEvent`	Fired when a tooltip in the map is closed.

Location events

Event	Data	Description
`locationerror`	`ErrorEvent`	Fired when geolocation (using the `locate` method) failed.
`locationfound`	`LocationEvent`	Fired when geolocation (using the `locate` method) went successfully.

Interaction events

Event	Data	Description
`click`	`MouseEvent`	Fired when the user clicks (or taps) the map.
`dblclick`	`MouseEvent`	Fired when the user double-clicks (or double-taps) the map.
`mousedown`	`MouseEvent`	Fired when the user pushes the mouse button on the map.
`mouseup`	`MouseEvent`	Fired when the user releases the mouse button on the map.
`mouseover`	`MouseEvent`	Fired when the mouse enters the map.
`mouseout`	`MouseEvent`	Fired when the mouse leaves the map.
`mousemove`	`MouseEvent`	Fired while the mouse moves over the map.
`contextmenu`	`MouseEvent`	Fired when the user pushes the right mouse button on the map, prevents default browser context menu from showing if there are listeners on this event. Also fired on mobile when the user holds a single touch for a second (also called long press).
`keypress`	`KeyboardEvent`	Fired when the user presses a key from the keyboard that produces a character value while the map is focused.
`keydown`	`KeyboardEvent`	Fired when the user presses a key from the keyboard while the map is focused. Unlike the `keypress` event, the `keydown` event is fired for keys that produce a character value and for keys that do not produce a character value.
`keyup`	`KeyboardEvent`	Fired when the user releases a key from the keyboard while the map is focused.
`preclick`	`MouseEvent`	Fired before mouse click on the map (sometimes useful when you want something to happen on click before any existing click handlers start running).

Other Events

Event	Data	Description
`zoomanim`	`ZoomAnimEvent`	Fired at least once per zoom animation. For continuous zoom, like pinch zooming, fired once per frame during zoom.

Methods

Method	Returns	Description
`getRenderer(<Path> layer)`	`Renderer`	Returns the instance of `Renderer` that should be used to render the given `Path`. It will ensure that the `renderer` options of the map and paths are respected, and that the renderers do exist on the map.

Methods for Layers and Controls

Method	Returns	Description
`addControl(<Control> control)`	`this`	Adds the given control to the map
`removeControl(<Control> control)`	`this`	Removes the given control from the map
`addLayer(<Layer> layer)`	`this`	Adds the given layer to the map
`removeLayer(<Layer> layer)`	`this`	Removes the given layer from the map.
`hasLayer(<Layer> layer)`	`Boolean`	Returns `true` if the given layer is currently added to the map

`eachLayer(<Function> fn, <Object> context?)`	`this`	Iterates over the layers of the map, optionally specifying context of the iterator function.
		`map.eachLayer(function(layer){`
		`layer.bindPopup('Hello');`
		`});`

`openPopup(<Popup> popup)`	`this`	Opens the specified popup while closing the previously opened (to make sure only one is opened at one time for usability).

Method	Returns	Description
`openPopup(<String|HTMLElement> content, <LatLng> latlng, <Popup options> options?)`	`this`	Creates a popup with the specified content and options and opens it in the given point on a map.
`closePopup(<Popup> popup?)`	`this`	Closes the popup previously opened with openPopup (or the given one).
`openTooltip(<Tooltip> tooltip)`	`this`	Opens the specified tooltip.
`openTooltip(<String|HTMLElement> content, <LatLng> latlng, <Tooltip options> options?)`	`this`	Creates a tooltip with the specified content and options and open it.
`closeTooltip(<Tooltip> tooltip)`	`this`	Closes the tooltip given as parameter.

Methods for modifying map state

Method	Returns	Description
`setView(<LatLng> center, <Number> zoom, <Zoom/pan options> options?)`	`this`	Sets the view of the map (geographical center and zoom) with the given animation options.
`setZoom(<Number> zoom, <Zoom/pan options> options?)`	`this`	Sets the zoom of the map.
`zoomIn(<Number> delta?, <Zoom options> options?)`	`this`	Increases the zoom of the map by `delta` (`zoomDelta` by default).
`zoomOut(<Number> delta?, <Zoom options> options?)`	`this`	Decreases the zoom of the map by `delta` (`zoomDelta` by default).
`setZoomAround(<LatLng> latlng, <Number> zoom, <Zoom options> options)`	`this`	Zooms the map while keeping a specified geographical point on the map stationary (e.g. used internally for scroll zoom and double-click zoom).
`setZoomAround(<Point> offset, <Number> zoom, <Zoom options> options)`	`this`	Zooms the map while keeping a specified pixel on the map (relative to the top-left corner) stationary.
`fitBounds(<LatLngBounds> bounds, <fitBounds options> options?)`	`this`	Sets a map view that contains the given geographical bounds with the maximum zoom level possible.
`fitWorld(<fitBounds options> options?)`	`this`	Sets a map view that mostly contains the whole world with the maximum zoom level possible.

Method	Returns	Description
`panTo(<LatLng> latlng, <Pan options> options?)`	this	Pans the map to a given center.
`panBy(<Point> offset, <Pan options> options?)`	this	Pans the map by a given number of pixels (animated).
`flyTo(<LatLng> latlng, <Number> zoom?, <Zoom/pan options> options?)`	this	Sets the view of the map (geographical center and zoom) performing a smooth pan-zoom animation.
`flyToBounds(<LatLngBounds> bounds, <fitBounds options> options?)`	this	Sets the view of the map with a smooth animation like `flyTo`, but takes a bounds parameter like `fitBounds`.
`setMaxBounds(<LatLngBounds> bounds)`	this	Restricts the map view to the given bounds (see the maxBounds option).
`setMinZoom(<Number> zoom)`	this	Sets the lower limit for the available zoom levels (see the minZoom option).
`setMaxZoom(<Number> zoom)`	this	Sets the upper limit for the available zoom levels (see the maxZoom option).
`panInsideBounds(<LatLngBounds> bounds, <Pan options> options?)`	this	Pans the map to the closest view that would lie inside the given bounds (if it's not already), controlling the animation using the options specific, if any.
`panInside(<LatLng> latlng, <padding options> options?)`	this	Pans the map the minimum amount to make the `latlng` visible. Use padding options to fit the display to more restricted bounds. If `latlng` is already within the (optionally padded) display bounds, the map will not be panned.
`invalidateSize(<Zoom/pan options> options)`	this	Checks if the map container size changed and updates the map if so — call it after you've changed the map size dynamically, also animating pan by default. If `options.pan` is `false`, panning will not occur. If `options.debounceMoveend` is `true`, it will delay `moveend` event so that it doesn't happen often even if

Method	Returns	Description
		the method is called many times in a row.
`invalidateSize(<Boolean> animate)`	`this`	Checks if the map container size changed and updates the map if so — call it after you've changed the map size dynamically, also animating pan by default.
`stop()`	`this`	Stops the currently running `panTo` or `flyTo` animation, if any.

Geolocation methods

Method	Returns	Description
`locate(<Locate options> options?)`	`this`	Tries to locate the user using the Geolocation API, firing a `locationfound` event with location data on success or a `locationerror` event on failure, and optionally sets the map view to the user's location with respect to detection accuracy (or to the world view if geolocation failed). Note that, if your page doesn't use HTTPS, this method will fail in modern browsers (Chrome 50 and newer) See `Locate options` for more details.
`stopLocate()`	`this`	Stops watching location previously initiated by `map.locate({watch: true})` and aborts resetting the map view if map.locate was called with `{setView: true}`.

Other Methods

Method	Returns	Description
`addHandler(<String> name, <Function> HandlerClass)`	`this`	Adds a new `Handler` to the map, given its name and constructor function.
`remove()`	`this`	Destroys the map and clears all related event listeners.
`createPane(<String> name, <HTMLElement> container?)`	`HTMLElement`	Creates a new map pane with the given name if it doesn't exist already, then returns it. The pane is created as a child of `container`, or as a child of the main map pane if not set.
`getPane(<String|HTMLElement> pane)`	`HTMLElement`	Returns a map pane, given its name or its HTML element (its identity).
`getPanes()`	`Object`	Returns a plain object containing the names of all panes as keys and

Method	Returns	Description
		the panes as values.
`getContainer()`	`HTMLElement`	Returns the HTML element that contains the map.
`whenReady(<Function> fn, <Object> context?)`	`this`	Runs the given function `fn` when the map gets initialized with a view (center and zoom) and at least one layer, or immediately if it's already initialized, optionally passing a function context.

Methods for Getting Map State

Method	Returns	Description
`getCenter()`	`LatLng`	Returns the geographical center of the map view
`getZoom()`	`Number`	Returns the current zoom level of the map view
`getBounds()`	`LatLngBounds`	Returns the geographical bounds visible in the current map view
`getMinZoom()`	`Number`	Returns the minimum zoom level of the map (if set in the `minZoom` option of the map or of any layers), or `0` by default.
`getMaxZoom()`	`Number`	Returns the maximum zoom level of the map (if set in the `maxZoom` option of the map or of any layers).
`getBoundsZoom(<LatLngBounds> bounds, <Boolean> inside?, <Point> padding?)`	`Number`	Returns the maximum zoom level on which the given bounds fit to the map view in its entirety. If `inside` (optional) is set to `true`, the method instead returns the minimum zoom level on which the map view fits into the given bounds in its entirety.
`getSize()`	`Point`	Returns the current size of the map container (in pixels).
`getPixelBounds()`	`Bounds`	Returns the bounds of the current map view in projected pixel coordinates (sometimes useful in layer and overlay implementations).
`getPixelOrigin()`	`Point`	Returns the projected pixel coordinates of the top left point of

Method	Returns	Description
		the map layer (useful in custom layer and overlay implementations).
`getPixelWorldBounds(<Number> zoom?)`	`Bounds`	Returns the world's bounds in pixel coordinates for zoom level `zoom`. If `zoom` is omitted, the map's current zoom level is used.

Conversion Methods

Method	Returns	Description
`getZoomScale(<Number> toZoom, <Number> fromZoom)`	`Number`	Returns the scale factor to be applied to a map transition from zoom level `fromZoom` to `toZoom`. Used internally to help with zoom animations.
`getScaleZoom(<Number> scale, <Number> fromZoom)`	`Number`	Returns the zoom level that the map would end up at, if it is at `fromZoom` level and everything is scaled by a factor of `scale`. Inverse of `getZoomScale`.
`project(<LatLng> latlng, <Number> zoom)`	`Point`	Projects a geographical coordinate `LatLng` according to the projection of the map's CRS, then scales it according to `zoom` and the CRS's `Transformation`. The result is pixel coordinate relative to the CRS origin.
`unproject(<Point> point, <Number> zoom)`	`LatLng`	Inverse of `project`.
`layerPointToLatLng(<Point> point)`	`LatLng`	Given a pixel coordinate relative to the origin pixel, returns the corresponding geographical coordinate (for the current zoom level).
`latLngToLayerPoint(<LatLng> latlng)`	`Point`	Given a geographical coordinate, returns the corresponding pixel

Method	Returns	Description
		coordinate relative to the origin pixel.
`wrapLatLng(<LatLng> latlng)`	`LatLng`	Returns a `LatLng` where `lat` and `lng` has been wrapped according to the map's CRS's `wrapLat` and `wrapLng` properties, if they are outside the CRS's bounds. By default this means longitude is wrapped around the dateline so its value is between -180 and +180 degrees.
`wrapLatLngBounds(<LatLngBounds> bounds)`	`LatLngBounds`	Returns a `LatLngBounds` with the same size as the given one, ensuring that its center is within the CRS's bounds. By default this means the center longitude is wrapped around the dateline so its value is between -180 and +180 degrees, and the majority of the bounds overlaps the CRS's bounds.
`distance(<LatLng> latlng1, <LatLng> latlng2)`	`Number`	Returns the distance between two geographical coordinates according to the map's CRS. By default this measures distance in meters.
`containerPointToLayerPoint(<Point> point)`	`Point`	Given a pixel coordinate relative to the map container, returns the corresponding pixel coordinate relative to the origin pixel.
`layerPointToContainerPoint(<Point> point)`	`Point`	Given a pixel coordinate relative to the origin pixel, returns the corresponding pixel

Method	Returns	Description
		coordinate relative to the map container.
`containerPointToLatLng(<Point> point)`	`LatLng`	Given a pixel coordinate relative to the map container, returns the corresponding geographical coordinate (for the current zoom level).
`latLngToContainerPoint(<LatLng> latlng)`	`Point`	Given a geographical coordinate, returns the corresponding pixel coordinate relative to the map container.
`mouseEventToContainerPoint(<MouseEvent> ev)`	`Point`	Given a MouseEvent object, returns the pixel coordinate relative to the map container where the event took place.
`mouseEventToLayerPoint(<MouseEvent> ev)`	`Point`	Given a MouseEvent object, returns the pixel coordinate relative to the origin pixel where the event took place.
`mouseEventToLatLng(<MouseEvent> ev)`	`LatLng`	Given a MouseEvent object, returns geographical coordinate where the event took place.

▶ Methods inherited from Evented

Properties

Controls

Property	Type	Description
`zoomControl`	`Control.Zoom`	The default zoom control (only available if the `zoomControl` option was `true` when creating the map).

Handlers

Property	Type	Description
`boxZoom`	`Handler`	Box (shift-drag with mouse) zoom handler.

Property	Type	Description
doubleClickZoom	Handler	Double click zoom handler.
dragging	Handler	Map dragging handler (by both mouse and touch).
keyboard	Handler	Keyboard navigation handler.
scrollWheelZoom	Handler	Scroll wheel zoom handler.
tapHold	Handler	Long tap handler to simulate contextmenu event (useful in mobile Safari).
touchZoom	Handler	Touch zoom handler.

Map panes

Panes are DOM elements used to control the ordering of layers on the map. You can access panes with map.getPane or map.getPanes methods. New panes can be created with the map.createPane method.

Every map has the following default panes that differ only in zIndex.

Pane	Type	Z-index	Description
mapPane	HTMLElement	'auto'	Pane that contains all other map panes
tilePane	HTMLElement	200	Pane for GridLayers and TileLayers
overlayPane	HTMLElement	400	Pane for vectors (Paths, like Polylines and Polygons), ImageOverlays and VideoOverlays
shadowPane	HTMLElement	500	Pane for overlay shadows (e.g. Marker shadows)
markerPane	HTMLElement	600	Pane for Icons of Markers
tooltipPane	HTMLElement	650	Pane for Tooltips.
popupPane	HTMLElement	700	Pane for Popups.

Locate options

Some of the geolocation methods for Map take in an options parameter. This is a plain javascript object with the following optional components:

Option	Type	Default	Description
watch	Boolean	false	If true, starts continuous watching of location changes (instead of detecting it once) using W3C watchPosition method.

Option	Type	Default	Description
			You can later stop watching using `map.stopLocate()` method.
`setView`	`Boolean`	`false`	If `true`, automatically sets the map view to the user location with respect to detection accuracy, or to world view if geolocation failed.
`maxZoom`	`Number`	`Infinity`	The maximum zoom for automatic view setting when using `setView` option.
`timeout`	`Number`	`10000`	Number of milliseconds to wait for a response from geolocation before firing a `locationerror` event.
`maximumAge`	`Number`	`0`	Maximum age of detected location. If less than this amount of milliseconds passed since last geolocation response, `locate` will return a cached location.
`enableHighAccuracy`	`Boolean`	`false`	Enables high accuracy, see description in the W3C spec.

Zoom options

Some of the `Map` methods which modify the zoom level take in an `options` parameter. This is a plain javascript object with the following optional components:

Option	Type	Default	Description
`animate`	`Boolean`		If not specified, zoom animation will happen if the zoom origin is inside the current view. If `true`, the map will attempt animating zoom disregarding where zoom origin is. Setting `false` will make it always reset the view completely without animation.

Pan options

Some of the `Map` methods which modify the center of the map take in an `options` parameter. This is a plain javascript object with the following optional components:

Option	Type	Default	Description
`animate`	`Boolean`		If `true`, panning will always be animated if possible. If `false`, it will not animate panning, either resetting the map view if panning more than a screen away, or just setting a new offset for the map pane (except for `panBy` which always does the latter).
`duration`	`Number`	`0.25`	Duration of animated panning, in seconds.

Option	Type	Default	Description
`easeLinearity`	`Number`	`0.25`	The curvature factor of panning animation easing (third parameter of the Cubic Bezier curve). 1.0 means linear animation, and the smaller this number, the more bowed the curve.
`noMoveStart`	`Boolean`	`false`	If `true`, panning won't fire `movestart` event on start (used internally for panning inertia).

