```
<code>
import tensorflow as tf
class ModuleLevel2(tf.Module):

    def __init__(self):
        self.state =
tf.TensorArray(size=0,
dynamic_size=True,
clear_after_read=False,
dtype=tf.float32)

    def __call__(self, x):
        self.state =
self.state.write(self.state.
size(), x)
        x = self.state.stack()
        x = tf.identity(x) #
Some op that requires all saved
inputs
        tf.print(x)

class ModuleLevel1(tf.Module):

    def __init__(self):
        self.module2 =
ModuleLevel2()

    def __call__(self, x):
        self.module2(x)

class Model(tf.Module):

    def __init__(self):
        self.module1 =
ModuleLevel1()

    def __call__(self, x):
        self.module1(x)

model = Model()

@tf.function
def search():
    for i in range(10):
        x =
(i*5)+tf.constant([1,2,3,4,5],
dtype=tf.float32)
        model(x)

search()
```

</code>

The snippet generates the error seen below in "Relevant log output".

If search is not a tf.function the snippet works which is in my opinion undesirable behavior.

What is possible is to create the state outside of the model and pass it as an argument but as seen in the below snippet I have to modify all arguments and return values of modules on an upper abstraction level:

<code>
```
import tensorflow as tf

class ModuleLevel2(tf.Module):

    def __init__(self):
        pass

    def __call__(self, x,
state):
        state =
state.write(state.size(), x)
        x = state.stack()
        x = tf.identity(x) #
Some op that requires all saved
inputs
        tf.print(x)
        return state

class ModuleLevel1(tf.Module):

    def __init__(self):
        self.module2 =
ModuleLevel2()

    def __call__(self, x,
state):
        state = self.module2(x,
state)
        return state

class Model(tf.Module):
```

```
    def __init__(self):
        self.module1 =
ModuleLevel1()

    def __call__(self, x,
state):
        state = self.module1(x,
state)
        return state

model = Model()

@tf.function
def search():
    state =
tf.TensorArray(size=0,
dynamic_size=True,
clear_after_read=False,
dtype=tf.float32)
    for i in range(10):
        x =
(i*5)+tf.constant([1,2,3,4,5],
dtype=tf.float32)
        state = model(x, state)

search()
</code>



unction check() {

expect(layer.getCenter()).to.
be.nearLatLng([50.
49998323576035,
30.50989603626345]);
                i++;
                if (i < 30) {
map.setZoom(i); }
}
map.on('zoomend', check);
```

```
@tf.function

def softmax_tffunction(x):

    return tf.nn.softmax(x)



if __name__ == "__main__":


softmax_tffunction(tf.random.
normal([10, 10]))
```

```
/* @preserve
 * Leaflet 1.9.1, a JS library
for interactive maps.
https://leafletjs.com
 * (c) 2010-2022 Vladimir
Agafonkin, (c) 2010-2011
CloudMade
 */

(function (global, factory) {
  typeof exports === 'object'
&& typeof module !==
'undefined' ? factory(exports)
:
  typeof define === 'function'
&& define.amd ?
define(['exports'], factory) :
  (global = typeof globalThis
!== 'undefined' ? globalThis :
global || self,
factory(global.leaflet = {}));
})(this, (function (exports) {
'use strict';

  var version = "1.9.1";

  /*
   * @namespace Util
   *
   * Various utility
functions, used by Leaflet
internally.
   */
```

```
  // @function extend(dest:
Object, src?: Object): Object
  // Merges the properties of
the `src` object (or multiple
objects) into `dest` object
and returns the latter. Has an
`L.extend` shortcut.
  function extend(dest) {
        var i, j, len, src;

        for (j = 1, len =
arguments.length; j < len;
j++) {
                src =
arguments[j];
                for (i in src)
{

dest[i] = src[i];
                }
        }
        return dest;
  }

  // @function create(proto:
Object, properties?: Object):
Object
  // Compatibility polyfill
for [Object.create]
(https://developer.mozilla.org
/docs/Web/JavaScript/Reference
/Global_Objects/Object/create)
  var create$2 = Object.create
|| (function () {
        function F() {}
        return function
(proto) {
                F.prototype =
proto;
                return new
F();
        };
  })();

  // @function bind(fn:
Function, …): Function
  // Returns a new function
```

```
bound to the arguments passed,
like [Function.prototype.bind]
(https://developer.mozilla.org
/docs/Web/JavaScript/Reference
/Global_Objects/Function/bind)
.
  // Has a `L.bind()`
shortcut.
  function bind(fn, obj) {
        var slice =
Array.prototype.slice;

        if (fn.bind) {
                return
fn.bind.apply(fn,
slice.call(arguments, 1));
        }

        var args =
slice.call(arguments, 2);

        return function () {
                return
fn.apply(obj, args.length ?
args.concat(slice.call(argumen
ts)) : arguments);
        };
  }

  // @property lastId: Number
  // Last unique ID used by
[`stamp()`](#util-stamp)
  var lastId = 0;

  // @function stamp(obj:
Object): Number
  // Returns the unique ID of
an object, assigning it one if
it doesn't have it.
  function stamp(obj) {
        if (!('_leaflet_id' in
obj)) {

obj['_leaflet_id'] = ++lastId;
        }
        return
obj._leaflet_id;
```

```
  }

  // @function throttle(fn:
Function, time: Number,
context: Object): Function
  // Returns a function which
executes function `fn` with
the given scope `context`
  // (so that the `this`
keyword refers to `context`
inside `fn`'s code). The
function
  // `fn` will be called no
more than one time per given
amount of `time`. The
arguments
  // received by the bound
function will be any arguments
passed when binding the
  // function, followed by any
arguments passed when invoking
the bound function.
  // Has an `L.throttle`
shortcut.
  function throttle(fn, time,
context) {
        var lock, args,
wrapperFn, later;

        later = function () {
                // reset lock
and call if queued
                lock = false;
                if (args) {

wrapperFn.apply(context,
args);
                        args =
false;
                }
        };

        wrapperFn = function
() {
                if (lock) {
                        //
called too soon, queue to call
```

```
later
                        args =
arguments;

                } else {
                        //
call and lock until later

fn.apply(context, arguments);

setTimeout(later, time);
                        lock =
true;
                }
        };

        return wrapperFn;
  }

  // @function wrapNum(num:
Number, range: Number[],
includeMax?: Boolean): Number
  // Returns the number `num`
modulo `range` in such a way
so it lies within
  // `range[0]` and
`range[1]`. The returned value
will be always smaller than
  // `range[1]` unless
`includeMax` is set to `true`.
  function wrapNum(x, range,
includeMax) {
        var max = range[1],
            min = range[0],
            d = max - min;
        return x === max &&
includeMax ? x : ((x - min) %
d + d) % d + min;
  }

  // @function falseFn():
Function
  // Returns a function which
always returns `false`.
  function falseFn() { return
false; }
```

```
  // @function formatNum(num:
Number, precision?:
Number|false): Number
  // Returns the number `num`
rounded with specified
`precision`.
  // The default `precision`
value is 6 decimal places.
  // `false` can be passed to
skip any processing (can be
useful to avoid round-off
errors).
  function formatNum(num,
precision) {
        if (precision ===
false) { return num; }
        var pow = Math.pow(10,
precision === undefined ? 6 :
precision);
        return Math.round(num
* pow) / pow;
  }

  // @function trim(str:
String): String
  // Compatibility polyfill
for [String.prototype.trim]
(https://developer.mozilla.org
/docs/Web/JavaScript/Reference
/Global_Objects/String/Trim)
  function trim(str) {
        return str.trim ?
str.trim() :
str.replace(/^\s+|\s+$/g, '');
  }

  // @function splitWords(str:
String): String[]
  // Trims and splits the
string on whitespace and
returns the array of parts.
  function splitWords(str) {
        return
trim(str).split(/\s+/);
  }

  // @function setOptions(obj:
```

```
Object, options: Object):
Object
  // Merges the given
properties to the `options` of
the `obj` object, returning
the resulting options. See
`Class options`. Has an
`L.setOptions` shortcut.
  function setOptions(obj,
options) {
        if
(!Object.prototype.hasOwnPrope
rty.call(obj, 'options')) {
                obj.options =
obj.options ?
create$2(obj.options) : {};
        }
        for (var i in options)
{
                obj.options[i]
= options[i];
        }
        return obj.options;
  }

  // @function
getParamString(obj: Object,
existingUrl?: String,
uppercase?: Boolean): String
  // Converts an object into a
parameter URL string, e.g.
`{a: "foo", b: "bar"}`
  // translates to `'?
a=foo&b=bar'`. If
`existingUrl` is set, the
parameters will
  // be appended at the end.
If `uppercase` is `true`, the
parameter names will
  // be uppercased (e.g. `'?
A=foo&B=bar'`)
  function getParamString(obj,
existingUrl, uppercase) {
        var params = [];
        for (var i in obj) {

params.push(encodeURIComponent
```

```
(uppercase ? i.toUpperCase() :
i) + '=' +
encodeURIComponent(obj[i]));
        }
        return ((!existingUrl
|| existingUrl.indexOf('?')
=== -1) ? '?' : '&') +
params.join('&');
  }

  var templateRe = /\{ *([\w_
-]+) *\}/g;

  // @function template(str:
String, data: Object): 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.
  function template(str, data)
{
        return
str.replace(templateRe,
function (str, key) {
                var value =
data[key];

                if (value ===
undefined) {
                        throw
new Error('No value provided
for variable ' + str);

                } else if
(typeof value === 'function')
{
                        value
= value(data);
```

```
                }
                return value;
        });
  }

  // @function isArray(obj):
Boolean
  // Compatibility polyfill
for [Array.isArray]
(https://developer.mozilla.org
/docs/Web/JavaScript/Reference
/Global_Objects/Array/isArray)
  var isArray = Array.isArray
|| function (obj) {
        return
(Object.prototype.toString.cal
l(obj) === '[object Array]');
  };

  // @function indexOf(array:
Array, el: Object): Number
  // Compatibility polyfill
for [Array.prototype.indexOf]
(https://developer.mozilla.org
/docs/Web/JavaScript/Reference
/Global_Objects/Array/indexOf)
  function indexOf(array, el)
{
        for (var i = 0; i <
array.length; i++) {
                if (array[i]
=== el) { return i; }
        }
        return -1;
  }

  // @property 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).
```

```
  var emptyImageUrl =
'
hAQABAAD/ACwAAAAAQABAAACADs='
;

  // inspired by
https://paulirish.com/2011/req
uestanimationframe-for-smart-
animating/

  function getPrefixed(name) {
        return window['webkit'
+ name] || window['moz' +
name] || window['ms' + name];
  }

  var lastTime = 0;

  // fallback for IE 7-8
  function timeoutDefer(fn) {
        var time = +new
Date(),
            timeToCall =
Math.max(0, 16 - (time -
lastTime));

        lastTime = time +
timeToCall;
        return
window.setTimeout(fn,
timeToCall);
  }

  var requestFn =
window.requestAnimationFrame
||
getPrefixed('RequestAnimationF
rame') || timeoutDefer;
  var cancelFn =
window.cancelAnimationFrame ||
getPrefixed('CancelAnimationFr
ame') ||

getPrefixed('CancelRequestAnim
ationFrame') || function (id)
{ window.clearTimeout(id); };
```

```
  // @function
requestAnimFrame(fn: Function,
context?: Object, immediate?:
Boolean): Number
  // Schedules `fn` to be
executed when the browser
repaints. `fn` is bound to
  // `context` if given. When
`immediate` is set, `fn` is
called immediately if
  // the browser doesn't have
native support for
  //
[`window.requestAnimationFrame
`]
(https://developer.mozilla.org
/docs/Web/API/window/requestAn
imationFrame),
  // otherwise it's delayed.
Returns a request ID that can
be used to cancel the request.
  function
requestAnimFrame(fn, context,
immediate) {
        if (immediate &&
requestFn === timeoutDefer) {

fn.call(context);
        } else {
                return
requestFn.call(window,
bind(fn, context));
        }
  }

  // @function
cancelAnimFrame(id: Number):
undefined
  // Cancels a previous
`requestAnimFrame`. See also
[window.cancelAnimationFrame]
(https://developer.mozilla.org
/docs/Web/API/window/cancelAni
mationFrame).
  function cancelAnimFrame(id)
{
        if (id) {
```

```
cancelFn.call(window, id);
        }
  }

  var Util = {
    __proto__: null,
    extend: extend,
    create: create$2,
    bind: bind,
    get lastId () { return
lastId; },
    stamp: stamp,
    throttle: throttle,
    wrapNum: wrapNum,
    falseFn: falseFn,
    formatNum: formatNum,
    trim: trim,
    splitWords: splitWords,
    setOptions: setOptions,
    getParamString:
getParamString,
    template: template,
    isArray: isArray,
    indexOf: indexOf,
    emptyImageUrl:
emptyImageUrl,
    requestFn: requestFn,
    cancelFn: cancelFn,
    requestAnimFrame:
requestAnimFrame,
    cancelAnimFrame:
cancelAnimFrame
  };

  // @class Class
  // @aka L.Class

  // @section
  // @uninheritable

  // Thanks to John Resig and
Dean Edwards for inspiration!

  function Class() {}

  Class.extend = function
```

```
(props) {

        // @function
extend(props: Object):
Function
        // [Extends the
current class](#class-
inheritance) given the
properties to be included.
        // Returns a
Javascript function that is a
class constructor (to be
called with `new`).
        var NewClass =
function () {


setOptions(this);

                // call the
constructor
                if
(this.initialize) {

this.initialize.apply(this,
arguments);
                }

                // call all
constructor hooks

this.callInitHooks();
        };

        var parentProto =
NewClass.__super__ =
this.prototype;

        var proto =
create$2(parentProto);
        proto.constructor =
NewClass;

        NewClass.prototype =
proto;

        // inherit parent's
```

```
statics
        for (var i in this) {
                if
(Object.prototype.hasOwnProper
ty.call(this, i) && i !==
'prototype' && i !==
'__super__') {

NewClass[i] = this[i];
                }
        }

        // mix static
properties into the class
        if (props.statics) {

extend(NewClass,
props.statics);
        }

        // mix includes into
the prototype
        if (props.includes) {

checkDeprecatedMixinEvents(pro
ps.includes);

extend.apply(null,
[proto].concat(props.includes)
);
        }

        // mix given
properties into the prototype
        extend(proto, props);
        delete proto.statics;
        delete proto.includes;

        // merge options
        if (proto.options) {
                proto.options
= parentProto.options ?
create$2(parentProto.options)
: {};

extend(proto.options,
props.options);
```

```
        }

        proto._initHooks = [];

        // add method for
calling all hooks
        proto.callInitHooks =
function () {

                if
(this._initHooksCalled) {
return; }

                if
(parentProto.callInitHooks) {

parentProto.callInitHooks.call
(this);
                }


this._initHooksCalled = true;

                for (var i =
0, len =
proto._initHooks.length; i <
len; i++) {

proto._initHooks[i].call(this)
;
                }
        };

        return NewClass;
  };


  // @function
include(properties: Object):
this
  // [Includes a mixin]
(#class-includes) into the
current class.
  Class.include = function
(props) {
        var parentOptions =
this.prototype.options;
```

```
        extend(this.prototype,
props);
        if (props.options) {

this.prototype.options =
parentOptions;

this.mergeOptions(props.option
s);
        }
        return this;
  };

  // @function
mergeOptions(options: Object):
this
  // [Merges `options`]
(#class-options) into the
defaults of the class.
  Class.mergeOptions =
function (options) {

extend(this.prototype.options,
options);
        return this;
  };

  // @function addInitHook(fn:
Function): this
  // Adds a [constructor hook]
(#class-constructor-hooks) to
the class.
  Class.addInitHook = function
(fn) { // (Function) ||
(String, args...)
        var args =
Array.prototype.slice.call(arg
uments, 1);

        var init = typeof fn
=== 'function' ? fn : function
() {

this[fn].apply(this, args);
        };
```

```
this.prototype._initHooks =
this.prototype._initHooks ||
[];

this.prototype._initHooks.push
(init);
        return this;
  };

  function
checkDeprecatedMixinEvents(inc
ludes) {
        if (typeof L ===
'undefined' || !L || !L.Mixin)
{ return; }

        includes =
isArray(includes) ? includes :
[includes];

        for (var i = 0; i <
includes.length; i++) {
                if
(includes[i] ===
L.Mixin.Events) {

console.warn('Deprecated
include of L.Mixin.Events: ' +

'this property will be removed
in future releases, ' +

'please inherit from L.Evented
instead.', new Error().stack);
                }
        }
  }

  /*
   * @class Evented
   * @aka L.Evented
   * @inherits Class
   *
   * A set of methods shared
between event-powered classes
(like `Map` and `Marker`).
Generally, events allow you to
```

```
execute some function when
something happens with an
object (e.g. the user clicks
on the map, causing the map to
fire `'click'` event).
   *
   * @example
   *
   * ```js
   * 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:
   *
   * ```js
   * function onClick(e) { ...
}
   *
   * map.on('click', onClick);
   * map.off('click',
onClick);
   * ```
   */

  var Events = {
        /* @method on(type:
String, fn: Function,
context?: Object): 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'`).
         *
         * @alternative
```

```
         * @method
on(eventMap: Object): this
         * Adds a set of
type/listener pairs, e.g.
`{click: onClick, mousemove:
onMouseMove}`
         */
        on: function (types,
fn, context) {

                // types can
be a map of types/handlers
                if (typeof
types === 'object') {
                        for
(var type in types) {

// we don't process space-
separated events here for
performance;

// it's a hot path since Layer
uses the on(obj) syntax

this._on(type, types[type],
fn);
                        }

                } else {
                        //
types can be a string of
space-separated words
                        types
= splitWords(types);

                        for
(var i = 0, len =
types.length; i < len; i++) {

this._on(types[i], fn,
context);
                        }
                }

                return this;
        },
```

```
        /* @method off(type:
String, fn?: Function,
context?: Object): this
         * Removes a
previously added listener
function. If no function is
specified, it will remove all
the listeners of that
particular event from the
object. Note that if you
passed a custom context to
`on`, you must pass the same
context to `off` in order to
remove the listener.
         *
         * @alternative
         * @method
off(eventMap: Object): this
         * Removes a set of
type/listener pairs.
         *
         * @alternative
         * @method off: this
         * Removes all
listeners to all events on the
object. This includes
implicitly attached events.
         */
        off: function (types,
fn, context) {

                if
(!arguments.length) {
                        //
clear all listeners if called
without arguments
                        delete
this._events;

                } else if
(typeof types === 'object') {
                        for
(var type in types) {

this._off(type, types[type],
fn);
                        }
```

```
                } else {
                        types
= splitWords(types);

                        var
removeAll = arguments.length
=== 1;
                        for
(var i = 0, len =
types.length; i < len; i++) {

if (removeAll) {

this._off(types[i]);

} else {

this._off(types[i], fn,
context);

}
                        }
                }

                return this;
        },

        // attach listener
(without syntactic sugar now)
        _on: function (type,
fn, context, _once) {
                if (typeof fn
!== 'function') {

console.warn('wrong listener
type: ' + typeof fn);

return;
                }

                // check if fn
already there
                if
(this._listens(type, fn,
context) !== false) {
```

```
return;
                }

                if (context
=== this) {
                        //
Less memory footprint.

context = undefined;
                }

                var
newListener = {fn: fn, ctx:
context};
                if (_once) {

newListener.once = true;
                }

                this._events =
this._events || {};

this._events[type] =
this._events[type] || [];

this._events[type].push(newLis
tener);
        },

        _off: function (type,
fn, context) {
                var listeners,
                    i,
                    len;

                if
(!this._events) {

return;
                }

                listeners =
this._events[type];
                if
(!listeners) {

return;
```

```
                }

                if
(arguments.length === 1) { //
remove all
                        if
(this._firingCount) {

// Set all removed listeners
to noop

// so they are not called if
remove happens in fire

for (i = 0, len =
listeners.length; i < len;
i++) {

listeners[i].fn = falseFn;

}
                        }
                        //
clear all listeners for a type
if function isn't specified
                        delete
this._events[type];

return;
                }

                if (typeof fn
!== 'function') {

console.warn('wrong listener
type: ' + typeof fn);

return;
                }

                // find fn and
remove it
                var index =
this._listens(type, fn,
context);
                if (index !==
false) {
```

```
                        var
listener = listeners[index];
                        if
(this._firingCount) {

// set the removed listener to
noop so that's not called if
remove happens in fire

listener.fn = falseFn;


/* copy array in case events
are being fired */

this._events[type] = listeners
= listeners.slice();
                        }

listeners.splice(index, 1);
                }
        },

        // @method fire(type:
String, data?: Object,
propagate?: Boolean): this
        // Fires an event of
the specified type. You can
optionally provide a data
        // object — the first
argument of the listener
function will contain its
        // properties. The
event can optionally be
propagated to event parents.
        fire: function (type,
data, propagate) {
                if
(!this.listens(type,
propagate)) { return this; }

                var event =
extend({}, data, {
                        type:
type,

target: this,
```

```
sourceTarget: data &&
data.sourceTarget || this
                });

                if
(this._events) {
                        var
listeners =
this._events[type];
                        if
(listeners) {

this._firingCount =
(this._firingCount + 1) || 1;

for (var i = 0, len =
listeners.length; i < len;
i++) {

var l = listeners[i];

// off overwrites l.fn, so we
need to copy fn to a var

var fn = l.fn;

if (l.once) {

this.off(type, fn, l.ctx);

}

fn.call(l.ctx || this, event);

}


this._firingCount--;
                        }
                }

                if (propagate)
{
                        //
propagate the event to parents
(set with addEventParent)
```

```
this._propagateEvent(event);
                }

                return this;
        },

        // @method
listens(type: String,
propagate?: Boolean): Boolean
        // @method
listens(type: String, fn:
Function, context?: Object,
propagate?: Boolean): Boolean
        // Returns `true` if a
particular event type has any
listeners attached to it.
        // The verification
can optionally be propagated,
it will return `true` if
parents have the listener
attached to it.
        listens: function
(type, fn, context, propagate)
{
                if (typeof
type !== 'string') {

console.warn('"string" type
argument expected');
                }

                // we don't
overwrite the input `fn`
value, because we need to use
it for propagation
                var _fn = fn;
                if (typeof fn
!== 'function') {

propagate = !!fn;
                        _fn =
undefined;

context = undefined;
                }
```

```
                var listeners
= this._events &&
this._events[type];
                if (listeners
&& listeners.length) {
                        if
(this._listens(type, _fn,
context) !== false) {

return true;
                        }
                }

                if (propagate)
{
                        //
also check parents for
listeners if event propagates
                        for
(var id in this._eventParents)
{

if
(this._eventParents[id].listen
s(type, fn, context,
propagate)) { return true; }
                        }
                }
                return false;
        },

        // returns the index
(number) or false
        _listens: function
(type, fn, context) {
                if
(!this._events) {
                        return
false;
                }

                var listeners
= this._events[type] || [];
                if (!fn) {
                        return
!!listeners.length;
                }
```

```
                if (context
=== this) {
                        //
Less memory footprint.

context = undefined;
                }

                for (var i =
0, len = listeners.length; i <
len; i++) {
                        if
(listeners[i].fn === fn &&
listeners[i].ctx === context)
{

return i;
                        }
                }
                return false;

        },

        // @method once(…):
this
        // Behaves as
[`on(…)`](#evented-on), except
the listener will only get
fired once and then removed.
        once: function (types,
fn, context) {

                // types can
be a map of types/handlers
                if (typeof
types === 'object') {
                        for
(var type in types) {

// we don't process space-
separated events here for
performance;

// it's a hot path since Layer
uses the on(obj) syntax
```

```
this._on(type, types[type],
fn, true);
                        }

                } else {
                        //
types can be a string of
space-separated words
                        types
= splitWords(types);

                        for
(var i = 0, len =
types.length; i < len; i++) {

this._on(types[i], fn,
context, true);
                        }
                }

                return this;
        },

        // @method
addEventParent(obj: Evented):
this
        // Adds an event
parent - an `Evented` that
will receive propagated events
        addEventParent:
function (obj) {

this._eventParents =
this._eventParents || {};

this._eventParents[stamp(obj)]
= obj;
                return this;
        },

        // @method
removeEventParent(obj:
Evented): this
        // Removes an event
parent, so it will stop
receiving propagated events
        removeEventParent:
```

```
function (obj) {
                if
(this._eventParents) {
                        delete
this._eventParents[stamp(obj)]
;
                }
                return this;
        },

        _propagateEvent:
function (e) {
                for (var id in
this._eventParents) {

this._eventParents[id].fire(e.
type, extend({

layer: e.target,

propagatedFrom: e.target
                        }, e),
true);
                }
        }
  };

  // aliases; we should ditch
those eventually

  // @method
addEventListener(…): this
  // Alias to [`on(…)`]
(#evented-on)
  Events.addEventListener =
Events.on;

  // @method
removeEventListener(…): this
  // Alias to [`off(…)`]
(#evented-off)

  // @method
clearAllEventListeners(…):
this
  // Alias to [`off()`]
(#evented-off)
```

```
  Events.removeEventListener =
Events.clearAllEventListeners
= Events.off;

  // @method
addOneTimeEventListener(…):
this
  // Alias to [`once(…)`]
(#evented-once)

Events.addOneTimeEventListener
= Events.once;

  // @method fireEvent(…):
this
  // Alias to [`fire(…)`]
(#evented-fire)
  Events.fireEvent =
Events.fire;

  // @method
hasEventListeners(…): Boolean
  // Alias to [`listens(…)`]
(#evented-listens)
  Events.hasEventListeners =
Events.listens;

  var Evented =
Class.extend(Events);

  /*
   * @class Point
   * @aka L.Point
   *
   * Represents a point with
`x` and `y` coordinates in
pixels.
   *
   * @example
   *
   * ```js
   * 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:
   *
   * ```js
   * map.panBy([200, 300]);
   * map.panBy(L.point(200,
300));
   * ```
   *
   * Note that `Point` does
not inherit from Leaflet's
`Class` object,
   * which means new classes
can't inherit from it, and new
methods
   * can't be added to it with
the `include` function.
   */

  function Point(x, y, round)
{
        // @property x:
Number; The `x` coordinate of
the point
        this.x = (round ?
Math.round(x) : x);
        // @property y:
Number; The `y` coordinate of
the point
        this.y = (round ?
Math.round(y) : y);
  }

  var trunc = Math.trunc ||
function (v) {
        return v > 0 ?
Math.floor(v) : Math.ceil(v);
  };

  Point.prototype = {

        // @method clone():
Point
        // Returns a copy of
the current point.
```

```
        clone: function () {
                return new
Point(this.x, this.y);
        },

        // @method
add(otherPoint: Point): Point
        // Returns the result
of addition of the current and
the given points.
        add: function (point)
{
                // non-
destructive, returns a new
point
                return
this.clone()._add(toPoint(poin
t));
        },

        _add: function (point)
{
                //
destructive, used directly for
performance in situations
where it's safe to modify
existing point
                this.x +=
point.x;
                this.y +=
point.y;
                return this;
        },

        // @method
subtract(otherPoint: Point):
Point
        // Returns the result
of subtraction of the given
point from the current.
        subtract: function
(point) {
                return
this.clone()._subtract(toPoint
(point));
        },
```

```
        _subtract: function
(point) {
                this.x -=
point.x;
                this.y -=
point.y;
                return this;
        },

        // @method
divideBy(num: Number): Point
        // Returns the result
of division of the current
point by the given number.
        divideBy: function
(num) {
                return
this.clone()._divideBy(num);
        },

        _divideBy: function
(num) {
                this.x /= num;
                this.y /= num;
                return this;
        },

        // @method
multiplyBy(num: Number): Point
        // Returns the result
of multiplication of the
current point by the given
number.
        multiplyBy: function
(num) {
                return
this.clone()._multiplyBy(num);
        },

        _multiplyBy: function
(num) {
                this.x *= num;
                this.y *= num;
                return this;
        },

        // @method
```

```
scaleBy(scale: Point): Point
        // Multiply each
coordinate of the current
point by each coordinate of
        // `scale`. In linear
algebra terms, multiply the
point by the
        // [scaling matrix]
(https://en.wikipedia.org/wiki
/Scaling_%28geometry%29#Matrix
_representation)
        // defined by `scale`.
        scaleBy: function
(point) {
                return new
Point(this.x * point.x, this.y
* point.y);
        },

        // @method
unscaleBy(scale: Point): Point
        // Inverse of
`scaleBy`. Divide each
coordinate of the current
point by
        // each coordinate of
`scale`.
        unscaleBy: function
(point) {
                return new
Point(this.x / point.x, this.y
/ point.y);
        },

        // @method round():
Point
        // Returns a copy of
the current point with rounded
coordinates.
        round: function () {
                return
this.clone()._round();
        },

        _round: function () {
                this.x =
Math.round(this.x);
```

```
                this.y =
Math.round(this.y);
                return this;
        },

        // @method floor():
Point
        // Returns a copy of
the current point with floored
coordinates (rounded down).
        floor: function () {
                return
this.clone()._floor();
        },

        _floor: function () {
                this.x =
Math.floor(this.x);
                this.y =
Math.floor(this.y);
                return this;
        },

        // @method ceil():
Point
        // Returns a copy of
the current point with ceiled
coordinates (rounded up).
        ceil: function () {
                return
this.clone()._ceil();
        },

        _ceil: function () {
                this.x =
Math.ceil(this.x);
                this.y =
Math.ceil(this.y);
                return this;
        },

        // @method trunc():
Point
        // Returns a copy of
the current point with
truncated coordinates (rounded
towards zero).
```

```
        trunc: function () {
                return
this.clone()._trunc();
        },

        _trunc: function () {
                this.x =
trunc(this.x);
                this.y =
trunc(this.y);
                return this;
        },

        // @method
distanceTo(otherPoint: Point):
Number
        // Returns the
cartesian distance between the
current and the given points.
        distanceTo: function
(point) {
                point =
toPoint(point);

                var x =
point.x - this.x,
                    y =
point.y - this.y;

                return
Math.sqrt(x * x + y * y);
        },

        // @method
equals(otherPoint: Point):
Boolean
        // Returns `true` if
the given point has the same
coordinates.
        equals: function
(point) {
                point =
toPoint(point);

                return point.x
=== this.x &&
                       point.y
```

```
=== this.y;
        },

        // @method
contains(otherPoint: Point):
Boolean
        // Returns `true` if
both coordinates of the given
point are less than the
corresponding current point
coordinates (in absolute
values).
        contains: function
(point) {
                point =
toPoint(point);

                return
Math.abs(point.x) <=
Math.abs(this.x) &&

Math.abs(point.y) <=
Math.abs(this.y);
        },

        // @method toString():
String
        // Returns a string
representation of the point
for debugging purposes.
        toString: function ()
{
                return
'Point(' +

formatNum(this.x) + ', ' +

formatNum(this.y) + ')';
        }
  };

  // @factory L.point(x:
Number, y: Number, round?:
Boolean)
  // Creates a Point object
with the given `x` and `y`
coordinates. If optional
```

```
`round` is set to true, rounds
the `x` and `y` values.

  // @alternative
  // @factory L.point(coords:
Number[])
  // Expects an array of the
form `[x, y]` instead.

  // @alternative
  // @factory L.point(coords:
Object)
  // Expects a plain object of
the form `{x: Number, y:
Number}` instead.
  function toPoint(x, y,
round) {
        if (x instanceof
Point) {
                return x;
        }
        if (isArray(x)) {
                return new
Point(x[0], x[1]);
        }
        if (x === undefined ||
x === null) {
                return x;
        }
        if (typeof x ===
'object' && 'x' in x && 'y' in
x) {
                return new
Point(x.x, x.y);
        }
        return new Point(x, y,
round);
  }

  /*
   * @class Bounds
   * @aka L.Bounds
   *
   * Represents a rectangular
area in pixel coordinates.
   *
   * @example
```

```
   *
   * ```js
   * 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:
   *
   * ```js
   *
otherBounds.intersects([[10,
10], [40, 60]]);
   * ```
   *
   * Note that `Bounds` does
not inherit from Leaflet's
`Class` object,
   * which means new classes
can't inherit from it, and new
methods
   * can't be added to it with
the `include` function.
   */

  function Bounds(a, b) {
        if (!a) { return; }

        var points = b ? [a,
b] : a;

        for (var i = 0, len =
points.length; i < len; i++) {

this.extend(points[i]);
        }
  }

  Bounds.prototype = {
        // @method
extend(point: Point): this
```

```
        // Extends the bounds
to contain the given point.

