Leaflet (http://leafletjs.com)

An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps

Star 6,309 Tweet Follow 6,490 follow ers Like 2.5k

This reference reflects **Leaflet 0.7**. Docs for 0.6 are available **in the source form (https://github.com/Leaflet/Leaflet/archive/gh-pages-0.6.zip)** (see **instructions for running docs (https://github.com/Leaflet/Leaflet/blob/master/CONTRIBUTING.md#improving-documentation)**).

Map

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( <HTMLElement|String> id, <Map options (#map-options)> options? )`	Instantiates a map object given a div element (or its id) and optionally an object literal with map options described below.

Options

Map State Options

Option	Type	Default	Description
`center`	LatLng (#latlng)	`null`	Initial geographical center of the map.
`zoom`	`Number`	`null`	Initial map zoom.
`layers`	ILayer (#ilayer)`[]`	`null`	Layers that will be added to the map initially.
`minZoom`	`Number`	`null`	Minimum zoom level of the map. Overrides any `minZoom` set on map layers.
`maxZoom`	`Number`	`null`	Maximum zoom level of the map. This overrides any `maxZoom` set on map layers.
`maxBounds`	LatLngBounds (#latlngbounds)	`null`	When this option is set, the map restricts the view to the given geographical bounds, bouncing the user back when he tries to pan outside the view, and also not allowing to zoom out to a view that's larger than the given bounds (depending on the map size). To set the restriction dynamically, use **setMaxBounds (#map-setmaxbounds)** method
`crs`	CRS (#icrs)	`L.CRS.EPSG3857`	Coordinate Reference System to use. Don't change this if you're not sure what it means.

Interaction Options

Option	Type	Default	Description
`dragging`	`Boolean`	`true`	Whether the map be draggable with mouse/touch or not.
`touchZoom`	`Boolean`	`true`	Whether the map can be zoomed by touch-dragging with two fingers.
`scrollWheelZoom`	`Boolean`	`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.
`doubleClickZoom`	`Boolean`	`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.
`boxZoom`	`Boolean`	`true`	Whether the map can be zoomed to a rectangular area specified by dragging the mouse while pressing shift.
`tap`	`Boolean`	`true`	Enables mobile hacks for supporting instant taps (fixing 200ms click delay on iOS/Android) and touch holds (fired as `contextmenu` events).
`tapTolerance`	`Number`	`15`	The max number of pixels a user can shift his finger during touch for it to be considered a valid tap.
`trackResize`	`Boolean`	`true`	Whether the map automatically handles browser window resize to update itself.
`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.
`closePopupOnClick`	`Boolean`	`true`	Set it to `false` if you don't want popups to close when user clicks the map.
`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.

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.
`keyboardPanOffset`	`Number`	`80`	Amount of pixels to pan when pressing an arrow key.
`keyboardZoomOffset`	`Number`	`1`	Number of zoom levels to change when pressing `+` or `-` key.

Panning Inertia Options

Option	Type	Default	Description
`inertia`	`Boolean`	`true`	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.
`inertiaDeceleration`	`Number`	`3000`	The rate with which the inertial movement slows down, in pixels/second^2.
`inertiaMaxSpeed`	`Number`	`1500`	Max speed of the inertial movement, in pixels/second.
`inertiaThreshold`	`Number`	`depends`	Number of milliseconds that should pass between stopping the movement and releasing the mouse or touch to prevent inertial movement. `32` for touch devices and `14` for the rest by default.

Control options

Option	Type	Default	Description
`zoomControl`	`Boolean`	`true`	Whether the **zoom control (#control-zoom)** is added to the map by default.
`attributionControl`	`Boolean`	`true`	Whether the **attribution control (#control-attribution)** is added to the map by default.

Animation options

Option	Type	Default	Description
`fadeAnimation`	`Boolean`	depends	Whether the tile fade animation is enabled. By default it's enabled in all browsers that support CSS3 Transitions except Android.
`zoomAnimation`	`Boolean`	depends	Whether the tile 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.
`markerZoomAnimation`	`Boolean`	depends	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.

Events

You can subscribe to the following events using **these methods (#events)**.

Event	Data	Description
`click`	`MouseEvent (#mouse-event)`	Fired when the user clicks (or taps) the map.
`dblclick`	`MouseEvent (#mouse-event)`	Fired when the user double-clicks (or double-taps) the map.
`mousedown`	`MouseEvent (#mouse-event)`	Fired when the user pushes the mouse button on the map.
`mouseup`	`MouseEvent (#mouse-event)`	Fired when the user pushes the mouse button on the map.
`mouseover`	`MouseEvent (#mouse-event)`	Fired when the mouse enters the map.
`mouseout`	`MouseEvent (#mouse-event)`	Fired when the mouse leaves the map.
`mousemove`	`MouseEvent (#mouse-event)`	Fired while the mouse moves over the map.
`contextmenu`	`MouseEvent (#mouse-event)`	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).
`focus`	`Event (#event)`	Fired when the user focuses the map either by tabbing to it or clicking/panning.
`blur`	`Event (#event)`	Fired when the map looses focus.
`preclick`	`MouseEvent (#mouse-event)`	Fired before mouse click on the map (sometimes useful when you want something to happen on click before any existing click handlers start running).
`load`	`Event (#event)`	Fired when the map is initialized (when its center and zoom are set for the first time).
`unload`	`Event (#event)`	Fired when the map is destroyed with **remove (#map-remove)** method.
`viewreset`	`Event (#event)`	Fired when the map needs to redraw its content (this usually happens on map zoom or load). Very useful for creating custom overlays.
`movestart`	`Event (#event)`	Fired when the view of the map starts changing (e.g. user starts dragging the map).
`move`	`Event (#event)`	Fired on any movement of the map view.
`moveend`	`Event (#event)`	Fired when the view of the map ends changed (e.g. user stopped dragging the map).
`dragstart`	`Event (#event)`	Fired when the user starts dragging the map.
`drag`	`Event (#event)`	Fired repeatedly while the user drags the map.
`dragend`	`DragEndEvent (#dragend-event)`	Fired when the user stops dragging the map.
`zoomstart`	`Event (#event)`	Fired when the map zoom is about to change (e.g. before zoom animation).
`zoomend`	`Event (#event)`	Fired when the map zoom changes.
`zoomlevelschange`	`Event (#event)`	Fired when the number of zoomlevels on the map is changed due to adding or removing a layer.
`resize`	`ResizeEvent (#resize-event)`	Fired when the map is resized.
`autopanstart`	`Event (#event)`	Fired when the map starts autopanning when opening a popup.
`layeradd`	`LayerEvent (#layer-event)`	Fired when a new layer is added to the map.
`layerremove`	`LayerEvent (#layer-event)`	Fired when some layer is removed from the map.
`baselayerchange`	`LayerEvent (#layer-event)`	Fired when the base layer is changed through the **layer control (#control-layers)**.
	`LayerEvent`	

overlayadd	(#layer-event)	Fired when an overlay is selected through the **layer control (#control-layers)**.
overlayremove	LayerEvent (#layer-event)	Fired when an overlay is deselected through the **layer control (#control-layers)**.
locationfound	LocationEvent (#location-event)	Fired when geolocation (using the **locate (#map-locate)** method) went successfully.
locationerror	ErrorEvent (#error-event)	Fired when geolocation (using the **locate (map-locate)** method) failed.
popupopen	PopupEvent (#popup-event)	Fired when a popup is opened (using `openPopup` method).
popupclose	PopupEvent (#popup-event)	Fired when a popup is closed (using `closePopup` method).

Methods for Modifying Map State

Method	Returns	Description
setView(<LatLng (#latlng)> *center*, <Number> *zoom?*, <zoom/pan options (#map-zoompanoptions)> *options?*)	this	Sets the view of the map (geographical center and zoom) with the given animation options.
setZoom(<Number> *zoom*, <zoom options (#map-zoomoptions)> *options?*)	this	Sets the zoom of the map.
zoomIn(<Number> *delta?*, <zoom options (#map-zoomoptions)> *options?*)	this	Increases the zoom of the map by `delta` (`1` by default).
zoomOut(<Number> *delta?*, <zoom options (#map-zoomoptions)> *options?*)	this	Decreases the zoom of the map by `delta` (`1` by default).
setZoomAround(<LatLng (#latlng)> *latlng*, <Number> *zoom*, <zoom options (#map-zoomoptions)> *options?*)	this	Zooms the map while keeping a specified point on the map stationary (e.g. used internally for scroll zoom and double-click zoom).
fitBounds(<LatLngBounds (#latlngbounds)> *bounds*, <fitBounds options (#map-fitboundsoptions)> *options?*)	this	Sets a map view that contains the given geographical bounds with the maximum zoom level possible.
fitWorld(<fitBounds options (#map-fitboundsoptions)> *options?*)	this	Sets a map view that mostly contains the whole world with the maximum zoom level possible.
panTo(<LatLng (#latlng)> *latlng*, <pan options (#map-panoptions)> *options?*)	this	Pans the map to a given center. Makes an animated pan if new center is not more than one screen away from the current one.
panInsideBounds(<LatLngBounds (#latlngbounds)> *bounds*, <pan options (#map-panoptions)> *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.
panBy(<Point (#point)> *point*, <pan options (#map-panoptions)> *options?*)	this	Pans the map by a given number of pixels (animated).
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.
invalidateSize(<zoom/pan options (#map-zoompanoptions)> *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 the method is called many times in a row.
setMaxBounds(<LatLngBounds (#latlngbounds)> *bounds*	this	Restricts the map view to the given bounds (see **map maxBounds (#map-maxbounds)** option).
locate(<Locate options (#map-locate-options)> *options?*)	this	Tries to locate the user using the **Geolocation API (https://en.wikipedia.org/wiki/W3C_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). See **Locate options (#map-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}`.
remove()	this	Destroys the map and clears all related event listeners.