Zoom/pan options

▶ Options inherited from Zoom options

▶ Options inherited from Pan options

Padding options

Option	Type	Default	Description
`paddingTopLeft`	`Point`	`[0, 0]`	Sets the amount of padding in the top left corner of a map container that shouldn't be accounted for when setting the view to fit bounds. Useful if you have some control overlays on the map like a sidebar and you don't want them to obscure objects you're zooming to.
`paddingBottomRight`	`Point`	`[0, 0]`	The same for the bottom right corner of the map.
`padding`	`Point`	`[0, 0]`	Equivalent of setting both top left and bottom right padding to the same value.

FitBounds options

Option	Type	Default	Description
`maxZoom`	`Number`	`null`	The maximum possible zoom to use.

▶ Options inherited from Zoom options

▶ Options inherited from Pan options

▶ Options inherited from Padding options

Marker

L.Marker is used to display clickable/draggable icons on the map. Extends `Layer`.

Usage example

```
L.marker([50.5, 30.5]).addTo(map);
```

Creation

Factory	Description
`L.marker(<LatLng> latlng, <Marker options> options?)`	Instantiates a Marker object given a geographical point and optionally an options object.

Options

Option	Type	Default	Description
`icon`	`Icon`	`*`	Icon instance to use for rendering the marker. See Icon documentation for details on how to customize the marker icon. If not specified, a common instance of `L.Icon.Default` is used.
`keyboard`	`Boolean`	`true`	Whether the marker can be tabbed to with a keyboard and clicked by pressing enter.
`title`	`String`	`''`	Text for the browser tooltip that appear on marker hover (no tooltip by default). Useful for accessibility.
`alt`	`String`	`'Marker'`	Text for the `alt` attribute of the icon image. Useful for accessibility.
`zIndexOffset`	`Number`	`0`	By default, marker images zIndex is set automatically based on its latitude. Use this option if you want to put the marker on top of all others (or below), specifying a high value like `1000` (or high negative value, respectively).
`opacity`	`Number`	`1.0`	The opacity of the marker.
`riseOnHover`	`Boolean`	`false`	If `true`, the marker will get on top of others when you hover the mouse over it.
`riseOffset`	`Number`	`250`	The z-index offset used for the `riseOnHover` feature.

Option	Type	Default	Description
`pane`	`String`	`'markerPane'`	`Map` pane where the markers icon will be added.
`shadowPane`	`String`	`'shadowPane'`	`Map` pane where the markers shadow will be added.
`bubblingMouseEvents`	`Boolean`	`false`	When `true`, a mouse event on this marker will trigger the same event on the map (unless `L.DomEvent.stopPropagation` is used).
`autoPanOnFocus`	`Boolean`	`true`	When `true`, the map will pan whenever the marker is focused (via e.g. pressing `tab` on the keyboard) to ensure the marker is visible within the map's bounds

Draggable marker options

Option	Type	Default	Description
`draggable`	`Boolean`	`false`	Whether the marker is draggable with mouse/touch or not.
`autoPan`	`Boolean`	`false`	Whether to pan the map when dragging this marker near its edge or not.
`autoPanPadding`	`Point`	`Point(50, 50)`	Distance (in pixels to the left/right and to the top/bottom) of the map edge to start panning the map.
`autoPanSpeed`	`Number`	`10`	Number of pixels the map should pan by.

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

Event	Data	Description
`move`	`Event`	Fired when the marker is moved via `setLatLng` or by dragging. Old and new coordinates are included in event arguments as `oldLatLng`, `latlng`.

Dragging events

Event	Data	Description
`dragstart`	`Event`	Fired when the user starts dragging the marker.
`movestart`	`Event`	Fired when the marker starts moving (because of dragging).

Event	Data	Description
drag	Event	Fired repeatedly while the user drags the marker.
dragend	DragEndEvent	Fired when the user stops dragging the marker.
moveend	Event	Fired when the marker stops moving (because of dragging).



▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

In addition to shared layer methods like `addTo()` and `remove()` and popup methods like bindPopup() you can also use the following methods:

Method	Returns	Description
getLatLng()	LatLng	Returns the current geographical position of the marker.
setLatLng(<LatLng> *latlng*)	this	Changes the marker position to the given point.
setZIndexOffset(<Number> *offset*)	this	Changes the zIndex offset of the marker.
getIcon()	Icon	Returns the current icon used by the marker
setIcon(<Icon> *icon*)	this	Changes the marker icon.
setOpacity(<Number> *opacity*)	this	Changes the opacity of the marker.

Other methods

Method	Returns	Description
toGeoJSON(<Number|false> *precision?*)	Object	Coordinates values are rounded with formatNum function with given precision. Returns a GeoJSON representation of the marker (as a GeoJSON Point Feature).

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented



Properties

Interaction handlers

Interaction handlers are properties of a marker instance that allow you to control interaction behavior in runtime, enabling or disabling certain features such as dragging (see `Handler` methods). Example:

```
marker.dragging.disable();
```

Property	Type	Description
`dragging`	`Handler`	Marker dragging handler (by both mouse and touch). Only valid when the marker is on the map (Otherwise set `marker.options.draggable`).

DivOverlay

Base model for L.Popup and L.Tooltip. Inherit from it for custom overlays like plugins.

Options

Option	Type	Default	Description
`interactive`	`Boolean`	`false`	If true, the popup/tooltip will listen to the mouse events.
`offset`	`Point`	`Point(0, 0)`	The offset of the overlay position.
`className`	`String`	`''`	A custom CSS class name to assign to the overlay.
`pane`	`String`	`undefined`	`Map` pane where the overlay will be added.
`content`	`String|HTMLElement|Function`	`''`	Sets the HTML content of the overlay while initializing. If a function is passed the source layer will be passed to the function. The function should return a `String` or `HTMLElement` to be used in the overlay.

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

DivOverlay events

Event	Data	Description
`contentupdate`	`Event`	Fired when the content of the overlay is updated

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`openOn(<Map> map)`	`this`	Adds the overlay to the map. Alternative to `map.openPopup(popup)/.openTooltip(tooltip)`.
`close()`	`this`	Closes the overlay. Alternative to `map.closePopup(popup)/.closeTooltip(tooltip)` and `layer.closePopup()/.closeTooltip()`.
`toggle(<Layer> layer?)`	`this`	Opens or closes the overlay bound to layer depending on its current state.

Method	Returns	Description
		Argument may be omitted only for overlay bound to layer. Alternative to `layer.togglePopup()`/`.toggleTooltip()`.
`getLatLng()`	`LatLng`	Returns the geographical point of the overlay.
`setLatLng(<LatLng> latlng)`	`this`	Sets the geographical point where the overlay will open.
`getContent()`	`String|HTMLElement`	Returns the content of the overlay.
`setContent(<String|HTMLElement|Function> htmlContent)`	`this`	Sets the HTML content of the overlay. If a function is passed the source layer will be passed to the function. The function should return a `String` or `HTMLElement` to be used in the overlay.
`getElement()`	`String|HTMLElement`	Returns the HTML container of the overlay.
`update()`	`null`	Updates the overlay content, layout and position. Useful for updating the overlay after something inside changed,

Method	Returns	Description
		e.g. image loaded.
`isOpen()`	`Boolean`	Returns `true` when the overlay is visible on the map.
`bringToFront()`	`this`	Brings this overlay in front of other overlays (in the same map pane).
`bringToBack()`	`this`	Brings this overlay to the back of other overlays (in the same map pane).

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Popup

Used to open popups in certain places of the map. Use Map.openPopup to open popups while making sure that only one popup is open at one time (recommended for usability), or use Map.addLayer to open as many as you want.

Usage example

If you want to just bind a popup to marker click and then open it, it's really easy:

```
marker.bindPopup(popupContent).openPopup();
```

Path overlays like polylines also have a `bindPopup` method.

A popup can be also standalone:

```
var popup = L.popup()
    .setLatLng(latlng)
    .setContent('<p>Hello world!<br />This is a nice popup.</p>')
    .openOn(map);
```

or

```
var popup = L.popup(latlng, {content: '<p>Hello world!<br />This is a nice popu
    .openOn(map);
```

Creation

Factory	Description
`L.popup(<Popup options> options?, <Layer> source?)`	Instantiates a Popup object given an optional `options` object that describes its appearance and location and an optional `source` object that is used to tag the popup with a reference to the Layer to which it refers.
`L.popup(<LatLng> latlng, <Popup options> options?)`	Instantiates a Popup object given `latlng` where the popup will open and an optional `options` object that describes its appearance and location.

Options

Option	Type	Default	Description
`pane`	`String`	`'popupPane'`	`Map` pane where the popup will be added.
`offset`	Point	`Point(0, 7)`	The offset of the popup position.
`maxWidth`	`Number`	`300`	Max width of the popup, in pixels.
`minWidth`	`Number`	`50`	Min width of the popup, in pixels.
`maxHeight`	`Number`	`null`	If set, creates a scrollable container of the given height inside a popup if its content exceeds it. The scrollable container can be styled using the `leaflet-popup-scrolled` CSS class selector.
`autoPan`	`Boolean`	`true`	Set it to `false` if you don't want the map to do panning

Option	Type	Default	Description
			animation to fit the opened popup.
`autoPanPaddingTopLeft`	Point	`null`	The margin between the popup and the top left corner of the map view after autopanning was performed.
`autoPanPaddingBottomRight`	Point	`null`	The margin between the popup and the bottom right corner of the map view after autopanning was performed.
`autoPanPadding`	Point	`Point(5, 5)`	Equivalent of setting both top left and bottom right autopan padding to the same value.
`keepInView`	`Boolean`	`false`	Set it to `true` if you want to prevent users from panning the popup off of the screen while it is open.
`closeButton`	`Boolean`	`true`	Controls the presence of a close button in the popup.
`autoClose`	`Boolean`	`true`	Set it to `false` if you want to override the default behavior of the popup closing when another popup is opened.
`closeOnEscapeKey`	`Boolean`	`true`	Set it to `false` if you want to override the default behavior of the ESC key for closing of the popup.
`closeOnClick`	`Boolean`	`*`	Set it if you want to override the default behavior of the popup closing when user clicks on the map. Defaults to the map's `closePopupOnClick` option.
`className`	`String`	`''`	A custom CSS class name to assign to the popup.

▶ Options inherited from DivOverlay

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ DivOverlay events inherited from DivOverlay

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`openOn(<Map> map)`	`this`	Alternative to `map.openPopup(popup)`. Adds the popup to the map and closes the previous one.

▶ Methods inherited from DivOverlay

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Tooltip

Used to display small texts on top of map layers.

Usage example

If you want to just bind a tooltip to marker:

```
marker.bindTooltip("my tooltip text").openTooltip();
```

Path overlays like polylines also have a `bindTooltip` method.

A tooltip can be also standalone:

```
var tooltip = L.tooltip()
    .setLatLng(latlng)
```

```
    .setContent('Hello world!<br />This is a nice tooltip.')
    .addTo(map);
```

or

```
var tooltip = L.tooltip(latlng, {content: 'Hello world!<br />This is a nice too
    .addTo(map);
```

Note about tooltip offset. Leaflet takes two options in consideration for computing tooltip offsetting:

- the `offset` Tooltip option: it defaults to [0, 0], and it's specific to one tooltip. Add a positive x offset to move the tooltip to the right, and a positive y offset to move it to the bottom. Negatives will move to the left and top.
- the `tooltipAnchor` Icon option: this will only be considered for Marker. You should adapt this value if you use a custom icon.

Creation

Factory	Description
`L.tooltip(<Tooltip options> options?, <Layer> source?)`	Instantiates a `Tooltip` object given an optional `options` object that describes its appearance and location and an optional `source` object that is used to tag the tooltip with a reference to the Layer to which it refers.
`L.tooltip(<LatLng> latlng, <Tooltip options> options?)`	Instantiates a `Tooltip` object given `latlng` where the tooltip will open and an optional `options` object that describes its appearance and location.

Options

Option	Type	Default	Description
`pane`	`String`	`'tooltipPane'`	`Map` pane where the tooltip will be added.
`offset`	`Point`	`Point(0, 0)`	Optional offset of the tooltip position.
`direction`	`String`	`'auto'`	Direction where to open the tooltip. Possible values are: `right`, `left`, `top`, `bottom`, `center`, `auto`. `auto` will dynamically switch between `right` and `left` according to the tooltip position on the map.
`permanent`	`Boolean`	`false`	Whether to open the tooltip permanently or only on mouseover.

Option	Type	Default	Description
`sticky`	`Boolean`	`false`	If true, the tooltip will follow the mouse instead of being fixed at the feature center.
`opacity`	`Number`	`0.9`	Tooltip container opacity.



▶ Options inherited from DivOverlay

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ DivOverlay events inherited from DivOverlay

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

▶ Methods inherited from DivOverlay

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

TileLayer

Used to load and display tile layers on the map. Note that most tile servers require attribution, which you can set under `Layer`. Extends `GridLayer`.

Usage example

```
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png?{foo}', {foo: 'bar'
```



URL template

A string of the following form:

```
'https://{s}.somedomain.com/blabla/{z}/{x}/{y}{r}.png'
```

`{s}` means one of the available subdomains (used sequentially to help with browser parallel requests per domain limitation; subdomain values are specified in options; `a`, `b` or `c` by default, can be omitted), `{z}` — zoom level, `{x}` and `{y}` — tile coordinates. `{r}` can be used to add "@2x" to the URL to load retina tiles.

You can use custom keys in the template, which will be evaluated from TileLayer options, like this:

```
L.tileLayer('https://{s}.somedomain.com/{foo}/{z}/{x}/{y}.png', {foo: 'bar'});
```

Creation

Extension methods

Factory	Description
`L.tilelayer(<String> urlTemplate, <TileLayer options> options?)`	Instantiates a tile layer object given a `URL template` and optionally an options object.

Options

Option	Type	Default	Description
`minZoom`	`Number`	`0`	The minimum zoom level down to which this layer will be displayed (inclusive).
`maxZoom`	`Number`	`18`	The maximum zoom level up to which this layer will be displayed (inclusive).
`subdomains`	`String|String[]`	`'abc'`	Subdomains of the tile service. Can be passed in the form of one string (where each letter is a subdomain name) or an array of strings.
`errorTileUrl`	`String`	`''`	URL to the tile image to show in place of the tile that failed to load.
`zoomOffset`	`Number`	`0`	The zoom number used in tile URLs will

Option	Type	Default	Description
			be offset with this value.
`tms`	`Boolean`	`false`	If `true`, inverses Y axis numbering for tiles (turn this on for TMS services).
`zoomReverse`	`Boolean`	`false`	If set to true, the zoom number used in tile URLs will be reversed (`maxZoom - zoom` instead of `zoom`)
`detectRetina`	`Boolean`	`false`	If `true` and user is on a retina display, it will request four tiles of half the specified size and a bigger zoom level in place of one to utilize the high resolution.
`crossOrigin`	`Boolean|String`	`false`	Whether the crossOrigin attribute will be added to the tiles. If a String is provided, all tiles will have their crossOrigin attribute set to the String provided. This is needed if you want to access tile pixel data. Refer to CORS Settings for valid String values.
`referrerPolicy`	`Boolean|String`	`false`	Whether the referrerPolicy attribute will be added to the tiles. If a String is provided, all tiles will have their referrerPolicy attribute set to the String provided. This may be needed if your map's rendering context has a strict default but your tile provider expects a valid referrer (e.g. to validate an API token). Refer to HTMLImageElement.referrerPolicy for valid String values.

▶ Options inherited from GridLayer

▶ Options inherited from Layer

Events

Extension methods

Event	Data	Description
`tileabort`	`TileEvent`	Fired when a tile was loading but is now not wanted.

▶ Events inherited from GridLayer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer



Methods

Method	Returns	Description
`setUrl(<String> url, <Boolean> noRedraw?)`	`this`	Updates the layer's URL template and redraws it (unless `noRedraw` is set to `true`). If the URL does not change, the layer will not be redrawn unless the noRedraw parameter is set to false.
`createTile(<Object> coords, <Function> done?)`	`HTMLElement`	Called only internally, overrides GridLayer's `createTile()` to return an `<img>` HTML element with the appropriate image URL given `coords`. The `done` callback is called when the tile has been loaded.

Extension methods

Layers extending `TileLayer` might reimplement the following method.