        // @alternative
        // @method
extend(otherBounds: Bounds):
this
        // Extend the bounds
to contain the given bounds
        extend: function (obj)
{
                var min2,
max2;
                if (!obj) {
return this; }

                if (obj
instanceof Point || typeof
obj[0] === 'number' || 'x' in
obj) {
                        min2 =
max2 = toPoint(obj);
                } else {
                        obj =
toBounds(obj);
                        min2 =
obj.min;
                        max2 =
obj.max;

                        if
(!min2 || !max2) { return
this; }
                }

                // @property
min: Point
                // The top
left corner of the rectangle.
                // @property
max: Point
                // The bottom
right corner of the rectangle.
                if (!this.min
&& !this.max) {

this.min = min2.clone();
```

```
this.max = max2.clone();
                } else {

this.min.x = Math.min(min2.x,
this.min.x);

this.max.x = Math.max(max2.x,
this.max.x);

this.min.y = Math.min(min2.y,
this.min.y);

this.max.y = Math.max(max2.y,
this.max.y);
                }
                return this;
        },

        // @method
getCenter(round?: Boolean):
Point
        // Returns the center
point of the bounds.
        getCenter: function
(round) {
                return
toPoint(

(this.min.x + this.max.x) / 2,

(this.min.y + this.max.y) / 2,
round);
        },

        // @method
getBottomLeft(): Point
        // Returns the bottom-
left point of the bounds.
        getBottomLeft:
function () {
                return
toPoint(this.min.x,
this.max.y);
        },

        // @method
```

```
getTopRight(): Point
        // Returns the top-
right point of the bounds.
        getTopRight: function
() { // -> Point
                return
toPoint(this.max.x,
this.min.y);
        },

        // @method
getTopLeft(): Point
        // Returns the top-
left point of the bounds (i.e.
[`this.min`](#bounds-min)).
        getTopLeft: function
() {
                return
this.min; // left, top
        },

        // @method
getBottomRight(): Point
        // Returns the bottom-
right point of the bounds
(i.e. [`this.max`](#bounds-
max)).
        getBottomRight:
function () {
                return
this.max; // right, bottom
        },

        // @method getSize():
Point
        // Returns the size of
the given bounds
        getSize: function () {
                return
this.max.subtract(this.min);
        },

        // @method
contains(otherBounds: Bounds):
Boolean
        // Returns `true` if
the rectangle contains the
```

```
given one.
        // @alternative
        // @method
contains(point: Point):
Boolean
        // Returns `true` if
the rectangle contains the
given point.
        contains: function
(obj) {
                var min, max;

                if (typeof
obj[0] === 'number' || obj
instanceof Point) {
                        obj =
toPoint(obj);
                } else {
                        obj =
toBounds(obj);
                }

                if (obj
instanceof Bounds) {
                        min =
obj.min;
                        max =
obj.max;
                } else {
                        min =
max = obj;
                }

                return (min.x
>= this.min.x) &&
                        (max.x
<= this.max.x) &&
                        (min.y
>= this.min.y) &&
                        (max.y
<= this.max.y);
        },

        // @method
intersects(otherBounds:
Bounds): Boolean
        // Returns `true` if
```

```
the rectangle intersects the
given bounds. Two bounds
        // intersect if they
have at least one point in
common.
        intersects: function
(bounds) { // (Bounds) ->
Boolean
                bounds =
toBounds(bounds);

                var min =
this.min,
                    max =
this.max,
                    min2 =
bounds.min,
                    max2 =
bounds.max,

xIntersects = (max2.x >=
min.x) && (min2.x <= max.x),

yIntersects = (max2.y >=
min.y) && (min2.y <= max.y);

                return
xIntersects && yIntersects;
        },

        // @method
overlaps(otherBounds: Bounds):
Boolean
        // Returns `true` if
the rectangle overlaps the
given bounds. Two bounds
        // overlap if their
intersection is an area.
        overlaps: function
(bounds) { // (Bounds) ->
Boolean
                bounds =
toBounds(bounds);

                var min =
this.min,
                    max =
```

```
this.max,
                    min2 =
bounds.min,
                    max2 =
bounds.max,
                    xOverlaps
= (max2.x > min.x) && (min2.x
< max.x),
                    yOverlaps
= (max2.y > min.y) && (min2.y
< max.y);

                return
xOverlaps && yOverlaps;
        },

        // @method isValid():
Boolean
        // Returns `true` if
the bounds are properly
initialized.
        isValid: function () {
                return !!
(this.min && this.max);
        },


        // @method
pad(bufferRatio: Number):
Bounds
        // Returns bounds
created by extending or
retracting the current bounds
by a given ratio in each
direction.
        // For example, a
ratio of 0.5 extends the
bounds by 50% in each
direction.
        // Negative values
will retract the bounds.
        pad: function
(bufferRatio) {
                var min =
this.min,
                max =
this.max,
```

```
                heightBuffer =
Math.abs(min.x - max.x) *
bufferRatio,
                widthBuffer =
Math.abs(min.y - max.y) *
bufferRatio;


                return
toBounds(

toPoint(min.x - heightBuffer,
min.y - widthBuffer),

toPoint(max.x + heightBuffer,
max.y + widthBuffer));
        },


        // @method
equals(otherBounds: Bounds,
maxMargin?: Number): Boolean
        // Returns `true` if
the rectangle is equivalent
(within a small margin of
error) to the given bounds.
The margin of error can be
overridden by setting
`maxMargin` to a small number.
        equals: function
(bounds) {
                if (!bounds) {
return false; }

                bounds =
toBounds(bounds);

                return
this.min.equals(bounds.getTopL
eft()) &&

this.max.equals(bounds.getBott
omRight());
        },
  };
```

```
  // @factory
L.bounds(corner1: Point,
corner2: Point)
  // Creates a Bounds object
from two corners coordinate
pairs.
  // @alternative
  // @factory L.bounds(points:
Point[])
  // Creates a Bounds object
from the given array of
points.
  function toBounds(a, b) {
        if (!a || a instanceof
Bounds) {
                return a;
        }
        return new Bounds(a,
b);
  }

  /*
   * @class LatLngBounds
   * @aka L.LatLngBounds
   *
   * Represents a rectangular
geographical area on a map.
   *
   * @example
   *
   * ```js
   * var corner1 =
L.latLng(40.712, -74.227),
   * corner2 =
L.latLng(40.774, -74.125),
   * bounds =
L.latLngBounds(corner1,
corner2);
   * ```
   *
   * All Leaflet methods that
accept LatLngBounds objects
also accept them in a simple
Array form (unless noted
otherwise), so the bounds
example above can be passed
like this:
```

```
   *
   * ```js
   * map.fitBounds([
   *    [40.712, -74.227],
   *    [40.774, -74.125]
   * ]);
   * ```
   *
   * Caution: if the area
crosses the antimeridian
(often confused with the
International Date Line), you
must specify corners _outside_
the [-180, 180] degrees
longitude range.
   *
   * Note that `LatLngBounds`
does not inherit from
Leaflet's `Class` object,
   * which means new classes
can't inherit from it, and new
methods
   * can't be added to it with
the `include` function.
   */

  function
LatLngBounds(corner1, corner2)
{ // (LatLng, LatLng) or
(LatLng[])
        if (!corner1) {
return; }

        var latlngs = corner2
? [corner1, corner2] :
corner1;

        for (var i = 0, len =
latlngs.length; i < len; i++)
{

this.extend(latlngs[i]);
        }
  }

  LatLngBounds.prototype = {
```

```
        // @method
extend(latlng: LatLng): this
        // Extend the bounds
to contain the given point

        // @alternative
        // @method
extend(otherBounds:
LatLngBounds): this
        // Extend the bounds
to contain the given bounds
        extend: function (obj)
{
                var sw =
this._southWest,
                    ne =
this._northEast,
                    sw2, ne2;

                if (obj
instanceof LatLng) {
                        sw2 =
obj;
                        ne2 =
obj;

                } else if (obj
instanceof LatLngBounds) {
                        sw2 =
obj._southWest;
                        ne2 =
obj._northEast;

                        if
(!sw2 || !ne2) { return this;
}

                } else {
                        return
obj ?
this.extend(toLatLng(obj) ||
toLatLngBounds(obj)) : this;
                }

                if (!sw &&
!ne) {
```

```
this._southWest = new
LatLng(sw2.lat, sw2.lng);

this._northEast = new
LatLng(ne2.lat, ne2.lng);
                } else {
                        sw.lat
= Math.min(sw2.lat, sw.lat);
                        sw.lng
= Math.min(sw2.lng, sw.lng);
                        ne.lat
= Math.max(ne2.lat, ne.lat);
                        ne.lng
= Math.max(ne2.lng, ne.lng);
                }

                return this;
        },

        // @method
pad(bufferRatio: Number):
LatLngBounds
        // Returns bounds
created by extending or
retracting the current bounds
by a given ratio in each
direction.
        // For example, a
ratio of 0.5 extends the
bounds by 50% in each
direction.
        // Negative values
will retract the bounds.
        pad: function
(bufferRatio) {
                var sw =
this._southWest,
                    ne =
this._northEast,

heightBuffer = Math.abs(sw.lat
- ne.lat) * bufferRatio,

widthBuffer = Math.abs(sw.lng
- ne.lng) * bufferRatio;

                return new
```

```
LatLngBounds(
                        new
LatLng(sw.lat - heightBuffer,
sw.lng - widthBuffer),
                        new
LatLng(ne.lat + heightBuffer,
ne.lng + widthBuffer));
        },

        // @method
getCenter(): LatLng
        // Returns the center
point of the bounds.
        getCenter: function ()
{
                return new
LatLng(

(this._southWest.lat +
this._northEast.lat) / 2,

(this._southWest.lng +
this._northEast.lng) / 2);
        },

        // @method
getSouthWest(): LatLng
        // Returns the south-
west point of the bounds.
        getSouthWest: function
() {
                return
this._southWest;
        },

        // @method
getNorthEast(): LatLng
        // Returns the north-
east point of the bounds.
        getNorthEast: function
() {
                return
this._northEast;
        },

        // @method
getNorthWest(): LatLng
```

```
        // Returns the north-
west point of the bounds.
        getNorthWest: function
() {
                return new
LatLng(this.getNorth(),
this.getWest());
        },

        // @method
getSouthEast(): LatLng
        // Returns the south-
east point of the bounds.
        getSouthEast: function
() {
                return new
LatLng(this.getSouth(),
this.getEast());
        },

        // @method getWest():
Number
        // Returns the west
longitude of the bounds
        getWest: function () {
                return
this._southWest.lng;
        },

        // @method getSouth():
Number
        // Returns the south
latitude of the bounds
        getSouth: function ()
{
                return
this._southWest.lat;
        },

        // @method getEast():
Number
        // Returns the east
longitude of the bounds
        getEast: function () {
                return
this._northEast.lng;
        },
```

```
        // @method getNorth(): Number
        // Returns the north latitude of the bounds
        getNorth: function () {
                return this._northEast.lat;
        },

        // @method contains(otherBounds: LatLngBounds): Boolean
        // Returns `true` if the rectangle contains the given one.

        // @alternative
        // @method contains(latlng: LatLng): Boolean
        // Returns `true` if the rectangle contains the given point.
        contains: function (obj) { // (LatLngBounds) or (LatLng) -> Boolean
                if (typeof obj[0] === 'number' || obj instanceof LatLng || 'lat' in obj) {
                        obj = toLatLng(obj);
                } else {
                        obj = toLatLngBounds(obj);
                }

                var sw = this._southWest,
                    ne = this._northEast,
                    sw2, ne2;

                if (obj instanceof LatLngBounds) {
                        sw2 =
```

```
obj.getSouthWest();
                        ne2 =
obj.getNorthEast();
                } else {
                        sw2 =
ne2 = obj;
                }

                return
(sw2.lat >= sw.lat) &&
(ne2.lat <= ne.lat) &&

(sw2.lng >= sw.lng) &&
(ne2.lng <= ne.lng);
        },

        // @method
intersects(otherBounds:
LatLngBounds): Boolean
        // Returns `true` if
the rectangle intersects the
given bounds. Two bounds
intersect if they have at
least one point in common.
        intersects: function
(bounds) {
                bounds =
toLatLngBounds(bounds);

                var sw =
this._southWest,
                    ne =
this._northEast,
                    sw2 =
bounds.getSouthWest(),
                    ne2 =
bounds.getNorthEast(),


latIntersects = (ne2.lat >=
sw.lat) && (sw2.lat <=
ne.lat),

lngIntersects = (ne2.lng >=
sw.lng) && (sw2.lng <=
ne.lng);
```

```
                return
latIntersects &&
lngIntersects;
        },

        // @method
overlaps(otherBounds:
LatLngBounds): Boolean
        // Returns `true` if
the rectangle overlaps the
given bounds. Two bounds
overlap if their intersection
is an area.
        overlaps: function
(bounds) {
                bounds =
toLatLngBounds(bounds);

                var sw =
this._southWest,
                    ne =
this._northEast,
                    sw2 =
bounds.getSouthWest(),
                    ne2 =
bounds.getNorthEast(),


latOverlaps = (ne2.lat >
sw.lat) && (sw2.lat < ne.lat),

lngOverlaps = (ne2.lng >
sw.lng) && (sw2.lng < ne.lng);

                return
latOverlaps && lngOverlaps;
        },

        // @method
toBBoxString(): String
        // Returns a string
with bounding box coordinates
in a
'southwest_lng,southwest_lat,n
ortheast_lng,northeast_lat'
format. Useful for sending
requests to web services that
```

```
return geo data.
        toBBoxString: function
() {
                return
[this.getWest(),
this.getSouth(),
this.getEast(),
this.getNorth()].join(',');
        },

        // @method
equals(otherBounds:
LatLngBounds, maxMargin?:
Number): Boolean
        // Returns `true` if
the rectangle is equivalent
(within a small margin of
error) to the given bounds.
The margin of error can be
overridden by setting
`maxMargin` to a small number.
        equals: function
(bounds, maxMargin) {
                if (!bounds) {
return false; }

                bounds =
toLatLngBounds(bounds);

                return
this._southWest.equals(bounds.
getSouthWest(), maxMargin) &&

this._northEast.equals(bounds.
getNorthEast(), maxMargin);
        },

        // @method isValid():
Boolean
        // Returns `true` if
the bounds are properly
initialized.
        isValid: function () {
                return !!
(this._southWest &&
this._northEast);
        }
```

```
  };

  // TODO International date
line?

  // @factory
L.latLngBounds(corner1:
LatLng, corner2: LatLng)
  // Creates a `LatLngBounds`
object by defining two
diagonally opposite corners of
the rectangle.

  // @alternative
  // @factory
L.latLngBounds(latlngs:
LatLng[])
  // Creates a `LatLngBounds`
object defined by the
geographical points it
contains. Very useful for
zooming the map to fit a
particular set of locations
with [`fitBounds`](#map-
fitbounds).
  function toLatLngBounds(a,
b) {
        if (a instanceof
LatLngBounds) {
                return a;
        }
        return new
LatLngBounds(a, b);
  }

  /* @class LatLng
   * @aka L.LatLng
   *
   * Represents a geographical
point with a certain latitude
and longitude.
   *
   * @example
   *
   * ```
   * var latlng =
L.latLng(50.5, 30.5);
```

```
   * ```
   *
   * All Leaflet methods that
accept LatLng objects also
accept them in a simple Array
form and simple object form
(unless noted otherwise), so
these lines are equivalent:
   *
   * ```
   * map.panTo([50, 30]);
   * map.panTo({lon: 30, lat:
50});
   * map.panTo({lat: 50, lng:
30});
   * map.panTo(L.latLng(50,
30));
   * ```
   *
   * Note that `LatLng` does
not inherit from Leaflet's
`Class` object,
   * which means new classes
can't inherit from it, and new
methods
   * can't be added to it with
the `include` function.
   */

  function LatLng(lat, lng,
alt) {
        if (isNaN(lat) ||
isNaN(lng)) {
                throw new
Error('Invalid LatLng object:
(' + lat + ', ' + lng + ')');
        }

        // @property lat:
Number
        // Latitude in degrees
        this.lat = +lat;

        // @property lng:
Number
        // Longitude in
degrees
```

```
        this.lng = +lng;

        // @property alt:
Number
        // Altitude in meters
(optional)
        if (alt !== undefined)
{
                this.alt =
+alt;
        }
  }

  LatLng.prototype = {
        // @method
equals(otherLatLng: LatLng,
maxMargin?: Number): Boolean
        // Returns `true` if
the given `LatLng` point is at
the same position (within a
small margin of error). The
margin of error can be
overridden by setting
`maxMargin` to a small number.
        equals: function (obj,
maxMargin) {
                if (!obj) {
return false; }

                obj =
toLatLng(obj);

                var margin =
Math.max(

Math.abs(this.lat - obj.lat),

Math.abs(this.lng - obj.lng));

                return margin
<= (maxMargin === undefined ?
1.0E-9 : maxMargin);
        },

        // @method toString():
String
        // Returns a string
```

```
representation of the point
(for debugging purposes).
        toString: function
(precision) {
                return
'LatLng(' +

formatNum(this.lat, precision)
+ ', ' +

formatNum(this.lng, precision)
+ ')';
        },

        // @method
distanceTo(otherLatLng:
LatLng): Number
        // Returns the
distance (in meters) to the
given `LatLng` calculated
using the [Spherical Law of
Cosines]
(https://en.wikipedia.org/wiki
/Spherical_law_of_cosines).
        distanceTo: function
(other) {
                return
Earth.distance(this,
toLatLng(other));
        },

        // @method wrap():
LatLng
        // Returns a new
`LatLng` object with the
longitude wrapped so it's
always between -180 and +180
degrees.
        wrap: function () {
                return
Earth.wrapLatLng(this);
        },

        // @method
toBounds(sizeInMeters:
Number): LatLngBounds
        // Returns a new
```

```
`LatLngBounds` object in which
each boundary is
`sizeInMeters/2` meters apart
from the `LatLng`.
        toBounds: function
(sizeInMeters) {
                var
latAccuracy = 180 *
sizeInMeters / 40075017,

lngAccuracy = latAccuracy /
Math.cos((Math.PI / 180) *
this.lat);

                return
toLatLngBounds(

[this.lat - latAccuracy,
this.lng - lngAccuracy],

[this.lat + latAccuracy,
this.lng + lngAccuracy]);
        },

        clone: function () {
                return new
LatLng(this.lat, this.lng,
this.alt);
        }
  };



  // @factory
L.latLng(latitude: Number,
longitude: Number, altitude?:
Number): LatLng
  // Creates an object
representing a geographical
point with the given latitude
and longitude (and optionally
altitude).

  // @alternative
  // @factory L.latLng(coords:
Array): LatLng
  // Expects an array of the
```

```
form `[Number, Number]` or
`[Number, Number, Number]`
instead.

  // @alternative
  // @factory L.latLng(coords:
Object): LatLng
  // Expects an plain object
of the form `{lat: Number,
lng: Number}` or `{lat:
Number, lng: Number, alt:
Number}` instead.

  function toLatLng(a, b, c) {
        if (a instanceof
LatLng) {
                return a;
        }
        if (isArray(a) &&
typeof a[0] !== 'object') {
                if (a.length
=== 3) {
                        return
new LatLng(a[0], a[1], a[2]);
                }
                if (a.length
=== 2) {
                        return
new LatLng(a[0], a[1]);
                }
                return null;
        }
        if (a === undefined ||
a === null) {
                return a;
        }
        if (typeof a ===
'object' && 'lat' in a) {
                return new
LatLng(a.lat, 'lng' in a ?
a.lng : a.lon, a.alt);
        }
        if (b === undefined) {
                return null;
        }
        return new LatLng(a,
b, c);
```

```
  }

  /*
   * @namespace CRS
   * @crs L.CRS.Base
   * Object that defines
coordinate reference systems
for projecting
   * geographical points into
pixel (screen) coordinates and
back (and to
   * coordinates in other
units for [WMS]
(https://en.wikipedia.org/wiki
/Web_Map_Service) services).
See
   * [spatial reference
system]
(https://en.wikipedia.org/wiki
/Spatial_reference_system).
   *
   * Leaflet defines the most
usual CRSs by default. If you
want to use a
   * CRS not defined by
default, take a look at the
   * [Proj4Leaflet]
(https://github.com/kartena/Pr
oj4Leaflet) plugin.
   *
   * Note that the CRS
instances do not inherit from
Leaflet's `Class` object,
   * and can't be
instantiated. Also, new
classes can't inherit from
them,
   * and methods can't be
added to them with the
`include` function.
   */

  var CRS = {
        // @method
latLngToPoint(latlng: LatLng,
zoom: Number): Point
        // Projects
```

```
geographical coordinates into
pixel coordinates for a given
zoom.
        latLngToPoint:
function (latlng, zoom) {
                var
projectedPoint =
this.projection.project(latlng
),
                    scale =
this.scale(zoom);

                return
this.transformation._transform
(projectedPoint, scale);
        },

        // @method
pointToLatLng(point: Point,
zoom: Number): LatLng
        // The inverse of
`latLngToPoint`. Projects
pixel coordinates on a given
        // zoom into
geographical coordinates.
        pointToLatLng:
function (point, zoom) {
                var scale =
this.scale(zoom),

untransformedPoint =
this.transformation.untransfor
m(point, scale);

                return
this.projection.unproject(untr
ansformedPoint);
        },

        // @method
project(latlng: LatLng): Point
        // Projects
geographical coordinates into
coordinates in units accepted
for
        // this CRS (e.g.
meters for EPSG:3857, for
```

```
passing it to WMS services).
        project: function
(latlng) {
                return
this.projection.project(latlng
);
        },

        // @method
unproject(point: Point):
LatLng
        // Given a projected
coordinate returns the
corresponding LatLng.
        // The inverse of
`project`.
        unproject: function
(point) {
                return
this.projection.unproject(poin
t);
        },

        // @method scale(zoom:
Number): 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.
        scale: function (zoom)
{
                return 256 *
Math.pow(2, zoom);
        },

        // @method zoom(scale:
Number): Number
        // Inverse of
`scale()`, returns the zoom
level corresponding to a scale
        // factor of `scale`.
        zoom: function (scale)
{
```

```
                return
Math.log(scale / 256) /
Math.LN2;
        },

        // @method
getProjectedBounds(zoom:
Number): Bounds
        // Returns the
projection's bounds scaled and
transformed for the provided
`zoom`.
        getProjectedBounds:
function (zoom) {
                if
(this.infinite) { return null;
}

                var b =
this.projection.bounds,
                    s =
this.scale(zoom),
                    min =
this.transformation.transform(
b.min, s),
                    max =
this.transformation.transform(
b.max, s);

                return new
Bounds(min, max);
        },

        // @method
distance(latlng1: LatLng,
latlng2: LatLng): Number
        // Returns the
distance between two
geographical coordinates.

        // @property code:
String
        // Standard code name
of the CRS passed into WMS
services (e.g. `'EPSG:3857'`)
        //
        // @property wrapLng:
```

```
Number[]
        // An array of two
numbers defining whether the
longitude (horizontal)
coordinate
        // axis wraps around a
given range and how. Defaults
to `[-180, 180]` in most
        // geographical CRSs.
If `undefined`, the longitude
axis does not wrap around.
        //
        // @property wrapLat:
Number[]
        // Like `wrapLng`, but
for the latitude (vertical)
axis.

        // wrapLng: [min,
max],
        // wrapLat: [min,
max],

        // @property infinite:
Boolean
        // If true, the
coordinate space will be
unbounded (infinite in both
axes)
        infinite: false,

        // @method
wrapLatLng(latlng: LatLng):
LatLng
        // Returns a `LatLng`
where lat and lng has been
wrapped according to the
        // CRS's `wrapLat` and
`wrapLng` properties, if they
are outside the CRS's bounds.
        wrapLatLng: function
(latlng) {
                var lng =
this.wrapLng ?
wrapNum(latlng.lng,
this.wrapLng, true) :
latlng.lng,
```

```
                    lat =
this.wrapLat ?
wrapNum(latlng.lat,
this.wrapLat, true) :
latlng.lat,
                    alt =
latlng.alt;

                return new
LatLng(lat, lng, alt);
        },

        // @method
wrapLatLngBounds(bounds:
LatLngBounds): LatLngBounds
        // Returns a
`LatLngBounds` with the same
size as the given one,
ensuring
        // that its center is
within the CRS's bounds.
        // Only accepts actual
`L.LatLngBounds` instances,
not arrays.
        wrapLatLngBounds:
function (bounds) {
                var center =
bounds.getCenter(),
                    newCenter
= this.wrapLatLng(center),
                    latShift =
center.lat - newCenter.lat,
                    lngShift =
center.lng - newCenter.lng;

                if (latShift
=== 0 && lngShift === 0) {
                        return
bounds;
                }

                var sw =
bounds.getSouthWest(),
                    ne =
bounds.getNorthEast(),
                    newSw =
new LatLng(sw.lat - latShift,
```

```
sw.lng - lngShift),
                    newNe =
new LatLng(ne.lat - latShift,
ne.lng - lngShift);

                return new
LatLngBounds(newSw, newNe);
        }
  };

  /*
   * @namespace CRS
   * @crs L.CRS.Earth
   *
   * Serves as the base for
CRS that are global such that
they cover the earth.
   * Can only be used as the
base for other CRS and cannot
be used directly,
   * since it does not have a
`code`, `projection` or
`transformation`. `distance()`
returns
   * meters.
   */

  var Earth = extend({}, CRS,
{
        wrapLng: [-180, 180],

        // Mean Earth Radius,
as recommended for use by
        // the International
Union of Geodesy and
Geophysics,
        // see
https://rosettacode.org/wiki/H
aversine_formula
        R: 6371000,

        // distance between
two geographical points using
spherical law of cosines
approximation
        distance: function
(latlng1, latlng2) {
```

```
                var rad =
Math.PI / 180,
                    lat1 =
latlng1.lat * rad,
                    lat2 =
latlng2.lat * rad,
                    sinDLat =
Math.sin((latlng2.lat -
latlng1.lat) * rad / 2),
                    sinDLon =
Math.sin((latlng2.lng -
latlng1.lng) * rad / 2),
                    a =
sinDLat * sinDLat +
Math.cos(lat1) *
Math.cos(lat2) * sinDLon *
sinDLon,
                    c = 2 *
Math.atan2(Math.sqrt(a),
Math.sqrt(1 - a));
                return this.R
* c;
        }
  });

  /*
   * @namespace Projection
   * @projection
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.
   */

  var earthRadius = 6378137;

  var SphericalMercator = {

        R: earthRadius,
        MAX_LATITUDE:
85.0511287798,
```

```
        project: function
(latlng) {
                var d =
Math.PI / 180,
                    max =
this.MAX_LATITUDE,
                    lat =
Math.max(Math.min(max,
latlng.lat), -max),
                    sin =
Math.sin(lat * d);

                return new
Point(
                        this.R
* latlng.lng * d,
                        this.R
* Math.log((1 + sin) / (1 -
sin)) / 2);
        },

        unproject: function
(point) {
                var d = 180 /
Math.PI;

                return new
LatLng(
                        (2 *
Math.atan(Math.exp(point.y /
this.R)) - (Math.PI / 2)) * d,

point.x * d / this.R);
        },

        bounds: (function () {
                var d =
earthRadius * Math.PI;
                return new
Bounds([-d, -d], [d, d]);
        })()
  };

  /*
   * @class Transformation
   * @aka L.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.
   *
   * @example
   *
   * ```js
   * var transformation =
L.transformation(2, 5, -1,
10),
   *    p = L.point(1, 2),
   *    p2 =
transformation.transform(p),
//  L.point(7, 8)
   *    p3 =
transformation.untransform(p2)
; //  L.point(1, 2)
   * ```
   */


  // factory new
L.Transformation(a: Number, b:
Number, c: Number, d: Number)
  // Creates a
`Transformation` object with
the given coefficients.
  function Transformation(a,
b, c, d) {
        if (isArray(a)) {
                // use array
properties
                this._a =
a[0];
                this._b =
a[1];
                this._c =
a[2];
                this._d =
```

```
a[3];
                return;
        }
        this._a = a;
        this._b = b;
        this._c = c;
        this._d = d;
  }

  Transformation.prototype = {
        // @method
transform(point: Point,
scale?: Number): Point
        // Returns a
transformed point, optionally
multiplied by the given scale.
        // Only accepts actual
`L.Point` instances, not
arrays.
        transform: function
(point, scale) { // (Point,
Number) -> Point
                return
this._transform(point.clone(),
scale);
        },

        // destructive
transform (faster)
        _transform: function
(point, scale) {
                scale = scale
|| 1;
                point.x =
scale * (this._a * point.x +
this._b);
                point.y =
scale * (this._c * point.y +
this._d);
                return point;
        },

        // @method
untransform(point: Point,
scale?: Number): Point
        // Returns the reverse
transformation of the given
```

```
point, optionally divided
        // by the given scale.
Only accepts actual `L.Point`
instances, not arrays.
        untransform: function
(point, scale) {
                scale = scale
|| 1;
                return new
Point(

(point.x / scale - this._b) /
this._a,

(point.y / scale - this._d) /
this._c);
        }
  };

  // factory
L.transformation(a: Number, b:
Number, c: Number, d: Number)

  // @factory
L.transformation(a: Number, b:
Number, c: Number, d: Number)
  // Instantiates a
Transformation object with the
given coefficients.