Methods for Getting Map State

Method	Returns	Description
getCenter()	LatLng (#latlng)	Returns the geographical center of the map view.
getZoom()	Number	Returns the current zoom of the map view.

getMinZoom()	Number	Returns the minimum zoom level of the map.
getMaxZoom()	Number	Returns the maximum zoom level of the map.
getBounds()	LatLngBounds (#latlngbounds)	Returns the LatLngBounds of the current map view.
getBoundsZoom(<LatLngBounds (#latlngbounds)> *bounds*, <Boolean> *inside?*)	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 (#point)	Returns the current size of the map container.
getPixelBounds()	Bounds	Returns the bounds of the current map view in projected pixel coordinates (sometimes useful in layer and overlay implementations).
getPixelOrigin()	Point (#point)	Returns the projected pixel coordinates of the top left point of the map layer (useful in custom layer and overlay implementations).

Methods for Layers and Controls

Method	Returns	Description
addLayer(<ILayer (#ilayer)> *layer*, <Boolean> *insertAtTheBottom?*)	this	Adds the given layer to the map. If optional `insertAtTheBottom` is set to `true`, the layer is inserted under all others (useful when switching base tile layers).
removeLayer(<ILayer (#ilayer)> *layer*)	this	Removes the given layer from the map.
hasLayer(<ILayer (#ilayer)> *layer*)	Boolean	Returns `true` if the given layer is currently added to the map.
openPopup(<Popup (#popup)> *popup*)	this	Opens the specified popup while closing the previously opened (to make sure only one is opened at one time for usability).
openPopup(<String> *html* | <HTMLElement> *el*, <LatLng (#latlng)> *latlng*, <Popup options (#popup-options)> *options?*)	this	Creates a popup with the specified options and opens it in the given point on a map.
closePopup(<Popup (#popup)> *popup?*)	this	Closes the popup previously opened with **openPopup (#map-openpopup)** (or the given one).
addControl(<IControl (#icontrol)> *control*)	this	Adds the given control to the map.
removeControl(<IControl (#icontrol)> *control*)	this	Removes the given control from the map.

Conversion Methods

Method	Returns	Description
latLngToLayerPoint(<LatLng (#latlng)> *latlng*)	Point (#point)	Returns the map layer point that corresponds to the given geographical coordinates (useful for placing overlays on the map).
layerPointToLatLng(<Point (#point)> *point*)	LatLng (#latlng)	Returns the geographical coordinates of a given map layer point.
containerPointToLayerPoint(<Point (#point)> *point*)	Point (#point)	Converts the point relative to the map container to a point relative to the map layer.
layerPointToContainerPoint(<Point (#point)> *point*)	Point (#point)	Converts the point relative to the map layer to a point relative to the map container.
latLngToContainerPoint(<LatLng (#latlng)> *latlng*)	Point (#point)	Returns the map container point that corresponds to the given geographical coordinates.
containerPointToLatLng(<Point (#point)> *point*)	LatLng (#latlng)	Returns the geographical coordinates of a given map container point.
project(<LatLng (#latlng)> *latlng*, <Number> *zoom?*)	Point (#point)	Projects the given geographical coordinates to absolute pixel coordinates for the given zoom level (current zoom level by default).
unproject(<Point (#point)> *point*, <Number> *zoom?*)	LatLng (#latlng)	Projects the given absolute pixel coordinates to geographical coordinates for the given zoom level (current zoom level by default).
mouseEventToContainerPoint(<MouseEvent> *event*)	Point (#point)	Returns the pixel coordinates of a mouse click (relative to the top left corner of the map) given its event object.
mouseEventToLayerPoint(<MouseEvent> *event*)	Point (#point)	Returns the pixel coordinates of a mouse click relative to the map layer given its event object.
mouseEventToLatLng(<MouseEvent> *event*)	LatLng (#latlng)	Returns the geographical coordinates of the point the mouse clicked on given the click's event object.

Other Methods

Method	Returns	Description
getContainer()	HTMLElement	Returns the container element of the map.
getPanes()	MapPanes (#map-panes)	Returns an object with different map panes (to render overlays in).

whenReady(<Function> *fn*, <Object> *context?*)	this	Runs the given callback when the map gets initialized with a place and zoom, or immediately if it happened already, optionally passing a function context.

Locate options

Option	Type	Default	Description
watch	Boolean	false	If true, starts continous watching of location changes (instead of detecting it once) using W3C watchPosition method. 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 (http://dev.w3.org/geo/api/spec-source.html#high-accuracy)**.

Zoom/pan options

Option	Type	Default	Description
reset	Boolean	false	If true, the map view will be completely reset (without any animations).
pan	pan options (#map-panoptions)	-	Sets the options for the panning (without the zoom change) if it occurs.
zoom	zoom options (#map-zoomoptions)	-	Sets the options for the zoom change if it occurs.
animate	Boolean	-	An equivalent of passing animate to both zoom and pan options (see below).

Pan options

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.
easeLinearity	Number	0.25	The curvature factor of panning animation easing (third parameter of the **Cubic Bezier curve (http://cubic-bezier.com/)**). 1.0 means linear animation, the less the more bowed the curve.
noMoveStart	Boolean	false	If true, panning won't fire movestart event on start (used internally for panning inertia).

Zoom options

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.

fitBounds options

The same as **zoom/pan options (#map-zoompanoptions)** and additionally:

Option	Type	Default	Description
paddingTopLeft	Point (#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 (#point)	[0, 0]	The same for bottom right corner of the map.
padding	Point (#point)	[0, 0]	Equivalent of setting both top left and bottom right padding to the same value.
maxZoom	Number	null	The maximum possible zoom to use.

Properties

Map properties include interaction handlers that allow you to control interaction behavior in runtime, enabling or disabling certain features such as dragging or touch zoom (see **IHandler (#ihandler)** methods). Example:

```
map.doubleClickZoom.disable();
```

You can also access default map controls like attribution control through map properties:

```
map.attributionControl.addAttribution("Earthquake data &copy; GeoNames");
```

Property	Type	Description
dragging	IHandler **(#ihandler)**	Map dragging handler (by both mouse and touch).
touchZoom	IHandler **(#ihandler)**	Touch zoom handler.
doubleClickZoom	IHandler **(#ihandler)**	Double click zoom handler.
scrollWheelZoom	IHandler **(#ihandler)**	Scroll wheel zoom handler.
boxZoom	IHandler **(#ihandler)**	Box (shift-drag with mouse) zoom handler.
keyboard	IHandler **(#ihandler)**	Keyboard navigation handler.
tap	IHandler **(#ihandler)**	Mobile touch hacks (quick tap and touch hold) handler.
zoomControl	Control.Zoom **(#control-zoom)**	Zoom control.
attributionControl	Control.Attribution **(#control-attribution)**	Attribution control.

Map Panes

An object literal (returned by **map.getPanes (#map-getpanes)**) that contains different map panes that you can use to put your custom overlays in. The difference is mostly in zIndex order that such overlays get.

Property	Type	Description
mapPane	HTMLElement	Pane that contains all other map panes.
tilePane	HTMLElement	Pane for tile layers.
objectsPane	HTMLElement	Pane that contains all the panes except tile pane.
shadowPane	HTMLElement	Pane for overlay shadows (e.g. marker shadows).
overlayPane	HTMLElement	Pane for overlays like polylines and polygons.
markerPane	HTMLElement	Pane for marker icons.
popupPane	HTMLElement	Pane for popups.

Marker

Used to put markers on the map.

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

Creation

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

Options

Option	Type	Default	Description
icon	**L.Icon (#icon)**	*	Icon class to use for rendering the marker. See **Icon documentation (#icon)** for details on how to customize the marker icon. Set to `new L.Icon.Default()` by default.
clickable	Boolean	true	If `false`, the marker will not emit mouse events and will act as a part of the underlying map.
draggable	Boolean	false	Whether the marker is draggable with mouse/touch or not.
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).
alt	String	''	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.

Events

You can subscribe to the following events using **these methods (#events)**.

Event	Data	Description
click	**MouseEvent (#mouse-event)**	Fired when the user clicks (or taps) the marker.
dblclick	**MouseEvent (#mouse-event)**	Fired when the user double-clicks (or double-taps) the marker.

mousedown	MouseEvent (#mouse-event)	Fired when the user pushes the mouse button on the marker.
mouseover	MouseEvent (#mouse-event)	Fired when the mouse enters the marker.
mouseout	MouseEvent (#mouse-event)	Fired when the mouse leaves the marker.
contextmenu	MouseEvent (#mouse-event)	Fired when the user right-clicks on the marker.
dragstart	Event (#event)	Fired when the user starts dragging the marker.
drag	Event (#event)	Fired repeatedly while the user drags the marker.
dragend	DragEndEvent (#dragend-event)	Fired when the user stops dragging the marker.
move	Event (#event)	Fired when the marker is moved via setLatLng. New coordinate include in event arguments.
add	Event (#event)	Fired when the marker is added to the map.
remove	Event (#event)	Fired when the marker is removed from the map.
popupopen	PopupEvent (#popup-event)	Fired when a popup bound to the marker is open.
popupclose	PopupEvent (#popup-event)	Fired when a popup bound to the marker is closed.