Method	Returns	Description
`getTileUrl(<Object> coords)`	`String`	Called only internally, returns the URL for a tile given its coordinates. Classes extending `TileLayer` can override this function to provide custom tile URL naming schemes.

▶ Methods inherited from GridLayer

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

TileLayer.WMS

Used to display WMS services as tile layers on the map. Extends `TileLayer`.

Usage example

```
var nexrad = L.tileLayer.wms("http://mesonet.agron.iastate.edu/cgi-bin/wms/nexr
    layers: 'nexrad-n0r-900913',
    format: 'image/png',
```

```
    transparent: true,
    attribution: "Weather data © 2012 IEM Nexrad"
});
```

Creation

Factory	Description
`L.tileLayer.wms(<String> baseUrl, <TileLayer.WMS options> options)`	Instantiates a WMS tile layer object given a base URL of the WMS service and a WMS parameters/options object.

Options

If any custom options not documented here are used, they will be sent to the WMS server as extra parameters in each request URL. This can be useful for non-standard vendor WMS parameters.

Option	Type	Default	Description
`layers`	`String`	`''`	**(required)** Comma-separated list of WMS layers to show.
`styles`	`String`	`''`	Comma-separated list of WMS styles.
`format`	`String`	`'image/jpeg'`	WMS image format (use `'image/png'` for layers with transparency).
`transparent`	`Boolean`	`false`	If `true`, the WMS service will return images with transparency.
`version`	`String`	`'1.1.1'`	Version of the WMS service to use
`crs`	`CRS`	`null`	Coordinate Reference System to use for the WMS requests, defaults to map CRS. Don't change this if you're not sure what it means.
`uppercase`	`Boolean`	`false`	If `true`, WMS request parameter keys will be uppercase.

▶ Options inherited from TileLayer

▶ Options inherited from GridLayer

▶ Options inherited from Layer

Events

▶ Extension methods inherited from TileLayer

▶ Events inherited from GridLayer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`setParams(<Object> params, <Boolean> noRedraw?)`	`this`	Merges an object with the new parameters and re-requests tiles on the current screen (unless `noRedraw` was set to true).

▶ Methods inherited from TileLayer

▶ Methods inherited from GridLayer

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

ImageOverlay

Used to load and display a single image over specific bounds of the map. Extends `Layer`.

Usage example

```
var imageUrl = 'https://maps.lib.utexas.edu/maps/historical/newark_nj_1922.jpg'
    imageBounds = [[40.712216, -74.22655], [40.773941, -74.12544]];
L.imageOverlay(imageUrl, imageBounds).addTo(map);
```

Creation

Factory	Description
`L.imageOverlay(<String> imageUrl, <LatLngBounds> bounds, <ImageOverlay options> options?)`	Instantiates an image overlay object given the URL of the image and the geographical bounds it is tied to.

Options

Option	Type	Default	Description
`opacity`	`Number`	`1.0`	The opacity of the image overlay.
`alt`	`String`	`''`	Text for the `alt` attribute of the image (useful for accessibility).
`interactive`	`Boolean`	`false`	If `true`, the image overlay will emit mouse events when clicked or hovered.
`crossOrigin`	`Boolean|String`	`false`	Whether the crossOrigin attribute will be added to the image. If a String is provided, the image will have its crossOrigin attribute set to the String provided. This is needed if you want to access image pixel data. Refer to CORS Settings for valid String values.
`errorOverlayUrl`	`String`	`''`	URL to the overlay image to show in place of the overlay that failed to load.
`zIndex`	`Number`	`1`	The explicit zIndex of the overlay layer.
`className`	`String`	`''`	A custom class name to assign to the image. Empty by default.

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

Event	Data	Description
`load`	`Event`	Fired when the ImageOverlay layer has loaded its image
`error`	`Event`	Fired when the ImageOverlay layer fails to load its image

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`setOpacity(<Number> opacity)`	`this`	Sets the opacity of the overlay.
`bringToFront()`	`this`	Brings the layer to the top of all overlays.
`bringToBack()`	`this`	Brings the layer to the bottom of all overlays.
`setUrl(<String> url)`	`this`	Changes the URL of the image.
`setBounds(<LatLngBounds> bounds)`	`this`	Update the bounds that this ImageOverlay covers
`setZIndex(<Number> value)`	`this`	Changes the zIndex of the image overlay.
`getBounds()`	`LatLngBounds`	Get the bounds that this ImageOverlay covers
`getElement()`	`HTMLElement`	Returns the instance of `HTMLImageElement` used by this overlay.
`getCenter()`	`LatLng`	Returns the center of the ImageOverlay.

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

VideoOverlay

Used to load and display a video player over specific bounds of the map. Extends `ImageOverlay`.

A video overlay uses the `<video>` HTML5 element.

Usage example

```
var videoUrl = 'https://www.mapbox.com/bites/00188/patricia_nasa.webm',
    videoBounds = [[ 32, -130], [ 13, -100]];
L.videoOverlay(videoUrl, videoBounds ).addTo(map);
```

Creation

Factory	Description
`L.videoOverlay(<String|Array|HTMLVideoElement> video, <LatLngBounds> bounds, <VideoOverlay options> options?)`	Instantiates an image overlay object given the URL of the video (or array of URLs, or even a video element) and the geographical bounds it is tied to.

Options

Option	Type	Default	Description
`autoplay`	`Boolean`	`true`	Whether the video starts playing automatically when loaded. On some browsers autoplay will only work with `muted: true`
`loop`	`Boolean`	`true`	Whether the video will loop back to the beginning when played.
`keepAspectRatio`	`Boolean`	`true`	Whether the video will save aspect ratio after the projection. Relevant for supported browsers. See browser compatibility
`muted`	`Boolean`	`false`	Whether the video starts on mute when loaded.
`playsInline`	`Boolean`	`true`	Mobile browsers will play the video right where it is instead of open it up in fullscreen mode.

▶ Options inherited from ImageOverlay

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

Event	Data	Description
`load`	`Event`	Fired when the video has finished loading the first frame

▶ Events inherited from ImageOverlay

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`getElement()`	`HTMLVideoElement`	Returns the instance of `HTMLVideoElement` used by this overlay.

▶ Methods inherited from ImageOverlay

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

SVGOverlay

Used to load, display and provide DOM access to an SVG file over specific bounds of the map. Extends `ImageOverlay`.

An SVG overlay uses the <svg> element.

Usage example

```
var svgElement = document.createElementNS("http://www.w3.org/2000/svg", "svg");
svgElement.setAttribute('xmlns', "http://www.w3.org/2000/svg");
svgElement.setAttribute('viewBox', "0 0 200 200");
svgElement.innerHTML = '<rect width="200" height="200"/><rect x="75" y="23" wid
var svgElementBounds = [ [ 32, -130 ], [ 13, -100 ] ];
L.svgOverlay(svgElement, svgElementBounds).addTo(map);
```

Creation

Factory	Description
`L.svgOverlay(<String|SVGElement> svg, <LatLngBounds> bounds, <SVGOverlay options> options?)`	Instantiates an image overlay object given an SVG element and the geographical bounds it is tied to. A viewBox attribute is required on the SVG element to zoom in and out properly.

Options

▶ Options inherited from ImageOverlay

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ Events inherited from ImageOverlay

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`getElement()`	`SVGElement`	Returns the instance of `SVGElement` used by this overlay.

▶ Methods inherited from ImageOverlay

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Path

An abstract class that contains options and constants shared between vector overlays (Polygon, Polyline, Circle). Do not use it directly. Extends `Layer`.

Options

Option	Type	Default	Description
`stroke`	`Boolean`	`true`	Whether to draw stroke along the path. Set it to `false` to disable borders on polygons or circles.
`color`	`String`	`'#3388ff'`	Stroke color
`weight`	`Number`	`3`	Stroke width in pixels
`opacity`	`Number`	`1.0`	Stroke opacity
`lineCap`	`String`	`'round'`	A string that defines shape to be used at the end of the stroke.
`lineJoin`	`String`	`'round'`	A string that defines shape to be used at the corners of the stroke.
`dashArray`	`String`	`null`	A string that defines the stroke dash pattern. Doesn't work on `Canvas`-powered layers in some old browsers.
`dashOffset`	`String`	`null`	A string that defines the distance into the dash pattern to start the dash. Doesn't work on `Canvas`-powered layers in some old browsers.
`fill`	`Boolean`	`depends`	Whether to fill the path with color. Set it to `false` to disable filling on polygons or circles.
`fillColor`	`String`	`*`	Fill color. Defaults to the value of the `color` option
`fillOpacity`	`Number`	`0.2`	Fill opacity.
`fillRule`	`String`	`'evenodd'`	A string that defines how the inside of a shape is determined.
`bubblingMouseEvents`	`Boolean`	`true`	When `true`, a mouse event on this path will trigger the same event on the map (unless `L.DomEvent.stopPropagation` is used).
`renderer`	`Renderer`		Use this specific instance of `Renderer` for this path. Takes precedence over the map's default renderer.
`className`	`String`	`null`	Custom class name set on an element. Only for SVG renderer.

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`redraw()`	`this`	Redraws the layer. Sometimes useful after you changed the coordinates that the path uses.
`setStyle(<Path options> style)`	`this`	Changes the appearance of a Path based on the options in the `Path options` object.
`bringToFront()`	`this`	Brings the layer to the top of all path layers.
`bringToBack()`	`this`	Brings the layer to the bottom of all path layers.

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Polyline

A class for drawing polyline overlays on a map. Extends `Path`.

Usage example

```
// create a red polyline from an array of LatLng points
var latlngs = [
    [45.51, -122.68],
    [37.77, -122.43],
    [34.04, -118.2]
];

var polyline = L.polyline(latlngs, {color: 'red'}).addTo(map);
```

```
// zoom the map to the polyline
map.fitBounds(polyline.getBounds());
```



You can also pass a multi-dimensional array to represent a `MultiPolyline` shape:

```
// create a red polyline from an array of arrays of LatLng points
var latlngs = [
    [[45.51, -122.68],
     [37.77, -122.43],
     [34.04, -118.2]],
    [[40.78, -73.91],
     [41.83, -87.62],
     [32.76, -96.72]]
];
```

Creation

Factory	Description
`L.polyline(<LatLng[]> latlngs, <Polyline options> options?)`	Instantiates a polyline object given an array of geographical points and optionally an options object. You can create a `Polyline` object with multiple separate lines (`MultiPolyline`) by passing an array of arrays of geographic points.

Options

Option	Type	Default	Description
`smoothFactor`	`Number`	`1.0`	How much to simplify the polyline on each zoom level. More means better performance and smoother look, and less means more accurate representation.
`noClip`	`Boolean`	`false`	Disable polyline clipping.

▶ Options inherited from Path

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer



Methods

Method	Returns	Description
`toGeoJSON(<Number|false> precision?)`	`Object`	Coordinates values are rounded with `formatNum` function with given `precision`. Returns a `GeoJSON` representation of the polyline (as a GeoJSON `LineString` or `MultiLineString` Feature).
`getLatLngs()`	`LatLng[]`	Returns an array of the points in the path, or nested arrays of points in case of multi-polyline.
`setLatLngs(<LatLng[]> latlngs)`	`this`	Replaces all the points in the polyline with the given array of geographical points.
`isEmpty()`	`Boolean`	Returns `true` if the Polyline has no LatLngs.
`closestLayerPoint(<Point> p)`	`Point`	Returns the point closest to `p` on the Polyline.
`getCenter()`	`LatLng`	Returns the center (centroid) of the polyline.
`getBounds()`	`LatLngBounds`	Returns the `LatLngBounds` of the path.
`addLatLng(<LatLng> latlng, <LatLng[]> latlngs?)`	`this`	Adds a given point to the polyline. By default, adds to the first ring of the polyline in case of a multi-polyline, but can be overridden by passing a specific ring as a LatLng array (that you can earlier access with `getLatLngs`).

▶ Methods inherited from Path

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Polygon

A class for drawing polygon overlays on a map. Extends `Polyline`.

Note that points you pass when creating a polygon shouldn't have an additional last point equal to the first one — it's better to filter out such points.

Usage example

```
// create a red polygon from an array of LatLng points
var latlngs = [[37, -109.05],[41, -109.03],[41, -102.05],[37, -102.04]];

var polygon = L.polygon(latlngs, {color: 'red'}).addTo(map);

// zoom the map to the polygon
map.fitBounds(polygon.getBounds());
```

You can also pass an array of arrays of latlngs, with the first array representing the outer shape and the other arrays representing holes in the outer shape:

```
var latlngs = [
  [[37, -109.05],[41, -109.03],[41, -102.05],[37, -102.04]], // outer ring
  [[37.29, -108.58],[40.71, -108.58],[40.71, -102.50],[37.29, -102.50]] // hole
];
```

Additionally, you can pass a multi-dimensional array to represent a MultiPolygon shape.

```
var latlngs = [
  [ // first polygon
    [[37, -109.05],[41, -109.03],[41, -102.05],[37, -102.04]], // outer ring
    [[37.29, -108.58],[40.71, -108.58],[40.71, -102.50],[37.29, -102.50]] // ho
  ],
  [ // second polygon
    [[41, -111.03],[45, -111.04],[45, -104.05],[41, -104.05]]
  ]
];
```

Creation

Factory	Description
`L.polygon(<LatLng[]> latlngs, <Polyline options> options?)`	

Options

▶ Options inherited from Polyline

▶ Options inherited from Path

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`toGeoJSON(<Number|false> precision?)`	`Object`	Coordinates values are rounded with `formatNum` function with given `precision`. Returns a `GeoJSON` representation of the polygon (as a GeoJSON `Polygon` or `MultiPolygon` Feature).
`getCenter()`	`LatLng`	Returns the center (centroid) of the Polygon.

▶ Methods inherited from Polyline

▶ Methods inherited from Path

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Rectangle

A class for drawing rectangle overlays on a map. Extends Polygon.



Usage example

```
// define rectangle geographical bounds
var bounds = [[54.559322, -5.767822], [56.1210604, -3.021240]];

// create an orange rectangle
L.rectangle(bounds, {color: "#ff7800", weight: 1}).addTo(map);

// zoom the map to the rectangle bounds
map.fitBounds(bounds);
```

Creation

Factory	Description
`L.rectangle(<LatLngBounds> latLngBounds, <Polyline options> options?)`	

Options

▶ Options inherited from Polyline

▶ Options inherited from Path

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`setBounds(<LatLngBounds> latLngBounds)`	`this`	Redraws the rectangle with the passed bounds.

▶ Methods inherited from Polygon

▶ Methods inherited from Polyline

▶ Methods inherited from Path

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Circle

A class for drawing circle overlays on a map. Extends `CircleMarker`.

It's an approximation and starts to diverge from a real circle closer to poles (due to projection distortion).

Usage example

```
L.circle([50.5, 30.5], {radius: 200}).addTo(map);
```

Creation

Factory	Description
`L.circle(<LatLng> latlng, <Circle options> options?)`	Instantiates a circle object given a geographical point, and an options object which contains the circle radius.
`L.circle(<LatLng> latlng, <Number> radius, <Circle options> options?)`	Obsolete way of instantiating a circle, for compatibility with 0.7.x code. Do not use in new applications or plugins.

Options

Option	Type	Default	Description
`radius`	`Number`		Radius of the circle, in meters.

▶ Options inherited from Path

▶ Options inherited from Interactive layer

▶ Options inherited from Layer



Events

▶ Events inherited from CircleMarker

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`setRadius(<Number> radius)`	`this`	Sets the radius of a circle. Units are in meters.
`getRadius()`	`Number`	Returns the current radius of a circle. Units are in meters.
`getBounds()`	`LatLngBounds`	Returns the `LatLngBounds` of the path.

▶ Methods inherited from CircleMarker

▶ Methods inherited from Path

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

CircleMarker

A circle of a fixed size with radius specified in pixels. Extends `Path`.

Creation

Factory	Description
`L.circleMarker(<LatLng> latlng, <CircleMarker options> options?)`	Instantiates a circle marker object given a geographical point, and an optional options object.

Options

Option	Type	Default	Description
`radius`	`Number`	`10`	Radius of the circle marker, in pixels

▶ Options inherited from Path

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

Event	Data	Description
`move`	`Event`	Fired when the marker is moved via `setLatLng`. Old and new coordinates are included in event arguments as `oldLatLng`, `latlng`.