  // @alternative
  // @factory
L.transformation(coefficients:
Array): Transformation
  // Expects an coefficients
array of the form
  // `[a: Number, b: Number,
c: Number, d: Number]`.

  function toTransformation(a,
b, c, d) {
        return new
Transformation(a, b, c, d);
  }

  /*
   * @namespace CRS
```

```
   * @crs 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.
   */

  var EPSG3857 = extend({},
Earth, {
        code: 'EPSG:3857',
        projection:
SphericalMercator,

        transformation:
(function () {
                var scale =
0.5 / (Math.PI *
SphericalMercator.R);
                return
toTransformation(scale, 0.5, -
scale, 0.5);
        }())
  });

  var EPSG900913 = extend({},
EPSG3857, {
        code: 'EPSG:900913'
  });

  // @namespace SVG; @section
  // There are several static
functions which can be called
without instantiating L.SVG:

  // @function create(name:
String): SVGElement
  // Returns a instance of
[SVGElement]
(https://developer.mozilla.org
/docs/Web/API/SVGElement),
  // corresponding to the
class name passed. For
example, using 'line' will
```

```
return
  // an instance of
[SVGLineElement]
(https://developer.mozilla.org
/docs/Web/API/SVGLineElement).
  function svgCreate(name) {
        return
document.createElementNS('http
://www.w3.org/2000/svg',
name);
  }

  // @function
pointsToPath(rings: Point[],
closed: Boolean): String
  // Generates a SVG path
string for multiple rings,
with each ring turning
  // into "M..L..L.."
instructions
  function pointsToPath(rings,
closed) {
        var str = '',
        i, j, len, len2,
points, p;

        for (i = 0, len =
rings.length; i < len; i++) {
                points =
rings[i];

                for (j = 0,
len2 = points.length; j <
len2; j++) {
                        p =
points[j];
                        str +=
(j ? 'L' : 'M') + p.x + ' ' +
p.y;
                }

                // closes the
ring for polygons; "x" is VML
syntax
                str += closed
? (Browser.svg ? 'z' : 'x') :
'';
```

```
        }

        // SVG complains about
empty path strings
        return str || 'M0 0';
  }

  /*
   * @namespace Browser
   * @aka L.Browser
   *
   * A namespace with static
properties for browser/feature
detection used by Leaflet
internally.
   *
   * @example
   *
   * ```js
   * if (L.Browser.ielt9) {
   *   alert('Upgrade your
browser, dude!');
   * }
   * ```
   */

  var style =
document.documentElement.style
;

  // @property ie: Boolean;
`true` for all Internet
Explorer versions (not Edge).
  var ie = 'ActiveXObject' in
window;

  // @property ielt9: Boolean;
`true` for Internet Explorer
versions less than 9.
  var ielt9 = ie &&
!document.addEventListener;

  // @property edge: Boolean;
`true` for the Edge web
browser.
  var edge = 'msLaunchUri' in
navigator && !('documentMode'
```

```
in document);

  // @property webkit:
Boolean;
  // `true` for webkit-based
browsers like Chrome and
Safari (including mobile
versions).
  var webkit =
userAgentContains('webkit');

  // @property android:
Boolean
  // **Deprecated.** `true`
for any browser running on an
Android platform.
  var android =
userAgentContains('android');

  // @property android23:
Boolean; **Deprecated.**
`true` for browsers running on
Android 2 or Android 3.
  var android23 =
userAgentContains('android 2')
|| userAgentContains('android
3');

  /* See
https://stackoverflow.com/a/17
961266 for details on
detecting stock Android */
  var webkitVer =
parseInt(/WebKit\/([0-
9]+)|$/.exec(navigator.userAge
nt)[1], 10); // also matches
AppleWebKit
  // @property androidStock:
Boolean; **Deprecated.**
`true` for the Android stock
browser (i.e. not Chrome)
  var androidStock = android
&& userAgentContains('Google')
&& webkitVer < 537 && !
('AudioNode' in window);

  // @property opera: Boolean;
```

```
`true` for the Opera browser
  var opera = !!window.opera;

  // @property chrome:
Boolean; `true` for the Chrome
browser.
  var chrome = !edge &&
userAgentContains('chrome');

  // @property gecko: Boolean;
`true` for gecko-based
browsers like Firefox.
  var gecko =
userAgentContains('gecko') &&
!webkit && !opera && !ie;

  // @property safari:
Boolean; `true` for the Safari
browser.
  var safari = !chrome &&
userAgentContains('safari');

  var phantom =
userAgentContains('phantom');

  // @property opera12:
Boolean
  // `true` for the Opera
browser supporting CSS
transforms (version 12 or
later).
  var opera12 = 'OTransition'
in style;

  // @property win: Boolean;
`true` when the browser is
running in a Windows platform
  var win =
navigator.platform.indexOf('Wi
n') === 0;

  // @property ie3d: Boolean;
`true` for all Internet
Explorer versions supporting
CSS transforms.
  var ie3d = ie &&
('transition' in style);
```

```
  // @property webkit3d:
Boolean; `true` for webkit-
based browsers supporting CSS
transforms.
  var webkit3d =
('WebKitCSSMatrix' in window)
&& ('m11' in new
window.WebKitCSSMatrix()) &&
!android23;

  // @property gecko3d:
Boolean; `true` for gecko-
based browsers supporting CSS
transforms.
  var gecko3d =
'MozPerspective' in style;

  // @property any3d: Boolean
  // `true` for all browsers
supporting CSS transforms.
  var any3d =
!window.L_DISABLE_3D && (ie3d
|| webkit3d || gecko3d) &&
!opera12 && !phantom;

  // @property mobile:
Boolean; `true` for all
browsers running in a mobile
device.
  var mobile = typeof
orientation !== 'undefined' ||
userAgentContains('mobile');

  // @property mobileWebkit:
Boolean; `true` for all
webkit-based browsers in a
mobile device.
  var mobileWebkit = mobile &&
webkit;

  // @property mobileWebkit3d:
Boolean
  // `true` for all webkit-
based browsers in a mobile
device supporting CSS
transforms.
```

```
  var mobileWebkit3d = mobile
&& webkit3d;

  // @property msPointer:
Boolean
  // `true` for browsers
implementing the Microsoft
touch events model (notably
IE10).
  var msPointer =
!window.PointerEvent &&
window.MSPointerEvent;

  // @property pointer:
Boolean
  // `true` for all browsers
supporting [pointer events]
(https://msdn.microsoft.com/en
-
us/library/dn433244%28v=vs.85%
29.aspx).
  var pointer = !!
(window.PointerEvent ||
msPointer);

  // @property touchNative:
Boolean
  // `true` for all browsers
supporting [touch events]
(https://developer.mozilla.org
/docs/Web/API/Touch_events).
  // **This does not
necessarily mean** that the
browser is running in a
computer with
  // a touchscreen, it only
means that the browser is
capable of understanding
  // touch events.
  var touchNative =
'ontouchstart' in window ||
!!window.TouchEvent;

  // @property touch: Boolean
  // `true` for all browsers
supporting either [touch]
(#browser-touch) or [pointer]
```

```
(#browser-pointer) events.
  // Note: pointer events will
be preferred (if available),
and processed for all `touch*`
listeners.
  var touch =
!window.L_NO_TOUCH &&
(touchNative || pointer);

  // @property mobileOpera:
Boolean; `true` for the Opera
browser in a mobile device.
  var mobileOpera = mobile &&
opera;

  // @property mobileGecko:
Boolean
  // `true` for gecko-based
browsers running in a mobile
device.
  var mobileGecko = mobile &&
gecko;

  // @property retina: Boolean
  // `true` for browsers on a
high-resolution "retina"
screen or on any screen when
browser's display zoom is more
than 100%.
  var retina =
(window.devicePixelRatio ||
(window.screen.deviceXDPI /
window.screen.logicalXDPI)) >
1;

  // @property passiveEvents:
Boolean
  // `true` for browsers that
support passive events.
  var passiveEvents =
(function () {
        var
supportsPassiveOption = false;
        try {
                var opts =
Object.defineProperty({},
'passive', {
```

```
                        get:
function () { // eslint-
disable-line getter-return

supportsPassiveOption = true;
                        }
                });

window.addEventListener('testP
assiveEventSupport', falseFn,
opts);

window.removeEventListener('te
stPassiveEventSupport',
falseFn, opts);
        } catch (e) {
                // Errors can
safely be ignored since this
is only a browser support
test.
        }
        return
supportsPassiveOption;
  }());

  // @property canvas: Boolean
  // `true` when the browser
supports [`<canvas>`]
(https://developer.mozilla.org
/docs/Web/API/Canvas_API).
  var canvas$1 = (function ()
{
        return
!!document.createElement('canv
as').getContext;
  }());

  // @property svg: Boolean
  // `true` when the browser
supports [SVG]
(https://developer.mozilla.org
/docs/Web/SVG).
  var svg$1 = !!
(document.createElementNS &&
svgCreate('svg').createSVGRect
);
```

```
  var inlineSvg = !!svg$1 &&
(function () {
        var div =
document.createElement('div');
        div.innerHTML =
'<svg/>';
        return (div.firstChild
&&
div.firstChild.namespaceURI)
===
'http://www.w3.org/2000/svg';
  })();

  // @property vml: Boolean
  // `true` if the browser
supports [VML]
(https://en.wikipedia.org/wiki
/Vector_Markup_Language).
  var vml = !svg$1 &&
(function () {
        try {
                var div =
document.createElement('div');
                div.innerHTML
= '<v:shape adj="1"/>';

                var shape =
div.firstChild;

shape.style.behavior =
'url(#default#VML)';

                return shape
&& (typeof shape.adj ===
'object');

        } catch (e) {
                return false;
        }
  }());


  // @property mac: Boolean;
`true` when the browser is
running in a Mac platform
  var mac =
navigator.platform.indexOf('Ma
```

```
c') === 0;

  // @property mac: Boolean;
`true` when the browser is
running in a Linux platform
  var linux =
navigator.platform.indexOf('Li
nux') === 0;

  function
userAgentContains(str) {
        return
navigator.userAgent.toLowerCas
e().indexOf(str) >= 0;
  }


  var Browser = {
        ie: ie,
        ielt9: ielt9,
        edge: edge,
        webkit: webkit,
        android: android,
        android23: android23,
        androidStock:
androidStock,
        opera: opera,
        chrome: chrome,
        gecko: gecko,
        safari: safari,
        phantom: phantom,
        opera12: opera12,
        win: win,
        ie3d: ie3d,
        webkit3d: webkit3d,
        gecko3d: gecko3d,
        any3d: any3d,
        mobile: mobile,
        mobileWebkit:
mobileWebkit,
        mobileWebkit3d:
mobileWebkit3d,
        msPointer: msPointer,
        pointer: pointer,
        touch: touch,
        touchNative:
touchNative,
```

```
        mobileOpera:
mobileOpera,
        mobileGecko:
mobileGecko,
        retina: retina,
        passiveEvents:
passiveEvents,
        canvas: canvas$1,
        svg: svg$1,
        vml: vml,
        inlineSvg: inlineSvg,
        mac: mac,
        linux: linux
  };

  /*
   * Extends L.DomEvent to
provide touch support for
Internet Explorer and Windows-
based devices.
   */

  var POINTER_DOWN =
Browser.msPointer ?
'MSPointerDown'   :
'pointerdown';
  var POINTER_MOVE =
Browser.msPointer ?
'MSPointerMove'   :
'pointermove';
  var POINTER_UP =
Browser.msPointer ?
'MSPointerUp'     :
'pointerup';
  var POINTER_CANCEL =
Browser.msPointer ?
'MSPointerCancel' :
'pointercancel';
  var pEvent = {
        touchstart  :
POINTER_DOWN,
        touchmove   :
POINTER_MOVE,
        touchend    :
POINTER_UP,
        touchcancel :
POINTER_CANCEL
```

```
  };
  var handle = {
        touchstart  :
_onPointerStart,
        touchmove   :
_handlePointer,
        touchend    :
_handlePointer,
        touchcancel :
_handlePointer
  };
  var _pointers = {};
  var _pointerDocListener =
false;

  // Provides a touch events
wrapper for (ms)pointer
events.
  // ref
https://www.w3.org/TR/pointere
vents/
https://www.w3.org/Bugs/Public
/show_bug.cgi?id=22890

  function
addPointerListener(obj, type,
handler) {
        if (type ===
'touchstart') {

_addPointerDocListener();
        }
        if (!handle[type]) {

console.warn('wrong event
specified:', type);
                return
L.Util.falseFn;
        }
        handler =
handle[type].bind(this,
handler);

obj.addEventListener(pEvent[ty
pe], handler, false);
        return handler;
  }
```

```
  function
removePointerListener(obj,
type, handler) {
        if (!pEvent[type]) {

console.warn('wrong event
specified:', type);
                return;
        }

obj.removeEventListener(pEvent
[type], handler, false);
  }

  function
_globalPointerDown(e) {
        _pointers[e.pointerId]
= e;
  }

  function
_globalPointerMove(e) {
        if
(_pointers[e.pointerId]) {

_pointers[e.pointerId] = e;
        }
  }

  function _globalPointerUp(e)
{
        delete
_pointers[e.pointerId];
  }

  function
_addPointerDocListener() {
        // need to keep track
of what pointers and how many
are active to provide
e.touches emulation
        if
(!_pointerDocListener) {
                // we listen
document as any drags that end
by moving the touch off the
```

```
screen get fired there

document.addEventListener(POIN
TER_DOWN, _globalPointerDown,
true);

document.addEventListener(POIN
TER_MOVE, _globalPointerMove,
true);

document.addEventListener(POIN
TER_UP, _globalPointerUp,
true);

document.addEventListener(POIN
TER_CANCEL, _globalPointerUp,
true);


_pointerDocListener = true;
        }
  }

  function
_handlePointer(handler, e) {
        if (e.pointerType ===
(e.MSPOINTER_TYPE_MOUSE ||
'mouse')) { return; }

        e.touches = [];
        for (var i in
_pointers) {

e.touches.push(_pointers[i]);
        }
        e.changedTouches =
[e];

        handler(e);
  }

  function
_onPointerStart(handler, e) {
        // IE10 specific:
MsTouch needs preventDefault.
See #2000
        if
```

```
(e.MSPOINTER_TYPE_TOUCH &&
e.pointerType ===
e.MSPOINTER_TYPE_TOUCH) {

preventDefault(e);
        }

_handlePointer(handler, e);
  }

  /*
   * Extends the event
handling code with double tap
support for mobile browsers.
   *
   * Note: currently most
browsers fire native dblclick,
with only a few exceptions
   * (see
https://github.com/Leaflet/Lea
flet/issues/7012#issuecomment-
595087386)
   */

  function makeDblclick(event)
{
        // in modern browsers
`type` cannot be just
overridden:
        //
https://developer.mozilla.org/
en-
US/docs/Web/JavaScript/Referen
ce/Errors/Getter_only
        var newEvent = {},
            prop, i;
        for (i in event) {
                prop =
event[i];
                newEvent[i] =
prop && prop.bind ?
prop.bind(event) : prop;
        }
        event = newEvent;
        newEvent.type =
'dblclick';
        newEvent.detail = 2;
```

```
        newEvent.isTrusted =
false;
        newEvent._simulated =
true; // for debug purposes
        return newEvent;
  }

  var delay = 200;
  function
addDoubleTapListener(obj,
handler) {
        // Most browsers
handle double tap natively

obj.addEventListener('dblclick
', handler);

        // On some platforms
the browser doesn't fire
native dblclicks for touch
events.
        // It seems that in
all such cases `detail`
property of `click` event is
always `1`.
        // So here we rely on
that fact to avoid excessive
'dblclick' simulation when not
needed.
        var last = 0,
            detail;
        function
simDblclick(e) {
                if (e.detail
!== 1) {
                        detail
= e.detail; // keep in sync to
avoid false dblclick in some
cases

return;
                }

                if
(e.pointerType === 'mouse' ||

(e.sourceCapabilities &&
```

```
!e.sourceCapabilities.firesTou
chEvents)) {


return;
                }

                // When
clicking on an <input>, the
browser generates a click on
its
                // <label>
(and vice versa) triggering
two clicks in quick
succession.
                // This
ignores clicks on elements
which are a label with a 'for'
                // attribute
(or children of such a label),
but not children of
                // a <input>.
                var path =
getPropagationPath(e);
                if
(path.some(function (el) {
                          return
el instanceof HTMLLabelElement
&& el.attributes.for;
                }) &&

!path.some(function (el) {

return (

el instanceof HTMLInputElement
||

el instanceof
HTMLSelectElement

);
                          })
                ) {

return;
                }
```

```
                var now =
Date.now();
                if (now - last
<= delay) {

detail++;
                        if
(detail === 2) {

handler(makeDblclick(e));
                        }
                } else {
                        detail
= 1;
                }
                last = now;
        }


obj.addEventListener('click',
simDblclick);

        return {
                dblclick:
handler,
                simDblclick:
simDblclick
        };
  }

  function
removeDoubleTapListener(obj,
handlers) {

obj.removeEventListener('dblcl
ick', handlers.dblclick);

obj.removeEventListener('click
', handlers.simDblclick);
  }

  /*
   * @namespace DomUtil
   *
   * Utility functions to work
with the [DOM]
```

```
(https://developer.mozilla.org
/docs/Web/API/Document_Object_
Model)
   * tree, used by Leaflet
internally.
   *
   * Most functions expecting
or returning a `HTMLElement`
also work for
   * SVG elements. The only
difference is that classes
refer to CSS classes
   * in HTML and SVG classes
in SVG.
   */


  // @property TRANSFORM:
String
  // Vendor-prefixed transform
style name (e.g.
`'webkitTransform'` for
WebKit).
  var TRANSFORM = testProp(
        ['transform',
'webkitTransform',
'OTransform', 'MozTransform',
'msTransform']);

  // webkitTransition comes
first because some browser
versions that drop vendor
prefix don't do
  // the same for the
transitionend event, in
particular the Android 4.1
stock browser

  // @property TRANSITION:
String
  // Vendor-prefixed
transition style name.
  var TRANSITION = testProp(
        ['webkitTransition',
'transition', 'OTransition',
'MozTransition',
'msTransition']);
```

```
  // @property TRANSITION_END:
String
  // Vendor-prefixed
transitionend event name.
  var TRANSITION_END =
        TRANSITION ===
'webkitTransition' ||
TRANSITION === 'OTransition' ?
TRANSITION + 'End' :
'transitionend';


  // @function get(id:
String|HTMLElement):
HTMLElement
  // Returns an element given
its DOM id, or returns the
element itself
  // if it was passed
directly.
  function get(id) {
        return typeof id ===
'string' ?
document.getElementById(id) :
id;
  }

  // @function getStyle(el:
HTMLElement, styleAttrib:
String): String
  // Returns the value for a
certain style attribute on an
element,
  // including computed values
or values set through CSS.
  function getStyle(el, style)
{
        var value =
el.style[style] ||
(el.currentStyle &&
el.currentStyle[style]);

        if ((!value || value
=== 'auto') &&
document.defaultView) {
                var css =
```

```
document.defaultView.getComput
edStyle(el, null);
                value = css ?
css[style] : null;
        }
        return value ===
'auto' ? null : value;
  }

  // @function create(tagName:
String, className?: String,
container?: HTMLElement):
HTMLElement
  // Creates an HTML element
with `tagName`, sets its class
to `className`, and optionally
appends it to `container`
element.
  function create$1(tagName,
className, container) {
        var el =
document.createElement(tagName
);
        el.className =
className || '';

        if (container) {

container.appendChild(el);
        }
        return el;
  }

  // @function remove(el:
HTMLElement)
  // Removes `el` from its
parent element
  function remove(el) {
        var parent =
el.parentNode;
        if (parent) {

parent.removeChild(el);
        }
  }

  // @function empty(el:
```

```
HTMLElement)
  // Removes all of `el`'s
children elements from `el`
  function empty(el) {
        while (el.firstChild)
{

el.removeChild(el.firstChild);
        }
  }

  // @function toFront(el:
HTMLElement)
  // Makes `el` the last child
of its parent, so it renders
in front of the other
children.
  function toFront(el) {
        var parent =
el.parentNode;
        if (parent &&
parent.lastChild !== el) {

parent.appendChild(el);
        }
  }

  // @function toBack(el:
HTMLElement)
  // Makes `el` the first
child of its parent, so it
renders behind the other
children.
  function toBack(el) {
        var parent =
el.parentNode;
        if (parent &&
parent.firstChild !== el) {

parent.insertBefore(el,
parent.firstChild);
        }
  }

  // @function hasClass(el:
HTMLElement, name: String):
Boolean
```

```
  // Returns `true` if the
element's class attribute
contains `name`.
  function hasClass(el, name)
{
        if (el.classList !==
undefined) {
                return
el.classList.contains(name);
        }
        var className =
getClass(el);
        return
className.length > 0 && new
RegExp('(^|\\s)' + name +
'(\\s|$)').test(className);
  }

  // @function addClass(el:
HTMLElement, name: String)
  // Adds `name` to the
element's class attribute.
  function addClass(el, name)
{
        if (el.classList !==
undefined) {
                var classes =
splitWords(name);
                for (var i =
0, len = classes.length; i <
len; i++) {

el.classList.add(classes[i]);
                }
        } else if
(!hasClass(el, name)) {
                var className
= getClass(el);
                setClass(el,
(className ? className + ' ' :
'') + name);
        }
  }

  // @function removeClass(el:
HTMLElement, name: String)
  // Removes `name` from the
```

```
element's class attribute.
  function removeClass(el,
name) {
        if (el.classList !==
undefined) {

el.classList.remove(name);
        } else {
                setClass(el,
trim((' ' + getClass(el) + '
').replace(' ' + name + ' ', '
')));
        }
  }

  // @function setClass(el:
HTMLElement, name: String)
  // Sets the element's class.
  function setClass(el, name)
{
        if
(el.className.baseVal ===
undefined) {
                el.className =
name;
        } else {
                // in case of
SVG element

el.className.baseVal = name;
        }
  }

  // @function getClass(el:
HTMLElement): String
  // Returns the element's
class.
  function getClass(el) {
        // Check if the
element is an
SVGElementInstance and use the
correspondingElement instead
        // (Required for
linked SVG elements in IE11.)
        if
(el.correspondingElement) {
                el =
```

```
el.correspondingElement;
        }
        return
el.className.baseVal ===
undefined ? el.className :
el.className.baseVal;
  }

  // @function setOpacity(el:
HTMLElement, opacity: Number)
  // Set the opacity of an
element (including old IE
support).
  // `opacity` must be a
number from `0` to `1`.
  function setOpacity(el,
value) {
        if ('opacity' in
el.style) {

el.style.opacity = value;
        } else if ('filter' in
el.style) {

_setOpacityIE(el, value);
        }
  }

  function _setOpacityIE(el,
value) {
        var filter = false,
            filterName =
'DXImageTransform.Microsoft.Al
pha';

        // filters collection
throws an error if we try to
retrieve a filter that doesn't
exist
        try {
                filter =
el.filters.item(filterName);
        } catch (e) {
                // don't set
opacity to 1 if we haven't
already set an opacity,
                // it isn't
```

```
needed and breaks transparent
pngs.
                if (value ===
1) { return; }
        }

        value =
Math.round(value * 100);

        if (filter) {
                filter.Enabled
= (value !== 100);
                filter.Opacity
= value;
        } else {

el.style.filter += ' progid:'
+ filterName + '(opacity=' +
value + ')';
        }
  }

  // @function testProp(props:
String[]): String|false
  // Goes through the array of
style names and returns the
first name
  // that is a valid style
name for an element. If no
such name is found,
  // it returns false. Useful
for vendor-prefixed styles
like `transform`.
  function testProp(props) {
        var style =
document.documentElement.style
;

        for (var i = 0; i <
props.length; i++) {
                if (props[i]
in style) {
                        return
props[i];
                }
        }
        return false;
```

```
  }

  // @function
setTransform(el: HTMLElement,
offset: Point, scale?: Number)
  // Resets the 3D CSS
transform of `el` so it is
translated by `offset` pixels
  // and optionally scaled by
`scale`. Does not have an
effect if the
  // browser doesn't support
3D CSS transforms.
  function setTransform(el,
offset, scale) {
        var pos = offset ||
new Point(0, 0);

        el.style[TRANSFORM] =
                (Browser.ie3d
?

'translate(' + pos.x + 'px,' +
pos.y + 'px)' :

'translate3d(' + pos.x + 'px,'
+ pos.y + 'px,0)') +
                (scale ? '
scale(' + scale + ')' : '');
  }

  // @function setPosition(el:
HTMLElement, position: Point)
  // Sets the position of `el`
to coordinates specified by
`position`,
  // using CSS translate or
top/left positioning depending
on the browser
  // (used by Leaflet
internally to position its
layers).
  function setPosition(el,
point) {

        /*eslint-disable */
        el._leaflet_pos =
```

```
point;
        /* eslint-enable */

        if (Browser.any3d) {

setTransform(el, point);
        } else {
                el.style.left
= point.x + 'px';
                el.style.top =
point.y + 'px';
        }
  }

  // @function getPosition(el:
HTMLElement): Point
  // Returns the coordinates
of an element previously
positioned with setPosition.
  function getPosition(el) {
        // this method is only
used for elements previously
positioned using setPosition,
        // so it's safe to
cache the position for
performance

        return el._leaflet_pos
|| new Point(0, 0);
  }

  // @function
disableTextSelection()
  // Prevents the user from
generating `selectstart` DOM
events, usually generated
  // when the user drags the
mouse through a page with
text. Used internally
  // by Leaflet to override
the behaviour of any click-
and-drag interaction on
  // the map. Affects drag
interactions on the whole
document.

  // @function
```

```
enableTextSelection()
  // Cancels the effects of a
previous
[`L.DomUtil.disableTextSelecti
on`](#domutil-
disabletextselection).
  var disableTextSelection;
  var enableTextSelection;
  var _userSelect;
  if ('onselectstart' in
document) {
        disableTextSelection =
function () {
                on(window,
'selectstart',
preventDefault);
        };
        enableTextSelection =
function () {
                off(window,
'selectstart',
preventDefault);
        };
  } else {
        var userSelectProperty
= testProp(
                ['userSelect',
'WebkitUserSelect',
'OUserSelect',
'MozUserSelect',
'msUserSelect']);

        disableTextSelection =
function () {
                if
(userSelectProperty) {
                        var
style =
document.documentElement.style
;

_userSelect =
style[userSelectProperty];

style[userSelectProperty] =
'none';
                }
```

```
        };
        enableTextSelection =
function () {
                if
(userSelectProperty) {

document.documentElement.style
[userSelectProperty] =
_userSelect;

_userSelect = undefined;
                }
        };
  }

  // @function
disableImageDrag()
  // As
[`L.DomUtil.disableTextSelecti
on`](#domutil-
disabletextselection), but
  // for `dragstart` DOM
events, usually generated when
the user drags an image.
  function disableImageDrag()
{
        on(window,
'dragstart', preventDefault);
  }

  // @function
enableImageDrag()
  // Cancels the effects of a
previous
[`L.DomUtil.disableImageDrag`]
(#domutil-
disabletextselection).
  function enableImageDrag() {
        off(window,
'dragstart', preventDefault);
  }

  var _outlineElement,
_outlineStyle;
  // @function
preventOutline(el:
HTMLElement)
```

```
  // Makes the [outline]
(https://developer.mozilla.org
/docs/Web/CSS/outline)
  // of the element `el`
invisible. Used internally by
Leaflet to prevent
  // focusable elements from
displaying an outline when the
user performs a
  // drag interaction on them.
  function
preventOutline(element) {
        while
(element.tabIndex === -1) {
                element =
element.parentNode;
        }
        if (!element.style) {
return; }
        restoreOutline();
        _outlineElement =
element;
        _outlineStyle =
element.style.outline;
        element.style.outline
= 'none';
        on(window, 'keydown',
restoreOutline);
  }

  // @function
restoreOutline()
  // Cancels the effects of a
previous
[`L.DomUtil.preventOutline`]
().
  function restoreOutline() {
        if (!_outlineElement)
{ return; }

_outlineElement.style.outline
= _outlineStyle;
        _outlineElement =
undefined;
        _outlineStyle =
undefined;
        off(window, 'keydown',
```

```
restoreOutline);
  }

  // @function
getSizedParentNode(el:
HTMLElement): HTMLElement
  // Finds the closest parent
node which size (width and
height) is not null.
  function
getSizedParentNode(element) {
        do {
                element =
element.parentNode;
        } while
((!element.offsetWidth ||
!element.offsetHeight) &&
element !== document.body);
        return element;
  }

  // @function getScale(el:
HTMLElement): Object
  // Computes the CSS scale
currently applied on the
element.
  // Returns an object with
`x` and `y` members as
horizontal and vertical scales
respectively,
  // and `boundingClientRect`
as the result of
[`getBoundingClientRect()`]
(https://developer.mozilla.org
/en-
US/docs/Web/API/Element/getBou
ndingClientRect).
  function getScale(element) {
        var rect =
element.getBoundingClientRect(
); // Read-only in old
browsers.

        return {
                x: rect.width
/ element.offsetWidth || 1,
                y: rect.height
```

```
/ element.offsetHeight || 1,

boundingClientRect: rect
        };
  }

  var DomUtil = {
    __proto__: null,
    TRANSFORM: TRANSFORM,
    TRANSITION: TRANSITION,
    TRANSITION_END:
TRANSITION_END,
    get: get,
    getStyle: getStyle,
    create: create$1,
    remove: remove,
    empty: empty,
    toFront: toFront,
    toBack: toBack,
    hasClass: hasClass,
    addClass: addClass,
    removeClass: removeClass,
    setClass: setClass,
    getClass: getClass,
    setOpacity: setOpacity,
    testProp: testProp,
    setTransform:
setTransform,
    setPosition: setPosition,
    getPosition: getPosition,
    get disableTextSelection
() { return
disableTextSelection; },
    get enableTextSelection ()
{ return enableTextSelection;
},
    disableImageDrag:
disableImageDrag,
    enableImageDrag:
enableImageDrag,
    preventOutline:
preventOutline,
    restoreOutline:
restoreOutline,
    getSizedParentNode:
getSizedParentNode,
    getScale: getScale
```

```
  };

  /*
   * @namespace DomEvent
   * Utility functions to work
with the [DOM events]
(https://developer.mozilla.org
/docs/Web/API/Event), used by
Leaflet internally.
   */

  // Inspired by John Resig,
Dean Edwards and YUI addEvent
implementations.

  // @function on(el:
HTMLElement, types: String,
fn: Function, context?:
Object): this
  // Adds a listener function
(`fn`) to a particular DOM
event type of the
  // element `el`. You can
optionally specify the context
of the listener
  // (object the `this`
keyword will point to). You
can also pass several
  // space-separated types
(e.g. `'click dblclick'`).

  // @alternative
  // @function on(el:
HTMLElement, eventMap: Object,
context?: Object): this
  // Adds a set of
type/listener pairs, e.g.
`{click: onClick, mousemove:
onMouseMove}`
  function on(obj, types, fn,
context) {

        if (types && typeof
types === 'object') {
                for (var type
in types) {
```

```
addOne(obj, type, types[type],
fn);
                }
        } else {
                types =
splitWords(types);

                for (var i =
0, len = types.length; i <
len; i++) {

addOne(obj, types[i], fn,
context);
                }
        }

        return this;
  }

  var eventsKey =
'_leaflet_events';

  // @function off(el:
HTMLElement, types: String,
fn: Function, context?:
Object): this
  // Removes a previously
added listener function.
  // Note that if you passed a
custom context to on, you must
pass the same
  // context to `off` in order
to remove the listener.

  // @alternative
  // @function off(el:
HTMLElement, eventMap: Object,
context?: Object): this
  // Removes a set of
type/listener pairs, e.g.
`{click: onClick, mousemove:
onMouseMove}`

  // @alternative
  // @function off(el:
HTMLElement, types: String):
this
```

```
  // Removes all previously
added listeners of given
types.