Methods

Method	Returns	Description
addTo(<Map (#map)> *map*)	this	Adds the marker to the map.
getLatLng()	LatLng (#latlng)	Returns the current geographical position of the marker.
setLatLng(<LatLng (#latlng)> *latlng*)	this	Changes the marker position to the given point.
setIcon(<Icon (#icon)> *icon*)	this	Changes the marker icon.
setZIndexOffset(<Number> *offset*)	this	Changes the **zIndex offset (#marker-zindexoffset)** of the marker.
setOpacity(<Number> *opacity*)	this	Changes the opacity of the marker.
update()	this	Updates the marker position, useful if coordinates of its latLng object were changed directly.
bindPopup(<String> *html* | <HTMLElement> *el* | <Popup (#popup)> *popup*, <Popup options (#popup-options)> *options?*)	this	Binds a popup with a particular HTML content to a click on this marker. You can also open the bound popup with the Marker **openPopup (#marker-openpopup)** method.
unbindPopup()	this	Unbinds the popup previously bound to the marker with bindPopup.
openPopup()	this	Opens the popup previously bound by the **bindPopup (#marker-bindpopup)** method.
getPopup()	Popup (#popup)> *popup*	Returns the popup previously bound by the **bindPopup (#marker-bindpopup)** method.
closePopup()	this	Closes the bound popup of the marker if it's opened.
togglePopup()	this	Toggles the popup previously bound by the **bindPopup (#marker-bindpopup)** method.
setPopupContent(<String> *html* | <HTMLElement> *el*, <Popup options (#popup-options)> *options?*)	this	Sets an HTML content of the popup of this marker.
toGeoJSON()	Object	Returns a **GeoJSON (http://en.wikipedia.org/wiki/GeoJSON)** representation of the marker (GeoJSON Point Feature).

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 **IHandler (#ihandler)** methods). Example:

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

Property	Type	Description
dragging	IHandler (#ihandler)	Marker dragging handler (by both mouse and touch).

Popup

Used to open popups in certain places of the map. Use **Map#openPopup (#map-openpopup)** to open popups while making sure that only one popup is open at one time (recommended for usability), or use **Map#addLayer (#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. Here's a more complicated way to open a popup on a map:

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

Creation

Factory	Description
L.popup(<[Popup options (#popup-options)](#popup-options)> *options?*, <[ILayer (#ilayer)](#ilayer)> *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 ILayer to which it refers.

Options

Option	Type	Default	Description
maxWidth	Number	300	Max width of the popup.
minWidth	Number	50	Min width of the popup.
maxHeight	Number	null	If set, creates a scrollable container of the given height inside a popup if its content exceeds it.
autoPan	Boolean	true	Set it to `false` if you don't want the map to do panning animation to fit the opened popup.
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 presense of a close button in the popup.
offset	[Point (#point)](#point)	Point(0, 6)	The offset of the popup position. Useful to control the anchor of the popup when opening it on some overlays.
autoPanPaddingTopLeft	[Point (#point)](#point)	null	The margin between the popup and the top left corner of the map view after autopanning was performed.
autoPanPaddingBottomRight	[Point (#point)](#point)	null	The margin between the popup and the bottom right corner of the map view after autopanning was performed.
autoPanPadding	[Point (#point)](#point)	Point(5, 5)	Equivalent of setting both top left and bottom right autopan padding to the same value.
zoomAnimation	Boolean	true	Whether to animate the popup on zoom. Disable it if you have problems with Flash content inside popups.
closeOnClick	Boolean	null	Set it to `false` if you want to override the default behavior of the popup closing when user clicks the map (set globally by the `Map closePopupOnClick` option).

Methods

Method	Returns	Description
addTo(<[Map (#map)](#map)> *map*)	this	Adds the popup to the map.
openOn(<[Map (#map)](#map)> *map*)	this	Adds the popup to the map and closes the previous one. The same as `map.openPopup(popup)`.
setLatLng(<[LatLng (#latlng)](#latlng)> *latlng*)	this	Sets the geographical point where the popup will open.
getLatLng()	[LatLng (#latlng)](#latlng)	Returns the geographical point of popup.
setContent(<String|HTMLElement> *htmlContent*)	this	Sets the HTML content of the popup.
getContent()	<String|HTMLElement>	Returns the content of the popup.
update()	this	Updates the popup content, layout and position. Useful for updating the popup after something inside changed, e.g. image loaded.

TileLayer

Used to load and display tile layers on the map, implements **[ILayer (#ilayer)](#ilayer)** interface.

Usage example

```
L.tileLayer('http://{s}.tile.cloudmade.com/{key}/{styleId}/256/{z}/{x}/{y}.png', {
    key: 'API-key',
    styleId: 997
}).addTo(map);
```

Creation

Factory	Description

L.tileLayer(<String> *urlTemplate (#url-template)*, <TileLayer options (#tilelayer-options)> *options?*)	Instantiates a tile layer object given a **URL template (#url-template)** and optionally an options object.

URL template

A string of the following form:

```
'http://{s}.somedomain.com/blabla/{z}/{x}/{y}.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.

You can use custom keys in the template, which will be **evaluated (#util-template)** from TileLayer options, like this:

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

Options

Option	Type	Default	Description
minZoom	`Number`	`0`	Minimum zoom number.
maxZoom	`Number`	`18`	Maximum zoom number.
maxNativeZoom	`Number`	`null`	Maximum zoom number the tiles source has available. If it is specified, the tiles on all zoom levels higher than `maxNativeZoom` will be loaded from `maxZoom` level and auto-scaled.
tileSize	`Number`	`256`	Tile size (width and height in pixels, assuming tiles are square).
subdomains	`String` or `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.
attribution	`String`	`''`	e.g. "© CloudMade" — the string used by the attribution control, describes the layer data.
tms	`Boolean`	`false`	If `true`, inverses Y axis numbering for tiles (turn this on for TMS services).
continuousWorld	`Boolean`	`false`	If set to `true`, the tile coordinates won't be wrapped by world width (-180 to 180 longitude) or clamped to lie within world height (-90 to 90). Use this if you use Leaflet for maps that don't reflect the real world (e.g. game, indoor or photo maps).
noWrap	`Boolean`	`false`	If set to `true`, the tiles just won't load outside the world width (-180 to 180 longitude) instead of repeating.
zoomOffset	`Number`	`0`	The zoom number used in tile URLs will be offset with this value.
zoomReverse	`Boolean`	`false`	If set to `true`, the zoom number used in tile URLs will be reversed (`maxZoom - zoom` instead of `zoom`)
opacity	`Number`	`1.0`	The opacity of the tile layer.
zIndex	`Number`	`null`	The explicit zIndex of the tile layer. Not set by default.
unloadInvisibleTiles	`Boolean`	depends	If `true`, all the tiles that are not visible after panning are removed (for better performance). `true` by default on mobile WebKit, otherwise `false`.
updateWhenIdle	`Boolean`	depends	If `false`, new tiles are loaded during panning, otherwise only after it (for better performance). `true` by default on mobile WebKit, otherwise `false`.
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.
reuseTiles	`Boolean`	`false`	If `true`, all the tiles that are not visible after panning are placed in a reuse queue from which they will be fetched when new tiles become visible (as opposed to dynamically creating new ones). This will in theory keep memory usage low and eliminate the need for reserving new memory whenever a new tile is needed.

Events

You can subscribe to the following events using **these methods (#events)**.

Event	Data	Description
loading	Event (#event)	Fired when the tile layer starts loading tiles.
load	Event (#event)	Fired when the tile layer loaded all visible tiles.
tileloadstart	TileEvent (#tile-event)	Fired when a tile is requested and starts loading.
tileload	TileEvent (#tile-event)	Fired when a tile loads.
tileunload	TileEvent (#tile-event)	Fired when a tile is removed (e.g. when you have `unloadInvisibleTiles` on).

Methods

Method	Returns	Description
addTo(<Map (#map)> *map*)	`this`	Adds the layer to the map.
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.
setOpacity(<Number> *opacity*)	this	Changes the opacity of the tile layer.
setZIndex(<Number> *zIndex*)	this	Sets the zIndex of the tile layer.
redraw()	this	Causes the layer to clear all the tiles and request them again.
setUrl(<String> ***urlTemplate (#url-template)***)	this	Updates the layer's URL template and redraws it.
getContainer()	HTMLElement	Returns the HTML element that contains the tiles for this layer.

TileLayer.WMS

Used to display WMS services as tile layers on the map. Extends **TileLayer (#tilelayer)**.

Usage example

```
var nexrad = L.tileLayer.wms("http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi", {
    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 (#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

Includes all **TileLayer options (#tilelayer-options)** and additionally:

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 (#icrs)	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.

Methods

Method	Returns	Description
setParams(<**WMS parameters (#tilelayer-wms-options)**> *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).

TileLayer.Canvas

Used to create Canvas-based tile layers where tiles get drawn on the browser side. Extends **TileLayer (#tilelayer)**.

Usage example

```
var canvasTiles = L.tileLayer.canvas();

canvasTiles.drawTile = function(canvas, tilePoint, zoom) {
    var ctx = canvas.getContext('2d');
    // draw something on the tile canvas
}
```

Creation

Factory	Description
L.tileLayer.canvas(<**TileLayer options (#tilelayer-options)**> *options?*)	Instantiates a Canvas tile layer object given an options object (optionally).

Options

Option	Type	Default	Description
async	Boolean	false	Indicates that tiles will be drawn asynchronously. **tileDrawn (#tilelayer-canvas-tiledrawn)** method should be called for each tile after drawing completion.

Methods

Method	Returns	Description
drawTile(<HTMLCanvasElement> *canvas*, <Point (#point)> *tilePoint*, <Number> *zoom*)	this	You need to define this method after creating the instance to draw tiles; canvas is the actual canvas tile on which you can draw, tilePoint represents the tile numbers, and zoom is the current zoom.
tileDrawn(<HTMLCanvasElement> *canvas*)	-	If async option is defined, this function should be called for each tile after drawing completion. canvas is the same canvas element, that was passed to **drawTile (#tilelayer-canvas-drawtile)**.