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`toGeoJSON(<Number|false> precision?)`	`Object`	Coordinates values are rounded with `formatNum` function with given `precision`. Returns a `GeoJSON` representation of the circle marker (as a GeoJSON `Point` Feature).
`setLatLng(<LatLng> latLng)`	`this`	Sets the position of a circle marker to a new location.
`getLatLng()`	`LatLng`	Returns the current geographical position of the circle marker
`setRadius(<Number> radius)`	`this`	Sets the radius of a circle marker. Units are in pixels.

Method	Returns	Description
`getRadius()`	`Number`	Returns the current radius of the circle



▶ Methods inherited from Path

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

SVG

Allows vector layers to be displayed with SVG. Inherits `Renderer`.

Due to technical limitations, SVG is not available in all web browsers, notably Android 2.x and 3.x.

Although SVG is not available on IE7 and IE8, these browsers support VML (a now deprecated technology), and the SVG renderer will fall back to VML in this case.

VML was deprecated in 2012, which means VML functionality exists only for backwards compatibility with old versions of Internet Explorer.

Usage example

Use SVG by default for all paths in the map:

```
var map = L.map('map', {
    renderer: L.svg()
});
```

Use a SVG renderer with extra padding for specific vector geometries:

```
var map = L.map('map');
var myRenderer = L.svg({ padding: 0.5 });
var line = L.polyline( coordinates, { renderer: myRenderer } );
var circle = L.circle( center, { renderer: myRenderer } );
```

Creation

Factory	Description
`L.svg(<Renderer options> options?)`	Creates a SVG renderer with the given options.



Options

▶ Options inherited from Renderer

▶ Options inherited from Layer

Events

▶ Events inherited from Renderer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Functions

There are several static functions which can be called without instantiating L.SVG:

Function	Returns	Description
`create(<String> name)`	`SVGElement`	Returns a instance of SVGElement, corresponding to the class name passed. For example, using 'line' will return an instance of SVGLineElement.
`pointsToPath(<Point[]> rings, <Boolean> closed)`	`String`	Generates a SVG path string for multiple rings, with each ring turning into "M..L..L.." instructions

Canvas

Allows vector layers to be displayed with `<canvas>`. Inherits `Renderer`.

Due to technical limitations, Canvas is not available in all web browsers, notably IE8, and overlapping geometries might not display properly in some edge cases.

Usage example

Use Canvas by default for all paths in the map:

```
var map = L.map('map', {
    renderer: L.canvas()
});
```

Use a Canvas renderer with extra padding for specific vector geometries:

```
var map = L.map('map');
var myRenderer = L.canvas({ padding: 0.5 });
var line = L.polyline( coordinates, { renderer: myRenderer } );
var circle = L.circle( center, { renderer: myRenderer } );
```

Creation

Factory	Description
`L.canvas(<Renderer options> options?)`	Creates a Canvas renderer with the given options.

Options

Option	Type	Default	Description
`tolerance`	`Number`	`0`	How much to extend the click tolerance around a path/object on the map.

▶ Options inherited from Renderer

▶ Options inherited from Layer

Events

▶ Events inherited from Renderer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

LayerGroup

Used to group several layers and handle them as one. If you add it to the map, any layers added or removed from the group will be added/removed on the map as well. Extends `Layer`.

Usage example

```
L.layerGroup([marker1, marker2])
    .addLayer(polyline)
    .addTo(map);
```

Creation

Factory	Description
`L.layerGroup(<Layer[]> layers?, <Object> options?)`	Create a layer group, optionally given an initial set of layers and an `options` object.

Options

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`toGeoJSON(<Number|false> precision?)`	`Object`	Coordinates values are rounded with `formatNum` function with given `precision`. Returns a `GeoJSON` representation of the layer group (as a GeoJSON `FeatureCollection`, `GeometryCollection`, or `MultiPoint`).
`addLayer(<Layer> layer)`	`this`	Adds the given layer to the group.
`removeLayer(<Layer> layer)`	`this`	Removes the given layer from the group.
`removeLayer(<Number> id)`	`this`	Removes the layer with the given internal ID from the group.
`hasLayer(<Layer> layer)`	`Boolean`	Returns `true` if the given layer is currently added to the group.
`hasLayer(<Number> id)`	`Boolean`	Returns `true` if the given internal ID is currently added to the group.
`clearLayers()`	`this`	Removes all the layers from the group.
`invoke(<String> methodName, …)`	`this`	Calls `methodName` on every layer contained in this group, passing any additional parameters. Has no effect if the layers contained do not implement `methodName`.
`eachLayer(<Function> fn, <Object> context?)`	`this`	Iterates over the layers of the group, optionally specifying context of the iterator function. `group.eachLayer(function (layer) { layer.bindPopup('Hello'); });`
`getLayer(<Number> id)`	`Layer`	Returns the layer with the given internal ID.
`getLayers()`	`Layer[]`	Returns an array of all the layers added to the group.
`setZIndex(<Number> zIndex)`	`this`	Calls `setZIndex` on every layer contained in this group, passing the z-index.
`getLayerId(<Layer> layer)`	`Number`	Returns the internal ID for a layer

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented



FeatureGroup

Extended `LayerGroup` that makes it easier to do the same thing to all its member layers:

- `bindPopup` binds a popup to all of the layers at once (likewise with `bindTooltip`)
- Events are propagated to the `FeatureGroup`, so if the group has an event handler, it will handle events from any of the layers. This includes mouse events and custom events.
- Has `layeradd` and `layerremove` events

Usage example

```
L.featureGroup([marker1, marker2, polyline])
    .bindPopup('Hello world!')
    .on('click', function() { alert('Clicked on a member of the group!'); })
    .addTo(map);
```

Creation

Factory	Description
`L.featureGroup(<Layer[]> layers?, <Object> options?)`	Create a feature group, optionally given an initial set of layers and an `options` object.

Options

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

Event	Data	Description
`layeradd`	`LayerEvent`	Fired when a layer is added to this `FeatureGroup`
`layerremove`	`LayerEvent`	Fired when a layer is removed from this `FeatureGroup`

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`setStyle(<Path options> style)`	`this`	Sets the given path options to each layer of the group that has a `setStyle` method.
`bringToFront()`	`this`	Brings the layer group to the top of all other layers
`bringToBack()`	`this`	Brings the layer group to the back of all other layers
`getBounds()`	`LatLngBounds`	Returns the LatLngBounds of the Feature Group (created from bounds and coordinates of its children).

▶ Methods inherited from LayerGroup

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

GeoJSON

Represents a GeoJSON object or an array of GeoJSON objects. Allows you to parse GeoJSON data and display it on the map. Extends `FeatureGroup`.

Usage example

```
L.geoJSON(data, {
    style: function (feature) {
        return {color: feature.properties.color};
    }
}).bindPopup(function (layer) {
```

```
    return layer.feature.properties.description;
}).addTo(map);
```

Creation

Factory	Description
`L.geoJSON(<Object> geojson?, <GeoJSON options> options?)`	Creates a GeoJSON layer. Optionally accepts an object in GeoJSON format to display on the map (you can alternatively add it later with `addData` method) and an `options` object.

Options

Option	Type	Default	Description
`pointToLayer`	`Function`	*	A `Function` defining how GeoJSON points spawn Leaflet layers. It is internally called when data is added, passing the GeoJSON point feature and its `LatLng`. The default is to spawn a default `Marker`: `function(geoJsonPoint, latlng) { return L.marker(latlng); }`
`style`	`Function`	*	A `Function` defining the `Path options` for styling GeoJSON lines and polygons, called internally when data is added. The default value is to not override any defaults: `function (geoJsonFeature) { return {} }`
`onEachFeature`	`Function`	*	A `Function` that will be called once for each created `Feature`, after it has been created and styled. Useful for attaching events and popups to features. The default is to do nothing with the newly created layers: `function (feature, layer) {}`

Option	Type	Default	Description
`filter`	`Function`	`*`	A `Function` that will be used to decide whether to include a feature or not. The default is to include all features: `function (geoJsonFeature) { return true; }` Note: dynamically changing the `filter` option will have effect only on newly added data. It will *not* re-evaluate already included features.
`coordsToLatLng`	`Function`	`*`	A `Function` that will be used for converting GeoJSON coordinates to `LatLng`s. The default is the `coordsToLatLng` static method.
`markersInheritOptions`	`Boolean`	`false`	Whether default Markers for "Point" type Features inherit from group options.

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

Events

▶ Events inherited from FeatureGroup

▶ Mouse events inherited from Interactive layer

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`addData(`*`data`*`)`	`this`	Adds a GeoJSON object to the layer.
`resetStyle(`*`layer?`*`)`	`this`	Resets the given vector layer's style to the original GeoJSON style, useful for resetting style after hover events. If `layer` is omitted, the style of all features in the current layer is reset.

Method	Returns	Description
`setStyle(style)`	`this`	Changes styles of GeoJSON vector layers with the given style function.



▶ Methods inherited from FeatureGroup

▶ Methods inherited from LayerGroup

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Functions

There are several static functions which can be called without instantiating L.GeoJSON:

Function	Returns	Description
`geometryToLayer(<Object> featureData, <GeoJSON options> options?)`	`Layer`	Creates a `Layer` from a given GeoJSON feature. Can use a custom `pointToLayer` and/or `coordsToLatLng` functions if provided as options.
`coordsToLatLng(<Array> coords)`	`LatLng`	Creates a `LatLng` object from an array of 2 numbers (longitude, latitude) or 3 numbers (longitude, latitude, altitude) used in GeoJSON for points.
`coordsToLatLngs(<Array> coords, <Number> levelsDeep?, <Function> coordsToLatLng?)`	`Array`	Creates a multidimensional array of `LatLng`s from a GeoJSON coordinates array. `levelsDeep` specifies the nesting level (0 is for an array of points, 1 for an array of arrays of points, etc., 0 by default). Can use a custom `coordsToLatLng` function.
`latLngToCoords(<LatLng> latlng, <Number|false> precision?)`	`Array`	Reverse of `coordsToLatLng` Coordinates values are rounded with `formatNum` function.
`latLngsToCoords(<Array> latlngs, <Number> levelsDeep?, <Boolean> closed?, <Number|false> precision?)`	`Array`	Reverse of `coordsToLatLngs` `closed` determines whether the first point should be appended to the end of the array to close the feature, only used when `levelsDeep` is 0. False by default. Coordinates values are rounded with `formatNum` function.

Function	Returns	Description
`asFeature(<Object> geojson)`	`Object`	Normalize GeoJSON geometries/features into GeoJSON features.



GridLayer

Generic class for handling a tiled grid of HTML elements. This is the base class for all tile layers and replaces `TileLayer.Canvas`. GridLayer can be extended to create a tiled grid of HTML elements like `<canvas>`, `<img>` or `<div>`. GridLayer will handle creating and animating these DOM elements for you.

Usage example

Synchronous usage

To create a custom layer, extend GridLayer and implement the `createTile()` method, which will be passed a `Point` object with the x, y, and z (zoom level) coordinates to draw your tile.

```
var CanvasLayer = L.GridLayer.extend({
    createTile: function(coords){
        // create a <canvas> element for drawing
        var tile = L.DomUtil.create('canvas', 'leaflet-tile');

        // setup tile width and height according to the options
        var size = this.getTileSize();
        tile.width = size.x;
        tile.height = size.y;

        // get a canvas context and draw something on it using coords.x, coords
        var ctx = tile.getContext('2d');

        // return the tile so it can be rendered on screen
        return tile;
    }
});
```

Asynchronous usage

Tile creation can also be asynchronous, this is useful when using a third-party drawing library. Once the tile is finished drawing it can be passed to the `done()` callback.

```
var CanvasLayer = L.GridLayer.extend({
    createTile: function(coords, done){
        var error;

        // create a <canvas> element for drawing
        var tile = L.DomUtil.create('canvas', 'leaflet-tile');

        // setup tile width and height according to the options
        var size = this.getTileSize();
        tile.width = size.x;
        tile.height = size.y;

        // draw something asynchronously and pass the tile to the done() callba
        setTimeout(function() {
            done(error, tile);
        }, 1000);

        return tile;
    }
});
```

Creation

Factory	Description
`L.gridLayer(<GridLayer options> options?)`	Creates a new instance of GridLayer with the supplied options.

Options

Option	Type	Default	Description
`tileSize`	`Number|Point`	`256`	Width and height of tiles in the grid. Use a number if width and height are equal, or `L.point(width, height)` otherwise.
`opacity`	`Number`	`1.0`	Opacity of the tiles. Can be used in the `createTile()` function.
`updateWhenIdle`	`Boolean`	`(depends)`	Load new tiles only when panning ends. `true` by default on mobile browsers, in order to avoid too many requests and keep smooth navigation. `false` otherwise in order to display new tiles *during* panning, since it is easy to pan outside the `keepBuffer` option in desktop browsers.

Option	Type	Default	Description
`updateWhenZooming`	`Boolean`	`true`	By default, a smooth zoom animation (during a touch zoom or a `flyTo()`) will update grid layers every integer zoom level. Setting this option to `false` will update the grid layer only when the smooth animation ends.
`updateInterval`	`Number`	`200`	Tiles will not update more than once every `updateInterval` milliseconds when panning.
`zIndex`	`Number`	`1`	The explicit zIndex of the tile layer.
`bounds`	`LatLngBounds`	`undefined`	If set, tiles will only be loaded inside the set `LatLngBounds`.
`minZoom`	`Number`	`0`	The minimum zoom level down to which this layer will be displayed (inclusive).
`maxZoom`	`Number`	`undefined`	The maximum zoom level up to which this layer will be displayed (inclusive).
`maxNativeZoom`	`Number`	`undefined`	Maximum zoom number the tile source has available. If it is specified, the tiles on all zoom levels higher than `maxNativeZoom` will be loaded from `maxNativeZoom` level and auto-scaled.
`minNativeZoom`	`Number`	`undefined`	Minimum zoom number the tile source has available. If it is specified, the tiles on all zoom levels lower than `minNativeZoom` will be loaded from `minNativeZoom` level and auto-scaled.
`noWrap`	`Boolean`	`false`	Whether the layer is wrapped around the antimeridian. If `true`, the GridLayer will only be displayed once at low zoom levels. Has no effect when the map CRS doesn't wrap around. Can be used in combination with `bounds` to prevent requesting tiles outside the CRS limits.
`pane`	`String`	`'tilePane'`	`Map` pane where the grid layer will be added.

Option	Type	Default	Description
`className`	`String`	`''`	A custom class name to assign to the tile layer. Empty by default.
`keepBuffer`	`Number`	`2`	When panning the map, keep this many rows and columns of tiles before unloading them.



▶ Options inherited from Layer

Events

Event	Data	Description
`loading`	`Event`	Fired when the grid layer starts loading tiles.
`tileunload`	`TileEvent`	Fired when a tile is removed (e.g. when a tile goes off the screen).
`tileloadstart`	`TileEvent`	Fired when a tile is requested and starts loading.
`tileerror`	`TileErrorEvent`	Fired when there is an error loading a tile.
`tileload`	`TileEvent`	Fired when a tile loads.
`load`	`Event`	Fired when the grid layer loaded all visible tiles.

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

Method	Returns	Description
`bringToFront()`	`this`	Brings the tile layer to the top of all tile layers.
`bringToBack()`	`this`	Brings the tile layer to the bottom of all tile layers.
`getContainer()`	`HTMLElement`	Returns the HTML element that contains the tiles for this layer.
`setOpacity(<Number> opacity)`	`this`	Changes the opacity of the grid layer.
`setZIndex(<Number> zIndex)`	`this`	Changes the zIndex of the grid layer.
`isLoading()`	`Boolean`	Returns `true` if any tile in the grid layer has

Method	Returns	Description
		not finished loading.
`redraw()`	`this`	Causes the layer to clear all the tiles and request them again.
`getTileSize()`	`Point`	Normalizes the tileSize option into a point. Used by the `createTile()` method.

Extension methods

Layers extending `GridLayer` shall reimplement the following method.

Method	Returns	Description
`createTile(<Object> coords, <Function> done?)`	`HTMLElement`	Called only internally, must be overridden by classes extending `GridLayer`. Returns the `HTMLElement` corresponding to the given `coords`. If the done callback is specified, it must be called when the tile has finished loading and drawing.

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

LatLng

Represents a geographical point with a certain latitude and longitude.

Usage example

```
var latlng = L.latLng(50.5, 30.5);
```

All Leaflet methods that accept LatLng objects also accept them in a simple Array form and simple object form (unless noted otherwise), so these lines are equivalent:

```
map.panTo([50, 30]);
map.panTo({lng: 30, lat: 50});
map.panTo({lat: 50, lng: 30});
map.panTo(L.latLng(50, 30));
```

Note that `LatLng` does not inherit from Leaflet's `Class` object, which means new classes can't inherit from it, and new methods can't be added to it with the `include` function.