  // @alternative
  // @function off(el:
HTMLElement): this
  // Removes all previously
added listeners from given
HTMLElement
  function off(obj, types, fn,
context) {

        if (arguments.length
=== 1) {

batchRemove(obj);
                delete
obj[eventsKey];

        } else if (types &&
typeof types === 'object') {
                for (var type
in types) {

removeOne(obj, type,
types[type], fn);
                }

        } else {
                types =
splitWords(types);

                if
(arguments.length === 2) {

batchRemove(obj, function
(type) {

return indexOf(types, type)
!== -1;
                        });
                } else {
                        for
(var i = 0, len =
types.length; i < len; i++) {
```

```
removeOne(obj, types[i], fn,
context);
                        }
                }
        }

        return this;
  }

  function batchRemove(obj,
filterFn) {
        for (var id in
obj[eventsKey]) {
                var type =
id.split(/\d/)[0];
                if (!filterFn
|| filterFn(type)) {

removeOne(obj, type, null,
null, id);
                }
        }
  }

  var mouseSubst = {
        mouseenter:
'mouseover',
        mouseleave:
'mouseout',
        wheel: !('onwheel' in
window) && 'mousewheel'
  };

  function addOne(obj, type,
fn, context) {
        var id = type +
stamp(fn) + (context ? '_' +
stamp(context) : '');

        if (obj[eventsKey] &&
obj[eventsKey][id]) { return
this; }

        var handler = function
(e) {
                return
fn.call(context || obj, e ||
```

```
window.event);
        };

        var originalHandler =
handler;

        if
(!Browser.touchNative &&
Browser.pointer &&
type.indexOf('touch') === 0) {
                // Needs
DomEvent.Pointer.js
                handler =
addPointerListener(obj, type,
handler);

        } else if
(Browser.touch && (type ===
'dblclick')) {
                handler =
addDoubleTapListener(obj,
handler);

        } else if
('addEventListener' in obj) {

                if (type ===
'touchstart' || type ===
'touchmove' || type ===
'wheel' ||  type ===
'mousewheel') {

obj.addEventListener(mouseSubs
t[type] || type, handler,
Browser.passiveEvents ?
{passive: false} : false);

                } else if
(type === 'mouseenter' || type
=== 'mouseleave') {

handler = function (e) {

e = e || window.event;

if (isExternalTarget(obj, e))
{
```

```
originalHandler(e);

}
                        };

obj.addEventListener(mouseSubs
t[type], handler, false);

                } else {

obj.addEventListener(type,
originalHandler, false);
                }

        } else {

obj.attachEvent('on' + type,
handler);
        }

        obj[eventsKey] =
obj[eventsKey] || {};
        obj[eventsKey][id] =
handler;
  }

  function removeOne(obj,
type, fn, context, id) {
        id = id || type +
stamp(fn) + (context ? '_' +
stamp(context) : '');
        var handler =
obj[eventsKey] &&
obj[eventsKey][id];

        if (!handler) { return
this; }

        if
(!Browser.touchNative &&
Browser.pointer &&
type.indexOf('touch') === 0) {

removePointerListener(obj,
type, handler);
```

```
        } else if
(Browser.touch && (type ===
'dblclick')) {

removeDoubleTapListener(obj,
handler);

        } else if
('removeEventListener' in obj)
{


obj.removeEventListener(mouseS
ubst[type] || type, handler,
false);

        } else {

obj.detachEvent('on' + type,
handler);
        }

        obj[eventsKey][id] =
null;
  }

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

        if (e.stopPropagation)
{
```

```
e.stopPropagation();
        } else if
(e.originalEvent) {  // In
case of Leaflet event.

e.originalEvent._stopped =
true;
        } else {
                e.cancelBubble
= true;
        }

        return this;
  }

  // @function
disableScrollPropagation(el:
HTMLElement): this
  // Adds `stopPropagation` to
the element's `'wheel'` events
(plus browser variants).
  function
disableScrollPropagation(el) {
        addOne(el, 'wheel',
stopPropagation);
        return this;
  }

  // @function
disableClickPropagation(el:
HTMLElement): this
  // Adds `stopPropagation` to
the element's `'click'`,
`'dblclick'`, `'contextmenu'`,
  // `'mousedown'` and
`'touchstart'` events (plus
browser variants).
  function
disableClickPropagation(el) {
        on(el, 'mousedown
touchstart dblclick
contextmenu',
stopPropagation);

el['_leaflet_disable_click'] =
true;
        return this;
```

```
  }

  // @function
preventDefault(ev: DOMEvent):
this
  // Prevents the default
action of the DOM Event `ev`
from happening (such as
  // following a link in the
href of the a element, or
doing a POST request
  // with page reload when a
`<form>` is submitted).
  // Use it inside listener
functions.
  function preventDefault(e) {
        if (e.preventDefault)
{

e.preventDefault();
        } else {
                e.returnValue
= false;
        }
        return this;
  }

  // @function stop(ev:
DOMEvent): this
  // Does `stopPropagation`
and `preventDefault` at the
same time.
  function stop(e) {
        preventDefault(e);
        stopPropagation(e);
        return this;
  }

  // @function
getPropagationPath(ev:
DOMEvent): Array
  // Compatibility polyfill
for [`Event.composedPath()`]
(https://developer.mozilla.org
/en-
US/docs/Web/API/Event/composed
Path).
```

```
  // Returns an array
containing the `HTMLElement`s
that the given DOM event
  // should propagate to (if
not stopped).
  function
getPropagationPath(ev) {
        if (ev.composedPath) {
                return
ev.composedPath();
        }

        var path = [];
        var el = ev.target;

        while (el) {
                path.push(el);
                el =
el.parentNode;
        }
        return path;
  }


  // @function
getMousePosition(ev: DOMEvent,
container?: HTMLElement):
Point
  // Gets normalized mouse
position from a DOM event
relative to the
  // `container` (border
excluded) or to the whole page
if not specified.
  function getMousePosition(e,
container) {
        if (!container) {
                return new
Point(e.clientX, e.clientY);
        }

        var scale =
getScale(container),
            offset =
scale.boundingClientRect; //
left and top  values are in
page scale (like the event
```

```
clientX/Y)

        return new Point(
                //
offset.left/top values are in
page scale (like clientX/Y),
                // whereas
clientLeft/Top (border width)
values are the original values
(before CSS scale applies).
                (e.clientX -
offset.left) / scale.x -
container.clientLeft,
                (e.clientY -
offset.top) / scale.y -
container.clientTop
        );
  }


  //  except , Safari and
  // We need double the scroll
pixels (see #7403 and #4538)
for all Browsers
  // except OSX (Mac) -> 3x,
Chrome running on Linux 1x

  var wheelPxFactor =
        (Browser.linux &&
Browser.chrome) ?
window.devicePixelRatio :
        Browser.mac ?
window.devicePixelRatio * 3 :

window.devicePixelRatio > 0 ?
2 * window.devicePixelRatio :
1;
  // @function
getWheelDelta(ev: DOMEvent):
Number
  // Gets normalized wheel
delta from a wheel DOM event,
in vertical
  // pixels scrolled (negative
if scrolling down).
  // Events from pointing
devices without precise
```

```
scrolling are mapped to
  // a best guess of 60
pixels.
  function getWheelDelta(e) {
        return (Browser.edge)
? e.wheelDeltaY / 2 : // Don't
trust window-geometry-based
delta
                (e.deltaY &&
e.deltaMode === 0) ? -e.deltaY
/ wheelPxFactor : // Pixels
                (e.deltaY &&
e.deltaMode === 1) ? -e.deltaY
* 20 : // Lines
                (e.deltaY &&
e.deltaMode === 2) ? -e.deltaY
* 60 : // Pages
                (e.deltaX ||
e.deltaZ) ? 0 : // Skip
horizontal/depth wheel events
                e.wheelDelta ?
(e.wheelDeltaY ||
e.wheelDelta) / 2 : // Legacy
IE pixels
                (e.detail &&
Math.abs(e.detail) < 32765) ?
-e.detail * 20 : // Legacy Moz
lines
                e.detail ?
e.detail / -32765 * 60 : //
Legacy Moz pages
                0;
  }

  // check if element really
left/entered the event target
(for mouseenter/mouseleave)
  function
isExternalTarget(el, e) {

        var related =
e.relatedTarget;

        if (!related) { return
true; }

        try {
```

```
                while (related
&& (related !== el)) {

related = related.parentNode;
                }
        } catch (err) {
                return false;
        }
        return (related !==
el);
  }

  var DomEvent = {
    __proto__: null,
    on: on,
    off: off,
    stopPropagation:
stopPropagation,
    disableScrollPropagation:
disableScrollPropagation,
    disableClickPropagation:
disableClickPropagation,
    preventDefault:
preventDefault,
    stop: stop,
    getPropagationPath:
getPropagationPath,
    getMousePosition:
getMousePosition,
    getWheelDelta:
getWheelDelta,
    isExternalTarget:
isExternalTarget,
    addListener: on,
    removeListener: off
  };

  /*
   * @class PosAnimation
   * @aka L.PosAnimation
   * @inherits Evented
   * Used internally for
panning animations, utilizing
CSS3 Transitions for modern
browsers and a timer fallback
for IE6-9.
   *
```

```
   * @example
   * ```js
   * var myPositionMarker =
L.marker([48.864716,
2.294694]).addTo(map);
   *
   *
myPositionMarker.on("click",
function() {
   *    var pos =
map.latLngToLayerPoint(myPosit
ionMarker.getLatLng());
   *    pos.y -= 25;
   *    var fx = new
L.PosAnimation();
   *
   *
fx.once('end',function() {
   *            pos.y += 25;
   *
fx.run(myPositionMarker._icon,
pos, 0.8);
   *    });
   *
   *
fx.run(myPositionMarker._icon,
pos, 0.3);
   * });
   *
   * ```
   *
   * @constructor
L.PosAnimation()
   * Creates a `PosAnimation`
object.
   *
   */

  var PosAnimation =
Evented.extend({

        // @method run(el:
HTMLElement, newPos: Point,
duration?: Number,
easeLinearity?: Number)
        // 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]
(https://cubic-
bezier.com/#0,0,.5,1),
        // `0.5` by default).
        run: function (el,
newPos, duration,
easeLinearity) {
                this.stop();

                this._el = el;

this._inProgress = true;
                this._duration
= duration || 0.25;

this._easeOutPower = 1 /
Math.max(easeLinearity || 0.5,
0.2);

                this._startPos
= getPosition(el);
                this._offset =
newPos.subtract(this._startPos
);

this._startTime = +new Date();

                // @event
start: Event
                // Fired when
the animation starts

this.fire('start');


this._animate();
        },

        // @method stop()
        // Stops the animation
(if currently running).
        stop: function () {
```

```
                if
(!this._inProgress) { return;
}


this._step(true);

this._complete();
        },

        _animate: function ()
{
                // animation
loop
                this._animId =
requestAnimFrame(this._animate
, this);
                this._step();
        },

        _step: function
(round) {
                var elapsed =
(+new Date()) -
this._startTime,
                    duration =
this._duration * 1000;

                if (elapsed <
duration) {

this._runFrame(this._easeOut(e
lapsed / duration), round);
                } else {

this._runFrame(1);

this._complete();
                }
        },

        _runFrame: function
(progress, round) {
                var pos =
this._startPos.add(this._offse
t.multiplyBy(progress));
                if (round) {
```

```
pos._round();
                }

setPosition(this._el, pos);

                // @event
step: Event
                // Fired
continuously during the
animation.

this.fire('step');
        },

        _complete: function ()
{

cancelAnimFrame(this._animId);


this._inProgress = false;
                // @event end:
Event
                // Fired when
the animation ends.

this.fire('end');
        },

        _easeOut: function (t)
{
                return 1 -
Math.pow(1 - t,
this._easeOutPower);
        }
  });

  /*
   * @class Map
   * @aka L.Map
   * @inherits Evented
   *
   * The central class of the
API — it is used to create a
map on a page and manipulate
it.
```

```
   *
   * @example
   *
   * ```js
   * // 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
   * });
   * ```
   *
   */

  var Map = Evented.extend({

        options: {
                // @section
Map State Options
                // @option
crs: CRS = L.CRS.EPSG3857
                // The
[Coordinate Reference System]
(#crs) to use. Don't change
this if you're not
                // sure what
it means.
                crs: EPSG3857,

                // @option
center: LatLng = undefined
                // Initial
geographic center of the map
                center:
undefined,

                // @option
zoom: Number = undefined
                // Initial map
zoom level
                zoom:
undefined,

                // @option
minZoom: Number = *
```

```
                // Minimum
zoom level of the map.
                // If not
specified and at least one
`GridLayer` or `TileLayer` is
in the map,
                // the lowest
of their `minZoom` options
will be used instead.
                minZoom:
undefined,

                // @option
maxZoom: Number = *
                // Maximum
zoom level of the map.
                // If not
specified and at least one
`GridLayer` or `TileLayer` is
in the map,
                // the highest
of their `maxZoom` options
will be used instead.
                maxZoom:
undefined,

                // @option
layers: Layer[] = []
                // Array of
layers that will be added to
the map initially
                layers: [],

                // @option
maxBounds: LatLngBounds = null
                // When this
option is set, the map
restricts the view to the
given
                //
geographical bounds, bouncing
the user back if the user
tries to pan
                // outside the
view. To set the restriction
dynamically, use
                //
```

```
[`setMaxBounds`](#map-
setmaxbounds) method.
                maxBounds:
undefined,

                // @option
renderer: Renderer = *
                // The default
method for drawing vector
layers on the map. `L.SVG`
                // or
`L.Canvas` by default
depending on browser support.
                renderer:
undefined,


                // @section
Animation Options
                // @option
zoomAnimation: Boolean = true
                // Whether the
map zoom animation is enabled.
By default it's enabled
                // in all
browsers that support CSS3
Transitions except Android.
                zoomAnimation:
true,

                // @option
zoomAnimationThreshold: Number
= 4
                // Won't
animate zoom if the zoom
difference exceeds this value.

zoomAnimationThreshold: 4,

                // @option
fadeAnimation: Boolean = true
                // Whether the
tile fade animation is
enabled. By default it's
enabled
                // in all
browsers that support CSS3
```

```
Transitions except Android.
		fadeAnimation:
true,

		// @option
markerZoomAnimation: Boolean =
true
		// Whether
markers animate their zoom
with the zoom animation, if
disabled
		// they will
disappear for the length of
the animation. By default it's
		// enabled in
all browsers that support CSS3
Transitions except Android.

markerZoomAnimation: true,

		// @option
transform3DLimit: Number =
2^23
		// Defines the
maximum size of a CSS
translation transform. The
default
		// value
should not be changed unless a
web browser positions layers
in
		// the wrong
place after doing a large
`panBy`.

transform3DLimit: 8388608, //
Precision limit of a 32-bit
float

		// @section
Interaction Options
		// @option
zoomSnap: Number = 1
		// Forces the
map's zoom level to always be
a multiple of this,
particularly
```

```
                // right after
a [`fitBounds()`](#map-
fitbounds) or a pinch-zoom.
                // By default,
the zoom level snaps to the
nearest integer; lower values
                // (e.g. `0.5`
or `0.1`) allow for greater
granularity. A value of `0`
                // means the
zoom level will not be snapped
after `fitBounds` or a pinch-
zoom.
                zoomSnap: 1,

                // @option
zoomDelta: Number = 1
                // Controls
how much the map's zoom level
will change after a
                //
[`zoomIn()`](#map-zoomin),
[`zoomOut()`](#map-zoomout),
pressing `+`
                // or `-` on
the keyboard, or using the
[zoom controls](#control-
zoom).
                // Values
smaller than `1` (e.g. `0.5`)
allow for greater granularity.
                zoomDelta: 1,

                // @option
trackResize: Boolean = true
                // Whether the
map automatically handles
browser window resize to
update itself.
                trackResize:
true
        },

        initialize: function
(id, options) { //
(HTMLElement or String,
Object)
```

```
                options =
setOptions(this, options);

                // Make sure
to assign internal flags at
the beginning,
                // to avoid
inconsistent state in some
edge cases.
                this._handlers
= [];
                this._layers =
{};

this._zoomBoundLayers = {};

this._sizeChanged = true;


this._initContainer(id);

this._initLayout();

                // hack for
https://github.com/Leaflet/Lea
flet/issues/1980
                this._onResize
= bind(this._onResize, this);


this._initEvents();

                if
(options.maxBounds) {

this.setMaxBounds(options.maxB
ounds);
                }

                if
(options.zoom !== undefined) {

this._zoom =
this._limitZoom(options.zoom);
                }

                if
```

```
(options.center &&
options.zoom !== undefined) {

this.setView(toLatLng(options.
center), options.zoom, {reset:
true});
                }


this.callInitHooks();

                // don't
animate on browsers without
hardware-accelerated
transitions or old
Android/Opera

this._zoomAnimated =
TRANSITION && Browser.any3d &&
!Browser.mobileOpera &&

this.options.zoomAnimation;

                // zoom
transitions run with the same
duration for all layers, so if
one of transitionend events
                // happens
after starting zoom animation
(propagating to the map pane),
we know that it ended globally
                if
(this._zoomAnimated) {

this._createAnimProxy();

on(this._proxy,
TRANSITION_END,
this._catchTransitionEnd,
this);
                }


this._addLayers(this.options.l
ayers);
        },
```

```
        // @section Methods
for modifying map state

        // @method
setView(center: LatLng, zoom:
Number, options?: Zoom/pan
options): this
        // Sets the view of
the map (geographical center
and zoom) with the given
        // animation options.
        setView: function
(center, zoom, options) {

                zoom = zoom
=== undefined ? this._zoom :
this._limitZoom(zoom);
                center =
this._limitCenter(toLatLng(cen
ter), zoom,
this.options.maxBounds);
                options =
options || {};

                this._stop();

                if
(this._loaded &&
!options.reset && options !==
true) {

                        if
(options.animate !==
undefined) {

options.zoom =
extend({animate:
options.animate},
options.zoom);

options.pan = extend({animate:
options.animate, duration:
options.duration},
options.pan);
                        }
```

```
                        // try
animating pan or zoom
                        var
moved = (this._zoom !== zoom)
?

this._tryAnimatedZoom &&
this._tryAnimatedZoom(center,
zoom, options.zoom) :

this._tryAnimatedPan(center,
options.pan);

                        if
(moved) {

// prevent resize handler
call, the view will refresh
after animation anyway

clearTimeout(this._sizeTimer);

return this;
                        }
                }

                // animation
didn't start, just reset the
map view

this._resetView(center, zoom,
options.pan &&
options.pan.noMoveStart);

                return this;
        },

        // @method
setZoom(zoom: Number,
options?: Zoom/pan options):
this
        // Sets the zoom of
the map.
        setZoom: function
(zoom, options) {
                if
(!this._loaded) {
```

```
this._zoom = zoom;
                        return
this;
                }
                return
this.setView(this.getCenter(),
zoom, {zoom: options});
        },

        // @method
zoomIn(delta?: Number,
options?: Zoom options): this
        // Increases the zoom
of the map by `delta`
([`zoomDelta`](#map-zoomdelta)
by default).
        zoomIn: function
(delta, options) {
                delta = delta
|| (Browser.any3d ?
this.options.zoomDelta : 1);
                return
this.setZoom(this._zoom +
delta, options);
        },

        // @method
zoomOut(delta?: Number,
options?: Zoom options): this
        // Decreases the zoom
of the map by `delta`
([`zoomDelta`](#map-zoomdelta)
by default).
        zoomOut: function
(delta, options) {
                delta = delta
|| (Browser.any3d ?
this.options.zoomDelta : 1);
                return
this.setZoom(this._zoom -
delta, options);
        },

        // @method
setZoomAround(latlng: LatLng,
zoom: Number, options: Zoom
```

```
options): this
        // Zooms the map while
keeping a specified
geographical point on the map
        // stationary (e.g.
used internally for scroll
zoom and double-click zoom).
        // @alternative
        // @method
setZoomAround(offset: Point,
zoom: Number, options: Zoom
options): this
        // Zooms the map while
keeping a specified pixel on
the map (relative to the top-
left corner) stationary.
        setZoomAround:
function (latlng, zoom,
options) {
                var scale =
this.getZoomScale(zoom),
                    viewHalf =
this.getSize().divideBy(2),

containerPoint = latlng
instanceof Point ? latlng :
this.latLngToContainerPoint(la
tlng),


centerOffset =
containerPoint.subtract(viewHa
lf).multiplyBy(1 - 1 / scale),
                    newCenter
=
this.containerPointToLatLng(vi
ewHalf.add(centerOffset));

                return
this.setView(newCenter, zoom,
{zoom: options});
        },

        _getBoundsCenterZoom:
function (bounds, options) {

                options =
```

```
options || {};
                bounds =
bounds.getBounds ?
bounds.getBounds() :
toLatLngBounds(bounds);

                var paddingTL
=
toPoint(options.paddingTopLeft
|| options.padding || [0, 0]),
                    paddingBR
=
toPoint(options.paddingBottomR
ight || options.padding || [0,
0]),

                    zoom =
this.getBoundsZoom(bounds,
false,
paddingTL.add(paddingBR));

                zoom = (typeof
options.maxZoom === 'number')
? Math.min(options.maxZoom,
zoom) : zoom;

                if (zoom ===
Infinity) {
                        return
{

center: bounds.getCenter(),

zoom: zoom
                        };
                }

                var
paddingOffset =
paddingBR.subtract(paddingTL).
divideBy(2),

                    swPoint =
this.project(bounds.getSouthWe
st(), zoom),
                    nePoint =
this.project(bounds.getNorthEa
```

```
st(), zoom),
                        center =
this.unproject(swPoint.add(neP
oint).divideBy(2).add(paddingO
ffset), zoom);

                return {

center: center,
                            zoom:
zoom
                };
        },

        // @method
fitBounds(bounds:
LatLngBounds, options?:
fitBounds options): this
        // Sets a map view
that contains the given
geographical bounds with the
        // maximum zoom level
possible.
        fitBounds: function
(bounds, options) {

                bounds =
toLatLngBounds(bounds);

                if
(!bounds.isValid()) {
                            throw
new Error('Bounds are not
valid.');
                }

                var target =
this._getBoundsCenterZoom(boun
ds, options);
                return
this.setView(target.center,
target.zoom, options);
        },

        // @method
fitWorld(options?: fitBounds
options): this
```

```
        // Sets a map view
that mostly contains the whole
world with the maximum
        // zoom level
possible.
        fitWorld: function
(options) {
                return
this.fitBounds([[-90, -180],
[90, 180]], options);
        },

        // @method
panTo(latlng: LatLng,
options?: Pan options): this
        // Pans the map to a
given center.
        panTo: function
(center, options) { //
(LatLng)
                return
this.setView(center,
this._zoom, {pan: options});
        },

        // @method
panBy(offset: Point, options?:
Pan options): this
        // Pans the map by a
given number of pixels
(animated).
        panBy: function
(offset, options) {
                offset =
toPoint(offset).round();
                options =
options || {};

                if (!offset.x
&& !offset.y) {
                        return
this.fire('moveend');
                }
                // If we pan
too far, Chrome gets issues
with tiles
                // and makes
```

```
them disappear or appear in
the wrong place (slightly
offset) #2602
                if
(options.animate !== true &&
!this.getSize().contains(offse
t)) {

this._resetView(this.unproject
(this.project(this.getCenter()
).add(offset)),
this.getZoom());
                        return
this;
                }

                if
(!this._panAnim) {

this._panAnim = new
PosAnimation();


this._panAnim.on({

'step':
this._onPanTransitionStep,

'end':
this._onPanTransitionEnd
                        },
this);
                }

                // don't fire
movestart if animating inertia
                if
(!options.noMoveStart) {

this.fire('movestart');
                }

                // animate pan
unless animate: false
specified
                if
(options.animate !== false) {
```

```
addClass(this._mapPane,
'leaflet-pan-anim');

                        var
newPos =
this._getMapPanePos().subtract
(offset).round();

this._panAnim.run(this._mapPan
e, newPos, options.duration ||
0.25, options.easeLinearity);
                } else {

this._rawPanBy(offset);

this.fire('move').fire('moveen
d');
                }

                return this;
        },

        // @method
flyTo(latlng: LatLng, zoom?:
Number, options?: Zoom/pan
options): this
        // Sets the view of
the map (geographical center
and zoom) performing a smooth
        // pan-zoom animation.
        flyTo: function
(targetCenter, targetZoom,
options) {

                options =
options || {};
                if
(options.animate === false ||
!Browser.any3d) {
                        return
this.setView(targetCenter,
targetZoom, options);
                }

                this._stop();
```

```
                var from =
this.project(this.getCenter())
,
                    to =
this.project(targetCenter),
                    size =
this.getSize(),
                    startZoom
= this._zoom;

                targetCenter =
toLatLng(targetCenter);
                targetZoom =
targetZoom === undefined ?
startZoom : targetZoom;

                var w0 =
Math.max(size.x, size.y),
                    w1 = w0 *
this.getZoomScale(startZoom,
targetZoom),
                    u1 =
(to.distanceTo(from)) || 1,
                    rho =
1.42,
                    rho2 = rho
* rho;

                function r(i)
{
                        var s1
= i ? -1 : 1,
                            s2
= i ? w1 : w0,
                            t1
= w1 * w1 - w0 * w0 + s1 *
rho2 * rho2 * u1 * u1,
                            b1
= 2 * s2 * rho2 * u1,
                            b
= t1 / b1,
                            sq
= Math.sqrt(b * b + 1) - b;

                            //
workaround for floating point
precision bug when sq = 0, log
```

```
= -Infinite,
                                //
thus triggering an infinite
loop in flyTo

var log = sq < 0.000000001 ?
-18 : Math.log(sq);

                          return
log;
                }

                function
sinh(n) { return (Math.exp(n)
- Math.exp(-n)) / 2; }
                function
cosh(n) { return (Math.exp(n)
+ Math.exp(-n)) / 2; }
                function
tanh(n) { return sinh(n) /
cosh(n); }

                var r0 = r(0);

                function w(s)
{ return w0 * (cosh(r0) /
cosh(r0 + rho * s)); }
                function u(s)
{ return w0 * (cosh(r0) *
tanh(r0 + rho * s) - sinh(r0))
/ rho2; }

                function
easeOut(t) { return 1 -
Math.pow(1 - t, 1.5); }

                var start =
Date.now(),
                    S = (r(1)
- r0) / rho,
                    duration =
options.duration ? 1000 *
options.duration : 1000 * S *
0.8;

                function
frame() {
```

```
                        var t
= (Date.now() - start) /
duration,
                            s
= easeOut(t) * S;

                        if (t
<= 1) {

this._flyToFrame =
requestAnimFrame(frame, this);


this._move(

this.unproject(from.add(to.sub
tract(from).multiplyBy(u(s) /
u1)), startZoom),

this.getScaleZoom(w0 / w(s),
startZoom),

{flyTo: true});

                        } else
{

this

._move(targetCenter,
targetZoom)

._moveEnd(true);
                        }
                }


this._moveStart(true,
options.noMoveStart);


frame.call(this);
                return this;
        },

        // @method
flyToBounds(bounds:
```

```
LatLngBounds, options?:
fitBounds options): this
        // Sets the view of
the map with a smooth
animation like [`flyTo`](#map-
flyto),
        // but takes a bounds
parameter like [`fitBounds`]
(#map-fitbounds).
        flyToBounds: function
(bounds, options) {
                var target =
this._getBoundsCenterZoom(boun
ds, options);
                return
this.flyTo(target.center,
target.zoom, options);
        },

        // @method
setMaxBounds(bounds:
LatLngBounds): this
        // Restricts the map
view to the given bounds (see
the [maxBounds](#map-
maxbounds) option).
        setMaxBounds: function
(bounds) {
                bounds =
toLatLngBounds(bounds);

                if
(this.listens('moveend',
this._panInsideMaxBounds)) {

this.off('moveend',
this._panInsideMaxBounds);
                }

                if
(!bounds.isValid()) {

this.options.maxBounds = null;
                        return
this;
                }
```

```
this.options.maxBounds =
bounds;

                if
(this._loaded) {

this._panInsideMaxBounds();
                }

                return
this.on('moveend',
this._panInsideMaxBounds);
        },

        // @method
setMinZoom(zoom: Number): this
        // Sets the lower
limit for the available zoom
levels (see the [minZoom]
(#map-minzoom) option).
        setMinZoom: function
(zoom) {
                var oldZoom =
this.options.minZoom;

this.options.minZoom = zoom;

                if
(this._loaded && oldZoom !==
zoom) {

this.fire('zoomlevelschange');

                        if
(this.getZoom() <
this.options.minZoom) {

return this.setZoom(zoom);
                        }
                }

                return this;
        },

        // @method
setMaxZoom(zoom: Number): this
```

```
        // Sets the upper
limit for the available zoom
levels (see the [maxZoom]
(#map-maxzoom) option).
        setMaxZoom: function
(zoom) {
                var oldZoom =
this.options.maxZoom;

this.options.maxZoom = zoom;

                if
(this._loaded && oldZoom !==
zoom) {

this.fire('zoomlevelschange');

                        if
(this.getZoom() >
this.options.maxZoom) {

return this.setZoom(zoom);
                        }
                }

                return this;
        },

        // @method
panInsideBounds(bounds:
LatLngBounds, options?: Pan
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.
        panInsideBounds:
function (bounds, options) {

this._enforcingBounds = true;
                var center =
this.getCenter(),
                    newCenter
= this._limitCenter(center,
this._zoom,
```

```
toLatLngBounds(bounds));

                if
(!center.equals(newCenter)) {

this.panTo(newCenter,
options);
                }


this._enforcingBounds = false;
                return this;
        },

        // @method
panInside(latlng: LatLng,
options?: padding options):
this
        // Pans the map the
minimum amount to make the
`latlng` visible. Use
        // padding options to
fit the display to more
restricted bounds.
        // If `latlng` is
already within the (optionally
padded) display bounds,
        // the map will not be
panned.
        panInside: function
(latlng, options) {
                options =
options || {};

                var paddingTL
=
toPoint(options.paddingTopLeft
|| options.padding || [0, 0]),
                    paddingBR
=
toPoint(options.paddingBottomR
ight || options.padding || [0,
0]),

pixelCenter =
this.project(this.getCenter())
,
```

```
                    pixelPoint
= this.project(latlng),

pixelBounds =
this.getPixelBounds(),

paddedBounds =
toBounds([pixelBounds.min.add(
paddingTL),
pixelBounds.max.subtract(paddi
ngBR)]),
                    paddedSize
= paddedBounds.getSize();

                if
(!paddedBounds.contains(pixelP
oint)) {

this._enforcingBounds = true;
                        var
centerOffset =
pixelPoint.subtract(paddedBoun
ds.getCenter());
                        var
offset =
paddedBounds.extend(pixelPoint
).getSize().subtract(paddedSiz
e);

pixelCenter.x +=
centerOffset.x < 0 ? -offset.x
: offset.x;

pixelCenter.y +=
centerOffset.y < 0 ? -offset.y
: offset.y;

this.panTo(this.unproject(pixe
lCenter), options);

this._enforcingBounds = false;
                }
                return this;
        },

        // @method
invalidateSize(options:
```

```
Zoom/pan 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.