ImageOverlay

Used to load and display a single image over specific bounds of the map, implements **ILayer (#ilayer)** interface.

Usage example

```
var imageUrl = 'http://www.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 (#latlngbounds)> *bounds*, <ImageOverlay options (#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.
attribution	String	''	The attribution text of the image overlay.

Methods

Method	Returns	Description
addTo(<Map (#map)> *map*)	this	Adds the overlay to the map.
setOpacity(<Number> *opacity*)	this	Sets the opacity of the overlay.
setUrl(<String> *imageUrl*)	this	Changes the URL of the image.
bringToFront()	this	Brings the layer to the top of all overlays.
bringToBack()	this	Brings the layer to the bottom of all overlays.

Path

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

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	'#03f'	Stroke color.
weight	Number	5	Stroke width in pixels.
opacity	Number	0.5	Stroke opacity.
fill	Boolean	depends	Whether to fill the path with color. Set it to false to disable filling on polygons or circles.
fillColor	String	same as color	Fill color.
fillOpacity	Number	0.2	Fill opacity.
dashArray	String	null	A string that defines the stroke **dash pattern (https://developer.mozilla.org/en/SVG/Attribute/stroke-dasharray)**. Doesn't work on canvas-powered layers (e.g. Android 2).
lineCap	String	null	A string that defines **shape to be used at the end (https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-**

			linecap) of the stroke.
lineJoin	String	null	A string that defines shape to be used at the corners (https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/stroke-linejoin) of the stroke.
clickable	Boolean	true	If `false`, the vector will not emit mouse events and will act as a part of the underlying map.
pointerEvents	String	null	Sets the `pointer-events` attribute on the path if SVG backend is used.
className	String	''	Custom class name set on an element.

Events

You can subscribe to the following events using **these methods (#events)**.

Event	Data	Description
click	MouseEvent (#mouse-event)	Fired when the user clicks (or taps) the object.
dblclick	MouseEvent (#mouse-event)	Fired when the user double-clicks (or double-taps) the object.
mousedown	MouseEvent (#mouse-event)	Fired when the user pushes the mouse button on the object.
mouseover	MouseEvent (#mouse-event)	Fired when the mouse enters the object.
mouseout	MouseEvent (#mouse-event)	Fired when the mouse leaves the object.
contextmenu	MouseEvent (#mouse-event)	Fired when the user pushes the right mouse button on the object, prevents default browser context menu from showing if there are listeners on this event.
add	Event (#event)	Fired when the path is added to the map.
remove	Event (#event)	Fired when the path is removed from the map.
popupopen	PopupEvent (#popup-event)	Fired when a popup bound to the path is open.
popupclose	PopupEvent (#popup-event)	Fired when a popup bound to the path is closed.

Methods

Method	Returns	Description
addTo(<Map (#map)> *map*)	this	Adds the layer to the map.
bindPopup(<String> *html* | <HTMLElement> *el* | <Popup (#popup)> *popup*, <Popup options (#popup-options)> *options?*)	this	Binds a popup with a particular HTML content to a click on this path.
bindPopup(<Popup (#popup)> *popup*, <Popup options (#popup-options)> *options?*)	this	Binds a given popup object to the path.
unbindPopup()	this	Unbinds the popup previously bound to the path with `bindPopup`.
openPopup(<LatLng (#latlng)> *latlng?*)	this	Opens the popup previously bound by the **bindPopup (#path-bindpopup)** method in the given point, or in one of the path's points if not specified.
closePopup()	this	Closes the path's bound popup if it is opened.
setStyle(<Path options (#path-options)> *object*)	this	Changes the appearance of a Path based on the options in the **Path options (#path-options)** object.
getBounds()	LatLngBounds (#latlngbounds)	Returns the LatLngBounds of the path.
bringToFront()	this	Brings the layer to the top of all path layers.
bringToBack()	this	Brings the layer to the bottom of all path layers.
redraw()	this	Redraws the layer. Sometimes useful after you changed the coordinates that the path uses.

Static properties

Constant	Type	Value	Description
SVG	Boolean	depends	True if SVG is used for vector rendering (true for most modern browsers).
VML	Boolean	depends	True if VML is used for vector rendering (IE 6-8).
CANVAS	Boolean	depends	True if Canvas is used for vector rendering (Android 2). You can also force this by setting global variable `L_PREFER_CANVAS` to `true` *before* the Leaflet include on your page — sometimes it can increase performance dramatically when rendering thousands of circle markers, but currently suffers from a bug that causes removing such layers to be extremely slow.

CLIP_PADDING	Number	0.5 for SVG 0.02 for VML	How much to extend the clip area around the map view (relative to its size, e.g. 0.5 is half the screen in each direction). Smaller values mean that you will see clipped ends of paths while you're dragging the map, and bigger values decrease drawing performance.

Polyline

A class for drawing polyline overlays on a map. Extends **Path (#path)**. Use **Map#addLayer (#map-addlayer)** to add it to the map.

Usage example

```
// create a red polyline from an arrays of LatLng points
var polyline = L.polyline(latlngs, {color: 'red'}).addTo(map);

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

Creation

Factory	Description
L.polyline(<LatLng (#latlng)[]> *latlngs*, <Polyline options (#polyline-options)> *options?*)	Instantiates a polyline object given an array of geographical points and optionally an options object.

Options

You can use **Path options (#path-options)** and additionally the following 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	Disabled polyline clipping.

Methods

You can use **Path methods (#path-methods)** and additionally the following methods:

Method	Returns	Description
addLatLng(<LatLng (#latlng)> *latlng*)	this	Adds a given point to the polyline.
setLatLngs(<LatLng (#latlng)[]> *latlngs*)	this	Replaces all the points in the polyline with the given array of geographical points.
getLatLngs()	LatLng (#latlng)[]	Returns an array of the points in the path.
spliceLatLngs(<Number> *index*, <Number> *pointsToRemove*, <LatLng (#latlng)> *latlng?*, …)	LatLng (#latlng)[]	Allows adding, removing or replacing points in the polyline. Syntax is the same as in **Array#splice (https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Array/splice)**. Returns the array of removed points (if any).
getBounds()	LatLngBounds (#latlngbounds)	Returns the LatLngBounds of the polyline.
toGeoJSON()	Object	Returns a **GeoJSON (http://en.wikipedia.org/wiki/GeoJSON)** representation of the polyline (GeoJSON LineString Feature).

MultiPolyline

Extends **FeatureGroup (#featuregroup)** to allow creating multi-polylines (single layer that consists of several polylines that share styling/popup).

Creation

Factory	Description
L.multiPolyline(<LatLng (#latlng)[][]> *latlngs*, <Polyline options (#polyline-options)> *options?*)	Instantiates a multi-polyline object given an array of arrays of geographical points (one for each individual polyline) and optionally an options object.

Methods

MultiPolylines accept all **Polyline methods (#polyline)** but have different behavior around their coordinate contents since they can contain multiple line features:

Method	Returns	Description

`setLatLngs( <LatLng (#latlng)[][]> latlngs )`	`this`	Replace all lines and their paths with the given array of arrays of geographical points.
`getLatLngs()`	`<LatLng (#latlng)[][]> latlngs`	Returns an array of arrays of geographical points in each line.
`openPopup()`	`this`	Opens the popup previously bound by **bindPopup (#path-bindpopup)**.
`toGeoJSON()`	`Object`	Returns a **GeoJSON (http://en.wikipedia.org/wiki/GeoJSON)** representation of the multipolyline (GeoJSON MultiLineString Feature).

Polygon

A class for drawing polygon overlays on a map. Extends **Polyline (#polyline)**. Use **Map#addLayer (#map-addlayer)** to add it to the map.

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.

Creation

Factory	Description
`L.polygon( <LatLng (#latlng)[]> latlngs, <Polyline options (#polyline-options)> options? )`	Instantiates a polygon object given an array of geographical points and optionally an options object (the same as for Polyline). You can also create a polygon with holes by passing an array of arrays of latlngs, with the first latlngs array representing the exterior ring while the remaining represent the holes inside.

Methods

Polygon has the same options and methods as Polyline, with the following differences:

Method	Returns	Description
`toGeoJSON()`	`Object`	Returns a **GeoJSON (http://en.wikipedia.org/wiki/GeoJSON)** representation of the polygon (GeoJSON Polygon Feature).

MultiPolygon

Extends **FeatureGroup (#featuregroup)** to allow creating multi-polygons (single layer that consists of several polygons that share styling/popup).

Creation

Factory	Description
`L.multiPolygon( <LatLng (#latlng)[][]> latlngs, <Polyline options (#polyline-options)> options? )`	Instantiates a multi-polygon object given an array of latlngs arrays (one for each individual polygon) and optionally an options object (the same as for MultiPolyline).

Methods

MultiPolygons accept all **Polyline methods (#polyline)** but have different behavior around their coordinate contents since they can contain multiple polygon features:

Method	Returns	Description
`setLatLngs( <LatLng (#latlng)[][]> latlngs )`	`this`	Replace all polygons and their paths with the given array of arrays of geographical points.
`getLatLngs()`	`<LatLng (#latlng)[][]> latlngs`	Returns an array of arrays of geographical points in each polygon.
`openPopup()`	`this`	Opens the popup previously bound by **bindPopup (#path-bindpopup)**.
`toGeoJSON()`	`Object`	Returns a **GeoJSON (http://en.wikipedia.org/wiki/GeoJSON)** representation of the multipolygon (GeoJSON MultiPolygon Feature).