Creation

Factory	Description
`L.latLng(<Number> latitude, <Number> longitude, <Number> altitude?)`	Creates an object representing a geographical point with the given latitude and longitude (and optionally altitude).
`L.latLng(<Array> coords)`	Expects an array of the form `[Number, Number]` or `[Number, Number, Number]` instead.
`L.latLng(<Object> coords)`	Expects an plain object of the form `{lat: Number, lng: Number}` or `{lat: Number, lng: Number, alt: Number}` instead.

Methods

Method	Returns	Description
`equals(<LatLng> otherLatLng, <Number> maxMargin?)`	`Boolean`	Returns `true` if the given `LatLng` point is at the same position (within a small margin of error). The margin of error can be overridden by setting `maxMargin` to a small number.
`toString()`	`String`	Returns a string representation of the point (for debugging purposes).
`distanceTo(<LatLng> otherLatLng)`	`Number`	Returns the distance (in meters) to the given `LatLng` calculated using the Spherical Law of Cosines.
`wrap()`	`LatLng`	Returns a new `LatLng` object with the longitude wrapped so it's always between -180 and +180 degrees.
`toBounds(<Number> sizeInMeters)`	`LatLngBounds`	Returns a new `LatLngBounds` object in which each boundary is `sizeInMeters/2` meters apart from the `LatLng`.

Properties

Property	Type	Description
`lat`	`Number`	Latitude in degrees
`lng`	`Number`	Longitude in degrees

Property	Type	Description
`alt`	`Number`	Altitude in meters (optional)



LatLngBounds

Represents a rectangular geographical area on a map.

Usage example

```
var corner1 = L.latLng(40.712, -74.227),
corner2 = L.latLng(40.774, -74.125),
bounds = L.latLngBounds(corner1, corner2);
```

All Leaflet methods that accept LatLngBounds objects also accept them in a simple Array form (unless noted otherwise), so the bounds example above can be passed like this:

```
map.fitBounds([
    [40.712, -74.227],
    [40.774, -74.125]
]);
```

Caution: if the area crosses the antimeridian (often confused with the International Date Line), you must specify corners *outside* the [-180, 180] degrees longitude range.

Note that `LatLngBounds` does not inherit from Leaflet's `Class` object, which means new classes can't inherit from it, and new methods can't be added to it with the `include` function.

Creation

Factory	Description
`L.latLngBounds(<LatLng> corner1, <LatLng> corner2)`	Creates a `LatLngBounds` object by defining two diagonally opposite corners of the rectangle.
`L.latLngBounds(<LatLng[]> latlngs)`	Creates a `LatLngBounds` object defined by the geographical points it contains. Very useful for zooming the map to fit a particular set of locations with `fitBounds`.

Methods

Method	Returns	Description
`extend(<LatLng> latlng)`	`this`	Extend the bounds to contain the given point
`extend(<LatLngBounds> otherBounds)`	`this`	Extend the bounds to contain the given bounds
`pad(<Number> bufferRatio)`	`LatLngBounds`	Returns bounds created by extending or retracting the current bounds by a given ratio in each direction. For example, a ratio of 0.5 extends the bounds by 50% in each direction. Negative values will retract the bounds.
`getCenter()`	`LatLng`	Returns the center point of the bounds.
`getSouthWest()`	`LatLng`	Returns the south-west point of the bounds.
`getNorthEast()`	`LatLng`	Returns the north-east point of the bounds.
`getNorthWest()`	`LatLng`	Returns the north-west point of the bounds.
`getSouthEast()`	`LatLng`	Returns the south-east point of the bounds.
`getWest()`	`Number`	Returns the west longitude of the bounds
`getSouth()`	`Number`	Returns the south latitude of the bounds
`getEast()`	`Number`	Returns the east longitude of the bounds
`getNorth()`	`Number`	Returns the north latitude of the bounds
`contains(<LatLngBounds> otherBounds)`	`Boolean`	Returns `true` if the rectangle contains the given one.
`contains(<LatLng> latlng)`	`Boolean`	Returns `true` if the rectangle contains the given point.
`intersects(<LatLngBounds> otherBounds)`	`Boolean`	Returns `true` if the rectangle intersects the given bounds. Two bounds intersect if they have at least one point in common.
`overlaps(<LatLngBounds> otherBounds)`	`Boolean`	Returns `true` if the rectangle overlaps the given bounds. Two bounds overlap if their intersection is an area.
`toBBoxString()`	`String`	Returns a string with bounding box coordinates in a 'southwest_lng,southwest_lat,northeast_lng,northeast_lat' format. Useful for sending requests to web services that return geo data.
`equals(<LatLngBounds> otherBounds, <Number> maxMargin?)`	`Boolean`	Returns `true` if the rectangle is equivalent (within a small margin of error) to the given bounds. The margin of error can be overridden by setting `maxMargin` to a small number.
`isValid()`	`Boolean`	Returns `true` if the bounds are properly initialized.

Point

Represents a point with x and y coordinates in pixels.

Usage example

```
var point = L.point(200, 300);
```

All Leaflet methods and options that accept `Point` objects also accept them in a simple Array form (unless noted otherwise), so these lines are equivalent:

```
map.panBy([200, 300]);
map.panBy(L.point(200, 300));
```

Note that `Point` does not inherit from Leaflet's `Class` object, which means new classes can't inherit from it, and new methods can't be added to it with the `include` function.

Creation

Factory	Description
`L.point(<Number> x, <Number> y, <Boolean> round?)`	Creates a Point object with the given x and y coordinates. If optional `round` is set to true, rounds the x and y values.
`L.point(<Number[]> coords)`	Expects an array of the form `[x, y]` instead.
`L.point(<Object> coords)`	Expects a plain object of the form `{x: Number, y: Number}` instead.

Methods

Method	Returns	Description
`clone()`	`Point`	Returns a copy of the current point.
`add(<Point> otherPoint)`	`Point`	Returns the result of addition of the current and the given points.
`subtract(<Point> otherPoint)`	`Point`	Returns the result of subtraction of the given point from the current.
`divideBy(<Number> num)`	`Point`	Returns the result of division of the current point by the given number.

Method	Returns	Description
`multiplyBy(<Number> num)`	`Point`	Returns the result of multiplication of the current point by the given number.
`scaleBy(<Point> scale)`	`Point`	Multiply each coordinate of the current point by each coordinate of `scale`. In linear algebra terms, multiply the point by the scaling matrix defined by `scale`.
`unscaleBy(<Point> scale)`	`Point`	Inverse of `scaleBy`. Divide each coordinate of the current point by each coordinate of `scale`.
`round()`	`Point`	Returns a copy of the current point with rounded coordinates.
`floor()`	`Point`	Returns a copy of the current point with floored coordinates (rounded down).
`ceil()`	`Point`	Returns a copy of the current point with ceiled coordinates (rounded up).
`trunc()`	`Point`	Returns a copy of the current point with truncated coordinates (rounded towards zero).
`distanceTo(<Point> otherPoint)`	`Number`	Returns the cartesian distance between the current and the given points.
`equals(<Point> otherPoint)`	`Boolean`	Returns `true` if the given point has the same coordinates.
`contains(<Point> otherPoint)`	`Boolean`	Returns `true` if both coordinates of the given point are less than the corresponding current point coordinates (in absolute values).
`toString()`	`String`	Returns a string representation of the point for debugging purposes.

Properties

Property	Type	Description
x	`Number`	The x coordinate of the point
y	`Number`	The y coordinate of the point

Bounds

Represents a rectangular area in pixel coordinates.

Usage example

```
var p1 = L.point(10, 10),
p2 = L.point(40, 60),
bounds = L.bounds(p1, p2);
```

All Leaflet methods that accept Bounds objects also accept them in a simple Array form (unless noted otherwise), so the bounds example above can be passed like this:

```
otherBounds.intersects([[10, 10], [40, 60]]);
```

Note that Bounds does not inherit from Leaflet's Class object, which means new classes can't inherit from it, and new methods can't be added to it with the `include` function.

Creation

Factory	Description
`L.bounds(<Point> corner1, <Point> corner2)`	Creates a Bounds object from two corners coordinate pairs.
`L.bounds(<Point[]> points)`	Creates a Bounds object from the given array of points.

Methods

Method	Returns	Description
`extend(<Point> point)`	`this`	Extends the bounds to contain the given point.
`extend(<Bounds> otherBounds)`	`this`	Extend the bounds to contain the given bounds
`getCenter(<Boolean> round?)`	`Point`	Returns the center point of the bounds.
`getBottomLeft()`	`Point`	Returns the bottom-left point of the bounds.
`getTopRight()`	`Point`	Returns the top-right point of the bounds.
`getTopLeft()`	`Point`	Returns the top-left point of the bounds (i.e. `this.min`).
`getBottomRight()`	`Point`	Returns the bottom-right point of the bounds (i.e. `this.max`).
`getSize()`	`Point`	Returns the size of the given bounds

Method	Returns	Description
`contains(<Bounds> otherBounds)`	`Boolean`	Returns `true` if the rectangle contains the given one.
`contains(<Point> point)`	`Boolean`	Returns `true` if the rectangle contains the given point.
`intersects(<Bounds> otherBounds)`	`Boolean`	Returns `true` if the rectangle intersects the given bounds. Two bounds intersect if they have at least one point in common.
`overlaps(<Bounds> otherBounds)`	`Boolean`	Returns `true` if the rectangle overlaps the given bounds. Two bounds overlap if their intersection is an area.
`isValid()`	`Boolean`	Returns `true` if the bounds are properly initialized.
`pad(<Number> bufferRatio)`	`Bounds`	Returns bounds created by extending or retracting the current bounds by a given ratio in each direction. For example, a ratio of 0.5 extends the bounds by 50% in each direction. Negative values will retract the bounds.
`equals(<Bounds> otherBounds)`	`Boolean`	Returns `true` if the rectangle is equivalent to the given bounds.

Properties

Property	Type	Description
`min`	`Point`	The top left corner of the rectangle.
`max`	`Point`	The bottom right corner of the rectangle.

Icon

Represents an icon to provide when creating a marker.

Usage example

```
var myIcon = L.icon({
    iconUrl: 'my-icon.png',
    iconSize: [38, 95],
    iconAnchor: [22, 94],
    popupAnchor: [-3, -76],
    shadowUrl: 'my-icon-shadow.png',
    shadowSize: [68, 95],
    shadowAnchor: [22, 94]
```

```
});

L.marker([50.505, 30.57], {icon: myIcon}).addTo(map);
```



`L.Icon.Default` extends `L.Icon` and is the blue icon Leaflet uses for markers by default.

Creation

Factory	Description
`L.icon(<Icon options> options)`	Creates an icon instance with the given options.

Options

Option	Type	Default	Description
`iconUrl`	`String`	`null`	**(required)** The URL to the icon image (absolute or relative to your script path).
`iconRetinaUrl`	`String`	`null`	The URL to a retina sized version of the icon image (absolute or relative to your script path). Used for Retina screen devices.
`iconSize`	`Point`	`null`	Size of the icon image in pixels.
`iconAnchor`	`Point`	`null`	The coordinates of the "tip" of the icon (relative to its top left corner). The icon will be aligned so that this point is at the marker's geographical location. Centered by default if size is specified, also can be set in CSS with negative margins.
`popupAnchor`	`Point`	`[0, 0]`	The coordinates of the point from which popups will "open", relative to the icon anchor.
`tooltipAnchor`	`Point`	`[0, 0]`	The coordinates of the point from which tooltips will "open", relative to the icon anchor.
`shadowUrl`	`String`	`null`	The URL to the icon shadow image. If not specified, no shadow image will be created.
`shadowRetinaUrl`	`String`	`null`	
`shadowSize`	`Point`	`null`	Size of the shadow image in pixels.
`shadowAnchor`	`Point`	`null`	The coordinates of the "tip" of the shadow (relative to its top left corner)

Option	Type	Default	Description
			(the same as iconAnchor if not specified).
`className`	`String`	`''`	A custom class name to assign to both icon and shadow images. Empty by default.
`crossOrigin`	`Boolean|String`	`false`	Whether the crossOrigin attribute will be added to the tiles. If a String is provided, all tiles will have their crossOrigin attribute set to the String provided. This is needed if you want to access tile pixel data. Refer to CORS Settings for valid String values.

Methods

Method	Returns	Description
`createIcon(<HTMLElement> oldIcon?)`	`HTMLElement`	Called internally when the icon has to be shown, returns a `<img>` HTML element styled according to the options.
`createShadow(<HTMLElement> oldIcon?)`	`HTMLElement`	As `createIcon`, but for the shadow beneath it.

Icon.Default

A trivial subclass of `Icon`, represents the icon to use in `Marker`s when no icon is specified. Points to the blue marker image distributed with Leaflet releases.

In order to customize the default icon, just change the properties of `L.Icon.Default.prototype.options` (which is a set of `Icon options`).

If you want to *completely* replace the default icon, override the `L.Marker.prototype.options.icon` with your own icon instead.

Option	Type	Default	Description
`imagePath`	`String`		`Icon.Default` will try to auto-detect the location of the blue icon images. If you are placing these images in a non-standard way, set this option to point to the right path.

DivIcon

Represents a lightweight icon for markers that uses a simple `<div>` element instead of an image. Inherits from `Icon` but ignores the `iconUrl` and shadow options.



Usage example

```
var myIcon = L.divIcon({className: 'my-div-icon'});
// you can set .my-div-icon styles in CSS

L.marker([50.505, 30.57], {icon: myIcon}).addTo(map);
```

By default, it has a 'leaflet-div-icon' CSS class and is styled as a little white square with a shadow.

Creation

Factory	Description
`L.divIcon(<DivIcon options> options)`	Creates a `DivIcon` instance with the given options.

Options

Option	Type	Default	Description
`html`	`String|HTMLElement`	`''`	Custom HTML code to put inside the div element, empty by default. Alternatively, an instance of `HTMLElement`.
`bgPos`	`Point`	`[0, 0]`	Optional relative position of the background, in pixels

▶ Options inherited from Icon

Methods

▶ Methods inherited from Icon

Control.Zoom

A basic zoom control with two buttons (zoom in and zoom out). It is put on the map by default unless you set its `zoomControl option` to `false`. Extends `Control`.

Creation

Factory	Description
`L.control.zoom(<Control.Zoom options> options)`	Creates a zoom control

Options

Option	Type	Default	Description
`zoomInText`	`String`	`'<span aria-hidden="true">+</span>'`	The text set on the 'zoom in' button.
`zoomInTitle`	`String`	`'Zoom in'`	The title set on the 'zoom in' button.
`zoomOutText`	`String`	`'<span aria-hidden="true">&#x2212;</span>'`	The text set on the 'zoom out' button.
`zoomOutTitle`	`String`	`'Zoom out'`	The title set on the 'zoom out' button.

▶ Options inherited from Control

Methods

▶ Methods inherited from Control

Control.Attribution

The attribution control allows you to display attribution data in a small text box on a map. It is put on the map by default unless you set its `attributionControl` option to `false`, and it fetches attribution texts from layers with the `getAttribution` method automatically. Extends Control.

Creation

Factory	Description
`L.control.attribution(<Control.Attribution options> options)`	Creates an attribution control.

Options

Option	Type	Default	Description
`prefix`	`String|false`	`'Leaflet'`	The HTML text shown before the attributions. Pass `false` to disable.

▶ Options inherited from Control

Methods

Method	Returns	Description
`setPrefix(<String|false> prefix)`	`this`	The HTML text shown before the attributions. Pass `false` to disable.
`addAttribution(<String> text)`	`this`	Adds an attribution text (e.g. `'&copy; OpenStreetMap contributors'`).
`removeAttribution(<String> text)`	`this`	Removes an attribution text.

▶ Methods inherited from Control

Control.Layers

The layers control gives users the ability to switch between different base layers and switch overlays on/off (check out the detailed example). Extends `Control`.