        // @alternative
        // @method
invalidateSize(animate:
Boolean): 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:
function (options) {
                if
(!this._loaded) { return this;
}

                options =
extend({

animate: false,
                        pan:
true
                }, options ===
true ? {animate: true} :
options);

                var oldSize =
```

```
this.getSize();

this._sizeChanged = true;

this._lastCenter = null;

                var newSize =
this.getSize(),
                    oldCenter
= oldSize.divideBy(2).round(),
                    newCenter
= newSize.divideBy(2).round(),
                    offset =
oldCenter.subtract(newCenter);

                if (!offset.x
&& !offset.y) { return this; }

                if
(options.animate &&
options.pan) {

this.panBy(offset);

                } else {
                        if
(options.pan) {

this._rawPanBy(offset);
                        }


this.fire('move');

                        if
(options.debounceMoveend) {

clearTimeout(this._sizeTimer);

this._sizeTimer =
setTimeout(bind(this.fire,
this, 'moveend'), 200);
                        } else
{

this.fire('moveend');
                        }
```

```
                }

                // @section
Map state change events
                // @event
resize: ResizeEvent
                // Fired when
the map is resized.
                return
this.fire('resize', {

oldSize: oldSize,

newSize: newSize
                });
        },

        // @section Methods
for modifying map state
        // @method stop():
this
        // Stops the currently
running `panTo` or `flyTo`
animation, if any.
        stop: function () {

this.setZoom(this._limitZoom(t
his._zoom));
                if
(!this.options.zoomSnap) {

this.fire('viewreset');
                }
                return
this._stop();
        },

        // @section
Geolocation methods
        // @method
locate(options?: Locate
options): this
        // Tries to locate the
user using the Geolocation
API, firing a
[`locationfound`](#map-
locationfound)
```

```
        // event with location
data on success or a
[`locationerror`](#map-
locationerror) event on
failure,
        // and optionally sets
the map view to the user's
location with respect to
        // detection accuracy
(or to the world view if
geolocation failed).
        // Note that, if your
page doesn't use HTTPS, this
method will fail in
        // modern browsers
([Chrome 50 and newer]
(https://sites.google.com/a/ch
romium.org/dev/Home/chromium-
security/deprecating-powerful-
features-on-insecure-origins))
        // See `Locate
options` for more details.
        locate: function
(options) {

                options =
this._locateOptions = extend({

timeout: 10000,
                        watch:
false
                        //
setView: false
                        //
maxZoom: <Number>
                        //
maximumAge: 0
                        //
enableHighAccuracy: false
                }, options);

                if (!
('geolocation' in navigator))
{

this._handleGeolocationError({
```

```
code: 0,

message: 'Geolocation not
supported.'
                        });
                        return
this;
                }

                var onResponse
=
bind(this._handleGeolocationRe
sponse, this),
                    onError =
bind(this._handleGeolocationEr
ror, this);

                if
(options.watch) {

this._locationWatchId =

navigator.geolocation.watchPos
ition(onResponse, onError,
options);
                } else {

navigator.geolocation.getCurre
ntPosition(onResponse,
onError, options);
                }
                return this;
        },

        // @method
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}`.
        stopLocate: function
() {
                if
(navigator.geolocation &&
```

```
navigator.geolocation.clearWat
ch) {

navigator.geolocation.clearWat
ch(this._locationWatchId);
                }
                if
(this._locateOptions) {

this._locateOptions.setView =
false;
                }
                return this;
        },


_handleGeolocationError:
function (error) {
                if
(!this._container._leaflet_id)
{ return; }

                var c =
error.code,
                    message =
error.message ||
                            (c
=== 1 ? 'permission denied' :
                            (c
=== 2 ? 'position unavailable'
: 'timeout'));

                if
(this._locateOptions.setView
&& !this._loaded) {

this.fitWorld();
                }

                // @section
Location events
                // @event
locationerror: ErrorEvent
                // Fired when
geolocation (using the
[`locate`](#map-locate)
method) failed.
```

```
this.fire('locationerror', {
                        code:
c,

message: 'Geolocation error: '
+ message + '.'
                });
        },


_handleGeolocationResponse:
function (pos) {
                if
(!this._container._leaflet_id)
{ return; }

                var lat =
pos.coords.latitude,
                    lng =
pos.coords.longitude,
                    latlng =
new LatLng(lat, lng),
                    bounds =
latlng.toBounds(pos.coords.acc
uracy * 2),
                    options =
this._locateOptions;

                if
(options.setView) {
                        var
zoom =
this.getBoundsZoom(bounds);

this.setView(latlng,
options.maxZoom ?
Math.min(zoom,
options.maxZoom) : zoom);
                }

                var data = {

latlng: latlng,

bounds: bounds,
```

```
timestamp: pos.timestamp
                };

                for (var i in
pos.coords) {
                        if
(typeof pos.coords[i] ===
'number') {

data[i] = pos.coords[i];
                        }
                }

                // @event
locationfound: LocationEvent
                // Fired when
geolocation (using the
[`locate`](#map-locate)
method)
                // went
successfully.

this.fire('locationfound',
data);
        },

        // TODO Appropriate
docs section?
        // @section Other
Methods
        // @method
addHandler(name: String,
HandlerClass: Function): this
        // Adds a new
`Handler` to the map, given
its name and constructor
function.
        addHandler: function
(name, HandlerClass) {
                if
(!HandlerClass) { return this;
}

                var handler =
this[name] = new
HandlerClass(this);
```

```
this._handlers.push(handler);

                if
(this.options[name]) {

handler.enable();
                }

                return this;
        },

        // @method remove(): this
        // Destroys the map and clears all related event listeners.
        remove: function () {


this._initEvents(true);
                if
(this.options.maxBounds) {
this.off('moveend',
this._panInsideMaxBounds); }

                if
(this._containerId !==
this._container._leaflet_id) {
                        throw
new Error('Map container is
being reused by another
instance');
                }

                try {
                        //
throws error in IE6-8
                        delete
this._container._leaflet_id;
                        delete
this._containerId;
                } catch (e) {

/*eslint-disable */

this._container._leaflet_id =
```

```
undefined;
                        /*
eslint-enable */

this._containerId = undefined;
                }

                if
(this._locationWatchId !==
undefined) {

this.stopLocate();
                }

                this._stop();


remove(this._mapPane);

                if
(this._clearControlPos) {

this._clearControlPos();
                }
                if
(this._resizeRequest) {

cancelAnimFrame(this._resizeRe
quest);

this._resizeRequest = null;
                }


this._clearHandlers();

                if
(this._loaded) {
                        //
@section Map state change
events
                        //
@event unload: Event
                        //
Fired when the map is
destroyed with [remove](#map-
remove) method.
```

```
this.fire('unload');
                }

                var i;
                for (i in
this._layers) {

this._layers[i].remove();
                }
                for (i in
this._panes) {

remove(this._panes[i]);
                }

                this._layers =
[];
                this._panes =
[];
                delete
this._mapPane;
                delete
this._renderer;

                return this;
        },

        // @section Other
Methods
        // @method
createPane(name: String,
container?: HTMLElement):
HTMLElement
        // Creates a new [map
pane](#map-pane) with the
given name if it doesn't exist
already,
        // then returns it.
The pane is created as a child
of `container`, or
        // as a child of the
main map pane if not set.
        createPane: function
(name, container) {
                var className
= 'leaflet-pane' + (name ? '
```

```
leaflet-' +
name.replace('Pane', '') + '-
pane' : ''),
                    pane =
create$1('div', className,
container || this._mapPane);

                if (name) {

this._panes[name] = pane;
                }
                return pane;
        },

        // @section Methods
for Getting Map State

        // @method
getCenter(): LatLng
        // Returns the
geographical center of the map
view
        getCenter: function ()
{

this._checkIfLoaded();

                if
(this._lastCenter &&
!this._moved()) {
                        return
this._lastCenter.clone();
                }
                return
this.layerPointToLatLng(this._
getCenterLayerPoint());
        },

        // @method getZoom():
Number
        // Returns the current
zoom level of the map view
        getZoom: function () {
                return
this._zoom;
        },
```

```
        // @method
getBounds(): LatLngBounds
        // Returns the
geographical bounds visible in
the current map view
        getBounds: function ()
{
                var bounds =
this.getPixelBounds(),
                    sw =
this.unproject(bounds.getBotto
mLeft()),
                    ne =
this.unproject(bounds.getTopRi
ght());

                return new
LatLngBounds(sw, ne);
        },

        // @method
getMinZoom(): Number
        // Returns the minimum
zoom level of the map (if set
in the `minZoom` option of the
map or of any layers), or `0`
by default.
        getMinZoom: function
() {
                return
this.options.minZoom ===
undefined ?
this._layersMinZoom || 0 :
this.options.minZoom;
        },

        // @method
getMaxZoom(): Number
        // Returns the maximum
zoom level of the map (if set
in the `maxZoom` option of the
map or of any layers).
        getMaxZoom: function
() {
                return
this.options.maxZoom ===
undefined ?
```

```
(this._layersMaxZoom ===
undefined ? Infinity :
this._layersMaxZoom) :

this.options.maxZoom;
        },

        // @method
getBoundsZoom(bounds:
LatLngBounds, inside?:
Boolean, padding?: Point):
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.
        getBoundsZoom:
function (bounds, inside,
padding) { // (LatLngBounds[,
Boolean, Point]) -> Number
                bounds =
toLatLngBounds(bounds);
                padding =
toPoint(padding || [0, 0]);

                var zoom =
this.getZoom() || 0,
                    min =
this.getMinZoom(),
                    max =
this.getMaxZoom(),
                    nw =
bounds.getNorthWest(),
                    se =
bounds.getSouthEast(),
                    size =
this.getSize().subtract(paddin
g),
```

```
                    boundsSize
= toBounds(this.project(se,
zoom), this.project(nw,
zoom)).getSize(),
                    snap =
Browser.any3d ?
this.options.zoomSnap : 1,
                    scalex =
size.x / boundsSize.x,
                    scaley =
size.y / boundsSize.y,
                    scale =
inside ? Math.max(scalex,
scaley) : Math.min(scalex,
scaley);

                zoom =
this.getScaleZoom(scale,
zoom);

                if (snap) {
                        zoom =
Math.round(zoom / (snap /
100)) * (snap / 100); // don't
jump if within 1% of a snap
level
                        zoom =
inside ? Math.ceil(zoom /
snap) * snap : Math.floor(zoom
/ snap) * snap;
                }

                return
Math.max(min, Math.min(max,
zoom));
        },

        // @method getSize():
Point
        // Returns the current
size of the map container (in
pixels).
        getSize: function () {
                if
(!this._size ||
this._sizeChanged) {
```

```
this._size = new Point(

this._container.clientWidth ||
0,

this._container.clientHeight
|| 0);


this._sizeChanged = false;
                }
                return
this._size.clone();
        },

        // @method
getPixelBounds(): Bounds
        // Returns the bounds
of the current map view in
projected pixel
        // coordinates
(sometimes useful in layer and
overlay implementations).
        getPixelBounds:
function (center, zoom) {
                var
topLeftPoint =
this._getTopLeftPoint(center,
zoom);
                return new
Bounds(topLeftPoint,
topLeftPoint.add(this.getSize(
)));
        },

        // TODO: Check
semantics - isn't the pixel
origin the 0,0 coord relative
to
        // the map pane? "left
point of the map layer" can be
confusing, specially
        // since there can be
negative offsets.
        // @method
getPixelOrigin(): Point
        // Returns the
```

```
projected pixel coordinates of
the top left point of
        // the map layer
(useful in custom layer and
overlay implementations).
        getPixelOrigin:
function () {

this._checkIfLoaded();
                return
this._pixelOrigin;
        },

        // @method
getPixelWorldBounds(zoom?:
Number): Bounds
        // Returns the world's
bounds in pixel coordinates
for zoom level `zoom`.
        // If `zoom` is
omitted, the map's current
zoom level is used.
        getPixelWorldBounds:
function (zoom) {
                return
this.options.crs.getProjectedB
ounds(zoom === undefined ?
this.getZoom() : zoom);
        },

        // @section Other
Methods

        // @method
getPane(pane:
String|HTMLElement):
HTMLElement
        // Returns a [map
pane](#map-pane), given its
name or its HTML element (its
identity).
        getPane: function
(pane) {
                return typeof
pane === 'string' ?
this._panes[pane] : pane;
        },
```

```
        // @method getPanes():
Object
        // Returns a plain
object containing the names of
all [panes](#map-pane) as keys
and
        // the panes as
values.
        getPanes: function ()
{
                return
this._panes;
        },

        // @method
getContainer: HTMLElement
        // Returns the HTML
element that contains the map.
        getContainer: function
() {
                return
this._container;
        },


        // @section Conversion
Methods

        // @method
getZoomScale(toZoom: Number,
fromZoom: Number): Number
        // Returns the scale
factor to be applied to a map
transition from zoom level
        // `fromZoom` to
`toZoom`. Used internally to
help with zoom animations.
        getZoomScale: function
(toZoom, fromZoom) {
                // TODO
replace with universal
implementation after
refactoring projections
                var crs =
this.options.crs;
                fromZoom =
```

```
fromZoom === undefined ?
this._zoom : fromZoom;
                return
crs.scale(toZoom) /
crs.scale(fromZoom);
        },

        // @method
getScaleZoom(scale: Number,
fromZoom: Number): Number
        // Returns the zoom
level that the map would end
up at, if it is at `fromZoom`
        // level and
everything is scaled by a
factor of `scale`. Inverse of
        // [`getZoomScale`]
(#map-getZoomScale).
        getScaleZoom: function
(scale, fromZoom) {
                var crs =
this.options.crs;
                fromZoom =
fromZoom === undefined ?
this._zoom : fromZoom;
                var zoom =
crs.zoom(scale *
crs.scale(fromZoom));
                return
isNaN(zoom) ? Infinity : zoom;
        },

        // @method
project(latlng: LatLng, zoom:
Number): Point
        // Projects a
geographical coordinate
`LatLng` according to the
projection
        // of the map's CRS,
then scales it according to
`zoom` and the CRS's
        // `Transformation`.
The result is pixel coordinate
relative to
        // the CRS origin.
        project: function
```

```
(latlng, zoom) {
                zoom = zoom
=== undefined ? this._zoom :
zoom;
                return
this.options.crs.latLngToPoint
(toLatLng(latlng), zoom);
        },

        // @method
unproject(point: Point, zoom:
Number): LatLng
        // Inverse of
[`project`](#map-project).
        unproject: function
(point, zoom) {
                zoom = zoom
=== undefined ? this._zoom :
zoom;
                return
this.options.crs.pointToLatLng
(toPoint(point), zoom);
        },

        // @method
layerPointToLatLng(point:
Point): LatLng
        // Given a pixel
coordinate relative to the
[origin pixel](#map-
getpixelorigin),
        // returns the
corresponding geographical
coordinate (for the current
zoom level).
        layerPointToLatLng:
function (point) {
                var
projectedPoint =
toPoint(point).add(this.getPix
elOrigin());
                return
this.unproject(projectedPoint)
;
        },

        // @method
```

```
latLngToLayerPoint(latlng:
LatLng): Point
        // Given a
geographical coordinate,
returns the corresponding
pixel coordinate
        // relative to the
[origin pixel](#map-
getpixelorigin).
        latLngToLayerPoint:
function (latlng) {
                var
projectedPoint =
this.project(toLatLng(latlng))
._round();
                return
projectedPoint._subtract(this.
getPixelOrigin());
        },

        // @method
wrapLatLng(latlng: LatLng):
LatLng
        // Returns a `LatLng`
where `lat` and `lng` has been
wrapped according to the
        // map's CRS's
`wrapLat` and `wrapLng`
properties, if they are
outside the
        // CRS's bounds.
        // By default this
means longitude is wrapped
around the dateline so its
        // value is between
-180 and +180 degrees.
        wrapLatLng: function
(latlng) {
                return
this.options.crs.wrapLatLng(to
LatLng(latlng));
        },

        // @method
wrapLatLngBounds(bounds:
LatLngBounds): LatLngBounds
        // Returns a
```

```
`LatLngBounds` with the same
size as the given one,
ensuring that
        // its center is
within the CRS's bounds.
        // By default this
means the center longitude is
wrapped around the dateline so
its
        // value is between
-180 and +180 degrees, and the
majority of the bounds
        // overlaps the CRS's
bounds.
        wrapLatLngBounds:
function (latlng) {
                return
this.options.crs.wrapLatLngBou
nds(toLatLngBounds(latlng));
        },

        // @method
distance(latlng1: LatLng,
latlng2: LatLng): Number
        // Returns the
distance between two
geographical coordinates
according to
        // the map's CRS. By
default this measures distance
in meters.
        distance: function
(latlng1, latlng2) {
                return
this.options.crs.distance(toLa
tLng(latlng1),
toLatLng(latlng2));
        },

        // @method
containerPointToLayerPoint(poi
nt: Point): Point
        // Given a pixel
coordinate relative to the map
container, returns the
corresponding
        // pixel coordinate
```

```
relative to the [origin pixel]
(#map-getpixelorigin).

containerPointToLayerPoint:
function (point) { // (Point)
                return
toPoint(point).subtract(this._
getMapPanePos());
        },

        // @method
layerPointToContainerPoint(poi
nt: Point): Point
        // Given a pixel
coordinate relative to the
[origin pixel](#map-
getpixelorigin),
        // returns the
corresponding pixel coordinate
relative to the map container.

layerPointToContainerPoint:
function (point) { // (Point)
                return
toPoint(point).add(this._getMa
pPanePos());
        },

        // @method
containerPointToLatLng(point:
Point): LatLng
        // Given a pixel
coordinate relative to the map
container, returns
        // the corresponding
geographical coordinate (for
the current zoom level).

containerPointToLatLng:
function (point) {
                var layerPoint
=
this.containerPointToLayerPoin
t(toPoint(point));
                return
this.layerPointToLatLng(layerP
oint);
```

```
        },

        // @method
latLngToContainerPoint(latlng:
LatLng): Point
        // Given a
geographical coordinate,
returns the corresponding
pixel coordinate
        // relative to the map
container.

latLngToContainerPoint:
function (latlng) {
                return
this.layerPointToContainerPoin
t(this.latLngToLayerPoint(toLa
tLng(latlng)));
        },

        // @method
mouseEventToContainerPoint(ev:
MouseEvent): Point
        // Given a MouseEvent
object, returns the pixel
coordinate relative to the
        // map container where
the event took place.

mouseEventToContainerPoint:
function (e) {
                return
getMousePosition(e,
this._container);
        },

        // @method
mouseEventToLayerPoint(ev:
MouseEvent): Point
        // Given a MouseEvent
object, returns the pixel
coordinate relative to
        // the [origin pixel]
(#map-getpixelorigin) where
the event took place.

mouseEventToLayerPoint:
```

```
function (e) {
                return
this.containerPointToLayerPoin
t(this.mouseEventToContainerPo
int(e));
        },

        // @method
mouseEventToLatLng(ev:
MouseEvent): LatLng
        // Given a MouseEvent
object, returns geographical
coordinate where the
        // event took place.
        mouseEventToLatLng:
function (e) { // (MouseEvent)
                return
this.layerPointToLatLng(this.m
ouseEventToLayerPoint(e));
        },


        // map initialization
methods

        _initContainer:
function (id) {
                var container
= this._container = get(id);

                if
(!container) {
                        throw
new Error('Map container not
found.');
                } else if
(container._leaflet_id) {
                        throw
new Error('Map container is
already initialized.');
                }

                on(container,
'scroll', this._onScroll,
this);

this._containerId =
```

```
stamp(container);
        },

        _initLayout: function
() {
                var container
= this._container;


this._fadeAnimated =
this.options.fadeAnimation &&
Browser.any3d;


addClass(container, 'leaflet-
container' +

(Browser.touch ? ' leaflet-
touch' : '') +

(Browser.retina ? ' leaflet-
retina' : '') +

(Browser.ielt9 ? ' leaflet-
oldie' : '') +

(Browser.safari ? ' leaflet-
safari' : '') +

(this._fadeAnimated ? '
leaflet-fade-anim' : ''));

                var position =
getStyle(container,
'position');

                if (position
!== 'absolute' && position !==
'relative' && position !==
'fixed') {

container.style.position =
'relative';
                }


this._initPanes();
```

```
                if
(this._initControlPos) {

this._initControlPos();
                }
        },

        _initPanes: function
() {
                var panes =
this._panes = {};

this._paneRenderers = {};

                // @section
                //
                // Panes are
DOM elements used to control
the ordering of layers on the
map. You
                // can access
panes with [`map.getPane`]
(#map-getpane) or
                //
[`map.getPanes`](#map-
getpanes) methods. New panes
can be created with the
                //
[`map.createPane`](#map-
createpane) method.
                //
                // Every map
has the following default
panes that differ only in
zIndex.
                //
                // @pane
mapPane: HTMLElement = 'auto'
                // Pane that
contains all other map panes

                this._mapPane
= this.createPane('mapPane',
this._container);

setPosition(this._mapPane, new
```

```
Point(0, 0));

                // @pane
tilePane: HTMLElement = 200
                // Pane for
`GridLayer`s and `TileLayer`s

this.createPane('tilePane');
                // @pane
overlayPane: HTMLElement = 400
                // Pane for
vectors (`Path`s, like
`Polyline`s and `Polygon`s),
`ImageOverlay`s and
`VideoOverlay`s

this.createPane('overlayPane')
;
                // @pane
shadowPane: HTMLElement = 500
                // Pane for
overlay shadows (e.g. `Marker`
shadows)

this.createPane('shadowPane');
                // @pane
markerPane: HTMLElement = 600
                // Pane for
`Icon`s of `Marker`s

this.createPane('markerPane');
                // @pane
tooltipPane: HTMLElement = 650
                // Pane for
`Tooltip`s.

this.createPane('tooltipPane')
;
                // @pane
popupPane: HTMLElement = 700
                // Pane for
`Popup`s.

this.createPane('popupPane');

                if
(!this.options.markerZoomAnima
```

```
tion) {

addClass(panes.markerPane,
'leaflet-zoom-hide');

addClass(panes.shadowPane,
'leaflet-zoom-hide');
                }
        },


        // private methods
that modify map state

        // @section Map state
change events
        _resetView: function
(center, zoom, noMoveStart) {

setPosition(this._mapPane, new
Point(0, 0));

                var loading =
!this._loaded;
                this._loaded =
true;
                zoom =
this._limitZoom(zoom);


this.fire('viewprereset');

                var
zoomChanged = this._zoom !==
zoom;
                this

._moveStart(zoomChanged,
noMoveStart)

._move(center, zoom)

._moveEnd(zoomChanged);

                // @event
viewreset: Event
                // Fired when
```

```
the map needs to redraw its
content (this usually happens
                // on map zoom
or load). Very useful for
creating custom overlays.

this.fire('viewreset');

                // @event
load: Event
                // Fired when
the map is initialized (when
its center and zoom are set
                // for the
first time).
                if (loading) {

this.fire('load');
                }
        },

        _moveStart: function
(zoomChanged, noMoveStart) {
                // @event
zoomstart: Event
                // Fired when
the map zoom is about to
change (e.g. before zoom
animation).
                // @event
movestart: Event
                // Fired when
the view of the map starts
changing (e.g. user starts
dragging the map).
                if
(zoomChanged) {

this.fire('zoomstart');
                }
                if
(!noMoveStart) {

this.fire('movestart');
                }
                return this;
        },
```

```
        _move: function
(center, zoom, data,
supressEvent) {
                if (zoom ===
undefined) {
                        zoom =
this._zoom;
                }
                var
zoomChanged = this._zoom !==
zoom;

                this._zoom =
zoom;

this._lastCenter = center;

this._pixelOrigin =
this._getNewPixelOrigin(center
);

                if
(!supressEvent) {
                        //
@event zoom: Event
                        //
Fired repeatedly during any
change in zoom level,
                        //
including zoom and fly
animations.
                        if
(zoomChanged || (data &&
data.pinch)) {  // Always fire
'zoom' if pinching because
#3530

this.fire('zoom', data);
                        }

                        //
@event move: Event
                        //
Fired repeatedly during any
movement of the map,
                        //
```

```
including pan and fly
animations.

this.fire('move', data);
                } else if
(data && data.pinch) {  //
Always fire 'zoom' if pinching
because #3530

this.fire('zoom', data);
                }
                return this;
        },

        _moveEnd: function
(zoomChanged) {
                // @event
zoomend: Event
                // Fired when
the map zoom changed, after
any animations.
                if
(zoomChanged) {

this.fire('zoomend');
                }

                // @event
moveend: Event
                // Fired when
the center of the map stops
changing
                // (e.g. user
stopped dragging the map or
after non-centered zoom).
                return
this.fire('moveend');
        },

        _stop: function () {

cancelAnimFrame(this._flyToFra
me);
                if
(this._panAnim) {

this._panAnim.stop();
```

```
                }
                return this;
        },

        _rawPanBy: function
(offset) {

setPosition(this._mapPane,
this._getMapPanePos().subtract
(offset));
        },

        _getZoomSpan: function
() {
                return
this.getMaxZoom() -
this.getMinZoom();
        },

        _panInsideMaxBounds:
function () {
                if
(!this._enforcingBounds) {

this.panInsideBounds(this.opti
ons.maxBounds);
                }
        },

        _checkIfLoaded:
function () {
                if
(!this._loaded) {
                        throw
new Error('Set map center and
zoom first.');
                }
        },

        // DOM event handling

        // @section
Interaction events
        _initEvents: function
(remove) {
                this._targets
= {};
```

```
this._targets[stamp(this._cont
ainer)] = this;

                var onOff =
remove ? off : on;

                // @event
click: MouseEvent
                // Fired when
the user clicks (or taps) the
map.
                // @event
dblclick: MouseEvent
                // Fired when
the user double-clicks (or
double-taps) the map.
                // @event
mousedown: MouseEvent
                // Fired when
the user pushes the mouse
button on the map.
                // @event
mouseup: MouseEvent
                // Fired when
the user releases the mouse
button on the map.
                // @event
mouseover: MouseEvent
                // Fired when
the mouse enters the map.
                // @event
mouseout: MouseEvent
                // Fired when
the mouse leaves the map.
                // @event
mousemove: MouseEvent
                // Fired while
the mouse moves over the map.
                // @event
contextmenu: MouseEvent
                // Fired when
the user pushes the right
mouse button on the map,
prevents
                // default
browser context menu from
```

```
showing if there are listeners
on
                // this event.
Also fired on mobile when the
user holds a single touch
                // for a
second (also called long
press).
                // @event
keypress: KeyboardEvent
                // Fired when
the user presses a key from
the keyboard that produces a
character value while the map
is focused.
                // @event
keydown: KeyboardEvent
                // Fired when
the user presses a key from
the keyboard while the map is
focused. Unlike the `keypress`
event,
                // the
`keydown` event is fired for
keys that produce a character
value and for keys
                // that do not
produce a character value.
                // @event
keyup: KeyboardEvent
                // Fired when
the user releases a key from
the keyboard while the map is
focused.

onOff(this._container, 'click
dblclick mousedown mouseup ' +

'mouseover mouseout mousemove
contextmenu keypress keydown
keyup', this._handleDOMEvent,
this);

                if
(this.options.trackResize) {

onOff(window, 'resize',
```

```
this._onResize, this);
                }

                if
(Browser.any3d &&
this.options.transform3DLimit)
{

(remove ? this.off :
this.on).call(this, 'moveend',
this._onMoveEnd);
                }
        },

        _onResize: function ()
{

cancelAnimFrame(this._resizeRe
quest);

this._resizeRequest =
requestAnimFrame(

function () {
this.invalidateSize({debounceM
oveend: true}); }, this);
        },

        _onScroll: function ()
{

this._container.scrollTop  =
0;

this._container.scrollLeft =
0;
        },

        _onMoveEnd: function
() {
                var pos =
this._getMapPanePos();
                if
(Math.max(Math.abs(pos.x),
Math.abs(pos.y)) >=
this.options.transform3DLimit)
{
```

```
                        //
https://bugzilla.mozilla.org/s
how_bug.cgi?id=1203873 but
Webkit also have
                        // a
pixel offset on very high
values, see:
https://jsfiddle.net/dg6r5hhb/

this._resetView(this.getCenter
(), this.getZoom());
                }
        },

        _findEventTargets:
function (e, type) {
                var targets =
[],
                    target,
                    isHover =
type === 'mouseout' || type
=== 'mouseover',
                    src =
e.target || e.srcElement,
                    dragging =
false;

                while (src) {
                        target
= this._targets[stamp(src)];
                        if
(target && (type === 'click'
|| type === 'preclick') &&
this._draggableMoved(target))
{

// Prevent firing click after
you just dragged an object.

dragging = true;

break;
                        }
                        if
(target &&
target.listens(type, true)) {
```

```
if (isHover &&
!isExternalTarget(src, e)) {
break; }

targets.push(target);

if (isHover) { break; }
                        }
                        if
(src === this._container) {
break; }
                        src =
src.parentNode;
                }
                if
(!targets.length && !dragging
&& !isHover &&
this.listens(type, true)) {

targets = [this];
                }
                return
targets;
        },

        _isClickDisabled:
function (el) {
                while (el &&
el !== this._container) {
                        if
(el['_leaflet_disable_click'])
{ return true; }
                        el =
el.parentNode;
                }
        },

        _handleDOMEvent:
function (e) {
                var el =
(e.target || e.srcElement);
                if
(!this._loaded ||
el['_leaflet_disable_events']
|| e.type === 'click' &&
this._isClickDisabled(el)) {
```

```
return;
                }

                var type =
e.type;

                if (type ===
'mousedown') {
                        //
prevents outline when clicking
on keyboard-focusable element

preventOutline(el);
                }


this._fireDOMEvent(e, type);
        },

        _mouseEvents:
['click', 'dblclick',
'mouseover', 'mouseout',
'contextmenu'],

        _fireDOMEvent:
function (e, type,
canvasTargets) {

                if (e.type ===
'click') {
                        //
Fire a synthetic 'preclick'
event which propagates up
(mainly for closing popups).
                        //
@event preclick: MouseEvent
                        //
Fired before mouse click on
the map (sometimes useful when
you
                        //
want something to happen on
click before any existing
click
                        //
handlers start running).
                        var
```

```
synth = extend({}, e);

synth.type = 'preclick';

this._fireDOMEvent(synth,
synth.type, canvasTargets);
                }

                // Find the
layer the event is propagating
from and its parents.
                var targets =
this._findEventTargets(e,
type);

                if
(canvasTargets) {
                        var
filtered = []; // pick only
targets with listeners
                        for
(var i = 0; i <
canvasTargets.length; i++) {

if
(canvasTargets[i].listens(type
, true)) {

filtered.push(canvasTargets[i]
);

}
                        }

targets =
filtered.concat(targets);
                }

                if
(!targets.length) { return; }

                if (type ===
'contextmenu') {

preventDefault(e);
                }
```

```
                var target =
targets[0];
                var data = {

originalEvent: e
                };

                if (e.type !==
'keypress' && e.type !==
'keydown' && e.type !==
'keyup') {
                        var
isMarker = target.getLatLng &&
(!target._radius ||
target._radius <= 10);

data.containerPoint = isMarker
?

this.latLngToContainerPoint(ta
rget.getLatLng()) :
this.mouseEventToContainerPoin
t(e);

data.layerPoint =
this.containerPointToLayerPoin
t(data.containerPoint);

data.latlng = isMarker ?
target.getLatLng() :
this.layerPointToLatLng(data.l
ayerPoint);
                }

                for (i = 0; i
< targets.length; i++) {

targets[i].fire(type, data,
true);
                        if
(data.originalEvent._stopped
||

(targets[i].options.bubblingMo
useEvents === false &&
indexOf(this._mouseEvents,
type) !== -1)) { return; }
```

```
                }
        },

        _draggableMoved:
function (obj) {
                obj =
obj.dragging &&
obj.dragging.enabled() ? obj :
this;
                return
(obj.dragging &&
obj.dragging.moved()) ||
(this.boxZoom &&
this.boxZoom.moved());
        },

        _clearHandlers:
function () {
                for (var i =
0, len =
this._handlers.length; i <
len; i++) {

this._handlers[i].disable();
                }
        },

        // @section Other
Methods

        // @method
whenReady(fn: Function,
context?: Object): this
        // Runs the given
function `fn` when the map
gets initialized with
        // a view (center and
zoom) and at least one layer,
or immediately
        // if it's already
initialized, optionally
passing a function context.
        whenReady: function
(callback, context) {
                if
(this._loaded) {
```

```
callback.call(context || this,
{target: this});
                } else {

this.on('load', callback,
context);
                }
                return this;
        },


        // private methods for
getting map state

        _getMapPanePos:
function () {
                return
getPosition(this._mapPane) ||
new Point(0, 0);
        },

        _moved: function () {
                var pos =
this._getMapPanePos();
                return pos &&
!pos.equals([0, 0]);
        },

        _getTopLeftPoint:
function (center, zoom) {
                var
pixelOrigin = center && zoom
!== undefined ?

this._getNewPixelOrigin(center
, zoom) :

this.getPixelOrigin();
                return
pixelOrigin.subtract(this._get
MapPanePos());
        },

        _getNewPixelOrigin:
function (center, zoom) {
                var viewHalf =
this.getSize()._divideBy(2);
```

```
                return
this.project(center,
zoom)._subtract(viewHalf)._add
(this._getMapPanePos())._round
();
        },


_latLngToNewLayerPoint:
function (latlng, zoom,
center) {
                var topLeft =
this._getNewPixelOrigin(center
, zoom);
                return
this.project(latlng,
zoom)._subtract(topLeft);
        },


_latLngBoundsToNewLayerBounds:
function (latLngBounds, zoom,
center) {
                var topLeft =
this._getNewPixelOrigin(center
, zoom);
                return
toBounds([

this.project(latLngBounds.getS
outhWest(),
zoom)._subtract(topLeft),

this.project(latLngBounds.getN
orthWest(),
zoom)._subtract(topLeft),

this.project(latLngBounds.getS
outhEast(),
zoom)._subtract(topLeft),

this.project(latLngBounds.getN
orthEast(),
zoom)._subtract(topLeft)
                ]);
        },
```

```
        // layer point of the
current center
        _getCenterLayerPoint:
function () {
                return
this.containerPointToLayerPoin
t(this.getSize()._divideBy(2))
;
        },

        // offset of the
specified place to the current
center in pixels
        _getCenterOffset:
function (latlng) {
                return
this.latLngToLayerPoint(latlng
).subtract(this._getCenterLaye
rPoint());
        },

        // adjust center for
view to get inside bounds
        _limitCenter: function
(center, zoom, bounds) {

                if (!bounds) {
return center; }

                var
centerPoint =
this.project(center, zoom),
                    viewHalf =
this.getSize().divideBy(2),
                    viewBounds
= new
Bounds(centerPoint.subtract(vi
ewHalf),
centerPoint.add(viewHalf)),
                    offset =
this._getBoundsOffset(viewBoun
ds, bounds, zoom);

                // If offset
is less than a pixel, ignore.
                // This
prevents unstable projections
```

```
from getting into
                // an infinite
loop of tiny offsets.
                if
(offset.round().equals([0,
0])) {
                        return
center;
                }

                return
this.unproject(centerPoint.add
(offset), zoom);
        },

        // adjust offset for
view to get inside bounds
        _limitOffset: function
(offset, bounds) {
                if (!bounds) {
return offset; }

                var viewBounds
= this.getPixelBounds(),
                    newBounds
= new
Bounds(viewBounds.min.add(offs
et),
viewBounds.max.add(offset));

                return
offset.add(this._getBoundsOffs
et(newBounds, bounds));
        },

        // returns offset
needed for pxBounds to get
inside maxBounds at a
specified zoom
        _getBoundsOffset:
function (pxBounds, maxBounds,
zoom) {
                var
projectedMaxBounds = toBounds(

this.project(maxBounds.getNort
hEast(), zoom),
```

```
this.project(maxBounds.getSouthWest(), zoom)
                    ),
                    minOffset = projectedMaxBounds.min.subtract(pxBounds.min),
                    maxOffset = projectedMaxBounds.max.subtract(pxBounds.max),

                    dx = this._rebound(minOffset.x, -maxOffset.x),
                    dy = this._rebound(minOffset.y, -maxOffset.y);

                return new Point(dx, dy);
        },

        _rebound: function (left, right) {
                return left + right > 0 ?
                        Math.round(left - right) / 2 :
                        Math.max(0, Math.ceil(left)) - Math.max(0, Math.floor(right));
        },