Rectangle

A class for drawing rectangle overlays on a map. Extends **Polygon (#polygon)**. Use **Map#addLayer (#map-addlayer)** to add it to the map.

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)> bounds, <Path options (#path-options)> options? )`	Instantiates a rectangle object with the given geographical bounds and optionally an options object.

Methods

You can use **Path methods (#path-methods)** and additionally the following methods:

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

Circle

A class for drawing circle overlays on a map. Extends **Path (#path)**. Use **Map#addLayer (#map-addlayer)** to add it to the map.

```
L.circle([50.5, 30.5], 200).addTo(map);
```

Creation

Factory	Description
`L.circle( <LatLng (#latlng)> latlng, <Number> radius, <Path options (#path-options)> options? )`	Instantiates a circle object given a geographical point, a radius in meters and optionally an options object.

Methods

Method	Returns	Description
`getLatLng()`	`LatLng (#latlng)`	Returns the current geographical position of the circle.
`getRadius()`	`Number`	Returns the current radius of a circle. Units are in meters.
`setLatLng( <LatLng (#latlng)> latlng )`	`this`	Sets the position of a circle to a new location.
`setRadius( <Number> radius )`	`this`	Sets the radius of a circle. Units are in meters.
`toGeoJSON()`	`Object`	Returns a **GeoJSON (http://en.wikipedia.org/wiki/GeoJSON)** representation of the circle (GeoJSON Point Feature).

CircleMarker

A circle of a fixed size with radius specified in pixels. Extends **Circle (#circle)**. Use **Map#addLayer (#map-addlayer)** to add it to the map.

Creation

Factory	Description
`L.circleMarker( <LatLng (#latlng)> latlng, <Path options (#path-options)> options? )`	Instantiates a circle marker given a geographical point and optionally an options object. The default radius is 10 and can be altered by passing a "radius" member in the path options object.

Methods

Method	Returns	Description
`setLatLng( <LatLng (#latlng)> latlng )`	`this`	Sets the position of a circle marker to a new location.
`setRadius( <Number> radius )`	`this`	Sets the radius of a circle marker. Units are in pixels.
`toGeoJSON()`	`Object`	Returns a **GeoJSON (http://en.wikipedia.org/wiki/GeoJSON)** representation of the circle marker (GeoJSON Point Feature).

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. Implements **ILayer (#ilayer)** interface.

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

Creation

Factory	Description
`L.LayerGroup( <ILayer (#ilayer)[]> layers? )`	Create a layer group, optionally given an initial set of layers.

Methods

Method	Returns	Description
`addTo( <Map (#map)> map )`	`this`	Adds the group of layers to the map.
`addLayer( <ILayer (#ilayer)> layer )`	`this`	Adds a given layer to the group.
`removeLayer( <ILayer (#ilayer)> layer )`	`this`	Removes a given layer from the group.
`removeLayer( <String> id )`	`this`	Removes a given layer of the given id from the group.
`hasLayer( <ILayer (#ilayer)> layer )`	`Boolean`	Returns `true` if the given layer is currently added to the group.
`getLayer( <String> id )`	`ILayer (#ilayer)`	Returns the layer with the given id.
`getLayers()`	`Array`	Returns an array of all the layers added to the group.
`clearLayers()`	`this`	Removes all the layers from the group.

`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');`
		`});`

`toGeoJSON()`	`Object`	Returns a **GeoJSON (http://en.wikipedia.org/wiki/GeoJSON)** representation of the layer group (GeoJSON FeatureCollection).

FeatureGroup

Extended **layerGroup (#layergroup)** that also has mouse events (propagated from members of the group) and a shared bindPopup method. Implements **ILayer (#ilayer)** interface.

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

Creation

Factory	Description
`L.featureGroup( <ILayer (#ilayer)[]> layers? )`	Create a layer group, optionally given an initial set of layers.

Methods

Has all **layerGroup (#layergroup)** methods and additionally:

Method	Returns	Description
`bindPopup( <String> htmlContent, <Popup options (#popup-options)> options? )`	`this`	Binds a popup with a particular HTML content to a click on any layer from the group that has a `bindPopup` method.
`getBounds()`	`LatLngBounds (#latlngbounds)`	Returns the LatLngBounds of the Feature Group (created from bounds and coordinates of its children).
`setStyle( <Path options (#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 bottom of all other layers.

Events

You can subscribe to the following events using **these methods (#events)**.

Event	Data	Description
`click`	`MouseEvent (#mouse-event)`	Fired when the user clicks (or taps) the group.
`dblclick`	`MouseEvent (#mouse-event)`	Fired when the user double-clicks (or double-taps) the group.
`mouseover`	`MouseEvent (#mouse-event)`	Fired when the mouse enters the group.
`mouseout`	`MouseEvent (#mouse-event)`	Fired when the mouse leaves the group.
`mousemove`	`MouseEvent (#mouse-event)`	Fired while the mouse moves over the layers of the group.

contextmenu	MouseEvent (#mouse-event)	Fired when the user right-clicks on one of the layers.
layeradd	LayerEvent (#layer-event)	Fired when a layer is added to the group.
layerremove	LayerEvent (#layer-event)	Fired when a layer is removed from the map.

GeoJson

Represents a **GeoJSON (http://geojson.org/geojson-spec.html)** layer. Allows you to parse GeoJSON data and display it on the map. Extends **FeatureGroup (#featuregroup)**.

```
L.geoJson(data, {
    style: function (feature) {
        return {color: feature.properties.color};
    },
    onEachFeature: function (feature, layer) {
        layer.bindPopup(feature.properties.description);
    }
}).addTo(map);
```

Each feature layer created by it gets a `feature` property that links to the GeoJSON feature data the layer was created from (so that you can access its properties later).

Creation

Factory	Description
L.geoJson(<Object> *geojson?*, <GeoJSON options (#geojson-options)> *options?*)	Creates a GeoJSON layer. Optionally accepts an object in **GeoJSON format (http://geojson.org/geojson-spec.html)** to display on the map (you can alternatively add it later with `addData` method) and an options object.

Options

Option	Description
pointToLayer(<GeoJSON> *featureData*, <LatLng (#latlng)> *latlng*)	Function that will be used for creating layers for GeoJSON points (if not specified, simple markers will be created).
style(<GeoJSON> *featureData*)	Function that will be used to get style options for vector layers created for GeoJSON features.
onEachFeature(<GeoJSON> *featureData*, <ILayer (#ilayer)> *layer*)	Function that will be called on each created feature layer. Useful for attaching events and popups to features.
filter(<GeoJSON> *featureData*, <ILayer (#ilayer)> *layer*)	Function that will be used to decide whether to show a feature or not.
coordsToLatLng(<Array> *coords*)	Function that will be used for converting GeoJSON coordinates to **LatLng (#latlng)** points (if not specified, coords will be assumed to be WGS84 — standard `[longitude, latitude]` values in degrees).

Additionally accepts all **Path options (#path-options)** for polylines and polygons.

Methods

Method	Returns	Description
addData(<GeoJSON> *data*)	`this`	Adds a GeoJSON object to the layer.
setStyle(<Function> ***style (#geojson-style)***)	`this`	Changes styles of GeoJSON vector layers with the given style function.
resetStyle(<Path (#path)> *layer*)	`this`	Resets the the given vector layer's style to the original GeoJSON style, useful for resetting style after hover events.

Static methods

Method	Returns	Description
geometryToLayer(<GeoJSON> *featureData*, <Function (#geojson-pointtolayer)> *pointToLayer?*)	ILayer (#ilayer)	Creates a layer from a given GeoJSON feature.
coordsToLatlng(<Array> *coords*, <Boolean> *reverse?*)	LatLng (#latlng)	Creates a LatLng object from an array of 2 numbers (latitude, longitude) used in GeoJSON for points. If `reverse` is set to `true`, the numbers will be interpreted as (longitude, latitude).
coordsToLatlngs(<Array> *coords*, <Number> *levelsDeep?*, <Boolean> *reverse?*)	`Array`	Creates a multidimensional array of LatLng objects 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). If `reverse` is set to `true`, the numbers will be interpreted as (longitude, latitude).

LatLng

Represents a geographical point with a certain latitude and longitude.

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

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).

Properties

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

Methods

Method	Returns	Description
`distanceTo``( <`[LatLng (#latlng)](#latlng)`> `*`otherLatlng`*` )`	`Number`	Returns the distance (in meters) to the given LatLng calculated using the Haversine formula. See **description on wikipedia (http://en.wikipedia.org/wiki/Haversine_formula)**
`equals``( <`[LatLng (#latlng)](#latlng)`> `*`otherLatlng`*` )`	`Boolean`	Returns `true` if the given LatLng point is at the same position (within a small margin of error).
`toString``()`	`String`	Returns a string representation of the point (for debugging purposes).
`wrap``( <Number> `*`left`*`, <Number> `*`right`*` )`	[LatLng (#latlng)](#latlng)	Returns a new `LatLng` object with the longitude wrapped around `left` and `right` boundaries (`-180` to `180` by default).

Constants

Constant	Type	Value	Description
`DEG_TO_RAD`	`Number`	`Math.PI / 180`	A multiplier for converting degrees into radians.
`RAD_TO_DEG`	`Number`	`180 / Math.PI`	A multiplier for converting radians into degrees.
`MAX_MARGIN`	`Number`	`1.0E-9`	Max margin of error for the equality check.

LatLngBounds

Represents a rectangular geographical area on a map.