Usage example

```
var baseLayers = {
    "Mapbox": mapbox,
    "OpenStreetMap": osm
};

var overlays = {
    "Marker": marker,
    "Roads": roadsLayer
};

L.control.layers(baseLayers, overlays).addTo(map);
```

The `baseLayers` and `overlays` parameters are object literals with layer names as keys and `Layer` objects as values:

```
{
    "<someName1>": layer1,
```

```
    "<someName2>": layer2
}
```

The layer names can contain HTML, which allows you to add additional styling to the items:

```
{"<img src='my-layer-icon' /> <span class='my-layer-item'>My Layer</span>": myL
```

Creation

Factory	Description
`L.control.layers(<Object> baselayers?, <Object> overlays?, <Control.Layers options> options?)`	Creates a layers control with the given layers. Base layers will be switched with radio buttons, while overlays will be switched with checkboxes. Note that all base layers should be passed in the base layers object, but only one should be added to the map during map instantiation.

Options

Option	Type	Default	Description
`collapsed`	`Boolean`	`true`	If `true`, the control will be collapsed into an icon and expanded on mouse hover, touch, or keyboard activation.
`autoZIndex`	`Boolean`	`true`	If `true`, the control will assign zIndexes in increasing order to all of its layers so that the order is preserved when switching them on/off.
`hideSingleBase`	`Boolean`	`false`	If `true`, the base layers in the control will be hidden when there is only one.
`sortLayers`	`Boolean`	`false`	Whether to sort the layers. When `false`, layers will keep the order in which they were added to the control.
`sortFunction`	`Function`	`*`	A compare function that will be used for sorting the layers, when `sortLayers` is `true`. The function receives both the `L.Layer` instances and their names, as in `sortFunction(layerA, layerB, nameA, nameB)`. By default, it sorts layers alphabetically by their name.

▶ Options inherited from Control

Methods

Method	Returns	Description
`addBaseLayer(<Layer> layer, <String> name)`	`this`	Adds a base layer (radio button entry) with the given name to the control.
`addOverlay(<Layer> layer, <String> name)`	`this`	Adds an overlay (checkbox entry) with the given name to the control.
`removeLayer(<Layer> layer)`	`this`	Remove the given layer from the control.
`expand()`	`this`	Expand the control container if collapsed.
`collapse()`	`this`	Collapse the control container if expanded.

▶ Methods inherited from Control

Control.Scale

A simple scale control that shows the scale of the current center of screen in metric (m/km) and imperial (mi/ft) systems. Extends `Control`.

Usage example

```
L.control.scale().addTo(map);
```

Creation

Factory	Description
`L.control.scale(<Control.Scale options> options?)`	Creates an scale control with the given options.

Options

Option	Type	Default	Description
`maxWidth`	`Number`	`100`	Maximum width of the control in pixels. The width is set dynamically to show round values (e.g. 100, 200, 500).
`metric`	`Boolean`	`True`	Whether to show the metric scale line (m/km).
`imperial`	`Boolean`	`True`	Whether to show the imperial scale line (mi/ft).
`updateWhenIdle`	`Boolean`	`false`	If `true`, the control is updated on `moveend`, otherwise it's always up-to-date (updated on

Option	Type	Default	Description
			move).

▶ Options inherited from Control

Methods

▶ Methods inherited from Control

Browser

A namespace with static properties for browser/feature detection used by Leaflet internally.

Usage example

```
if (L.Browser.ielt9) {
  alert('Upgrade your browser, dude!');
}
```

Properties

Property	Type	Description
ie	Boolean	true for all Internet Explorer versions (not Edge).
ielt9	Boolean	true for Internet Explorer versions less than 9.
edge	Boolean	true for the Edge web browser.
webkit	Boolean;	true for webkit-based browsers like Chrome and Safari (including mobile versions).
android	Boolean	**Deprecated.** true for any browser running on an Android platform.
android23	Boolean	**Deprecated.** true for browsers running on Android 2 or Android 3.
androidStock	Boolean	**Deprecated.** true for the Android stock browser (i.e. not Chrome)
opera	Boolean	true for the Opera browser
chrome	Boolean	true for the Chrome browser.
gecko	Boolean	true for gecko-based browsers like Firefox.

Property	Type	Description
`safari`	`Boolean`	`true` for the Safari browser.
`opera12`	`Boolean`	`true` for the Opera browser supporting CSS transforms (version 12 or later).
`win`	`Boolean`	`true` when the browser is running in a Windows platform
`ie3d`	`Boolean`	`true` for all Internet Explorer versions supporting CSS transforms.
`webkit3d`	`Boolean`	`true` for webkit-based browsers supporting CSS transforms.
`gecko3d`	`Boolean`	`true` for gecko-based browsers supporting CSS transforms.
`any3d`	`Boolean`	`true` for all browsers supporting CSS transforms.
`mobile`	`Boolean`	`true` for all browsers running in a mobile device.
`mobileWebkit`	`Boolean`	`true` for all webkit-based browsers in a mobile device.
`mobileWebkit3d`	`Boolean`	`true` for all webkit-based browsers in a mobile device supporting CSS transforms.
`msPointer`	`Boolean`	`true` for browsers implementing the Microsoft touch events model (notably IE10).
`pointer`	`Boolean`	`true` for all browsers supporting pointer events.
`touchNative`	`Boolean`	`true` for all browsers supporting touch events. **This does not necessarily mean** that the browser is running in a computer with a touchscreen, it only means that the browser is capable of understanding touch events.
`touch`	`Boolean`	`true` for all browsers supporting either touch or pointer events. Note: pointer events will be preferred (if available), and processed for all `touch*` listeners.
`mobileOpera`	`Boolean`	`true` for the Opera browser in a mobile device.
`mobileGecko`	`Boolean`	`true` for gecko-based browsers running in a mobile device.
`retina`	`Boolean`	`true` for browsers on a high-resolution "retina" screen or on any screen when browser's display zoom is more than 100%.
`passiveEvents`	`Boolean`	`true` for browsers that support passive events.
`canvas`	`Boolean`	`true` when the browser supports `<canvas>`.
`svg`	`Boolean`	`true` when the browser supports SVG.
`vml`	`Boolean`	`true` if the browser supports VML.
`mac`	`Boolean`	`true` when the browser is running in a Mac platform `true` when the browser is running in a Linux platform

Util

Various utility functions, used by Leaflet internally.

Functions

Function	Returns	Description
`extend(<Object> dest, <Object> src?)`	`Object`	Merges the properties of the `src` object (or multiple objects) into `dest` object and returns the latter. Has an `L.extend` shortcut.
`create(<Object> proto, <Object> properties?)`	`Object`	Compatibility polyfill for Object.create
`bind(<Function> fn, …)`	`Function`	Returns a new function bound to the arguments passed, like Function.prototype.bind. Has a `L.bind()` shortcut.
`stamp(<Object> obj)`	`Number`	Returns the unique ID of an object, assigning it one if it doesn't have it.
`throttle(<Function> fn, <Number> time, <Object> context)`	`Function`	Returns a function which executes function `fn` with the given scope `context` (so that the `this` keyword refers to `context` inside `fn`'s code). The function `fn` will be called no more than one time per given amount of `time`. The arguments received by the bound function will be any arguments passed when binding the function, followed by any arguments passed when invoking the bound function. Has an `L.throttle` shortcut.
`wrapNum(<Number> num, <Number[]> range, <Boolean> includeMax?)`	`Number`	Returns the number `num` modulo `range` in such a way so it lies within `range[0]` and `range[1]`. The returned value will be always smaller than `range[1]` unless `includeMax` is set to `true`.
`falseFn()`	`Function`	Returns a function which always returns `false`.
`formatNum(<Number> num, <Number|false> precision?)`	`Number`	Returns the number `num` rounded with specified `precision`. The default `precision` value is 6 decimal places. `false` can be passed to skip any processing (can be useful to avoid round-off errors).

Function	Returns	Description
`trim(<String> str)`	`String`	Compatibility polyfill for String.prototype.trim
`splitWords(<String> str)`	`String[]`	Trims and splits the string on whitespace and returns the array of parts.
`setOptions(<Object> obj, <Object> options)`	`Object`	Merges the given properties to the `options` of the `obj` object, returning the resulting options. See `Class options`. Has an `L.setOptions` shortcut.
`getParamString(<Object> obj, <String> existingUrl?, <Boolean> uppercase?)`	`String`	Converts an object into a parameter URL string, e.g. `{a: "foo", b: "bar"}` translates to `'?a=foo&b=bar'`. If `existingUrl` is set, the parameters will be appended at the end. If `uppercase` is `true`, the parameter names will be uppercased (e.g. `'?A=foo&B=bar'`)
`template(<String> str, <Object> data)`	`String`	Simple templating facility, accepts a template string of the form `'Hello {a}, {b}'` and a data object like `{a: 'foo', b: 'bar'}`, returns evaluated string (`'Hello foo, bar'`). You can also specify functions instead of strings for data values — they will be evaluated passing `data` as an argument.
`isArray(obj)`	`Boolean`	Compatibility polyfill for Array.isArray
`indexOf(<Array> array, <Object> el)`	`Number`	Compatibility polyfill for Array.prototype.indexOf
`requestAnimFrame(<Function> fn, <Object> context?, <Boolean> immediate?)`	`Number`	Schedules `fn` to be executed when the browser repaints. `fn` is bound to `context` if given. When `immediate` is set, `fn` is called immediately if the browser doesn't have native support for `window.requestAnimationFrame`, otherwise it's delayed. Returns a request ID that can be used to cancel the request.
`cancelAnimFrame(<Number> id)`	`undefined`	Cancels a previous `requestAnimFrame`. See also window.cancelAnimationFrame.

Properties

Property	Type	Description
`lastId`	`Number`	Last unique ID used by `stamp()`

Property	Type	Description
`emptyImageUrl`	`String`	Data URI string containing a base64-encoded empty GIF image. Used as a hack to free memory from unused images on WebKit-powered mobile devices (by setting image `src` to this string).



Transformation

Represents an affine transformation: a set of coefficients `a`, `b`, `c`, `d` for transforming a point of a form `(x, y)` into `(a*x + b, c*y + d)` and doing the reverse. Used by Leaflet in its projections code.

Usage example

```
var transformation = L.transformation(2, 5, -1, 10),
    p = L.point(1, 2),
    p2 = transformation.transform(p), //  L.point(7, 8)
    p3 = transformation.untransform(p2); //  L.point(1, 2)
```

Creation

Factory	Description
`L.transformation(<Number> a, <Number> b, <Number> c, <Number> d)`	Instantiates a Transformation object with the given coefficients.
`L.transformation(<Array> coefficients)`	Expects an coefficients array of the form `[a: Number, b: Number, c: Number, d: Number]`.

Methods

Method	Returns	Description
`transform(<Point> point, <Number> scale?)`	`Point`	Returns a transformed point, optionally multiplied by the given scale. Only accepts actual `L.Point` instances, not arrays.
`untransform(<Point> point, <Number> scale?)`	`Point`	Returns the reverse transformation of the given point, optionally divided by the given scale. Only accepts actual `L.Point` instances, not arrays.

LineUtil

Various utility functions for polyline points processing, used by Leaflet internally to make polylines lightning-fast.

Functions

Function	Returns	Description
`simplify(<Point[]> points, <Number> tolerance)`	`Point[]`	Dramatically reduces the number of points in a polyline while retaining its shape and returns a new array of simplified points, using the Ramer-Douglas-Peucker algorithm. Used for a huge performance boost when processing/displaying Leaflet polylines for each zoom level and also reducing visual noise. tolerance affects the amount of simplification (lesser value means higher quality but slower and with more points). Also released as a separated micro-library Simplify.js.
`pointToSegmentDistance(<Point> p, <Point> p1, <Point> p2)`	`Number`	Returns the distance between point p and segment p1 to p2.
`closestPointOnSegment(<Point> p, <Point> p1, <Point> p2)`	`Number`	Returns the closest point from a point p on a segment p1 to p2.
`clipSegment(<Point> a, <Point> b, <Bounds> bounds, <Boolean> useLastCode?, <Boolean> round?)`	`Point[]|Boolean`	Clips the segment a to b by rectangular bounds with the Cohen-Sutherland algorithm (modifying the segment points directly!). Used by Leaflet to only show polyline points that are on the screen or near, increasing performance.
`isFlat(<LatLng[]> latlngs)`	`Boolean`	Returns true if `latlngs` is a flat array, false is nested.
`polylineCenter(<LatLng[]> latlngs, <CRS> crs)`	`LatLng`	Returns the center (centroid) of the passed LatLngs (first ring) from a polyline.

PolyUtil

Various utility functions for polygon geometries.

Functions

Function	Returns	Description
`clipPolygon(<Point[]> points, <Bounds> bounds, <Boolean> round?)`	`Point[]`	Clips the polygon geometry defined by the given `points` by the given bounds (using the Sutherland-Hodgman algorithm). Used by Leaflet to only show polygon points that are on the screen or near, increasing performance. Note that polygon points needs different algorithm for clipping than polyline, so there's a separate method for it.
`polygonCenter(<LatLng[]> latlngs, <CRS> crs)`	`LatLng`	Returns the center (centroid) of the passed LatLngs (first ring) from a polygon.
`centroid(<LatLng[]> latlngs)`	`LatLng`	Returns the 'center of mass' of the passed LatLngs.

DomEvent

Utility functions to work with the DOM events, used by Leaflet internally.

Functions

Function	Returns	Description
`on(<HTMLElement> el, <String> types, <Function> fn, <Object> context?)`	`this`	Adds a listener function (`fn`) to a particular DOM event type of the element `el`. You can optionally specify the context of the listener (object the `this` keyword will point to). You can also pass several space-separated types (e.g. `'click dblclick'`).
`on(<HTMLElement> el, <Object> eventMap, <Object> context?)`	`this`	Adds a set of type/listener pairs, e.g. `{click: onClick, mousemove: onMouseMove}`

Function	Returns	Description
`off(<HTMLElement> el, <String> types, <Function> fn, <Object> context?)`	`this`	Removes a previously added listener function. Note that if you passed a custom context to on, you must pass the same context to `off` in order to remove the listener.
`off(<HTMLElement> el, <Object> eventMap, <Object> context?)`	`this`	Removes a set of type/listener pairs, e.g. `{click: onClick, mousemove: onMouseMove}`
`off(<HTMLElement> el, <String> types)`	`this`	Removes all previously added listeners of given types.
`off(<HTMLElement> el)`	`this`	Removes all previously added listeners from given HTMLElement

`stopPropagation(<DOMEvent> ev)`	`this`	Stop the given event from propagation to parent elements. Used inside the listener functions:
		`L.DomEvent.on(div, 'click', function (ev) {`
		`L.DomEvent.stopPropagation(ev);`
		`});`

`disableScrollPropagation(<HTMLElement> el)`	`this`	Adds `stopPropagation` to the element's `'wheel'` events (plus browser variants).
`disableClickPropagation(<HTMLElement> el)`	`this`	Adds `stopPropagation` to the element's `'click'`, `'dblclick'`, `'contextmenu'`, `'mousedown'` and `'touchstart'` events (plus browser variants).
`preventDefault(<DOMEvent> ev)`	`this`	Prevents the default action of the DOM Event `ev` from happening (such as following a link in the href of the a element, or doing a POST request with page reload when a `<form>` is submitted).

Function	Returns	Description
		Use it inside listener functions.
`stop(<DOMEvent> ev)`	`this`	Does `stopPropagation` and `preventDefault` at the same time.
`getPropagationPath(<DOMEvent> ev)`	`Array`	Compatibility polyfill for `Event.composedPath()`. Returns an array containing the `HTMLElements` that the given DOM event should propagate to (if not stopped).
`getMousePosition(<DOMEvent> ev, <HTMLElement> container?)`	`Point`	Gets normalized mouse position from a DOM event relative to the `container` (border excluded) or to the whole page if not specified.
`getWheelDelta(<DOMEvent> ev)`	`Number`	Gets normalized wheel delta from a wheel DOM event, in vertical pixels scrolled (negative if scrolling down). Events from pointing devices without precise scrolling are mapped to a best guess of 60 pixels.
`addListener(…)`	`this`	Alias to `L.DomEvent.on`
`removeListener(…)`	`this`	Alias to `L.DomEvent.off`

DomUtil

Utility functions to work with the DOM tree, used by Leaflet internally.

Most functions expecting or returning a `HTMLElement` also work for SVG elements. The only difference is that classes refer to CSS classes in HTML and SVG classes in SVG.

Functions

Function	Returns	Description
`get(<String|HTMLElement> id)`	`HTMLElement`	Returns an element given its DOM id, or returns the element itself if it was passed directly.