        _limitZoom: function (zoom) {
                var min = this.getMinZoom(),
                    max = this.getMaxZoom(),
                    snap = Browser.any3d ? this.options.zoomSnap : 1;
                if (snap) {
                        zoom =
```

```
Math.round(zoom / snap) *
snap;
                }
                return
Math.max(min, Math.min(max,
zoom));
        },

        _onPanTransitionStep:
function () {

this.fire('move');
        },

        _onPanTransitionEnd:
function () {

removeClass(this._mapPane,
'leaflet-pan-anim');

this.fire('moveend');
        },

        _tryAnimatedPan:
function (center, options) {
                // difference
between the new and current
centers in pixels
                var offset =
this._getCenterOffset(center).
_trunc();

                // don't
animate too far unless
animate: true specified in
options
                if ((options
&& options.animate) !== true
&&
!this.getSize().contains(offse
t)) { return false; }


this.panBy(offset, options);

                return true;
        },
```

```
        _createAnimProxy:
function () {

                var proxy =
this._proxy = create$1('div',
'leaflet-proxy leaflet-zoom-
animated');

this._panes.mapPane.appendChil
d(proxy);


this.on('zoomanim', function
(e) {
                        var
prop = TRANSFORM,

transform =
this._proxy.style[prop];


setTransform(this._proxy,
this.project(e.center,
e.zoom),
this.getZoomScale(e.zoom, 1));

                        //
workaround for case when
transform is the same and so
transitionend event is not
fired
                        if
(transform ===
this._proxy.style[prop] &&
this._animatingZoom) {

this._onZoomTransitionEnd();
                        }
                }, this);

                this.on('load
moveend', this._animMoveEnd,
this);


this._on('unload',
```

```
this._destroyAnimProxy, this);
        },

        _destroyAnimProxy:
function () {

remove(this._proxy);
                this.off('load
moveend', this._animMoveEnd,
this);
                delete
this._proxy;
        },

        _animMoveEnd: function
() {
                var c =
this.getCenter(),
                    z =
this.getZoom();

setTransform(this._proxy,
this.project(c, z),
this.getZoomScale(z, 1));
        },

        _catchTransitionEnd:
function (e) {
                if
(this._animatingZoom &&
e.propertyName.indexOf('transf
orm') >= 0) {

this._onZoomTransitionEnd();
                }
        },

        _nothingToAnimate:
function () {
                return
!this._container.getElementsBy
ClassName('leaflet-zoom-
animated').length;
        },

        _tryAnimatedZoom:
function (center, zoom,
```

```
options) {

                if
(this._animatingZoom) { return
true; }

                options =
options || {};

                // don't
animate if disabled, not
supported or zoom difference
is too large
                if
(!this._zoomAnimated ||
options.animate === false ||
this._nothingToAnimate() ||

Math.abs(zoom - this._zoom) >
this.options.zoomAnimationThre
shold) { return false; }

                // offset is
the pixel coords of the zoom
origin relative to the current
center
                var scale =
this.getZoomScale(zoom),
                    offset =
this._getCenterOffset(center).
_divideBy(1 - 1 / scale);

                // don't
animate if the zoom origin
isn't within one screen from
the current center, unless
forced
                if
(options.animate !== true &&
!this.getSize().contains(offse
t)) { return false; }


requestAnimFrame(function () {
                        this

._moveStart(true, false)
```

```
._animateZoom(center, zoom,
true);
                }, this);

                return true;
        },

        _animateZoom: function
(center, zoom, startAnim,
noUpdate) {
                if
(!this._mapPane) { return; }

                if (startAnim)
{

this._animatingZoom = true;

                        //
remember what center/zoom to
set after animation

this._animateToCenter =
center;

this._animateToZoom = zoom;


addClass(this._mapPane,
'leaflet-zoom-anim');
                }

                // @section
Other Events
                // @event
zoomanim: ZoomAnimEvent
                // Fired at
least once per zoom animation.
For continuous zoom, like
pinch zooming, fired once per
frame during zoom.

this.fire('zoomanim', {

center: center,
                        zoom:
```

```
zoom,

noUpdate: noUpdate
                });

                if
(!this._tempFireZoomEvent) {

this._tempFireZoomEvent =
this._zoom !==
this._animateToZoom;
                }


this._move(this._animateToCent
er, this._animateToZoom,
undefined, true);

                // Work around
webkit not firing
'transitionend', see
https://github.com/Leaflet/Lea
flet/issues/3689, 2693

setTimeout(bind(this._onZoomTr
ansitionEnd, this), 250);
        },

        _onZoomTransitionEnd:
function () {
                if
(!this._animatingZoom) {
return; }

                if
(this._mapPane) {

removeClass(this._mapPane,
'leaflet-zoom-anim');
                }


this._animatingZoom = false;


this._move(this._animateToCent
er, this._animateToZoom,
```

```
undefined, true);

                if
(this._tempFireZoomEvent) {

this.fire('zoom');
                }
                delete
this._tempFireZoomEvent;


this.fire('move');


this._moveEnd(true);
        }
  });

  // @section

  // @factory L.map(id:
String, options?: Map options)
  // Instantiates a map object
given the DOM ID of a `<div>`
element
  // and optionally an object
literal with `Map options`.
  //
  // @alternative
  // @factory L.map(el:
HTMLElement, options?: Map
options)
  // Instantiates a map object
given an instance of a `<div>`
HTML element
  // and optionally an object
literal with `Map options`.
  function createMap(id,
options) {
        return new Map(id,
options);
  }

  /*
   * @class Control
   * @aka L.Control
   * @inherits Class
```

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

  var Control = Class.extend({
        // @section
        // @aka Control
Options
        options: {
                // @option
position: String = 'topright'
                // The
position of the control (one
of the map corners). Possible
values are `'topleft'`,
                //
`'topright'`, `'bottomleft'`
or `'bottomright'`
                position:
'topright'
        },

        initialize: function
(options) {

setOptions(this, options);
        },

        /* @section
         * Classes extending
L.Control will inherit the
following methods:
         *
         * @method
getPosition: string
         * Returns the
position of the control.
         */
        getPosition: function
() {
                return
this.options.position;
        },
```

```
        // @method
setPosition(position: string):
this
        // Sets the position
of the control.
        setPosition: function
(position) {
                var map =
this._map;

                if (map) {

map.removeControl(this);
                }


this.options.position =
position;

                if (map) {

map.addControl(this);
                }

                return this;
        },

        // @method
getContainer: HTMLElement
        // Returns the
HTMLElement that contains the
control.
        getContainer: function
() {
                return
this._container;
        },

        // @method addTo(map:
Map): this
        // Adds the control to
the given map.
        addTo: function (map)
{
                this.remove();
                this._map =
```

```
map;

                var container
= this._container =
this.onAdd(map),
                    pos =
this.getPosition(),
                    corner =
map._controlCorners[pos];


addClass(container, 'leaflet-
control');

                if
(pos.indexOf('bottom') !== -1)
{

corner.insertBefore(container,
corner.firstChild);
                } else {

corner.appendChild(container);
                }


this._map.on('unload',
this.remove, this);

                return this;
        },

        // @method remove:
this
        // Removes the control
from the map it is currently
active on.
        remove: function () {
                if
(!this._map) {
                        return
this;
                }


remove(this._container);
```

```
                if
(this.onRemove) {

this.onRemove(this._map);
                }


this._map.off('unload',
this.remove, this);
                this._map =
null;

                return this;
        },

        _refocusOnMap:
function (e) {
                // if map
exists and event is not a
keyboard event
                if (this._map
&& e && e.screenX > 0 &&
e.screenY > 0) {

this._map.getContainer().focus
();
                }
        }
  });

  var control = function
(options) {
        return new
Control(options);
  };

  /* @section Extension
methods
   * @uninheritable
   *
   * Every control should
extend from `L.Control` and
(re-)implement the following
methods.
   *
   * @method onAdd(map: Map):
HTMLElement
```

```
   * Should return the
container DOM element for the
control and add listeners on
relevant map events. Called on
[`control.addTo(map)`]
(#control-addTo).
   *
   * @method onRemove(map:
Map)
   * Optional method. Should
contain all clean up code that
removes the listeners
previously added in [`onAdd`]
(#control-onadd). Called on
[`control.remove()`](#control-
remove).
   */

  /* @namespace Map
   * @section Methods for
Layers and Controls
   */
  Map.include({
        // @method
addControl(control: Control):
this
        // Adds the given
control to the map
        addControl: function
(control) {

control.addTo(this);
                return this;
        },

        // @method
removeControl(control:
Control): this
        // Removes the given
control from the map
        removeControl:
function (control) {

control.remove();
                return this;
        },
```

```
        _initControlPos:
function () {
                var corners =
this._controlCorners = {},
                    l =
'leaflet-',
                    container
= this._controlContainer =

create$1('div', l + 'control-
container', this._container);

                function
createCorner(vSide, hSide) {
                        var
className = l + vSide + ' ' +
l + hSide;


corners[vSide + hSide] =
create$1('div', className,
container);
                }


createCorner('top', 'left');

createCorner('top', 'right');

createCorner('bottom',
'left');

createCorner('bottom',
'right');
        },

        _clearControlPos:
function () {
                for (var i in
this._controlCorners) {

remove(this._controlCorners[i]
);
                }

remove(this._controlContainer)
;
```

```
                delete
this._controlCorners;
                delete
this._controlContainer;
        }
  });

  /*
   * @class Control.Layers
   * @aka L.Control.Layers
   * @inherits Control
   *
   * The layers control gives
users the ability to switch
between different base layers
and switch overlays on/off
(check out the [detailed
example]
(https://leafletjs.com/example
s/layers-control/)). Extends
`Control`.
   *
   * @example
   *
   * ```js
   * var baseLayers = {
   *    "Mapbox": mapbox,
   *    "OpenStreetMap": osm
   * };
   *
   * var overlays = {
   *    "Marker": marker,
   *    "Roads": roadsLayer
   * };
   *
   *
L.control.layers(baseLayers,
overlays).addTo(map);
   * ```
   *
   * The `baseLayers` and
`overlays` parameters are
object literals with layer
names as keys and `Layer`
objects as values:
   *
   * ```js
```

```
   * {
   *     "<someName1>":
layer1,
   *     "<someName2>": layer2
   * }
   * ```
   *
   * The layer names can
contain HTML, which allows you
to add additional styling to
the items:
   *
   * ```js
   * {"<img src='my-layer-
icon' /> <span class='my-
layer-item'>My Layer</span>":
myLayer}
   * ```
   */

  var Layers =
Control.extend({
        // @section
        // @aka Control.Layers
options
        options: {
                // @option
collapsed: Boolean = true
                // If `true`,
the control will be collapsed
into an icon and expanded on
mouse hover, touch, or
keyboard activation.
                collapsed:
true,
                position:
'topright',

                // @option
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.
                autoZIndex:
```

```
true,

                // @option
hideSingleBase: Boolean =
false
                // If `true`,
the base layers in the control
will be hidden when there is
only one.

hideSingleBase: false,

                // @option
sortLayers: Boolean = false
                // Whether to
sort the layers. When `false`,
layers will keep the order
                // in which
they were added to the
control.
                sortLayers:
false,

                // @option
sortFunction: Function = *
                // A [compare
function]
(https://developer.mozilla.org
/docs/Web/JavaScript/Reference
/Global_Objects/Array/sort)
                // that will
be used for sorting the
layers, when `sortLayers` is
`true`.
                // The
function receives both the
`L.Layer` instances and their
names, as in
                //
`sortFunction(layerA, layerB,
nameA, nameB)`.
                // By default,
it sorts layers alphabetically
by their name.
                sortFunction:
function (layerA, layerB,
nameA, nameB) {
```

```
                        return
nameA < nameB ? -1 : (nameB <
nameA ? 1 : 0);
                }
        },

        initialize: function
(baseLayers, overlays,
options) {

setOptions(this, options);


this._layerControlInputs = [];
                this._layers =
[];

this._lastZIndex = 0;

this._handlingClick = false;

                for (var i in
baseLayers) {

this._addLayer(baseLayers[i],
i);
                }

                for (i in
overlays) {

this._addLayer(overlays[i], i,
true);
                }
        },

        onAdd: function (map)
{

this._initLayout();

this._update();

                this._map =
map;

map.on('zoomend',
```

```
this._checkDisabledLayers,
this);

                for (var i =
0; i < this._layers.length;
i++) {

this._layers[i].layer.on('add
remove', this._onLayerChange,
this);
                }

                return
this._container;
        },

        addTo: function (map)
{

Control.prototype.addTo.call(t
his, map);
                // Trigger
expand after Layers Control
has been inserted into DOM so
that is now has an actual
height.
                return
this._expandIfNotCollapsed();
        },

        onRemove: function ()
{

this._map.off('zoomend',
this._checkDisabledLayers,
this);

                for (var i =
0; i < this._layers.length;
i++) {

this._layers[i].layer.off('add
remove', this._onLayerChange,
this);
                }
        },
```

```
        // @method
addBaseLayer(layer: Layer,
name: String): this
        // Adds a base layer
(radio button entry) with the
given name to the control.
        addBaseLayer: function
(layer, name) {

this._addLayer(layer, name);
                return
(this._map) ? this._update() :
this;
        },

        // @method
addOverlay(layer: Layer, name:
String): this
        // Adds an overlay
(checkbox entry) with the
given name to the control.
        addOverlay: function
(layer, name) {

this._addLayer(layer, name,
true);
                return
(this._map) ? this._update() :
this;
        },

        // @method
removeLayer(layer: Layer):
this
        // Remove the given
layer from the control.
        removeLayer: function
(layer) {
                layer.off('add
remove', this._onLayerChange,
this);

                var obj =
this._getLayer(stamp(layer));
                if (obj) {

this._layers.splice(this._laye
```

```
rs.indexOf(obj), 1);
                }
                return
(this._map) ? this._update() :
this;
        },

        // @method expand():
this
        // Expand the control
container if collapsed.
        expand: function () {

addClass(this._container,
'leaflet-control-layers-
expanded');

this._section.style.height =
null;
                var
acceptableHeight =
this._map.getSize().y -
(this._container.offsetTop +
50);
                if
(acceptableHeight <
this._section.clientHeight) {

addClass(this._section,
'leaflet-control-layers-
scrollbar');

this._section.style.height =
acceptableHeight + 'px';
                } else {

removeClass(this._section,
'leaflet-control-layers-
scrollbar');
                }

this._checkDisabledLayers();
                return this;
        },

        // @method collapse():
this
```

```
        // Collapse the
control container if expanded.
        collapse: function ()
{

removeClass(this._container,
'leaflet-control-layers-
expanded');
                return this;
        },

        _initLayout: function
() {
                var className
= 'leaflet-control-layers',
                    container
= this._container =
create$1('div', className),
                    collapsed
= this.options.collapsed;

                // makes this
work on IE touch devices by
stopping it from firing a
mouseout event when the touch
is released

container.setAttribute('aria-
haspopup', true);


disableClickPropagation(contai
ner);

disableScrollPropagation(conta
iner);

                var section =
this._section =
create$1('section', className
+ '-list');

                if (collapsed)
{

this._map.on('click',
this.collapse, this);
```

```
on(container, {

mouseenter: function () {

on(section, 'click',
preventDefault);

this.expand();

setTimeout(function () {

off(section, 'click',
preventDefault);

});

},

mouseleave: this.collapse
                        },
this);
                }

                var link =
this._layersLink =
create$1('a', className + '-
toggle', container);
                link.href =
'#';
                link.title =
'Layers';

link.setAttribute('role',
'button');

                on(link,
'click', preventDefault); //
prevent link function
                on(link,
'focus', this.expand, this);

                if
(!collapsed) {

this.expand();
```

```
                }


this._baseLayersList =
create$1('div', className + '-
base', section);

this._separator =
create$1('div', className + '-
separator', section);

this._overlaysList =
create$1('div', className + '-
overlays', section);


container.appendChild(section)
;
        },

        _getLayer: function
(id) {
                for (var i =
0; i < this._layers.length;
i++) {

                        if
(this._layers[i] &&
stamp(this._layers[i].layer)
=== id) {

return this._layers[i];
                        }
                }
        },

        _addLayer: function
(layer, name, overlay) {
                if (this._map)
{

layer.on('add remove',
this._onLayerChange, this);
                }


this._layers.push({
```

```
                        layer:
layer,
                        name:
name,

overlay: overlay
                });

                if
(this.options.sortLayers) {

this._layers.sort(bind(functio
n (a, b) {

return
this.options.sortFunction(a.la
yer, b.layer, a.name, b.name);
                        },
this));
                }

                if
(this.options.autoZIndex &&
layer.setZIndex) {

this._lastZIndex++;

layer.setZIndex(this._lastZInd
ex);
                }


this._expandIfNotCollapsed();
        },

        _update: function () {
                if
(!this._container) { return
this; }


empty(this._baseLayersList);

empty(this._overlaysList);


this._layerControlInputs = [];
```

```
                var
baseLayersPresent,
overlaysPresent, i, obj,
baseLayersCount = 0;

                for (i = 0; i
< this._layers.length; i++) {
                        obj =
this._layers[i];

this._addItem(obj);

overlaysPresent =
overlaysPresent ||
obj.overlay;

baseLayersPresent =
baseLayersPresent ||
!obj.overlay;

baseLayersCount +=
!obj.overlay ? 1 : 0;
                }

                // Hide base
layers section if there's only
one layer.
                if
(this.options.hideSingleBase)
{

baseLayersPresent =
baseLayersPresent &&
baseLayersCount > 1;

this._baseLayersList.style.dis
play = baseLayersPresent ? ''
: 'none';
                }


this._separator.style.display
= overlaysPresent &&
baseLayersPresent ? '' :
'none';

                return this;
```

```
        },

        _onLayerChange:
function (e) {
                if
(!this._handlingClick) {

this._update();
                }

                var obj =
this._getLayer(stamp(e.target)
);

                // @namespace
Map
                // @section
Layer events
                // @event
baselayerchange:
LayersControlEvent
                // Fired when
the base layer is changed
through the [layers control]
(#control-layers).
                // @event
overlayadd: LayersControlEvent
                // Fired when
an overlay is selected through
the [layers control](#control-
layers).
                // @event
overlayremove:
LayersControlEvent
                // Fired when
an overlay is deselected
through the [layers control]
(#control-layers).
                // @namespace
Control.Layers
                var type =
obj.overlay ?

(e.type === 'add' ?
'overlayadd' :
'overlayremove') :
```

```
(e.type === 'add' ?
'baselayerchange' : null);

                if (type) {

this._map.fire(type, obj);
                }
        },

        // IE7 bugs out if you
create a radio dynamically, so
you have to do it this hacky
way (see
https://stackoverflow.com/a/11
9079)
        _createRadioElement:
function (name, checked) {

                var radioHtml
= '<input type="radio"
class="leaflet-control-layers-
selector" name="' +

name + '"' + (checked ? '
checked="checked"' : '') +
'/>';

                var
radioFragment =
document.createElement('div');

radioFragment.innerHTML =
radioHtml;

                return
radioFragment.firstChild;
        },

        _addItem: function
(obj) {
                var label =
document.createElement('label'
),
                    checked =
this._map.hasLayer(obj.layer),
                    input;
```

```
                if
(obj.overlay) {
                        input
=
document.createElement('input'
);

input.type = 'checkbox';

input.className = 'leaflet-
control-layers-selector';

input.defaultChecked =
checked;
                } else {
                        input
=
this._createRadioElement('leaf
let-base-layers_' +
stamp(this), checked);
                }


this._layerControlInputs.push(
input);
                input.layerId
= stamp(obj.layer);

                on(input,
'click', this._onInputClick,
this);

                var name =
document.createElement('span')
;
                name.innerHTML
= ' ' + obj.name;

                // Helps from
preventing layer control
flicker when checkboxes are
disabled
                //
https://github.com/Leaflet/Lea
flet/issues/2771
                var holder =
document.createElement('span')
```

```
;


label.appendChild(holder);

holder.appendChild(input);

holder.appendChild(name);

                var container
= obj.overlay ?
this._overlaysList :
this._baseLayersList;

container.appendChild(label);


this._checkDisabledLayers();
                return label;
        },

        _onInputClick:
function () {
                var inputs =
this._layerControlInputs,
                    input,
layer;
                var
addedLayers = [],

removedLayers = [];


this._handlingClick = true;

                for (var i =
inputs.length - 1; i >= 0; i--
) {
                        input
= inputs[i];
                        layer
=
this._getLayer(input.layerId).
layer;

                        if
(input.checked) {
```

```
addedLayers.push(layer);
                        } else
if (!input.checked) {

removedLayers.push(layer);
                        }
                }

                // Bugfix
issue 2318: Should remove all
old layers before readding new
ones
                for (i = 0; i
< removedLayers.length; i++) {
                        if
(this._map.hasLayer(removedLay
ers[i])) {

this._map.removeLayer(removedL
ayers[i]);
                        }
                }
                for (i = 0; i
< addedLayers.length; i++) {
                        if
(!this._map.hasLayer(addedLaye
rs[i])) {

this._map.addLayer(addedLayers
[i]);
                        }
                }


this._handlingClick = false;


this._refocusOnMap();
        },

        _checkDisabledLayers:
function () {
                var inputs =
this._layerControlInputs,
                    input,
                    layer,
```

```
                    zoom =
this._map.getZoom();

                for (var i =
inputs.length - 1; i >= 0; i--
) {
                        input
= inputs[i];
                        layer
=
this._getLayer(input.layerId).
layer;

input.disabled =
(layer.options.minZoom !==
undefined && zoom <
layer.options.minZoom) ||

(layer.options.maxZoom !==
undefined && zoom >
layer.options.maxZoom);

                }
        },

        _expandIfNotCollapsed:
function () {
                if (this._map
&& !this.options.collapsed) {

this.expand();
                }
                return this;
        }

  });


  // @factory
L.control.layers(baselayers?:
Object, overlays?: Object,
options?: Control.Layers
options)
  // Creates a layers control
with the given layers. Base
layers will be switched with
radio buttons, while overlays
```

```
will be switched with
checkboxes. Note that all base
layers should be passed in the
base layers object, but only
one should be added to the map
during map instantiation.
  var layers = function
(baseLayers, overlays,
options) {
        return new
Layers(baseLayers, overlays,
options);
  };

  /*
   * @class Control.Zoom
   * @aka L.Control.Zoom
   * @inherits Control
   *
   * 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](#map-
zoomcontrol) to `false`.
Extends `Control`.
   */

  var Zoom = Control.extend({
        // @section
        // @aka Control.Zoom
options
        options: {
                position:
'topleft',

                // @option
zoomInText: String = '<span
aria-hidden="true">+</span>'
                // The text
set on the 'zoom in' button.
                zoomInText:
'<span aria-hidden="true">+
</span>',

                // @option
zoomInTitle: String = 'Zoom
```

```
in'
                // The title
set on the 'zoom in' button.
                zoomInTitle:
'Zoom in',

                // @option
zoomOutText: String = '<span
aria-hidden="true">&#x2212;
</span>'
                // The text
set on the 'zoom out' button.
                zoomOutText:
'<span aria-
hidden="true">&#x2212;
</span>',

                // @option
zoomOutTitle: String = 'Zoom
out'
                // The title
set on the 'zoom out' button.
                zoomOutTitle:
'Zoom out'
        },

        onAdd: function (map)
{
                var zoomName =
'leaflet-control-zoom',
                    container
= create$1('div', zoomName + '
leaflet-bar'),
                    options =
this.options;


this._zoomInButton  =
this._createButton(options.zoo
mInText, options.zoomInTitle,

zoomName + '-in',  container,
this._zoomIn);

this._zoomOutButton =
this._createButton(options.zoo
mOutText,
```

```
options.zoomOutTitle,

zoomName + '-out', container,
this._zoomOut);


this._updateDisabled();

map.on('zoomend
zoomlevelschange',
this._updateDisabled, this);

                return
container;
        },

        onRemove: function
(map) {

map.off('zoomend
zoomlevelschange',
this._updateDisabled, this);
        },

        disable: function () {
                this._disabled
= true;

this._updateDisabled();
                return this;
        },

        enable: function () {
                this._disabled
= false;

this._updateDisabled();
                return this;
        },

        _zoomIn: function (e)
{
                if
(!this._disabled &&
this._map._zoom <
this._map.getMaxZoom()) {
```

```
this._map.zoomIn(this._map.opt
ions.zoomDelta * (e.shiftKey ?
3 : 1));
                }
        },

        _zoomOut: function (e)
{
                if
(!this._disabled &&
this._map._zoom >
this._map.getMinZoom()) {

this._map.zoomOut(this._map.op
tions.zoomDelta * (e.shiftKey
? 3 : 1));
                }
        },

        _createButton:
function (html, title,
className, container, fn) {
                var link =
create$1('a', className,
container);
                link.innerHTML
= html;
                link.href =
'#';
                link.title =
title;

                /*
                 * Will force
screen readers like VoiceOver
to read this as "Zoom in -
button"
                 */

link.setAttribute('role',
'button');

link.setAttribute('aria-
label', title);


disableClickPropagation(link);
```

```
                on(link,
'click', stop);
                on(link,
'click', fn, this);
                on(link,
'click', this._refocusOnMap,
this);

                return link;
        },

        _updateDisabled:
function () {
                var map =
this._map,
                    className
= 'leaflet-disabled';


removeClass(this._zoomInButton
, className);

removeClass(this._zoomOutButto
n, className);

this._zoomInButton.setAttribut
e('aria-disabled', 'false');

this._zoomOutButton.setAttribu
te('aria-disabled', 'false');

                if
(this._disabled || map._zoom
=== map.getMinZoom()) {

addClass(this._zoomOutButton,
className);

this._zoomOutButton.setAttribu
te('aria-disabled', 'true');
                }
                if
(this._disabled || map._zoom
=== map.getMaxZoom()) {

addClass(this._zoomInButton,
className);
```

```
this._zoomInButton.setAttribut
e('aria-disabled', 'true');
                }
        }
  });

  // @namespace Map
  // @section Control options
  // @option zoomControl:
Boolean = true
  // Whether a [zoom control]
(#control-zoom) is added to
the map by default.
  Map.mergeOptions({
        zoomControl: true
  });

  Map.addInitHook(function ()
{
        if
(this.options.zoomControl) {
                // @section
Controls
                // @property
zoomControl: Control.Zoom
                // The default
zoom control (only available
if the
                //
[`zoomControl` option](#map-
zoomcontrol) was `true` when
creating the map).

this.zoomControl = new Zoom();

this.addControl(this.zoomContr
ol);
        }
  });

  // @namespace Control.Zoom
  // @factory
L.control.zoom(options:
Control.Zoom options)
  // Creates a zoom control
  var zoom = function
```

```
(options) {
        return new
Zoom(options);
  };

  /*
   * @class Control.Scale
   * @aka L.Control.Scale
   * @inherits Control
   *
   * 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`.
   *
   * @example
   *
   * ```js
   *
L.control.scale().addTo(map);
   * ```
   */

  var Scale = Control.extend({
        // @section
        // @aka Control.Scale
options
        options: {
                position:
'bottomleft',

                // @option
maxWidth: Number = 100
                // Maximum
width of the control in
pixels. The width is set
dynamically to show round
values (e.g. 100, 200, 500).
                maxWidth: 100,

                // @option
metric: Boolean = True
                // Whether to
show the metric scale line
(m/km).
```

```
                metric: true,

                // @option
imperial: Boolean = True
                // Whether to
show the imperial scale line
(mi/ft).
                imperial: true

                // @option
updateWhenIdle: Boolean =
false
                // If `true`,
the control is updated on
[`moveend`](#map-moveend),
otherwise it's always up-to-
date (updated on [`move`]
(#map-move)).
        },

        onAdd: function (map)
{
                var className
= 'leaflet-control-scale',
                    container
= create$1('div', className),
                    options =
this.options;


this._addScales(options,
className + '-line',
container);


map.on(options.updateWhenIdle
? 'moveend' : 'move',
this._update, this);

map.whenReady(this._update,
this);

                return
container;
        },

        onRemove: function
```

```
(map) {

map.off(this.options.updateWhe
nIdle ? 'moveend' : 'move',
this._update, this);
        },

        _addScales: function
(options, className,
container) {
                if
(options.metric) {

this._mScale = create$1('div',
className, container);
                }
                if
(options.imperial) {

this._iScale = create$1('div',
className, container);
                }
        },

        _update: function () {
                var map =
this._map,
                    y =
map.getSize().y / 2;

                var maxMeters
= map.distance(

map.containerPointToLatLng([0,
y]),

map.containerPointToLatLng([th
is.options.maxWidth, y]));


this._updateScales(maxMeters);
        },

        _updateScales:
function (maxMeters) {
                if
(this.options.metric &&
```

```
maxMeters) {

this._updateMetric(maxMeters);
                }
                if
(this.options.imperial &&
maxMeters) {

this._updateImperial(maxMeters
);
                }
        },

        _updateMetric:
function (maxMeters) {
                var meters =
this._getRoundNum(maxMeters),
                    label =
meters < 1000 ? meters + ' m'
: (meters / 1000) + ' km';


this._updateScale(this._mScale
, label, meters / maxMeters);
        },

        _updateImperial:
function (maxMeters) {
                var maxFeet =
maxMeters * 3.2808399,
                    maxMiles,
miles, feet;

                if (maxFeet >
5280) {

maxMiles = maxFeet / 5280;
                        miles
= this._getRoundNum(maxMiles);

this._updateScale(this._iScale
, miles + ' mi', miles /
maxMiles);

                } else {
                        feet =
this._getRoundNum(maxFeet);
```

```
this._updateScale(this._iScale
, feet + ' ft', feet /
maxFeet);
                }
        },

        _updateScale: function
(scale, text, ratio) {

scale.style.width =
Math.round(this.options.maxWid
th * ratio) + 'px';

scale.innerHTML = text;
        },

        _getRoundNum: function
(num) {
                var pow10 =
Math.pow(10, (Math.floor(num)
+ '').length - 1),
                    d = num /
pow10;

                d = d >= 10 ?
10 :
                    d >= 5 ? 5
:
                    d >= 3 ? 3
:
                    d >= 2 ? 2
: 1;

                return pow10 *
d;
        }
  });


  // @factory
L.control.scale(options?:
Control.Scale options)
  // Creates an scale control
with the given options.
  var scale = function
(options) {
```

```
        return new
Scale(options);
  };

  var ukrainianFlag = '<svg
aria-hidden="true"
xmlns="http://www.w3.org/2000/
svg" width="12" height="8"
viewBox="0 0 12 8"
class="leaflet-attribution-
flag"><path fill="#4C7BE1"
d="M0 0h12v4H0z"/><path
fill="#FFD500" d="M0
4h12v3H0z"/><path
fill="#E0BC00" d="M0
7h12v1H0z"/></svg>';


  /*
   * @class
Control.Attribution
   * @aka
L.Control.Attribution
   * @inherits Control
   *
   * 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]
(#map-attributioncontrol) to
`false`, and it fetches
attribution texts from layers
with the [`getAttribution`
method](#layer-getattribution)
automatically. Extends
Control.
   */

  var Attribution =
Control.extend({
        // @section
        // @aka
Control.Attribution options
        options: {
```

```
                position:
'bottomright',

                // @option
prefix: String|false =
'Leaflet'
                // The HTML
text shown before the
attributions. Pass `false` to
disable.
                prefix: '<a
href="https://leafletjs.com"
title="A JavaScript library
for interactive maps">' +
(Browser.inlineSvg ?
ukrainianFlag + ' ' : '') +
'Leaflet</a>'
        },

        initialize: function
(options) {

setOptions(this, options);


this._attributions = {};
        },

        onAdd: function (map)
{

map.attributionControl = this;

this._container =
create$1('div', 'leaflet-
control-attribution');

disableClickPropagation(this._
container);

                // TODO ugly,
refactor
                for (var i in
map._layers) {
                        if
(map._layers[i].getAttribution
) {
```

```
this.addAttribution(map._layer
s[i].getAttribution());
                        }
                }


this._update();


map.on('layeradd',
this._addAttribution, this);

                return
this._container;
        },

        onRemove: function
(map) {

map.off('layeradd',
this._addAttribution, this);
        },

        _addAttribution:
function (ev) {
                if
(ev.layer.getAttribution) {

this.addAttribution(ev.layer.g
etAttribution());

ev.layer.once('remove',
function () {

this.removeAttribution(ev.laye
r.getAttribution());
                        },
this);
                }
        },

        // @method
setPrefix(prefix:
String|false): this
        // The HTML text shown
before the attributions. Pass
```

```
`false` to disable.
        setPrefix: function
(prefix) {

this.options.prefix = prefix;

this._update();
                return this;
        },

        // @method
addAttribution(text: String):
this
        // Adds an attribution
text (e.g. `'&copy;
OpenStreetMap contributors'`).
        addAttribution:
function (text) {
                if (!text) {
return this; }

                if
(!this._attributions[text]) {

this._attributions[text] = 0;
                }

this._attributions[text]++;


this._update();

                return this;
        },

        // @method
removeAttribution(text:
String): this
        // Removes an
attribution text.
        removeAttribution:
function (text) {
                if (!text) {
return this; }

                if
(this._attributions[text]) {
```

```
this._attributions[text]--;

this._update();
                }

                return this;
        },

        _update: function () {
                if
(!this._map) { return; }

                var attribs =
[];

                for (var i in
this._attributions) {
                        if
(this._attributions[i]) {

attribs.push(i);
                        }
                }

                var
prefixAndAttribs = [];

                if
(this.options.prefix) {

prefixAndAttribs.push(this.opt
ions.prefix);
                }
                if
(attribs.length) {

prefixAndAttribs.push(attribs.
join(', '));
                }


this._container.innerHTML =
prefixAndAttribs.join(' <span
aria-hidden="true">|</span>
');
        }
```

```
  });

  // @namespace Map
  // @section Control options
  // @option
attributionControl: Boolean =
true
  // Whether a [attribution
control](#control-attribution)
is added to the map by
default.
  Map.mergeOptions({
        attributionControl:
true
  });

  Map.addInitHook(function ()
{
        if
(this.options.attributionContr
ol) {
                new
Attribution().addTo(this);
        }
  });

  // @namespace
Control.Attribution
  // @factory
L.control.attribution(options:
Control.Attribution options)
  // Creates an attribution
control.
  var attribution = function
(options) {
        return new
Attribution(options);
  };

  Control.Layers = Layers;
  Control.Zoom = Zoom;
  Control.Scale = Scale;
  Control.Attribution =
Attribution;

  control.layers = layers;
  control.zoom = zoom;
```

```
  control.scale = scale;
  control.attribution =
attribution;

  /*
        L.Handler is a base
class for handler classes that
are used internally to inject
        interaction features
like dragging to classes like
Map and Marker.
  */

  // @class Handler
  // @aka L.Handler
  // Abstract class for map
interaction handlers

  var Handler = Class.extend({
        initialize: function
(map) {
                this._map =
map;
        },

        // @method enable():
this
        // Enables the handler
        enable: function () {
                if
(this._enabled) { return this;
}

                this._enabled
= true;

this.addHooks();
                return this;
        },

        // @method disable():
this
        // Disables the
handler
        disable: function () {
                if
(!this._enabled) { return
```

```
this; }

                this._enabled
= false;

this.removeHooks();
                return this;
        },

        // @method enabled():
Boolean
        // Returns `true` if
the handler is enabled
        enabled: function () {
                return
!!this._enabled;
        }

        // @section Extension
methods
        // Classes inheriting
from `Handler` must implement
the two following methods:
        // @method addHooks()
        // Called when the
handler is enabled, should add
event hooks.
        // @method
removeHooks()
        // Called when the
handler is disabled, should
remove the event hooks added
previously.
  });

  // @section There is static
function which can be called
without instantiating
L.Handler:
  // @function addTo(map: Map,
name: String): this
  // Adds a new Handler to the
given map with the given name.
  Handler.addTo = function
(map, name) {
        map.addHandler(name,
this);
```

```
        return this;
  };

  var Mixin = {Events:
Events};

  /*
   * @class Draggable
   * @aka L.Draggable
   * @inherits Evented
   *
   * A class for making DOM
elements draggable (including
touch support).
   * Used internally for map
and marker dragging. Only
works for elements
   * that were positioned with
[`L.DomUtil.setPosition`]
(#domutil-setposition).
   *
   * @example
   * ```js
   * var draggable = new
L.Draggable(elementToDrag);
   * draggable.enable();
   * ```
   */

  var START = Browser.touch ?
'touchstart mousedown' :
'mousedown';

  var Draggable =
Evented.extend({

        options: {
                // @section
                // @aka
Draggable options
                // @option
clickTolerance: Number = 3
                // The max
number of pixels a user can
shift the mouse pointer during
a click
                // for it to
```

```
be considered a valid click
(as opposed to a mouse drag).

clickTolerance: 3
        },

        // @constructor
L.Draggable(el: HTMLElement,
dragHandle?: HTMLElement,
preventOutline?: Boolean,
options?: Draggable options)
        // Creates a
`Draggable` object for moving
`el` when you start dragging
the `dragHandle` element
(equals `el` itself by
default).
        initialize: function
(element, dragStartTarget,
preventOutline, options) {

setOptions(this, options);

                this._element
= element;

this._dragStartTarget =
dragStartTarget || element;

this._preventOutline =
preventOutline;
        },

        // @method enable()
        // Enables the
dragging ability
        enable: function () {
                if
(this._enabled) { return; }


on(this._dragStartTarget,
START, this._onDown, this);

                this._enabled
= true;
        },
```

```
        // @method disable()
        // Disables the
dragging ability
        disable: function () {
                if
(!this._enabled) { return; }

                // If we're
currently dragging this
draggable,
                // disabling
it counts as first ending the
drag.
                if
(Draggable._dragging === this)
{

this.finishDrag(true);
                }


off(this._dragStartTarget,
START, this._onDown, this);

                this._enabled
= false;
                this._moved =
false;
        },

        _onDown: function (e)
{
                // Ignore the
event if disabled; this
happens in IE11
                // under some
circumstances, see #3666.
                if
(!this._enabled) { return; }

                this._moved =
false;

                if
(hasClass(this._element,
'leaflet-zoom-anim')) {
```

```
return; }

                if (e.touches
&& e.touches.length !== 1) {
                        //
Finish dragging to avoid
conflict with touchZoom
                        if
(Draggable._dragging === this)
{

this.finishDrag();
                        }

return;
                }

                if
(Draggable._dragging ||
e.shiftKey || ((e.which !== 1)
&& (e.button !== 1) &&
!e.touches)) { return; }

Draggable._dragging = this;
// Prevent dragging multiple
objects at once.

                if
(this._preventOutline) {

preventOutline(this._element);
                }


disableImageDrag();

disableTextSelection();

                if
(this._moving) { return; }

                // @event
down: Event
                // Fired when
a drag is about to start.

this.fire('down');
```

```
                var first =
e.touches ? e.touches[0] : e,

sizedParent =
getSizedParentNode(this._eleme
nt);


this._startPoint = new
Point(first.clientX,
first.clientY);
                this._startPos
= getPosition(this._element);

                // Cache the
scale, so that we can
continuously compensate for it
during drag (_onMove).

this._parentScale =
getScale(sizedParent);

                var mouseevent
= e.type === 'mousedown';
                on(document,
mouseevent ? 'mousemove' :
'touchmove', this._onMove,
this);
                on(document,
mouseevent ? 'mouseup' :
'touchend touchcancel',
this._onUp, this);
        },

        _onMove: function (e)
{
                // Ignore the
event if disabled; this
happens in IE11
                // under some
circumstances, see #3666.
                if
(!this._enabled) { return; }

                if (e.touches
&& e.touches.length > 1) {
```

```
this._moved = true;

return;
                }

                var first =
(e.touches && e.touches.length
=== 1 ? e.touches[0] : e),
                    offset =
new Point(first.clientX,
first.clientY)._subtract(this.
_startPoint);

                if (!offset.x
&& !offset.y) { return; }
                if
(Math.abs(offset.x) +
Math.abs(offset.y) <
this.options.clickTolerance) {
return; }

                // We assume
that the parent container's
position, border and scale do
not change for the duration of
the drag.
                // Therefore
there is no need to account
for the position and border
(they are eliminated by the
subtraction)
                // and we can
use the cached value for the
scale.
                offset.x /=
this._parentScale.x;
                offset.y /=
this._parentScale.y;


preventDefault(e);

                if
(!this._moved) {
                    //
@event dragstart: Event
```

```
                        //
Fired when a drag starts

this.fire('dragstart');


this._moved = true;


addClass(document.body,
'leaflet-dragging');


this._lastTarget = e.target ||
e.srcElement;
                        // IE
and Edge do not give the <use>
element, so fetch it
                        // if
necessary
                        if
(window.SVGElementInstance &&
this._lastTarget instanceof
window.SVGElementInstance) {

this._lastTarget =
this._lastTarget.corresponding
UseElement;
                        }

addClass(this._lastTarget,
'leaflet-drag-target');
                }

                this._newPos =
this._startPos.add(offset);
                this._moving =
true;


this._lastEvent = e;

this._updatePosition();
        },

        _updatePosition:
function () {
```

```
                var e =
{originalEvent:
this._lastEvent};

                // @event
predrag: Event
                // Fired
continuously during dragging
*before* each corresponding
                // update of
the element's position.

this.fire('predrag', e);

setPosition(this._element,
this._newPos);

                // @event
drag: Event
                // Fired
continuously during dragging.

this.fire('drag', e);
        },

        _onUp: function () {
                // Ignore the
event if disabled; this
happens in IE11
                // under some
circumstances, see #3666.
                if
(!this._enabled) { return; }

this.finishDrag();
        },

        finishDrag: function
(noInertia) {

removeClass(document.body,
'leaflet-dragging');

                if
(this._lastTarget) {

removeClass(this._lastTarget,
```

```
'leaflet-drag-target');

this._lastTarget = null;
                }

                off(document,
'mousemove touchmove',
this._onMove, this);
                off(document,
'mouseup touchend
touchcancel', this._onUp,
this);


enableImageDrag();

enableTextSelection();

                if
(this._moved && this._moving)
{

                        //
@event dragend: DragEndEvent
                        //
Fired when the drag ends.

this.fire('dragend', {

noInertia: noInertia,

distance:
this._newPos.distanceTo(this._
startPos)
                        });
                }

                this._moving =
false;

Draggable._dragging = false;
        }

  });

  /*
   * @namespace LineUtil
```