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

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]
]);
```

Creation

Factory	Description
`L.latLngBounds``( <`[LatLng (#latlng)](#latlng)`> `*`southWest`*`, <`[LatLng (#latlng)](#latlng)`> `*`northEast`*` )`	Creates a latLngBounds object by defining south-west and north-east corners of the rectangle.
`L.latLngBounds``(`	Creates a LatLngBounds object defined by the geographical points it contains. Very useful for

<LatLng (#latlng)[]> *latlngs*)	zooming the map to fit a particular set of locations with **fitBounds (#map-fitbounds)**.

Methods

Method	Returns	Description
extend(<LatLng (#latlng)|LatLngBounds (#latlngbounds)> *latlng*)	this	Extends the bounds to contain the given point or bounds.
getSouthWest()	LatLng (#latlng)	Returns the south-west point of the bounds.
getNorthEast()	LatLng (#latlng)	Returns the north-east point of the bounds.
getNorthWest()	LatLng (#latlng)	Returns the north-west point of the bounds.
getSouthEast()	LatLng (#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.
getCenter()	LatLng (#latlng)	Returns the center point of the bounds.
contains(<LatLngBounds (#latlngbounds)> *otherBounds*)	Boolean	Returns `true` if the rectangle contains the given one.
contains(<LatLng (#latlng)> *latlng*)	Boolean	Returns `true` if the rectangle contains the given point.
intersects(<LatLngBounds (#latlngbounds)> *otherBounds*)	Boolean	Returns `true` if the rectangle intersects the given bounds.
equals(<LatLngBounds (#latlngbounds)> *otherBounds*)	Boolean	Returns `true` if the rectangle is equivalent (within a small margin of error) to the given bounds.
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.
pad(<Number> *bufferRatio*)	LatLngBounds (#latlngbounds)	Returns bigger bounds created by extending the current bounds by a given percentage in each direction.
isValid()	Boolean	Returns `true` if the bounds are properly initialized.

Point

Represents a point with x and y coordinates in pixels.

```
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));
```

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.

Properties

Property	Type	Description
x	Number	The x coordinate.
y	Number	The y coordinate.

Methods

Method	Returns	Description
add(<Point (#point)> *otherPoint*)	Point (#point)	Returns the result of addition of the current and the given points.
subtract(<Point (#point)> *otherPoint*)	Point (#point)	Returns the result of subtraction of the given point from the current.

multiplyBy(<Number> *number*)	Point (#point)	Returns the result of multiplication of the current point by the given number.
divideBy(<Number> *number*, <Boolean> *round?*)	Point (#point)	Returns the result of division of the current point by the given number. If optional `round` is set to `true`, returns a rounded result.
distanceTo(<Point (#point)> *otherPoint*)	Number	Returns the distance between the current and the given points.
clone()	Point (#point)	Returns a copy of the current point.
round()	Point (#point)	Returns a copy of the current point with rounded coordinates.
floor()	Point (#point)	Returns a copy of the current point with floored coordinates (rounded down).
equals(<Point (#point)> *otherPoint*)	Boolean	Returns `true` if the given point has the same coordinates.
contains(<Point (#point)> *otherPoint*)	Boolean	Returns `true` if the 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.

Bounds

Represents a rectangular area in pixel coordinates.

```
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]]);
```

Creation

Factory	Description
L.bounds(<Point (#point)> *topLeft*, <Point (#point)> *bottomRight*)	Creates a Bounds object from two coordinates (usually top-left and bottom-right corners).
L.bounds(<Point (#point)[]> *points*)	Creates a Bounds object defined by the points it contains.

Properties

Property	Type	Description
min	Point (#point)	The top left corner of the rectangle.
max	Point (#point)	The bottom right corner of the rectangle.

Methods

Method	Returns	Description
extend(<Point (#point)> *point*)	-	Extends the bounds to contain the given point.
getCenter()	Point (#point)	Returns the center point of the bounds.
contains(<Bounds (#bounds)> *otherBounds*)	Boolean	Returns `true` if the rectangle contains the given one.
contains(<Point (#point)> *point*)	Boolean	Returns `true` if the rectangle contains the given point.
intersects(<Bounds (#bounds)> *otherBounds*)	Boolean	Returns `true` if the rectangle intersects the given bounds.
isValid()	Boolean	Returns `true` if the bounds are properly initialized.
getSize()	Point (#point)	Returns the size of the given bounds.

Icon

Represents an icon to provide when creating a marker.

```
var myIcon = L.icon({
    iconUrl: 'my-icon.png',
    iconRetinaUrl: 'my-icon@2x.png',
    iconSize: [38, 95],
    iconAnchor: [22, 94],
    popupAnchor: [-3, -76],
    shadowUrl: 'my-icon-shadow.png',
    shadowRetinaUrl: 'my-icon-shadow@2x.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 (#icon-options)]> *options*)	Creates an icon instance with the given options.

Options

Option	Type	Description
iconUrl	String	(required) The URL to the icon image (absolute or relative to your script path).
iconRetinaUrl	String	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 (#point)	Size of the icon image in pixels.
iconAnchor	Point (#point)	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.
shadowUrl	String	The URL to the icon shadow image. If not specified, no shadow image will be created.
shadowRetinaUrl	String	The URL to the retina sized version of the icon shadow image. If not specified, no shadow image will be created. Used for Retina screen devices.
shadowSize	Point (#point)	Size of the shadow image in pixels.
shadowAnchor	Point (#point)	The coordinates of the "tip" of the shadow (relative to its top left corner) (the same as `iconAnchor` if not specified).
popupAnchor	Point (#point)	The coordinates of the point from which popups will "open", relative to the icon anchor.
className	String	A custom class name to assign to both icon and shadow images. Empty by default.

DivIcon

Represents a lightweight icon for markers that uses a simple `div` element instead of an image.

```
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'` class and is styled as a little white square with a shadow.

Creation

Factory	Description
L.divIcon(<[DivIcon options (#divicon-options)]> *options*)	Creates a div icon instance with the given options.

Options

Option	Type	Description
iconSize	Point (#point)	Size of the icon in pixels. Can be also set through CSS.
iconAnchor	Point (#point)	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.
className	String	A custom class name to assign to the icon. `'leaflet-div-icon'` by default.
html	String	A custom HTML code to put inside the div element, empty by default.

Control

The base class for all Leaflet controls. Implements **IControl (#icontrol)** interface. You can add controls to the map like this:

```
control.addTo(map);
// the same as
map.addControl(control);
```

Creation

Factory	Description
`L.control( <Control options (#control-options)> options? )`	Creates a control with the given options.

Options

Option	Type	Default	Description
position	`String`	`'topright'`	The initial position of the control (one of the map corners). See **control positions (#control-positions)**.

Methods

Method	Returns	Description
`setPosition( <String> position )`	`this`	Sets the position of the control. See **control positions (#control-positions)**.
`getPosition()`	`String`	Returns the current position of the control.
`addTo( <Map (#map)> map )`	`this`	Adds the control to the map.
`removeFrom( <Map (#map)> map )`	`this`	Removes the control from the map.
`getContainer()`	`HTMLElement`	Returns the HTML container of the control.

Control Positions

Control positions (map corner to put a control to) are set using strings. Margins between controls and the map border are set with CSS, so that you can easily override them.

Position	Description
`'topleft'`	Top left of the map.
`'topright'`	Top right of the map.
`'bottomleft'`	Bottom left of the map.
`'bottomright'`	Bottom right of the map.

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 (#control)**.

Creation

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

Options

Option	Type	Default	Description
position	`String`	`'topleft'`	The position of the control (one of the map corners). See **control positions (#control-positions)**.
zoomInText	`String`	`'+'`	The text set on the zoom in button.
zoomOutText	`String`	`'-'`	The text set on the zoom out button.
zoomInTitle	`String`	`'Zoom in'`	The title set on the zoom in button.
zoomInTitle	`String`	`'Zoom out'`	The title set on the zoom out button.

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 `getAttribution` method automatically. Extends **Control (#control)**.

Creation

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

Options

Option	Type	Default	Description
position	`String`	`'bottomright'`	The position of the control (one of the map corners). See **control positions (#control-positions)**.

prefix	String `'Leaflet'`	The HTML text shown before the attributions. Pass `false` to disable.

Methods

Method	Returns	Description
setPrefix(<String> *prefix*)	`this`	Sets the text before the attributions.
addAttribution(<String> *text*)	`this`	Adds an attribution text (e.g. `'Vector data &copy; CloudMade'`).
removeAttribution(<String> *text*)	`this`	Removes an attribution text.

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 (examples/layers-control.html)**). Extends **Control (#control)**.

```
var baseLayers = {
    "CloudMade": cloudmade,
    "OpenStreetMap": osm
};

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

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

Creation

Factory	Description
L.control.layers(<Layer Config (#control-layers-config)> *baseLayers?*, <Layer Config (#control-layers-config)> *overlays?*, <Control.Layers options (#control-layers-options)> *options?*)	Creates an attribution 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.

Methods

Method	Returns	Description
addBaseLayer(<ILayer (#ilayer)> *layer*, <String> *name*)	`this`	Adds a base layer (radio button entry) with the given name to the control.
addOverlay(<ILayer (#ilayer)> *layer*, <String> *name*)	`this`	Adds an overlay (checkbox entry) with the given name to the control.
removeLayer(<ILayer (#ilayer)> *layer*)	`this`	Remove the given layer from the control.

Options

Option	Type	Default	Description
position	String	`'topright'`	The position of the control (one of the map corners). See **control positions (#control-positions)**.
collapsed	Boolean	`true`	If `true`, the control will be collapsed into an icon and expanded on mouse hover or touch.
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.