Function	Returns	Description
`getStyle(<HTMLElement> el, <String> styleAttrib)`	`String`	Returns the value for a certain style attribute on an element, including computed values or values set through CSS.
`create(<String> tagName, <String> className?, <HTMLElement> container?)`	`HTMLElement`	Creates an HTML element with `tagName`, sets its class to `className`, and optionally appends it to `container` element.
`remove(<HTMLElement> el)`		Removes `el` from its parent element
`empty(<HTMLElement> el)`		Removes all of `el`'s children elements from `el`
`toFront(<HTMLElement> el)`		Makes `el` the last child of its parent, so it renders in front of the other children.
`toBack(<HTMLElement> el)`		Makes `el` the first child of its parent, so it renders behind the other children.
`hasClass(<HTMLElement> el, <String> name)`	`Boolean`	Returns `true` if the element's class attribute contains `name`.
`addClass(<HTMLElement> el, <String> name)`		Adds `name` to the element's class attribute.
`removeClass(<HTMLElement> el, <String> name)`		Removes `name` from the element's class attribute.
`setClass(<HTMLElement> el, <String> name)`		Sets the element's class.
`getClass(<HTMLElement> el)`	`String`	Returns the element's class.
`setOpacity(<HTMLElement> el, <Number> opacity)`		Set the opacity of an element (including old IE support). `opacity` must be a number from `0` to `1`.
`testProp(<String[]> props)`	`String|false`	Goes through the array of style names and returns the first name that is a valid style name for an element. If no such name is found, it returns false. Useful for vendor-prefixed styles like `transform`.

Function	Returns	Description
`setTransform(<HTMLElement> el, <Point> offset, <Number> scale?)`		Resets the 3D CSS transform of `el` so it is translated by `offset` pixels and optionally scaled by `scale`. Does not have an effect if the browser doesn't support 3D CSS transforms.
`setPosition(<HTMLElement> el, <Point> position)`		Sets the position of `el` to coordinates specified by `position`, using CSS translate or top/left positioning depending on the browser (used by Leaflet internally to position its layers).
`getPosition(<HTMLElement> el)`	`Point`	Returns the coordinates of an element previously positioned with setPosition.
`disableTextSelection()`		Prevents the user from generating `selectstart` DOM events, usually generated when the user drags the mouse through a page with text. Used internally by Leaflet to override the behaviour of any click-and-drag interaction on the map. Affects drag interactions on the whole document.
`enableTextSelection()`		Cancels the effects of a previous `L.DomUtil.disableTextSelection`.
`disableImageDrag()`		As `L.DomUtil.disableTextSelection`, but for `dragstart` DOM events, usually generated when the user drags an image.
`enableImageDrag()`		Cancels the effects of a previous `L.DomUtil.disableImageDrag`.
`preventOutline(<HTMLElement> el)`		Makes the outline of the element `el` invisible. Used internally by Leaflet to prevent focusable elements from

Function	Returns	Description
		displaying an outline when the user performs a drag interaction on them.
`restoreOutline()`		Cancels the effects of a previous `L.DomUtil.preventOutline`.
`getSizedParentNode(<HTMLElement> el)`	`HTMLElement`	Finds the closest parent node which size (width and height) is not null.
`getScale(<HTMLElement> el)`	`Object`	Computes the CSS scale currently applied on the element. Returns an object with x and y members as horizontal and vertical scales respectively, and `boundingClientRect` as the result of `getBoundingClientRect()`.

Properties

Property	Type	Description
`TRANSFORM`	`String`	Vendor-prefixed transform style name (e.g. `'webkitTransform'` for WebKit).
`TRANSITION`	`String`	Vendor-prefixed transition style name.
`TRANSITION_END`	`String`	Vendor-prefixed transitionend event name.

PosAnimation

Used internally for panning animations, utilizing CSS3 Transitions for modern browsers and a timer fallback for IE6-9.

Usage example

```
var myPositionMarker = L.marker([48.864716, 2.294694]).addTo(map);

myPositionMarker.on("click", function() {
    var pos = map.latLngToLayerPoint(myPositionMarker.getLatLng());
    pos.y -= 25;
    var fx = new L.PosAnimation();
```

```
    fx.once('end',function() {
        pos.y += 25;
        fx.run(myPositionMarker._icon, pos, 0.8);
    });

    fx.run(myPositionMarker._icon, pos, 0.3);
});
```



Constructor

Constructor	Description
`L.PosAnimation()`	Creates a `PosAnimation` object.

Events

Event	Data	Description
`start`	`Event`	Fired when the animation starts
`step`	`Event`	Fired continuously during the animation.
`end`	`Event`	Fired when the animation ends.

Methods

Method	Returns	Description
`run(<HTMLElement> el, <Point> newPos, <Number> duration?, <Number> easeLinearity?)`		Run an animation of a given element to a new position, optionally setting duration in seconds (`0.25` by default) and easing linearity factor (3rd argument of the cubic bezier curve, `0.5` by default).
`stop()`		Stops the animation (if currently running).

▶ Methods inherited from Evented

Draggable

A class for making DOM elements draggable (including touch support). Used internally for map and marker dragging. Only works for elements that were positioned with `L.DomUtil.setPosition`.

Usage example

```
var draggable = new L.Draggable(elementToDrag);
draggable.enable();
```

Constructor

Constructor	Description
`L.Draggable(<HTMLElement> el, <HTMLElement> dragHandle?, <Boolean> preventOutline?, <Draggable options> options?)`	Creates a `Draggable` object for moving `el` when you start dragging the `dragHandle` element (equals `el` itself by default).

Options

Option	Type	Default	Description
`clickTolerance`	`Number`	`3`	The max number of pixels a user can shift the mouse pointer during a click for it to be considered a valid click (as opposed to a mouse drag).

Events

Event	Data	Description
`down`	`Event`	Fired when a drag is about to start.
`dragstart`	`Event`	Fired when a drag starts
`predrag`	`Event`	Fired continuously during dragging *before* each corresponding update of the element's position.
`drag`	`Event`	Fired continuously during dragging.
`dragend`	`DragEndEvent`	Fired when the drag ends.

Methods

Method	Returns	Description
`enable()`		Enables the dragging ability
`disable()`		Disables the dragging ability

▶ Methods inherited from Evented

Class

L.Class powers the OOP facilities of Leaflet and is used to create almost all of the Leaflet classes documented here.

In addition to implementing a simple classical inheritance model, it introduces several special properties for convenient code organization — options, includes and statics.

Usage example

```
var MyClass = L.Class.extend({
initialize: function (greeter) {
    this.greeter = greeter;
    // class constructor
},

greet: function (name) {
    alert(this.greeter + ', ' + name)
    }
});

// create instance of MyClass, passing "Hello" to the constructor
var a = new MyClass("Hello");

// call greet method, alerting "Hello, World"
a.greet("World");
```

Class Factories

You may have noticed that Leaflet objects are created without using the `new` keyword. This is achieved by complementing each class with a lowercase factory method:

```
new L.Map('map'); // becomes:
L.map('map');
```

The factories are implemented very easily, and you can do this for your own classes:

```
L.map = function (id, options) {
    return new L.Map(id, options);
};
```

Inheritance

You use L.Class.extend to define new classes, but you can use the same method on any class to inherit from it:

```
var MyChildClass = MyClass.extend({
    // ... new properties and methods
});
```



This will create a class that inherits all methods and properties of the parent class (through a proper prototype chain), adding or overriding the ones you pass to extend. It will also properly react to instanceof:

```
var a = new MyChildClass();
a instanceof MyChildClass; // true
a instanceof MyClass; // true
```

You can call parent methods (including constructor) from corresponding child ones (as you do with super calls in other languages) by accessing parent class prototype and using JavaScript's call or apply:

```
var MyChildClass = MyClass.extend({
    initialize: function () {
        MyClass.prototype.initialize.call(this, "Yo");
    },

    greet: function (name) {
        MyClass.prototype.greet.call(this, 'bro ' + name + '!');
    }
});

var a = new MyChildClass();
a.greet('Jason'); // alerts "Yo, bro Jason!"
```

Options

`options` is a special property that unlike other objects that you pass to `extend` will be merged with the parent one instead of overriding it completely, which makes managing configuration of objects and default values convenient:

```
var MyClass = L.Class.extend({
    options: {
        myOption1: 'foo',
        myOption2: 'bar'
    }
});

var MyChildClass = MyClass.extend({
    options: {
        myOption1: 'baz',
        myOption3: 5
    }
});
```

```
var a = new MyChildClass();
a.options.myOption1; // 'baz'
a.options.myOption2; // 'bar'
a.options.myOption3; // 5
```



There's also `L.Util.setOptions`, a method for conveniently merging options passed to constructor with the defaults defines in the class:

```
var MyClass = L.Class.extend({
    options: {
        foo: 'bar',
        bla: 5
    },

    initialize: function (options) {
        L.Util.setOptions(this, options);
        ...
    }
});

var a = new MyClass({bla: 10});
a.options; // {foo: 'bar', bla: 10}
```

Note that the options object allows any keys, not just the options defined by the class and its base classes. This means you can use the options object to store application specific information, as long as you avoid keys that are already used by the class in question.

Includes

`includes` is a special class property that merges all specified objects into the class (such objects are called mixins).

```
 var MyMixin = {
    foo: function () { ... },
    bar: 5
};

var MyClass = L.Class.extend({
    includes: MyMixin
});

var a = new MyClass();
a.foo();
```

You can also do such includes in runtime with the `include` method:

```
MyClass.include(MyMixin);
```

`statics` is just a convenience property that injects specified object properties as the static properties of the class, useful for defining constants:



```
var MyClass = L.Class.extend({
    statics: {
        FOO: 'bar',
        BLA: 5
    }
});

MyClass.FOO; // 'bar'
```

Constructor hooks

If you're a plugin developer, you often need to add additional initialization code to existing classes (e.g. editing hooks for `L.Polyline`). Leaflet comes with a way to do it easily using the `addInitHook` method:

```
MyClass.addInitHook(function () {
    // ... do something in constructor additionally
    // e.g. add event listeners, set custom properties etc.
});
```

You can also use the following shortcut when you just need to make one additional method call:

```
MyClass.addInitHook('methodName', arg1, arg2, …);
```

Functions

Function	Returns	Description
`extend(<Object> props)`	`Function`	Extends the current class given the properties to be included. Returns a Javascript function that is a class constructor (to be called with new).
`include(<Object> properties)`	`this`	Includes a mixin into the current class.
`mergeOptions(<Object> options)`	`this`	Merges `options` into the defaults of the class.
`addInitHook(<Function> fn)`	`this`	Adds a constructor hook to the class.

Evented

A set of methods shared between event-powered classes (like `Map` and `Marker`). Generally, events allow you to execute some function when something happens with an object (e.g. the user clicks on the map, causing the map to fire `'click'` event).

Usage example

```
map.on('click', function(e) {
    alert(e.latlng);
} );
```

Leaflet deals with event listeners by reference, so if you want to add a listener and then remove it, define it as a function:

```
function onClick(e) { ... }

map.on('click', onClick);
map.off('click', onClick);
```

Methods

Method	Returns	Description
`on(<String> type, <Function> fn, <Object> context?)`	`this`	Adds a listener function (`fn`) to a particular event type of the object. You can optionally specify the context of the listener (object the this keyword will point to). You can also pass several space-separated types (e.g. `'click dblclick'`).
`on(<Object> eventMap)`	`this`	Adds a set of type/listener pairs, e.g. `{click: onClick, mousemove: onMouseMove}`
`off(<String> type, <Function> fn?, <Object> context?)`	`this`	Removes a previously added listener function. If no function is specified, it will remove all the listeners of that particular event from the object. Note that if you passed a custom context to `on`, you must pass the same context to `off` in order to remove the listener.
`off(<Object> eventMap)`	`this`	Removes a set of type/listener pairs.
`off()`	`this`	Removes all listeners to all events on the object. This includes implicitly attached

Method	Returns	Description
		events.
`fire(<String> type, <Object> data?, <Boolean> propagate?)`	`this`	Fires an event of the specified type. You can optionally provide a data object — the first argument of the listener function will contain its properties. The event can optionally be propagated to event parents.
`listens(<String> type, <Boolean> propagate?)`	`Boolean`	Returns `true` if a particular event type has any listeners attached to it. The verification can optionally be propagated, it will return `true` if parents have the listener attached to it.
`once(…)`	`this`	Behaves as `on(…)`, except the listener will only get fired once and then removed.
`addEventParent(<Evented> obj)`	`this`	Adds an event parent - an `Evented` that will receive propagated events
`removeEventParent(<Evented> obj)`	`this`	Removes an event parent, so it will stop receiving propagated events
`addEventListener(…)`	`this`	Alias to `on(…)`
`removeEventListener(…)`	`this`	Alias to `off(…)`
`clearAllEventListeners(…)`	`this`	Alias to `off()`
`addOneTimeEventListener(…)`	`this`	Alias to `once(…)`
`fireEvent(…)`	`this`	Alias to `fire(…)`
`hasEventListeners(…)`	`Boolean`	Alias to `listens(…)`

Layer

A set of methods from the Layer base class that all Leaflet layers use. Inherits all methods, options and events from `L.Evented`.

Usage example

```
var layer = L.marker(latlng).addTo(map);
layer.addTo(map);
layer.remove();
```

Options

Option	Type	Default	Description
`pane`	`String`	`'overlayPane'`	By default the layer will be added to the map's overlay pane. Overriding this option will cause the layer to be placed on another pane by default.
`attribution`	`String`	`null`	String to be shown in the attribution control, e.g. "© OpenStreetMap contributors". It describes the layer data and is often a legal obligation towards copyright holders and tile providers.

Events

Event	Data	Description
`add`	`Event`	Fired after the layer is added to a map
`remove`	`Event`	Fired after the layer is removed from a map

Popup events

Event	Data	Description
`popupopen`	`PopupEvent`	Fired when a popup bound to this layer is opened
`popupclose`	`PopupEvent`	Fired when a popup bound to this layer is closed

Tooltip events

Event	Data	Description
`tooltipopen`	`TooltipEvent`	Fired when a tooltip bound to this layer is opened.
`tooltipclose`	`TooltipEvent`	Fired when a tooltip bound to this layer is closed.

Methods

Classes extending `L.Layer` will inherit the following methods:

Method	Returns	Description
`addTo(<Map|LayerGroup> map)`	`this`	Adds the layer to the given map or layer group.
`remove()`	`this`	Removes the layer from the map it is currently active on.
`removeFrom(<Map> map)`	`this`	Removes the layer from the given map

Method	Returns	Description
`removeFrom(<LayerGroup> group)`	`this`	Removes the layer from the given `LayerGroup`
`getPane(<String> name?)`	`HTMLElement`	Returns the `HTMLElement` representing the named pane on the map. If `name` is omitted, returns the pane for this layer.
`getAttribution()`	`String`	Used by the `attribution control`, returns the attribution option.

Extension methods

Every layer should extend from `L.Layer` and (re-)implement the following methods.

Method	Returns	Description
`onAdd(<Map> map)`	`this`	Should contain code that creates DOM elements for the layer, adds them to `map panes` where they should belong and puts listeners on relevant map events. Called on `map.addLayer(layer)`.
`onRemove(<Map> map)`	`this`	Should contain all clean up code that removes the layer's elements from the DOM and removes listeners previously added in `onAdd`. Called on `map.removeLayer(layer)`.
`getEvents()`	`Object`	This optional method should return an object like `{ viewreset: this._reset }` for `addEventListener`. The event handlers in this object will be automatically added and removed from the map with your layer.
`getAttribution()`	`String`	This optional method should return a string containing HTML to be shown on the `Attribution control` whenever the layer is visible.
`beforeAdd(<Map> map)`	`this`	Optional method. Called on `map.addLayer(layer)`, before the layer is added to the map, before events are initialized, without waiting until the map is in a usable state. Use for early initialization only.

Popup methods

All layers share a set of methods convenient for binding popups to it.

```
var layer = L.Polygon(latlngs).bindPopup('Hi There!').addTo(map);
layer.openPopup();
layer.closePopup();
```

Popups will also be automatically opened when the layer is clicked on and closed when the layer is removed from the map or another popup is opened.

Method	Returns	Description
`bindPopup(<String|HTMLElement|Function|Popup> content, <Popup options> options?)`	`this`	Binds a popup to the layer with the passed `content` and sets up

Method	Returns	Description
		the necessary event listeners. If a `Function` is passed it will receive the layer as the first argument and should return a `String` or `HTMLElement`.
`unbindPopup()`	`this`	Removes the popup previously bound with `bindPopup`.
`openPopup(<LatLng> latlng?)`	`this`	Opens the bound popup at the specified `latlng` or at the default popup anchor if no `latlng` is passed.
`closePopup()`	`this`	Closes the popup bound to this layer if it is open.
`togglePopup()`	`this`	Opens or closes the popup bound to this layer depending on its current state.
`isPopupOpen()`	`boolean`	Returns `true` if the popup bound to this layer is currently open.
`setPopupContent(<String|HTMLElement|Popup> content)`	`this`	Sets the content of the popup bound to this layer.
`getPopup()`	Popup	Returns the popup bound to this layer.