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

  // Simplify polyline with
vertex reduction and Douglas-
Peucker simplification.
  // Improves rendering
performance dramatically by
lessening the number of points
to draw.

  // @function
simplify(points: Point[],
tolerance: Number): Point[]
  // Dramatically reduces the
number of points in a polyline
while retaining
  // its shape and returns a
new array of simplified
points, using the
  // [Ramer-Douglas-Peucker
algorithm]
(https://en.wikipedia.org/wiki
/Ramer-Douglas-
Peucker_algorithm).
  // Used for a huge
performance boost when
processing/displaying Leaflet
polylines for
  // each zoom level and also
reducing visual noise.
tolerance affects the amount
of
  // simplification (lesser
value means higher quality but
slower and with more points).
  // Also released as a
separated micro-library
[Simplify.js]
(https://mourner.github.io/sim
plify-js/).
  function simplify(points,
```

```
tolerance) {
        if (!tolerance ||
!points.length) {
                return
points.slice();
        }

        var sqTolerance =
tolerance * tolerance;

            // stage 1: vertex
reduction
            points =
_reducePoints(points,
sqTolerance);

            // stage 2:
Douglas-Peucker simplification
            points =
_simplifyDP(points,
sqTolerance);

        return points;
  }

  // @function
pointToSegmentDistance(p:
Point, p1: Point, p2: Point):
Number
  // Returns the distance
between point `p` and segment
`p1` to `p2`.
  function
pointToSegmentDistance(p, p1,
p2) {
        return
Math.sqrt(_sqClosestPointOnSeg
ment(p, p1, p2, true));
  }

  // @function
closestPointOnSegment(p:
Point, p1: Point, p2: Point):
Number
  // Returns the closest point
from a point `p` on a segment
`p1` to `p2`.
```

```
  function
closestPointOnSegment(p, p1,
p2) {
        return
_sqClosestPointOnSegment(p,
p1, p2);
  }

  // Ramer-Douglas-Peucker
simplification, see
https://en.wikipedia.org/wiki/
Ramer-Douglas-
Peucker_algorithm
  function _simplifyDP(points,
sqTolerance) {

        var len =
points.length,
            ArrayConstructor =
typeof Uint8Array !==
undefined + '' ? Uint8Array :
Array,
            markers = new
ArrayConstructor(len);

            markers[0] =
markers[len - 1] = 1;


_simplifyDPStep(points,
markers, sqTolerance, 0, len -
1);

        var i,
            newPoints = [];

        for (i = 0; i < len;
i++) {
                if
(markers[i]) {

newPoints.push(points[i]);
                }
        }

        return newPoints;
  }
```

```
  function
_simplifyDPStep(points,
markers, sqTolerance, first,
last) {

        var maxSqDist = 0,
        index, i, sqDist;

        for (i = first + 1; i
<= last - 1; i++) {
                sqDist =
_sqClosestPointOnSegment(point
s[i], points[first],
points[last], true);

                if (sqDist >
maxSqDist) {
                        index
= i;

maxSqDist = sqDist;
                }
        }

        if (maxSqDist >
sqTolerance) {
                markers[index]
= 1;


_simplifyDPStep(points,
markers, sqTolerance, first,
index);

_simplifyDPStep(points,
markers, sqTolerance, index,
last);
        }
  }

  // reduce points that are
too close to each other to a
single point
  function
_reducePoints(points,
sqTolerance) {
```

```
        var reducedPoints =
[points[0]];

        for (var i = 1, prev =
0, len = points.length; i <
len; i++) {
                if
(_sqDist(points[i],
points[prev]) > sqTolerance) {

reducedPoints.push(points[i]);
                        prev =
i;
                }
        }
        if (prev < len - 1) {

reducedPoints.push(points[len
- 1]);
        }
        return reducedPoints;
  }

  var _lastCode;

  // @function clipSegment(a:
Point, b: Point, bounds:
Bounds, useLastCode?: Boolean,
round?: Boolean):
Point[]|Boolean
  // Clips the segment a to b
by rectangular bounds with the
  // [Cohen-Sutherland
algorithm]
(https://en.wikipedia.org/wiki
/Cohen%E2%80%93Sutherland_algo
rithm)
  // (modifying the segment
points directly!). Used by
Leaflet to only show polyline
  // points that are on the
screen or near, increasing
performance.
  function clipSegment(a, b,
bounds, useLastCode, round) {
        var codeA =
useLastCode ? _lastCode :
```

```
_getBitCode(a, bounds),
            codeB =
_getBitCode(b, bounds),

            codeOut, p,
newCode;

            // save 2nd code
to avoid calculating it on the
next segment
            _lastCode = codeB;

        while (true) {
                // if a,b is
inside the clip window
(trivial accept)
                if (!(codeA |
codeB)) {
                        return
[a, b];
                }

                // if a,b is
outside the clip window
(trivial reject)
                if (codeA &
codeB) {
                        return
false;
                }

                // other cases
                codeOut =
codeA || codeB;
                p =
_getEdgeIntersection(a, b,
codeOut, bounds, round);
                newCode =
_getBitCode(p, bounds);

                if (codeOut
=== codeA) {
                        a = p;
                        codeA
= newCode;
                } else {
                        b = p;
```

```
                        codeB
= newCode;
                }
        }
  }

  function
_getEdgeIntersection(a, b,
code, bounds, round) {
        var dx = b.x - a.x,
            dy = b.y - a.y,
            min = bounds.min,
            max = bounds.max,
            x, y;

        if (code & 8) { // top
                x = a.x + dx *
(max.y - a.y) / dy;
                y = max.y;

        } else if (code & 4) {
// bottom
                x = a.x + dx *
(min.y - a.y) / dy;
                y = min.y;

        } else if (code & 2) {
// right
                x = max.x;
                y = a.y + dy *
(max.x - a.x) / dx;

        } else if (code & 1) {
// left
                x = min.x;
                y = a.y + dy *
(min.x - a.x) / dx;
        }

        return new Point(x, y,
round);
  }

  function _getBitCode(p,
bounds) {
        var code = 0;
```

```
        if (p.x <
bounds.min.x) { // left
                code |= 1;
        } else if (p.x >
bounds.max.x) { // right
                code |= 2;
        }

        if (p.y <
bounds.min.y) { // bottom
                code |= 4;
        } else if (p.y >
bounds.max.y) { // top
                code |= 8;
        }

        return code;
  }

  // square distance (to avoid
unnecessary Math.sqrt calls)
  function _sqDist(p1, p2) {
        var dx = p2.x - p1.x,
            dy = p2.y - p1.y;
        return dx * dx + dy *
dy;
  }

  // return closest point on
segment or distance to that
point
  function
_sqClosestPointOnSegment(p,
p1, p2, sqDist) {
        var x = p1.x,
            y = p1.y,
            dx = p2.x - x,
            dy = p2.y - y,
            dot = dx * dx + dy
* dy,
            t;

        if (dot > 0) {
                t = ((p.x - x)
* dx + (p.y - y) * dy) / dot;

                if (t > 1) {
```

```
                        x =
p2.x;
                        y =
p2.y;
                } else if (t >
0) {
                        x +=
dx * t;
                        y +=
dy * t;
                }
        }

        dx = p.x - x;
        dy = p.y - y;

        return sqDist ? dx *
dx + dy * dy : new Point(x,
y);
  }


  // @function isFlat(latlngs:
LatLng[]): Boolean
  // Returns true if `latlngs`
is a flat array, false is
nested.
  function isFlat(latlngs) {
        return
!isArray(latlngs[0]) ||
(typeof latlngs[0][0] !==
'object' && typeof latlngs[0]
[0] !== 'undefined');
  }

  function _flat(latlngs) {

console.warn('Deprecated use
of _flat, please use
L.LineUtil.isFlat instead.');
        return
isFlat(latlngs);
  }

  /* @function
polylineCenter(latlngs:
LatLng[], crs: CRS): LatLng
```

```
   * Returns the center
([centroid]
(http://en.wikipedia.org/wiki/
Centroid)) of the passed
LatLngs (first ring) from a
polyline.
   */
  function
polylineCenter(latlngs, crs) {
        var i, halfDist,
segDist, dist, p1, p2, ratio,
center;

        if (!latlngs ||
latlngs.length === 0) {
                throw new
Error('latlngs not passed');
        }

        if (!isFlat(latlngs))
{

console.warn('latlngs are not
flat! Only the first ring will
be used');
                latlngs =
latlngs[0];
        }

        var points = [];
        for (var j in latlngs)
{

points.push(crs.project(toLatL
ng(latlngs[j])));
        }

        var len =
points.length;

        for (i = 0, halfDist =
0; i < len - 1; i++) {
                halfDist +=
points[i].distanceTo(points[i
+ 1]) / 2;
        }
```

```
        // The line is so
small in the current view that
all points are on the same
pixel.
        if (halfDist === 0) {
                center =
points[0];
        } else {
                for (i = 0,
dist = 0; i < len - 1; i++) {
                        p1 =
points[i];
                        p2 =
points[i + 1];

segDist = p1.distanceTo(p2);
                        dist
+= segDist;

                        if
(dist > halfDist) {

ratio = (dist - halfDist) /
segDist;

center = [

p2.x - ratio * (p2.x - p1.x),

p2.y - ratio * (p2.y - p1.y)

];

break;
                        }
                }
        }
        return
crs.unproject(toPoint(center))
;
  }

  var LineUtil = {
    __proto__: null,
    simplify: simplify,
    pointToSegmentDistance:
pointToSegmentDistance,
```

```
    closestPointOnSegment:
closestPointOnSegment,
    clipSegment: clipSegment,
    _getEdgeIntersection:
_getEdgeIntersection,
    _getBitCode: _getBitCode,
    _sqClosestPointOnSegment:
_sqClosestPointOnSegment,
    isFlat: isFlat,
    _flat: _flat,
    polylineCenter:
polylineCenter
  };

  /*
   * @namespace PolyUtil
   * Various utility functions
for polygon geometries.
   */

  /* @function
clipPolygon(points: Point[],
bounds: Bounds, round?:
Boolean): Point[]
   * Clips the polygon
geometry defined by the given
`points` by the given bounds
(using the [Sutherland-Hodgman
algorithm]
(https://en.wikipedia.org/wiki
/Sutherland%E2%80%93Hodgman_al
gorithm)).
   * Used by Leaflet to only
show polygon points that are
on the screen or near,
increasing
   * performance. Note that
polygon points needs different
algorithm for clipping
   * than polyline, so there's
a separate method for it.
   */
  function clipPolygon(points,
bounds, round) {
        var clippedPoints,
            edges = [1, 4, 2,
8],
```

```
            i, j, k,
            a, b,
            len, edge, p;

        for (i = 0, len =
points.length; i < len; i++) {

points[i]._code =
_getBitCode(points[i],
bounds);
        }

        // for each edge
(left, bottom, right, top)
        for (k = 0; k < 4;
k++) {
                edge =
edges[k];
                clippedPoints
= [];

                for (i = 0,
len = points.length, j = len -
1; i < len; j = i++) {
                        a =
points[i];
                        b =
points[j];

                        // if
a is inside the clip window
                        if (!
(a._code & edge)) {

// if b is outside the clip
window (a->b goes out of
screen)

if (b._code & edge) {

p = _getEdgeIntersection(b, a,
edge, bounds, round);

p._code = _getBitCode(p,
bounds);

clippedPoints.push(p);
```

```
}

clippedPoints.push(a);

                        //
else if b is inside the clip
window (a->b enters the
screen)
                        } else
if (!(b._code & edge)) {

p = _getEdgeIntersection(b, a,
edge, bounds, round);

p._code = _getBitCode(p,
bounds);

clippedPoints.push(p);
                        }
                }
                points =
clippedPoints;
        }

        return points;
  }

  /* @function
polygonCenter(latlngs:
LatLng[] crs: CRS): LatLng
   * Returns the center
([centroid]
(http://en.wikipedia.org/wiki/
Centroid)) of the passed
LatLngs (first ring) from a
polygon.
   */
  function
polygonCenter(latlngs, crs) {
        var i, j, p1, p2, f,
area, x, y, center;

        if (!latlngs ||
latlngs.length === 0) {
                throw new
Error('latlngs not passed');
```

```
        }

        if (!isFlat(latlngs))
{

console.warn('latlngs are not
flat! Only the first ring will
be used');
                latlngs =
latlngs[0];
        }

        var points = [];
        for (var k in latlngs)
{

points.push(crs.project(toLatL
ng(latlngs[k])));
        }

        var len =
points.length;
        area = x = y = 0;

        // polygon centroid
algorithm;
        for (i = 0, j = len -
1; i < len; j = i++) {
                p1 =
points[i];
                p2 =
points[j];

                f = p1.y *
p2.x - p2.y * p1.x;
                x += (p1.x +
p2.x) * f;
                y += (p1.y +
p2.y) * f;
                area += f * 3;
        }

        if (area === 0) {
                // Polygon is
so small that all points are
on same pixel.
                center =
```

```
points[0];
        } else {
                center = [x /
area, y / area];
        }
        return
crs.unproject(toPoint(center))
;
  }

  var PolyUtil = {
    __proto__: null,
    clipPolygon: clipPolygon,
    polygonCenter:
polygonCenter
  };

  /*
   * @namespace Projection
   * @section
   * Leaflet comes with a set
of already defined Projections
out of the box:
   *
   * @projection
L.Projection.LonLat
   *
   * Equirectangular, or Plate
Carree projection — the most
simple projection,
   * mostly used by GIS
enthusiasts. Directly maps `x`
as longitude, and `y` as
   * latitude. Also suitable
for flat worlds, e.g. game
maps. Used by the
   * `EPSG:4326` and `Simple`
CRS.
   */

  var LonLat = {
        project: function
(latlng) {
                return new
Point(latlng.lng, latlng.lat);
        },
```

```
        unproject: function
(point) {
                return new
LatLng(point.y, point.x);
        },

        bounds: new
Bounds([-180, -90], [180, 90])
  };

  /*
   * @namespace Projection
   * @projection
L.Projection.Mercator
   *
   * Elliptical Mercator
projection — more complex than
Spherical Mercator. Assumes
that Earth is an ellipsoid.
Used by the EPSG:3395 CRS.
   */

  var Mercator = {
        R: 6378137,
        R_MINOR:
6356752.314245179,

        bounds: new
Bounds([-20037508.34279,
-15496570.73972],
[20037508.34279,
18764656.23138]),

        project: function
(latlng) {
                var d =
Math.PI / 180,
                    r =
this.R,
                    y =
latlng.lat * d,
                    tmp =
this.R_MINOR / r,
                    e =
Math.sqrt(1 - tmp * tmp),
                    con = e *
Math.sin(y);
```

```
                var ts =
Math.tan(Math.PI / 4 - y / 2)
/ Math.pow((1 - con) / (1 +
con), e / 2);
                y = -r *
Math.log(Math.max(ts, 1E-10));

                return new
Point(latlng.lng * d * r, y);
        },

        unproject: function
(point) {
                var d = 180 /
Math.PI,
                    r =
this.R,
                    tmp =
this.R_MINOR / r,
                    e =
Math.sqrt(1 - tmp * tmp),
                    ts =
Math.exp(-point.y / r),
                    phi =
Math.PI / 2 - 2 *
Math.atan(ts);

                for (var i =
0, dphi = 0.1, con; i < 15 &&
Math.abs(dphi) > 1e-7; i++) {
                        con =
e * Math.sin(phi);
                        con =
Math.pow((1 - con) / (1 +
con), e / 2);
                        dphi =
Math.PI / 2 - 2 * Math.atan(ts
* con) - phi;
                        phi +=
dphi;
                }

                return new
LatLng(phi * d, point.x * d /
r);
        }
```

```
  };

  /*
   * @class Projection

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

   * @property bounds: Bounds
   * The bounds (specified in
CRS units) where the
projection is valid

   * @method project(latlng:
LatLng): Point
   * Projects geographical
coordinates into a 2D point.
   * Only accepts actual
`L.LatLng` instances, not
arrays.

   * @method unproject(point:
Point): LatLng
   * The inverse of `project`.
Projects a 2D point into a
geographical location.
   * Only accepts actual
`L.Point` instances, not
arrays.

   * Note that the projection
instances do not inherit from
Leaflet's `Class` object,
   * and can't be
instantiated. Also, new
classes can't inherit from
them,
   * and methods can't be
added to them with the
`include` function.

   */
```

```
  var index = {
    __proto__: null,
    LonLat: LonLat,
    Mercator: Mercator,
    SphericalMercator:
SphericalMercator
  };

  /*
   * @namespace CRS
   * @crs L.CRS.EPSG3395
   *
   * Rarely used by some
commercial tile providers.
Uses Elliptical Mercator
projection.
   */
  var EPSG3395 = extend({},
Earth, {
        code: 'EPSG:3395',
        projection: Mercator,

        transformation:
(function () {
                var scale =
0.5 / (Math.PI * Mercator.R);
                return
toTransformation(scale, 0.5, -
scale, 0.5);
        }())
  });

  /*
   * @namespace CRS
   * @crs L.CRS.EPSG4326
   *
   * A common CRS among GIS
enthusiasts. Uses simple
Equirectangular projection.
   *
   * Leaflet 1.0.x complies
with the [TMS coordinate
scheme for EPSG:4326]
(https://wiki.osgeo.org/wiki/T
ile_Map_Service_Specification#
global-geodetic),
```

```
   * which is a breaking
change from 0.7.x behaviour.
If you are using a `TileLayer`
   * with this CRS, ensure
that there are two 256x256
pixel tiles covering the
   * whole earth at zoom level
zero, and that the tile
coordinate origin is
(-180,+90),
   * or (-180,-90) for
`TileLayer`s with [the `tms`
option](#tilelayer-tms) set.
   */

  var EPSG4326 = extend({},
Earth, {
        code: 'EPSG:4326',
        projection: LonLat,
        transformation:
toTransformation(1 / 180, 1,
-1 / 180, 0.5)
  });

  /*
   * @namespace CRS
   * @crs L.CRS.Simple
   *
   * A simple CRS that maps
longitude and latitude into
`x` and `y` directly.
   * May be used for maps of
flat surfaces (e.g. game
maps). Note that the `y`
   * axis should still be
inverted (going from bottom to
top). `distance()` returns
   * simple euclidean
distance.
   */

  var Simple = extend({}, CRS,
{
        projection: LonLat,
        transformation:
toTransformation(1, 0, -1, 0),
```

```
        scale: function (zoom)
{
                return
Math.pow(2, zoom);
        },

        zoom: function (scale)
{
                return
Math.log(scale) / Math.LN2;
        },

        distance: function
(latlng1, latlng2) {
                var dx =
latlng2.lng - latlng1.lng,
                    dy =
latlng2.lat - latlng1.lat;

                return
Math.sqrt(dx * dx + dy * dy);
        },

        infinite: true
  });

  CRS.Earth = Earth;
  CRS.EPSG3395 = EPSG3395;
  CRS.EPSG3857 = EPSG3857;
  CRS.EPSG900913 = EPSG900913;
  CRS.EPSG4326 = EPSG4326;
  CRS.Simple = Simple;

  /*
   * @class Layer
   * @inherits Evented
   * @aka L.Layer
   * @aka ILayer
   *
   * A set of methods from the
Layer base class that all
Leaflet layers use.
   * Inherits all methods,
options and events from
`L.Evented`.
   *
   * @example
```

```
   *
   * ```js
   * var layer =
L.marker(latlng).addTo(map);
   * layer.addTo(map);
   * layer.remove();
   * ```
   *
   * @event add: Event
   * Fired after the layer is
added to a map
   *
   * @event remove: Event
   * Fired after the layer is
removed from a map
   */


  var Layer = Evented.extend({

        // Classes extending
`L.Layer` will inherit the
following options:
        options: {
                // @option
pane: String = 'overlayPane'
                // By default
the layer will be added to the
map's [overlay pane](#map-
overlaypane). Overriding this
option will cause the layer to
be placed on another pane by
default.
                pane:
'overlayPane',

                // @option
attribution: String = null
                // String to
be shown in the attribution
control, e.g. "© OpenStreetMap
contributors". It describes
the layer data and is often a
legal obligation towards
copyright holders and tile
providers.
                attribution:
```

```
null,


bubblingMouseEvents: true
        },

        /* @section
         * Classes extending
`L.Layer` will inherit the
following methods:
         *
         * @method addTo(map:
Map|LayerGroup): this
         * Adds the layer to
the given map or layer group.
         */
        addTo: function (map)
{

map.addLayer(this);
                return this;
        },

        // @method remove:
this
        // Removes the layer
from the map it is currently
active on.
        remove: function () {
                return
this.removeFrom(this._map ||
this._mapToAdd);
        },

        // @method
removeFrom(map: Map): this
        // Removes the layer
from the given map
        //
        // @alternative
        // @method
removeFrom(group: LayerGroup):
this
        // Removes the layer
from the given `LayerGroup`
        removeFrom: function
(obj) {
```

```
                if (obj) {

obj.removeLayer(this);
                }
                return this;
        },

        // @method
getPane(name? : String):
HTMLElement
        // Returns the
`HTMLElement` representing the
named pane on the map. If
`name` is omitted, returns the
pane for this layer.
        getPane: function
(name) {
                return
this._map.getPane(name ?
(this.options[name] || name) :
this.options.pane);
        },

        addInteractiveTarget:
function (targetEl) {

this._map._targets[stamp(targe
tEl)] = this;
                return this;
        },


removeInteractiveTarget:
function (targetEl) {
                delete
this._map._targets[stamp(targe
tEl)];
                return this;
        },

        // @method
getAttribution: String
        // Used by the
`attribution control`, returns
the [attribution option]
(#gridlayer-attribution).
        getAttribution:
```

```
function () {
                return
this.options.attribution;
        },

        _layerAdd: function
(e) {
                var map =
e.target;