Layer Config

An object literal 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>": myLayer}
```

Events

You can subscribe to the following events on the **Map (#map)** object using **these methods (#events)**.

Event	Data	Description
baselayerchange	LayersControlEvent (#layers-control-event)	Fired when the base layer is changed through the control.
overlayadd	LayersControlEvent (#layers-control-event)	Fired when an overlay is selected through the control.

overlayremove	LayersControlEvent (#layers-control-event)	Fired when an overlay is deselected through the 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 (#control)**.

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

Creation

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

Options

Option	Type	Default	Description
position	String	'bottomleft'	The position of the control (one of the map corners). See **control positions (#control-positions)**.
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 move).

Events methods

A set of methods shared between event-powered classes (like Map). 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 'fire' event).

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
addEventListener(<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').
addOneTimeEventListener(<String> *type*, <Function> *fn*, <Object> *context?*)	this	The same as above except the listener will only get fired once and then removed.
addEventListener(<Object> *eventMap*, <Object> *context?*)	this	Adds a set of type/listener pairs, e.g. {click: onClick, mousemove: onMouseMove}
removeEventListener(<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.
removeEventListener(<Object> *eventMap*, <Object> *context?*)	this	Removes a set of type/listener pairs.
removeEventListener()	this	Removes all listeners. An alias to clearAllEventListeners when you use it without arguments.
hasEventListeners(<String> *type*)	Boolean	Returns true if a particular event type has some listeners attached to it.
fireEvent(<String> *type*, <Object> *data?*)	this	Fires an event of the specified type. You can optionally provide an data object — the first argument of the listener function will contain its properties.
clearAllEventListeners()	this	Removes all listeners to all events on the object.
on(…)	this	Alias to addEventListener.

once(…)	this	Alias to addOneTimeEventListener.
off(…)	this	Alias to removeEventListener.
fire(…)	this	Alias to fireEvent.

Event objects

Event object is an object that you recieve as an argument in a listener function when some event is fired, containing useful information about that event. For example:

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

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.

MouseEvent

property	type	description
latlng	LatLng (#latlng)	The geographical point where the mouse event occured.
layerPoint	Point (#point)	Pixel coordinates of the point where the mouse event occured relative to the map layer.
containerPoint	Point (#point)	Pixel coordinates of the point where the mouse event occured relative to the map container.
originalEvent	DOMMouseEvent	The original DOM mouse event fired by the browser.

LocationEvent

property	type	description
latlng	LatLng (#latlng)	Detected geographical location of the user.
bounds	LatLngBounds (#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.

ErrorEvent

property	type	description
message	String	Error message.
code	Number	Error code (if applicable).

LayerEvent

property	type	description
layer	ILayer (#ilayer)	The layer that was added or removed.

LayersControlEvent

property	type	description
layer	ILayer (#ilayer)	The layer that was added or removed.
name	String	The name of the layer that was added or removed.

TileEvent

property	type	description
tile	HTMLElement	The tile element (image).
url	String	The source URL of the tile.

ResizeEvent

property	type	description
oldSize	Point (#point)	The old size before resize event.
newSize	Point (#point)	The new size after the resize event.

GeoJSON event

property	type	description
layer	ILayer (#ilayer)	The layer for the GeoJSON feature that is being added to the map.
properties	Object	GeoJSON properties of the feature.
geometryType	String	GeoJSON geometry type of the feature.
id	String	GeoJSON ID of the feature (if present).

Popup event

property	type	description
popup	Popup (#popup)	The popup that was opened or closed.

DragEndEvent

property	type	description
distance	Number	The distance in pixels the draggable element was moved by.

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.

```
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("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 = L.Class.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 defauls 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}
```

Includes

`includes` is a special class property that merges all specified objects into the class (such objects are called mixins). A good example of this is `L.Mixin.Events` that **event-related methods (#events)** like `on`, `off` and `fire` to the class.

```
 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

`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, …);
```

Browser

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

```
if (L.Browser.ie6) {
    alert('Upgrade your browser, dude!');
}
```

property	type	description
`ie`	`Boolean`	`true` for all Internet Explorer versions.
`ie6`	`Boolean`	`true` for Internet Explorer 6.
`ie7`	`Boolean`	`true` for Internet Explorer 7.
`webkit`	`Boolean`	`true` for webkit-based browsers like Chrome and Safari (including mobile versions).
`webkit3d`	`Boolean`	`true` for webkit-based browsers that support CSS 3D transformations.
`android`	`Boolean`	`true` for Android mobile browser.
`android23`	`Boolean`	`true` for old Android stock browsers (2 and 3).
`mobile`	`Boolean`	`true` for modern mobile browsers (including iOS Safari and different Android browsers).
`mobileWebkit`	`Boolean`	`true` for mobile webkit-based browsers.
`mobileOpera`	`Boolean`	`true` for mobile Opera.
`touch`	`Boolean`	`true` for all browsers on touch devices.
`msTouch`	`Boolean`	`true` for browsers with Microsoft touch model (e.g. IE10).
`retina`	`Boolean`	`true` for devices with Retina screens.

Util

Various utility functions, used by Leaflet internally.

Methods

Method	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.
`bind``( <Function> `*`fn`*`, <Object> `*`obj`*` )`	`Function`	Returns a function which executes function `fn` with the given scope `obj` (so that `this` keyword refers to `obj` inside the function code). Has an `L.bind` shortcut.
`stamp``( <Object> `*`obj`*` )`	`String`	Applies a unique key to the object and returns that key. Has an `L.stamp` shortcut.
`limitExecByInterval``( <Function> `*`fn`*`, <Number> `*`time`*`, <Object> `*`context?`*` )`	`Function`	Returns a wrapper around the function `fn` that makes sure it's called not more often than a certain time interval `time`, but as fast as possible otherwise (for example, it is used for checking and requesting new tiles while dragging the map), optionally passing the scope (`context`) in which the function will be called.
`falseFn``()`	`Function`	Returns a function which always returns `false`.
`formatNum``( <Number> `*`num`*`,`	`Number`	Returns the number `num` rounded to `digits` decimals.

<Number> *digits*)		
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 (#class-options)**. Has an `L.setOptions` shortcut.
getParamString(<Object> *obj*)	String	Converts an object into a parameter URL string, e.g. `{a: "foo", b: "bar"}` translates to `'?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(<Object> *obj*)	Boolean	Returns `true` if the given object is an array.
trim(<String> *str*)	String	Trims the whitespace from both ends of the string and returns the result.

Properties

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.

```
var transformation = new 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

Creation	Description
new **L.Transformation**(<Number> *a*, <Number> *b*, <Number> *c*, <Number> *d*)	Creates a transformation object with the given coefficients.

Methods

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

LineUtil

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

Methods

Method	Returns	Description
simplify(<Point (#point)[]> *points*, <Number> *tolerance*)	Point (#point)[]	Dramatically reduces the number of points in a polyline while retaining its shape and returns a new array of simplified points. 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 (http://mourner.github.com/simplify-js/)**.
pointToSegmentDistance(<Point (#point)> *p*, <Point (#point)> *p1*, <Point (#point)> *p2*)	Number	Returns the distance between point `p` and segment `p1` to `p2`.
closestPointOnSegment(<Point (#point)> *p*, <Point (#point)> *p1*, <Point (#point)> *p2*)	Point (#point)	Returns the closest point from a point `p` on a segment `p1` to `p2`.
clipSegment(<Point (#point)> *a*, <Point (#point)> *b*, <Bounds (#bounds)> *bounds*)	-	Clips the segment `a` to `b` by rectangular bounds (modifying the segment points directly!). Used by Leaflet to only show polyline points that are on the screen or near, increasing performance.

PolyUtil

Various utility functions for polygon geometries.

Methods

Method	Returns	Description
clipPolygon(<Point (#point)[]> *points*, <Bounds (#bounds)> *bounds*)	Point (#point)[]	Clips the polygon geometry defined by the given points by rectangular bounds. 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 seperate method for it.

DomEvent

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

Methods

Method	Returns	Description
addListener(<HTMLElement> *el*, <String> *type*, <Function> *fn*, <Object> *context?*)	`this`	Adds a listener `fn` to the element's DOM event of the specified type. `this` keyword inside the listener will point to `context`, or to the element if not specified.
removeListener(<HTMLElement> *el*, <String> *type*, <Function> *fn*)	`this`	Removes an event listener from the element.
stopPropagation(<DOMEvent> *e*)	`this`	Stop the given event from propagation to parent elements. Used inside the listener functions: `L.DomEvent.addListener(div, 'click', function (e) {` `L.DomEvent.stopPropagation(e);` `});`
preventDefault(<DOMEvent> *e*)	`this`	Prevents the default action of the event from happening (such as following a link in the `href` of the `a` element, or doing a `POST` request with page reload when `form` is submitted). Use it inside listener functions.
stop(<DOMEvent> *e*)	`this`	Does `stopPropagation` and `preventDefault` at the same time.
disableClickPropagation(<HTMLElement> *el*)	`this`	Adds `stopPropagation` to the element's `'click'`, `'doubleclick'`, `'mousedown'` and `'touchstart'` events.
getMousePosition(<DOMEvent> *e*, <HTMLElement> *container?*)	Point (#point)	Gets normalized mouse position from a DOM event relative to the container or to the whole page if not specified.
getWheelDelta(<DOMEvent> *e*)	Number	Gets normalized wheel delta from a `mousewheel` DOM event.