Tooltip methods

All layers share a set of methods convenient for binding tooltips to it.

```
var layer = L.Polygon(latlngs).bindTooltip('Hi There!').addTo(map);
layer.openTooltip();
layer.closeTooltip();
```

Method	Returns	Description
`bindTooltip(<String|HTMLElement|Function|Tooltip> content, <Tooltip options> options?)`	`this`	Binds a tooltip to the layer with the passed

Method	Returns	Description
		`content` and sets up the necessary event listeners. If a `Function` is passed it will receive the layer as the first argument and should return a `String` or `HTMLElement`.
`unbindTooltip()`	`this`	Removes the tooltip previously bound with `bindTooltip`.
`openTooltip(<LatLng> latlng?)`	`this`	Opens the bound tooltip at the specified `latlng` or at the default tooltip anchor if no `latlng` is passed.
`closeTooltip()`	`this`	Closes the tooltip bound to this layer if it is open.
`toggleTooltip()`	`this`	Opens or closes the tooltip bound to this layer depending on its current state.
`isTooltipOpen()`	`boolean`	Returns `true` if the tooltip bound to this layer is currently open.
`setTooltipContent(<String|HTMLElement|Tooltip> content)`	`this`	Sets the content of the tooltip bound to this layer.
`getTooltip()`	`Tooltip`	Returns the tooltip bound to this layer.

▶ Methods inherited from Evented



Interactive layer

Some `Layer`s can be made interactive - when the user interacts with such a layer, mouse events like `click` and `mouseover` can be handled. Use the event handling methods to handle these events.

Options

Option	Type	Default	Description
`interactive`	`Boolean`	`true`	If `false`, the layer will not emit mouse events and will act as a part of the underlying map.
`bubblingMouseEvents`	`Boolean`	`true`	When `true`, a mouse event on this layer will trigger the same event on the map (unless `L.DomEvent.stopPropagation` is used).

▶ Options inherited from Layer

Events

Mouse events

Event	Data	Description
`click`	`MouseEvent`	Fired when the user clicks (or taps) the layer.
`dblclick`	`MouseEvent`	Fired when the user double-clicks (or double-taps) the layer.
`mousedown`	`MouseEvent`	Fired when the user pushes the mouse button on the layer.
`mouseup`	`MouseEvent`	Fired when the user releases the mouse button pushed on the layer.
`mouseover`	`MouseEvent`	Fired when the mouse enters the layer.
`mouseout`	`MouseEvent`	Fired when the mouse leaves the layer.
`contextmenu`	`MouseEvent`	Fired when the user right-clicks on the layer, prevents default browser context menu from showing if there are listeners on this event. Also fired on mobile when the user holds a single touch for a second (also called long press).

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Control

L.Control is a base class for implementing map controls. Handles positioning. All other controls extend from this class.

Options

Option	Type	Default	Description
`position`	`String`	`'topright'`	The position of the control (one of the map corners). Possible values are `'topleft'`, `'topright'`, `'bottomleft'` or `'bottomright'`

Methods

Classes extending L.Control will inherit the following methods:

Method	Returns	Description
`getPosition()`	`string`	Returns the position of the control.
`setPosition(<string> position)`	`this`	Sets the position of the control.
`getContainer()`	`HTMLElement`	Returns the HTMLElement that contains the control.
`addTo(<Map> map)`	`this`	Adds the control to the given map.
`remove()`	`this`	Removes the control from the map it is currently active on.

Extension methods

Every control should extend from `L.Control` and (re-)implement the following methods.

Method	Returns	Description
`onAdd(<Map> map)`	`HTMLElement`	Should return the container DOM element for the control and add listeners on relevant map events. Called on `control.addTo(map)`.
`onRemove(<Map> map)`		Optional method. Should contain all clean up code that removes the listeners previously added in onAdd. Called on `control.remove()`.

Handler

Abstract class for map interaction handlers

Methods

Method	Returns	Description
`enable()`	`this`	Enables the handler
`disable()`	`this`	Disables the handler
`enabled()`	`Boolean`	Returns `true` if the handler is enabled

Extension methods

Classes inheriting from `Handler` must implement the two following methods:

Method	Returns	Description
`addHooks()`		Called when the handler is enabled, should add event hooks.
`removeHooks()`		Called when the handler is disabled, should remove the event hooks added previously.

Functions

There is static function which can be called without instantiating L.Handler:

Function	Returns	Description
`addTo(<Map> map, <String> name)`	`this`	Adds a new Handler to the given map with the given name.

Projection

An object with methods for projecting geographical coordinates of the world onto a flat surface (and back). See Map projection.

Methods

Method	Returns	Description
`project(<LatLng> latlng)`	`Point`	Projects geographical coordinates into a 2D point. Only accepts actual `L.LatLng` instances, not arrays.
`unproject(<Point> point)`	`LatLng`	The inverse of `project`. Projects a 2D point into a geographical location. Only accepts actual `L.Point` instances, not arrays. Note that the projection instances do not inherit from Leaflet's `Class` object, and can't be instantiated. Also, new classes can't inherit from them, and methods can't be added to them with the `include` function.

Properties

Property	Type	Description
`bounds`	`Bounds`	The bounds (specified in CRS units) where the projection is valid

Defined projections

Leaflet comes with a set of already defined Projections out of the box:

Projection	Description
`L.Projection.LonLat`	Equirectangular, or Plate Carree projection — the most simple projection, mostly used by GIS enthusiasts. Directly maps x as longitude, and y as latitude. Also suitable for flat worlds, e.g. game maps. Used by the `EPSG:4326` and `Simple` CRS.
`L.Projection.Mercator`	Elliptical Mercator projection — more complex than Spherical Mercator. Assumes that Earth is an ellipsoid. Used by the EPSG:3395 CRS.
`L.Projection.SphericalMercator`	Spherical Mercator projection — the most common projection for online maps, used by almost all free and commercial tile providers. Assumes that Earth is a sphere. Used by the `EPSG:3857` CRS.

CRS

Methods

Method	Returns	Description
`latLngToPoint(<LatLng> latlng, <Number> zoom)`	`Point`	Projects geographical coordinates into pixel coordinates for a given zoom.
`pointToLatLng(<Point> point, <Number> zoom)`	`LatLng`	The inverse of `latLngToPoint`. Projects pixel coordinates on a given zoom into geographical coordinates.
`project(<LatLng> latlng)`	`Point`	Projects geographical coordinates into coordinates in units accepted for this CRS (e.g. meters for EPSG:3857, for passing it to WMS services).
`unproject(<Point> point)`	`LatLng`	Given a projected coordinate returns the corresponding LatLng. The inverse of `project`.
`scale(<Number> zoom)`	`Number`	Returns the scale used when transforming projected coordinates into pixel coordinates for a particular zoom. For example, it returns `256 * 2^zoom` for Mercator-based CRS.
`zoom(<Number> scale)`	`Number`	Inverse of `scale()`, returns the zoom level corresponding to a scale factor of `scale`.
`getProjectedBounds(<Number> zoom)`	`Bounds`	Returns the projection's bounds scaled and transformed for the provided `zoom`.
`distance(<LatLng> latlng1, <LatLng> latlng2)`	`Number`	Returns the distance between two geographical coordinates.
`wrapLatLng(<LatLng> latlng)`	`LatLng`	Returns a `LatLng` where lat and lng has been wrapped according to the CRS's `wrapLat` and `wrapLng` properties, if they are outside the CRS's bounds.
`wrapLatLngBounds(<LatLngBounds> bounds)`	`LatLngBounds`	Returns a `LatLngBounds` with the same size as the given one, ensuring that its center is within the CRS's bounds. Only accepts actual `L.LatLngBounds` instances, not arrays.

Properties

Property	Type	Description
`code`	`String`	Standard code name of the CRS passed into WMS services (e.g. `'EPSG:3857'`)
`wrapLng`	`Number[]`	An array of two numbers defining whether the longitude (horizontal) coordinate axis wraps around a given range and how. Defaults to `[-180, 180]` in most geographical CRSs. If `undefined`, the longitude axis does not wrap around.
`wrapLat`	`Number[]`	Like `wrapLng`, but for the latitude (vertical) axis.
`infinite`	`Boolean`	If true, the coordinate space will be unbounded (infinite in both axes)

Defined CRSs

CRS	Description
`L.CRS.Earth`	Serves as the base for CRS that are global such that they cover the earth. Can only be used as the base for other CRS and cannot be used directly, since it does not have a `code`, `projection` or `transformation`. `distance()` returns meters.
`L.CRS.EPSG3395`	Rarely used by some commercial tile providers. Uses Elliptical Mercator projection.
`L.CRS.EPSG3857`	The most common CRS for online maps, used by almost all free and commercial tile providers. Uses Spherical Mercator projection. Set in by default in Map's `crs` option.
`L.CRS.EPSG4326`	A common CRS among GIS enthusiasts. Uses simple Equirectangular projection. Leaflet 1.0.x complies with the TMS coordinate scheme for EPSG:4326, which is a breaking change from 0.7.x behaviour. If you are using a `TileLayer` with this CRS, ensure that there are two 256x256 pixel tiles covering the whole earth at zoom level zero, and that the tile coordinate origin is (-180,+90), or (-180,-90) for `TileLayer`s with the `tms` option set.
`L.CRS.Base`	Object that defines coordinate reference systems for projecting geographical points into pixel (screen) coordinates and back (and to coordinates in other units for WMS services). See spatial reference system. Leaflet defines the most usual CRSs by default. If you want to use a CRS not defined by default, take a look at the Proj4Leaflet plugin. Note that the CRS instances do not inherit from Leaflet's `Class` object, and can't be instantiated. Also, new classes can't inherit from them, and methods can't be added to them with the `include` function.
`L.CRS.Simple`	A simple CRS that maps longitude and latitude into `x` and `y` directly. May be used for maps of flat surfaces (e.g. game maps). Note that the `y` axis should still be inverted (going from bottom to top). `distance()` returns simple euclidean distance.

Renderer

Base class for vector renderer implementations (SVG, `Canvas`). Handles the DOM container of the renderer, its bounds, and its zoom animation.

A `Renderer` works as an implicit layer group for all `Path`s - the renderer itself can be added or removed to the map. All paths use a renderer, which can be implicit (the map will decide the type of renderer and use it automatically) or explicit (using the `renderer` option of the path).

Do not use this class directly, use SVG and `Canvas` instead.

Options

Option	Type	Default	Description
`padding`	`Number`	`0.1`	How much to extend the clip area around the map view (relative to its size) e.g. 0.1 would be 10% of map view in each direction

▶ Options inherited from Layer

Events

Event	Data	Description
`update`	`Event`	Fired when the renderer updates its bounds, center and zoom, for example when its map has moved

▶ Events inherited from Layer

▶ Popup events inherited from Layer

▶ Tooltip events inherited from Layer

Methods

▶ Methods inherited from Layer

▶ Popup methods inherited from Layer

▶ Tooltip methods inherited from Layer

▶ Methods inherited from Evented

Event objects

Whenever a class inheriting from `Evented` fires an event, a listener function will be called with an event argument, which is a plain object containing information about the event. For example:

```
map.on('click', function(ev) {
    alert(ev.latlng); // ev is an event object (MouseEvent in this case)
});
```

The information available depends on the event type:

Event

The base event object. All other event objects contain these properties too.

Property	Type	Description
`type`	`String`	The event type (e.g. `'click'`).
`target`	`Object`	The object that fired the event. For propagated events, the last object in the propagation chain that fired the event.
`sourceTarget`	`Object`	The object that originally fired the event. For non-propagated events, this will be the same as the `target`.
`propagatedFrom`	`Object`	For propagated events, the last object that propagated the event to its event parent.
`layer`	`Object`	**Deprecated.** The same as `propagatedFrom`.

KeyboardEvent

Property	Type	Description
`originalEvent`	`DOMEvent`	The original DOM `KeyboardEvent` that triggered this Leaflet event.

▶ Properties inherited from Event

MouseEvent

Property	Type	Description
`latlng`	`LatLng`	The geographical point where the mouse event occurred.
`layerPoint`	`Point`	Pixel coordinates of the point where the mouse event occurred relative to the map layer.

Property	Type	Description
containerPoint	Point	Pixel coordinates of the point where the mouse event occurred relative to the map container.
originalEvent	DOMEvent	The original DOM MouseEvent or DOM TouchEvent that triggered this Leaflet event.

▶ Properties inherited from Event

LocationEvent

Property	Type	Description
latlng	LatLng	Detected geographical location of the user.
bounds	LatLngBounds	Geographical bounds of the area user is located in (with respect to the accuracy of location).
accuracy	Number	Accuracy of location in meters.
altitude	Number	Height of the position above the WGS84 ellipsoid in meters.
altitudeAccuracy	Number	Accuracy of altitude in meters.
heading	Number	The direction of travel in degrees counting clockwise from true North.
speed	Number	Current velocity in meters per second.
timestamp	Number	The time when the position was acquired.

▶ Properties inherited from Event

ErrorEvent

Property	Type	Description
message	String	Error message.
code	Number	Error code (if applicable).

▶ Properties inherited from Event

LayerEvent

Property	Type	Description
layer	Layer	The layer that was added or removed.

▶ Properties inherited from Event

LayersControlEvent

Property	Type	Description
`layer`	`Layer`	The layer that was added or removed.
`name`	`String`	The name of the layer that was added or removed.

▶ Properties inherited from Event

TileEvent

Property	Type	Description
`tile`	`HTMLElement`	The tile element (image).
`coords`	`Point`	Point object with the tile's x, y, and z (zoom level) coordinates.

▶ Properties inherited from Event

TileErrorEvent

Property	Type	Description
`tile`	`HTMLElement`	The tile element (image).
`coords`	`Point`	Point object with the tile's x, y, and z (zoom level) coordinates.
`error`	`*`	Error passed to the tile's `done()` callback.

▶ Properties inherited from Event

ResizeEvent

Property	Type	Description
`oldSize`	`Point`	The old size before resize event.
`newSize`	`Point`	The new size after the resize event.

▶ Properties inherited from Event

GeoJSONEvent

Property	Type	Description
`layer`	`Layer`	The layer for the GeoJSON feature that is being added to the map.
`properties`	`Object`	GeoJSON properties of the feature.

Property	Type	Description
geometryType	String	GeoJSON geometry type of the feature.
id	String	GeoJSON ID of the feature (if present).

▶ Properties inherited from Event

PopupEvent

Property	Type	Description
popup	Popup	The popup that was opened or closed.

▶ Properties inherited from Event

TooltipEvent

Property	Type	Description
tooltip	Tooltip	The tooltip that was opened or closed.

▶ Properties inherited from Event

DragEndEvent

Property	Type	Description
distance	Number	The distance in pixels the draggable element was moved by.

▶ Properties inherited from Event

ZoomAnimEvent

Property	Type	Description
center	LatLng	The current center of the map
zoom	Number	The current zoom level of the map
noUpdate	Boolean	Whether layers should update their contents due to this event

▶ Properties inherited from Event

Global Switches

Global switches are created for rare cases and generally make Leaflet to not detect a particular browser feature even if it's there. You need to set the switch as a global variable to true before including Leaflet on the page, like this:

```
<script>L_NO_TOUCH = true;</script>
<script src="leaflet.js"></script>
```

Switch	Description
L_NO_TOUCH	Forces Leaflet to not use touch events even if it detects them.
L_DISABLE_3D	Forces Leaflet to not use hardware-accelerated CSS 3D transforms for positioning (which may cause glitches in some rare environments) even if they're supported.

noConflict

This method restores the L global variable to the original value it had before Leaflet inclusion, and returns the real Leaflet namespace so you can put it elsewhere, like this:

```
<script src='libs/l.js'>
<!-- L points to some other library -->

<script src='leaflet.js'>
<!-- you include Leaflet, it replaces the L variable to Leaflet namespace -->

<script>
var Leaflet = L.noConflict();
// now L points to that other library again, and you can use Leaflet.Map etc.
</script>
```

version

A constant that represents the Leaflet version in use.

```
L.version; // contains "1.0.0" (or whatever version is currently in use)
```