DomUtil

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

Methods

Method	Returns	Description
get(<String or HTMLElement> *id*)	HTMLElement	Returns an element with the given id if a string was passed, or just returns the element if it was passed directly.
getStyle(<HTMLElement> *el*, <String> *style*)	String	Returns the value for a certain style attribute on an element, including computed values or values set through CSS.
getViewportOffset(<HTMLElement> *el*)	Point (#point)	Returns the offset to the viewport for the requested element.
create(<String> *tagName*, <String> *className*, <HTMLElement> *container?*)	HTMLElement	Creates an element with `tagName`, sets the `className`, and optionally appends it to `container` element.
disableTextSelection()	-	Makes sure text cannot be selected, for example during dragging.
enableTextSelection()	-	Makes text selection possible again.
hasClass(<HTMLElement> *el*, <String> *name*)	Boolean	Returns `true` if the element 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.
setOpacity(<HTMLElement> *el*, <Number> *value*)	-	Set the opacity of an element (including old IE support). Value must be from `0` to `1`.

testProp(<String[]> *props*)	String or 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`.
getTranslateString(<Point (#point)> *point*)	String	Returns a CSS transform string to move an element by the offset provided in the given point. Uses 3D translate on WebKit for hardware-accelerated transforms and 2D on other browsers.
getScaleString(<Number> *scale*, <Point (#point)> *origin*)	String	Returns a CSS transform string to scale an element (with the given scale origin).
setPosition(<HTMLElement> *el*, <Point (#point)> *point*, <Boolean> *disable3D?*)	-	Sets the position of an element to coordinates specified by `point`, using CSS translate or top/left positioning depending on the browser (used by Leaflet internally to position its layers). Forces top/left positioning if `disable3D` is `true`.
getPosition(<HTMLElement> *el*)	Point (#point)	Returns the coordinates of an element previously positioned with `setPosition`.

Properties

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

PosAnimation

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

```
var fx = new L.PosAnimation();
fx.run(el, [300, 500], 0.5);
```

Creation

Creation	Description
new **L.PosAnimation**()	Creates a PosAnimation object.

Methods

Method	Returns	Description
run(<HTMLElement> *element*, <Point (#point)> *newPos*, <Number> *duration?*, <Number> *easeLinearity?*)	this	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 (http://cubic-bezier.com/#0,0,.5,1)**, `0.5` by default)

Events

You can subscribe to the following events using **these methods (#events)**.

Event	Data	Description
start	Event (#event)	Fired when the animation starts.
step	Event (#event)	Fired continuously during the animation.
end	Event (#event)	Fired when the animation ends.

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 **DomUtil#setPosition (#domutil-setposition)**

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

Creation

Creation	Description
new **L.Draggable**(<HTMLElement> *element*, <HTMLElement> *dragHandle?*)	Creates a Draggable object for moving the given element when you start dragging the `dragHandle` element (equals the element itself by default).

Events

You can subscribe to the following events using **these methods (#events)**.

Event	Data	Description
dragstart	Event (#event)	Fired when the dragging starts.
predrag	Event (#event)	Fired continuously during dragging *before* each corresponding update of the element position.
drag	Event (#event)	Fired continuously during dragging.
dragend	Event (#event)	Fired when the dragging ends.

Methods

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

IHandler

An interface implemented by **interaction handlers (#map-interaction-handlers)**.

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

ILayer

Represents an object attached to a particular location (or a set of locations) on a map. Implemented by **tile layers (#tilelayer)**, **markers (#marker)**, **popups (#popup)**, **image overlays (#imageoverlay)**, **vector layers (#path)** and **layer groups (#layergroup)**.

Methods

Method	Returns	Description
onAdd(<Map (#map)> *map*)	-	Should contain code that creates DOM elements for the overlay, adds them to **map panes (#map-panes)** where they should belong and puts listeners on relevant map events. Called on `map.addLayer(layer)`.
onRemove(<Map (#map)> *map*)	-	Should contain all clean up code that removes the overlay's elements from the DOM and removes listeners previously added in `onAdd`. Called on `map.removeLayer(layer)`.

Implementing Custom Layers

The most important things know about when implementing custom layers are Map **viewreset (#map-viewreset)** event and **latLngToLayerPoint (#map-latlngtolayerpoint)** method. `viewreset` is fired when the map needs to reposition its layers (e.g. on zoom), and `latLngToLayerPoint` is used to get coordinates for the layer's new position.

Another event often used in layer implementations is **moveend (#map-moveend)** which fires after any movement of the map (panning, zooming, etc.).

Another thing to note is that you'll usually need to add `leaflet-zoom-hide` class to the DOM elements you create for the layer so that it hides during zoom animation. Implementing zoom animation for custom layers is a complex topic and will be documented separately in future, but meanwhile you can take a look at how it's done for Leaflet layers (e.g. `ImageOverlay`) in the source.

Custom Layer Example

Here's how a custom layer implementation usually looks:

```
var MyCustomLayer = L.Class.extend({

    initialize: function (latlng) {
        // save position of the layer or any options from the constructor
        this._latlng = latlng;
    },

    onAdd: function (map) {
        this._map = map;

        // create a DOM element and put it into one of the map panes
        this._el = L.DomUtil.create('div', 'my-custom-layer leaflet-zoom-hide');
        map.getPanes().overlayPane.appendChild(this._el);

        // add a viewreset event listener for updating layer's position, do the latter
        map.on('viewreset', this._reset, this);
        this._reset();
    },

    onRemove: function (map) {
        // remove layer's DOM elements and listeners
        map.getPanes().overlayPane.removeChild(this._el);
        map.off('viewreset', this._reset, this);
    },

    _reset: function () {
        // update layer's position
        var pos = this._map.latLngToLayerPoint(this._latlng);
        L.DomUtil.setPosition(this._el, pos);
    }
});

map.addLayer(new MyCustomLayer(latlng));
```

IControl

Represents a UI element in one of the corners of the map. Implemented by **zoom (#control-zoom)**, **attribution (#control-attribution)**, **scale (#control-scale)** and **layers (#control-layers)** controls.

Methods

Every control in Leaflet should extend from **Control (#control)** class and additionally have the following methods:

Method	Returns	Description
onAdd(<Map (#map)> *map*)	`HTMLElement`	Should contain code that creates all the neccessary DOM elements for the control, adds listeners on relevant map events, and returns the element containing the control. Called on `map.addControl(control)` or `control.addTo(map)`.
onRemove(<Map (#map)> *map*)	-	Optional, should contain all clean up code (e.g. removes control's event listeners). Called on `map.removeControl(control)` or `control.removeFrom(map)`. The control's DOM container is removed automatically.

Custom Control Example

```
var MyControl = L.Control.extend({
    options: {
        position: 'topright'
    },

    onAdd: function (map) {
        // create the control container with a particular class name
        var container = L.DomUtil.create('div', 'my-custom-control');

        // ... initialize other DOM elements, add listeners, etc.

        return container;
    }
});

map.addControl(new MyControl());
```

If specify your own constructor for the control, you'll also probably want to process options properly:

```
var MyControl = L.Control.extend({
    initialize: function (foo, options) {
        // ...
        L.Util.setOptions(this, options);
    },
    // ...
});
```

This will allow you to pass options like `position` when creating the control instances:

```
map.addControl(new MyControl('bar', {position: 'bottomleft'}));
```

IProjection

An object with methods for projecting geographical coordinates of the world onto a flat surface (and back). See **Map projection (http://en.wikipedia.org/wiki/Map_projection)**.

Methods

Method	Returns	Description
project(<LatLng (#latlng)> *latlng*)	Point (#point)	Projects geographical coordinates into a 2D point.
unproject(<Point (#point)> *point*)	LatLng (#latlng)	The inverse of `project`. Projects a 2D point into geographical location.

Defined Projections

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

Projection	Description
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.
L.Projection.Mercator	Elliptical Mercator projection — more complex than Spherical Mercator. Takes into account that Earth is a geoid, not a perfect sphere. Used by the `EPSG:3395` CRS.
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:3395` and `Simple` CRS.

ICRS

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 (http://en.wikipedia.org/wiki/Coordinate_reference_system)**.

Methods

Method	Returns	Description
latLngToPoint(<LatLng (#latlng)> *latlng*, <Number> *zoom*)	Point (#point)	Projects geographical coordinates on a given zoom into pixel coordinates.
pointToLatLng(<Point (#point)> *point*, <Number> *zoom*)	LatLng (#latlng)	The inverse of `latLngToPoint`. Projects pixel coordinates on a given zoom into geographical coordinates.
project(<LatLng (#latlng)> *latlng*)	Point (#point)	Projects geographical coordinates into coordinates in units accepted for this CRS (e.g. meters for `EPSG:3857`, for passing it to WMS services).
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.
getSize(<Number> *zoom*)	Point (#point)	Returns the size of the world in pixels for a particular zoom.

Properties

Property	Type	Description
projection	IProjection (#iprojection)	Projection that this CRS uses.
transformation	Transformation (#transformation)	Transformation that this CRS uses to turn projected coordinates into screen coordinates for a particular tile service.
code	String	Standard code name of the CRS passed into WMS services (e.g. `'EPSG:3857'`).

Defined CRS

Leaflet comes with a set of already defined CRS to use out of the box:

Projection	Description
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.
L.CRS.EPSG3395	Rarely used by some commercial tile providers. Uses Elliptical Mercator projection.
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).

If you want to use some obscure CRS not listed here, take a look at the **Proj4Leaflet (https://github.com/kartena/Proj4Leaflet)** plugin.

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

Switch	Description
L_PREFER_CANVAS	Forces Leaflet to use the Canvas back-end (if available) for vector layers instead of SVG. This can increase performance considerably in some cases (e.g. many thousands of circle markers on the map).
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:

```
// L points to some other library
...
// you include Leaflet, it replaces the L variable to Leaflet namespace

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

version

A constant that represents the Leaflet version in use.

```
L.version // returns "0.5" (or whatever version is currently in use)
```

(http://github.com/Leaflet/Leaflet)