                // check in
case layer gets added and then
removed before the map is
ready
                if
(!map.hasLayer(this)) {
return; }

                this._map =
map;

this._zoomAnimated =
map._zoomAnimated;

                if
(this.getEvents) {
                        var
events = this.getEvents();

map.on(events, this);

this.once('remove', function
() {

map.off(events, this);
                        },
this);
                }


this.onAdd(map);


this.fire('add');

map.fire('layeradd', {layer:
this});
```

```
        }
  });

  /* @section Extension
methods
   * @uninheritable
   *
   * Every layer should extend
from `L.Layer` and (re-
)implement the following
methods.
   *
   * @method onAdd(map: Map):
this
   * Should contain code that
creates DOM elements for the
layer, adds them to `map
panes` where they should
belong and puts listeners on
relevant map events. Called on
[`map.addLayer(layer)`](#map-
addlayer).
   *
   * @method onRemove(map:
Map): this
   * Should contain all clean
up code that removes the
layer's elements from the DOM
and removes listeners
previously added in [`onAdd`]
(#layer-onadd). Called on
[`map.removeLayer(layer)`]
(#map-removelayer).
   *
   * @method getEvents():
Object
   * This optional method
should return an object like
`{ viewreset: this._reset }`
for [`addEventListener`]
(#evented-addeventlistener).
The event handlers in this
object will be automatically
added and removed from the map
with your layer.
   *
   * @method getAttribution():
```

```
String
   * This optional method
should return a string
containing HTML to be shown on
the `Attribution control`
whenever the layer is visible.
   *
   * @method beforeAdd(map:
Map): this
   * Optional method. Called
on [`map.addLayer(layer)`]
(#map-addlayer), before the
layer is added to the map,
before events are initialized,
without waiting until the map
is in a usable state. Use for
early initialization only.
   */


  /* @namespace Map
   * @section Layer events
   *
   * @event layeradd:
LayerEvent
   * Fired when a new layer is
added to the map.
   *
   * @event layerremove:
LayerEvent
   * Fired when some layer is
removed from the map
   *
   * @section Methods for
Layers and Controls
   */
  Map.include({
        // @method
addLayer(layer: Layer): this
        // Adds the given
layer to the map
        addLayer: function
(layer) {
                if
(!layer._layerAdd) {
                        throw
new Error('The provided object
```

```
is not a Layer.');
                }

                var id =
stamp(layer);
                if
(this._layers[id]) { return
this; }

this._layers[id] = layer;


layer._mapToAdd = this;

                if
(layer.beforeAdd) {

layer.beforeAdd(this);
                }


this.whenReady(layer._layerAdd
, layer);

                return this;
        },

        // @method
removeLayer(layer: Layer):
this
        // Removes the given
layer from the map.
        removeLayer: function
(layer) {
                var id =
stamp(layer);

                if
(!this._layers[id]) { return
this; }

                if
(this._loaded) {

layer.onRemove(this);
                }
```

```
                delete
this._layers[id];

                if
(this._loaded) {

this.fire('layerremove',
{layer: layer});

layer.fire('remove');
                }

                layer._map =
layer._mapToAdd = null;

                return this;
        },

        // @method
hasLayer(layer: Layer):
Boolean
        // Returns `true` if
the given layer is currently
added to the map
        hasLayer: function
(layer) {
                return
stamp(layer) in this._layers;
        },

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

```
                for (var i in
this._layers) {

method.call(context,
this._layers[i]);
                }
                return this;
        },

        _addLayers: function
(layers) {
                layers =
layers ? (isArray(layers) ?
layers : [layers]) : [];

                for (var i =
0, len = layers.length; i <
len; i++) {

this.addLayer(layers[i]);
                }
        },

        _addZoomLimit:
function (layer) {
                if
(!isNaN(layer.options.maxZoom)
||
!isNaN(layer.options.minZoom))
{

this._zoomBoundLayers[stamp(la
yer)] = layer;

this._updateZoomLevels();
                }
        },

        _removeZoomLimit:
function (layer) {
                var id =
stamp(layer);

                if
(this._zoomBoundLayers[id]) {
                        delete
this._zoomBoundLayers[id];
```

```
this._updateZoomLevels();
                }
        },

        _updateZoomLevels:
function () {
                var minZoom =
Infinity,
                    maxZoom =
-Infinity,

oldZoomSpan =
this._getZoomSpan();

                for (var i in
this._zoomBoundLayers) {
                        var
options =
this._zoomBoundLayers[i].optio
ns;


minZoom = options.minZoom ===
undefined ? minZoom :
Math.min(minZoom,
options.minZoom);

maxZoom = options.maxZoom ===
undefined ? maxZoom :
Math.max(maxZoom,
options.maxZoom);
                }


this._layersMaxZoom = maxZoom
=== -Infinity ? undefined :
maxZoom;

this._layersMinZoom = minZoom
=== Infinity ? undefined :
minZoom;

                // @section
Map state change events
                // @event
zoomlevelschange: Event
```

```
                // Fired when
the number of zoomlevels on
the map is changed due
                // to adding
or removing a layer.
                if
(oldZoomSpan !==
this._getZoomSpan()) {

this.fire('zoomlevelschange');
                }

                if
(this.options.maxZoom ===
undefined &&
this._layersMaxZoom &&
this.getZoom() >
this._layersMaxZoom) {

this.setZoom(this._layersMaxZo
om);
                }
                if
(this.options.minZoom ===
undefined &&
this._layersMinZoom &&
this.getZoom() <
this._layersMinZoom) {

this.setZoom(this._layersMinZo
om);
                }
        }
  });

  /*
   * @class LayerGroup
   * @aka L.LayerGroup
   * @inherits Interactive
layer
   *
   * Used to group several
layers and handle them as one.
If you add it to the map,
   * any layers added or
removed from the group will be
added/removed on the map as
```

```
   * well. Extends `Layer`.
   *
   * @example
   *
   * ```js
   * L.layerGroup([marker1,
marker2])
   *    .addLayer(polyline)
   *    .addTo(map);
   * ```
   */

  var LayerGroup =
Layer.extend({

        initialize: function
(layers, options) {

setOptions(this, options);

                this._layers =
{};

                var i, len;

                if (layers) {
                        for (i
= 0, len = layers.length; i <
len; i++) {

this.addLayer(layers[i]);
                        }
                }
        },

        // @method
addLayer(layer: Layer): this
        // Adds the given
layer to the group.
        addLayer: function
(layer) {
                var id =
this.getLayerId(layer);


this._layers[id] = layer;
```

```
                if (this._map)
{

this._map.addLayer(layer);
                }

                return this;
        },

        // @method
removeLayer(layer: Layer):
this
        // Removes the given
layer from the group.
        // @alternative
        // @method
removeLayer(id: Number): this
        // Removes the layer
with the given internal ID
from the group.
        removeLayer: function
(layer) {
                var id = layer
in this._layers ? layer :
this.getLayerId(layer);

                if (this._map
&& this._layers[id]) {

this._map.removeLayer(this._la
yers[id]);
                }

                delete
this._layers[id];

                return this;
        },

        // @method
hasLayer(layer: Layer):
Boolean
        // Returns `true` if
the given layer is currently
added to the group.
        // @alternative
        // @method
```

```
hasLayer(id: Number): Boolean
        // Returns `true` if
the given internal ID is
currently added to the group.
        hasLayer: function
(layer) {
                var layerId =
typeof layer === 'number' ?
layer :
this.getLayerId(layer);
                return layerId
in this._layers;
        },

        // @method
clearLayers(): this
        // Removes all the
layers from the group.
        clearLayers: function
() {
                return
this.eachLayer(this.removeLaye
r, this);
        },

        // @method
invoke(methodName: String, …):
this
        // Calls `methodName`
on every layer contained in
this group, passing any
        // additional
parameters. Has no effect if
the layers contained do not
        // implement
`methodName`.
        invoke: function
(methodName) {
                var args =
Array.prototype.slice.call(arg
uments, 1),
                    i, layer;

                for (i in
this._layers) {
                        layer
= this._layers[i];
```

```
                        if
(layer[methodName]) {

layer[methodName].apply(layer,
args);
                        }
                }

                return this;
        },

        onAdd: function (map)
{

this.eachLayer(map.addLayer,
map);
        },

        onRemove: function
(map) {

this.eachLayer(map.removeLayer
, map);
        },

        // @method
eachLayer(fn: Function,
context?: Object): this
        // Iterates over the
layers of the group,
optionally specifying context
of the iterator function.
        // ```js
        //
group.eachLayer(function
(layer) {
        //
layer.bindPopup('Hello');
        // });
        // ```
        eachLayer: function
(method, context) {
                for (var i in
this._layers) {

method.call(context,
```

```
this._layers[i]);
                }
                return this;
        },

        // @method
getLayer(id: Number): Layer
        // Returns the layer
with the given internal ID.
        getLayer: function
(id) {
                return
this._layers[id];
        },

        // @method
getLayers(): Layer[]
        // Returns an array of
all the layers added to the
group.
        getLayers: function ()
{
                var layers =
[];

this.eachLayer(layers.push,
layers);
                return layers;
        },

        // @method
setZIndex(zIndex: Number):
this
        // Calls `setZIndex`
on every layer contained in
this group, passing the z-
index.
        setZIndex: function
(zIndex) {
                return
this.invoke('setZIndex',
zIndex);
        },

        // @method
getLayerId(layer: Layer):
Number
```

```
        // Returns the
internal ID for a layer
        getLayerId: function
(layer) {
                return
stamp(layer);
        }
  });


  // @factory
L.layerGroup(layers?: Layer[],
options?: Object)
  // Create a layer group,
optionally given an initial
set of layers and an `options`
object.
  var layerGroup = function
(layers, options) {
        return new
LayerGroup(layers, options);
  };

  /*
   * @class FeatureGroup
   * @aka L.FeatureGroup
   * @inherits LayerGroup
   *
   * Extended `LayerGroup`
that makes it easier to do the
same thing to all its member
layers:
   *  * [`bindPopup`](#layer-
bindpopup) binds a popup to
all of the layers at once
(likewise with [`bindTooltip`]
(#layer-bindtooltip))
   *  * Events are propagated
to the `FeatureGroup`, so if
the group has an event
   * handler, it will handle
events from any of the layers.
This includes mouse events
   * and custom events.
   *  * Has `layeradd` and
`layerremove` events
   *
```

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

  var FeatureGroup =
LayerGroup.extend({

        addLayer: function
(layer) {
                if
(this.hasLayer(layer)) {
                        return
this;
                }


layer.addEventParent(this);


LayerGroup.prototype.addLayer.
call(this, layer);

                // @event
layeradd: LayerEvent
                // Fired when
a layer is added to this
`FeatureGroup`
                return
this.fire('layeradd', {layer:
layer});
        },

        removeLayer: function
(layer) {
                if
(!this.hasLayer(layer)) {
                        return
```

```
this;
                }
                if (layer in
this._layers) {
                        layer
= this._layers[layer];
                }


layer.removeEventParent(this);


LayerGroup.prototype.removeLay
er.call(this, layer);

                // @event
layerremove: LayerEvent
                // Fired when
a layer is removed from this
`FeatureGroup`
                return
this.fire('layerremove',
{layer: layer});
        },

        // @method
setStyle(style: Path options):
this
        // Sets the given path
options to each layer of the
group that has a `setStyle`
method.
        setStyle: function
(style) {
                return
this.invoke('setStyle',
style);
        },

        // @method
bringToFront(): this
        // Brings the layer
group to the top of all other
layers
        bringToFront: function
() {
                return
```

```
this.invoke('bringToFront');
        },

        // @method
bringToBack(): this
        // Brings the layer
group to the back of all other
layers
        bringToBack: function
() {
                return
this.invoke('bringToBack');
        },

        // @method
getBounds(): LatLngBounds
        // Returns the
LatLngBounds of the Feature
Group (created from bounds and
coordinates of its children).
        getBounds: function ()
{
                var bounds =
new LatLngBounds();

                for (var id in
this._layers) {
                        var
layer = this._layers[id];

bounds.extend(layer.getBounds
? layer.getBounds() :
layer.getLatLng());
                }
                return bounds;
        }
  });

  // @factory
L.featureGroup(layers?:
Layer[], options?: Object)
  // Create a feature group,
optionally given an initial
set of layers and an `options`
object.
  var featureGroup = function
(layers, options) {
```

```
        return new
FeatureGroup(layers, options);
  };

  /*
   * @class Icon
   * @aka L.Icon
   *
   * Represents an icon to
provide when creating a
marker.
   *
   * @example
   *
   * ```js
   * 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.
   *
   */

  var Icon = Class.extend({

        /* @section
```

```
	 * @aka Icon options
	 *
	 * @option iconUrl:
String = null
	 * **(required)** The
URL to the icon image
(absolute or relative to your
script path).
	 *
	 * @option
iconRetinaUrl: String = null
	 * The URL to a retina
sized version of the icon
image (absolute or relative to
your
	 * script path). Used
for Retina screen devices.
	 *
	 * @option iconSize:
Point = null
	 * Size of the icon
image in pixels.
	 *
	 * @option iconAnchor:
Point = null
	 * The coordinates of
the "tip" of the icon
(relative to its top left
corner). The icon
	 * will be aligned so
that this point is at the
marker's geographical
location. Centered
	 * by default if size
is specified, also can be set
in CSS with negative margins.
	 *
	 * @option
popupAnchor: Point = [0, 0]
	 * The coordinates of
the point from which popups
will "open", relative to the
icon anchor.
	 *
	 * @option
tooltipAnchor: Point = [0, 0]
	 * The coordinates of
```

```
the point from which tooltips
will "open", relative to the
icon anchor.
         *
         * @option shadowUrl:
String = null
         * The URL to the icon
shadow image. If not
specified, no shadow image
will be created.
         *
         * @option
shadowRetinaUrl: String = null
         *
         * @option shadowSize:
Point = null
         * Size of the shadow
image in pixels.
         *
         * @option
shadowAnchor: Point = null
         * The coordinates of
the "tip" of the shadow
(relative to its top left
corner) (the same
         * as iconAnchor if
not specified).
         *
         * @option className:
String = ''
         * A custom class name
to assign to both icon and
shadow images. Empty by
default.
         */

        options: {
                popupAnchor:
[0, 0],
                tooltipAnchor:
[0, 0],

                // @option
crossOrigin: Boolean|String =
false
                // Whether the
crossOrigin attribute will be
```

```
added to the tiles.
                // If a String
is provided, all tiles will
have their crossOrigin
attribute set to the String
provided. This is needed if
you want to access tile pixel
data.
                // Refer to
[CORS Settings]
(https://developer.mozilla.org
/en-
US/docs/Web/HTML/CORS_settings
_attributes) for valid String
values.
                crossOrigin:
false
        },

        initialize: function
(options) {

setOptions(this, options);
        },

        // @method
createIcon(oldIcon?:
HTMLElement): HTMLElement
        // Called internally
when the icon has to be shown,
returns a `<img>` HTML element
        // styled according to
the options.
        createIcon: function
(oldIcon) {
                return
this._createIcon('icon',
oldIcon);
        },

        // @method
createShadow(oldIcon?:
HTMLElement): HTMLElement
        // As `createIcon`,
but for the shadow beneath it.
        createShadow: function
(oldIcon) {
```

```
                return
this._createIcon('shadow',
oldIcon);
        },

        _createIcon: function
(name, oldIcon) {
                var src =
this._getIconUrl(name);

                if (!src) {
                        if
(name === 'icon') {

throw new Error('iconUrl not
set in Icon options (see the
docs).');
                        }
                        return
null;
                }

                var img =
this._createImg(src, oldIcon
&& oldIcon.tagName === 'IMG' ?
oldIcon : null);

this._setIconStyles(img,
name);

                if
(this.options.crossOrigin ||
this.options.crossOrigin ===
'') {

img.crossOrigin =
this.options.crossOrigin ===
true ? '' :
this.options.crossOrigin;
                }

                return img;
        },

        _setIconStyles:
function (img, name) {
                var options =
```

```
this.options;
                var sizeOption
= options[name + 'Size'];

                if (typeof
sizeOption === 'number') {

sizeOption = [sizeOption,
sizeOption];
                }

                var size =
toPoint(sizeOption),
                    anchor =
toPoint(name === 'shadow' &&
options.shadowAnchor ||
options.iconAnchor ||

size && size.divideBy(2,
true));

                img.className
= 'leaflet-marker-' + name + '
' + (options.className || '');

                if (anchor) {

img.style.marginLeft = (-
anchor.x) + 'px';

img.style.marginTop  = (-
anchor.y) + 'px';
                }

                if (size) {

img.style.width  = size.x +
'px';

img.style.height = size.y +
'px';
                }
        },

        _createImg: function
(src, el) {
                el = el ||
```

```
document.createElement('img');
                el.src = src;
                return el;
        },

        _getIconUrl: function
(name) {
                return
Browser.retina &&
this.options[name +
'RetinaUrl'] ||
this.options[name + 'Url'];
        }
  });


  // @factory L.icon(options:
Icon options)
  // Creates an icon instance
with the given options.
  function icon(options) {
        return new
Icon(options);
  }

  /*
   * @miniclass Icon.Default
(Icon)
   * @aka L.Icon.Default
   * @section
   *
   * A trivial subclass of
`Icon`, represents the icon to
use in `Marker`s when
   * no icon is specified.
Points to the blue marker
image distributed with Leaflet
   * releases.
   *
   * In order to customize the
default icon, just change the
properties of
`L.Icon.Default.prototype.opti
ons`
   * (which is a set of `Icon
options`).
   *
```

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

  var IconDefault =
Icon.extend({

        options: {
                iconUrl:
'marker-icon.png',
                iconRetinaUrl:
'marker-icon-2x.png',
                shadowUrl:
'marker-shadow.png',
                iconSize:
[25, 41],
                iconAnchor:
[12, 41],
                popupAnchor:
[1, -34],
                tooltipAnchor:
[16, -28],
                shadowSize:
[41, 41]
        },

        _getIconUrl: function
(name) {
                if (typeof
IconDefault.imagePath !==
'string') {     // Deprecated,
backwards-compatibility only

IconDefault.imagePath =
this._detectIconPath();
                }

                // @option
imagePath: String
                //
`Icon.Default` will try to
auto-detect the location of
```

```
the
                // blue icon
images. If you are placing
these images in a non-standard
                // way, set
this option to point to the
right path.
                return
(this.options.imagePath ||
IconDefault.imagePath) +
Icon.prototype._getIconUrl.cal
l(this, name);
        },

        _stripUrl: function
(path) {        // separate
function to use in tests
                var strip =
function (str, re, idx) {
                        var
match = re.exec(str);
                        return
match && match[idx];
                };
                path =
strip(path, /^url\((['"])?
(.+)\1\)$/, 2);
                return path &&
strip(path, /^(.*)marker-
icon\.png$/, 1);
        },

        _detectIconPath:
function () {
                var el =
create$1('div',  'leaflet-
default-icon-path',
document.body);
                var path =
getStyle(el, 'background-
image') ||

getStyle(el,
'backgroundImage');     // IE8


document.body.removeChild(el);
```

```
                path =
this._stripUrl(path);
                if (path) {
return path; }
                var link =
document.querySelector('link[h
ref$="leaflet.css"]');
                if (!link) {
return ''; }
                return
link.href.substring(0,
link.href.length -
'leaflet.css'.length - 1);
        }
  });

  /*
   * L.Handler.MarkerDrag is
used internally by L.Marker to
make the markers draggable.
   */


  /* @namespace Marker
   * @section Interaction
handlers
   *
   * Interaction handlers are
properties of a marker
instance that allow you to
control interaction behavior
in runtime, enabling or
disabling certain features
such as dragging (see
`Handler` methods). Example:
   *
   * ```js
   *
marker.dragging.disable();
   * ```
   *
   * @property dragging:
Handler
   * Marker dragging handler
(by both mouse and touch).
Only valid when the marker is
on the map (Otherwise set
```

```
[`marker.options.draggable`]
(#marker-draggable)).
   */

  var MarkerDrag =
Handler.extend({
        initialize: function
(marker) {
                this._marker =
marker;
        },

        addHooks: function ()
{
                var icon =
this._marker._icon;

                if
(!this._draggable) {

this._draggable = new
Draggable(icon, icon, true);
                }


this._draggable.on({

dragstart: this._onDragStart,

predrag: this._onPreDrag,
                        drag:
this._onDrag,

dragend: this._onDragEnd
                },
this).enable();

                addClass(icon,
'leaflet-marker-draggable');
        },

        removeHooks: function
() {

this._draggable.off({

dragstart: this._onDragStart,
```

```
predrag: this._onPreDrag,
                        drag:
this._onDrag,

dragend: this._onDragEnd
                },
this).disable();

                if
(this._marker._icon) {

removeClass(this._marker._icon
, 'leaflet-marker-draggable');
                }
        },

        moved: function () {
                return
this._draggable &&
this._draggable._moved;
        },

        _adjustPan: function
(e) {
                var marker =
this._marker,
                    map =
marker._map,
                    speed =
this._marker.options.autoPanSp
eed,
                    padding =
this._marker.options.autoPanPa
dding,
                    iconPos =
getPosition(marker._icon),
                    bounds =
map.getPixelBounds(),
                    origin =
map.getPixelOrigin();

                var panBounds
= toBounds(

bounds.min._subtract(origin).a
dd(padding),
```

```
bounds.max._subtract(origin).s
ubtract(padding)
                );

                if
(!panBounds.contains(iconPos))
{
                        //
Compute incremental movement
                        var
movement = toPoint(

(Math.max(panBounds.max.x,
iconPos.x) - panBounds.max.x)
/ (bounds.max.x -
panBounds.max.x) -

(Math.min(panBounds.min.x,
iconPos.x) - panBounds.min.x)
/ (bounds.min.x -
panBounds.min.x),


(Math.max(panBounds.max.y,
iconPos.y) - panBounds.max.y)
/ (bounds.max.y -
panBounds.max.y) -

(Math.min(panBounds.min.y,
iconPos.y) - panBounds.min.y)
/ (bounds.min.y -
panBounds.min.y)

).multiplyBy(speed);


map.panBy(movement, {animate:
false});


this._draggable._newPos._add(m
ovement);

this._draggable._startPos._add
(movement);
```

```
setPosition(marker._icon,
this._draggable._newPos);

this._onDrag(e);


this._panRequest =
requestAnimFrame(this._adjustP
an.bind(this, e));
                }
        },

        _onDragStart: function
() {
                // @section
Dragging events
                // @event
dragstart: Event
                // Fired when
the user starts dragging the
marker.

                // @event
movestart: Event
                // Fired when
the marker starts moving
(because of dragging).


this._oldLatLng =
this._marker.getLatLng();

                // When using
ES6 imports it could not be
set when `Popup` was not
imported as well

this._marker.closePopup &&
this._marker.closePopup();

                this._marker

.fire('movestart')

.fire('dragstart');
        },
```

```
        _onPreDrag: function
(e) {
                if
(this._marker.options.autoPan)
{

cancelAnimFrame(this._panReque
st);

this._panRequest =
requestAnimFrame(this._adjustP
an.bind(this, e));
                }
        },

        _onDrag: function (e)
{
                var marker =
this._marker,
                    shadow =
marker._shadow,
                    iconPos =
getPosition(marker._icon),
                    latlng =
marker._map.layerPointToLatLng
(iconPos);

                // update
shadow position
                if (shadow) {

setPosition(shadow, iconPos);
                }

                marker._latlng
= latlng;
                e.latlng =
latlng;
                e.oldLatLng =
this._oldLatLng;

                // @event
drag: Event
                // Fired
repeatedly while the user
drags the marker.
```

```
                marker

.fire('move', e)

.fire('drag', e);
        },

        _onDragEnd: function
(e) {
                // @event
dragend: DragEndEvent
                // Fired when
the user stops dragging the
marker.


cancelAnimFrame(this._panReque
st);

                // @event
moveend: Event
                // Fired when
the marker stops moving
(because of dragging).
                delete
this._oldLatLng;
                this._marker

.fire('moveend')

.fire('dragend', e);
        }
  });

  /*
   * @class Marker
   * @inherits Interactive
layer
   * @aka L.Marker
   * L.Marker is used to
display clickable/draggable
icons on the map. Extends
`Layer`.
   *
   * @example
   *
   * ```js
```

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

  var Marker = Layer.extend({

        // @section
        // @aka Marker options
        options: {
                // @option
icon: Icon = *
                // Icon
instance to use for rendering
the marker.
                // See [Icon
documentation](#L.Icon) for
details on how to customize
the marker icon.
                // If not
specified, a common instance
of `L.Icon.Default` is used.
                icon: new
IconDefault(),

                // Option
inherited from "Interactive
layer" abstract class
                interactive:
true,

                // @option
keyboard: Boolean = true
                // Whether the
marker can be tabbed to with a
keyboard and clicked by
pressing enter.
                keyboard:
true,

                // @option
title: String = ''
                // Text for
the browser tooltip that
appear on marker hover (no
tooltip by default).
                // [Useful for
```

```
accessibility]
(https://leafletjs.com/example
s/accessibility/#markers-must-
be-labelled).
                title: '',

                // @option
alt: String = 'Marker'
                // Text for
the `alt` attribute of the
icon image.
                // [Useful for
accessibility]
(https://leafletjs.com/example
s/accessibility/#markers-must-
be-labelled).
                alt: 'Marker',

                // @option
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).
                zIndexOffset:
0,

                // @option
opacity: Number = 1.0
                // The opacity
of the marker.
                opacity: 1,

                // @option
riseOnHover: Boolean = false
                // If `true`,
the marker will get on top of
others when you hover the
mouse over it.
                riseOnHover:
false,
```

```
                // @option
riseOffset: Number = 250
                // The z-index
offset used for the
`riseOnHover` feature.
                riseOffset:
250,

                // @option
pane: String = 'markerPane'
                // `Map pane`
where the markers icon will be
added.
                pane:
'markerPane',

                // @option
shadowPane: String =
'shadowPane'
                // `Map pane`
where the markers shadow will
be added.
                shadowPane:
'shadowPane',

                // @option
bubblingMouseEvents: Boolean =
false
                // When
`true`, a mouse event on this
marker will trigger the same
event on the map
                // (unless
[`L.DomEvent.stopPropagation`]
(#domevent-stoppropagation) is
used).

bubblingMouseEvents: false,

                // @option
autoPanOnFocus: Boolean = true
                // When
`true`, the map will pan
whenever the marker is focused
(via
                // e.g.
pressing `tab` on the
```

```
keyboard) to ensure the marker
is
                // visible
within the map's bounds

autoPanOnFocus: true,

                // @section
Draggable marker options
                // @option
draggable: Boolean = false
                // Whether the
marker is draggable with
mouse/touch or not.
                draggable:
false,

                // @option
autoPan: Boolean = false
                // Whether to
pan the map when dragging this
marker near its edge or not.
                autoPan:
false,

                // @option
autoPanPadding: Point =
Point(50, 50)
                // Distance
(in pixels to the left/right
and to the top/bottom) of the
                // map edge to
start panning the map.

autoPanPadding: [50, 50],

                // @option
autoPanSpeed: Number = 10
                // Number of
pixels the map should pan by.
                autoPanSpeed:
10
        },

        /* @section
         *
         * In addition to
```

```
[shared layer methods](#Layer)
like `addTo()` and `remove()`
and [popup methods](#Popup)
like bindPopup() you can also
use the following methods:
         */

        initialize: function
(latlng, options) {

setOptions(this, options);
                this._latlng =
toLatLng(latlng);
        },

        onAdd: function (map)
{

this._zoomAnimated =
this._zoomAnimated &&
map.options.markerZoomAnimatio
n;

                if
(this._zoomAnimated) {

map.on('zoomanim',
this._animateZoom, this);
                }


this._initIcon();
                this.update();
        },

        onRemove: function
(map) {
                if
(this.dragging &&
this.dragging.enabled()) {

this.options.draggable = true;

this.dragging.removeHooks();
                }
                delete
this.dragging;
```

```
                if
(this._zoomAnimated) {

map.off('zoomanim',
this._animateZoom, this);
                }


this._removeIcon();

this._removeShadow();
        },

        getEvents: function ()
{
                return {
                        zoom:
this.update,

viewreset: this.update
                };
        },

        // @method getLatLng:
LatLng
        // Returns the current
geographical position of the
marker.
        getLatLng: function ()
{
                return
this._latlng;
        },

        // @method
setLatLng(latlng: LatLng):
this
        // Changes the marker
position to the given point.
        setLatLng: function
(latlng) {
                var oldLatLng
= this._latlng;
                this._latlng =
toLatLng(latlng);
                this.update();
```

```
                // @event
move: Event
                // Fired when
the marker is moved via
[`setLatLng`](#marker-
setlatlng) or by [dragging]
(#marker-dragging). Old and
new coordinates are included
in event arguments as
`oldLatLng`, `latlng`.
                return
this.fire('move', {oldLatLng:
oldLatLng, latlng:
this._latlng});
        },

        // @method
setZIndexOffset(offset:
Number): this
        // Changes the [zIndex
offset](#marker-zindexoffset)
of the marker.
        setZIndexOffset:
function (offset) {

this.options.zIndexOffset =
offset;
                return
this.update();
        },

        // @method getIcon:
Icon
        // Returns the current
icon used by the marker
        getIcon: function () {
                return
this.options.icon;
        },

        // @method
setIcon(icon: Icon): this
        // Changes the marker
icon.
        setIcon: function
(icon) {
```

```
this.options.icon = icon;

                if (this._map)
{

this._initIcon();

this.update();
                }

                if
(this._popup) {

this.bindPopup(this._popup,
this._popup.options);
                }

                return this;
        },

        getElement: function
() {
                return
this._icon;
        },

        update: function () {

                if (this._icon
&& this._map) {
                        var
pos =
this._map.latLngToLayerPoint(t
his._latlng).round();

this._setPos(pos);
                }

                return this;
        },

        _initIcon: function ()
{
                var options =
this.options,
```

```
                    classToAdd
= 'leaflet-zoom-' +
(this._zoomAnimated ?
'animated' : 'hide');

                var icon =
options.icon.createIcon(this._
icon),
                    addIcon =
false;

                // if we're
not reusing the icon, remove
the old one and init new one
                if (icon !==
this._icon) {
                        if
(this._icon) {

this._removeIcon();
                        }

addIcon = true;

                        if
(options.title) {

icon.title = options.title;
                        }

                        if
(icon.tagName === 'IMG') {

icon.alt = options.alt || '';
                        }
                }

                addClass(icon,
classToAdd);

                if
(options.keyboard) {

icon.tabIndex = '0';

icon.setAttribute('role',
'button');
```

```
                }

                this._icon =
icon;

                if
(options.riseOnHover) {

this.on({

mouseover: this._bringToFront,

mouseout: this._resetZIndex
                        });
                }

                if
(this.options.autoPanOnFocus)
{

on(icon, 'focus',
this._panOnFocus, this);
                }

                var newShadow
=
options.icon.createShadow(this
._shadow),
                    addShadow
= false;

                if (newShadow
!== this._shadow) {

this._removeShadow();

addShadow = true;
                }

                if (newShadow)
{

addClass(newShadow,
classToAdd);

newShadow.alt = '';
                }
```

```
                this._shadow =
newShadow;


                if
(options.opacity < 1) {

this._updateOpacity();
                }


                if (addIcon) {

this.getPane().appendChild(thi
s._icon);
                }

this._initInteraction();
                if (newShadow
&& addShadow) {

this.getPane(options.shadowPan
e).appendChild(this._shadow);
                }
        },

        _removeIcon: function
() {
                if
(this.options.riseOnHover) {

this.off({

mouseover: this._bringToFront,

mouseout: this._resetZIndex
                        });
                }

                if
(this.options.autoPanOnFocus)
{

off(this._icon, 'focus',
this._panOnFocus, this);
                }
```

```
remove(this._icon);

this.removeInteractiveTarget(t
his._icon);

                this._icon =
null;
        },

        _removeShadow:
function () {
                if
(this._shadow) {

remove(this._shadow);
                }
                this._shadow =
null;
        },

        _setPos: function
(pos) {

                if
(this._icon) {

setPosition(this._icon, pos);
                }

                if
(this._shadow) {

setPosition(this._shadow,
pos);
                }

                this._zIndex =
pos.y +
this.options.zIndexOffset;


this._resetZIndex();
        },

        _updateZIndex:
function (offset) {
```

```
                if
(this._icon) {

this._icon.style.zIndex =
this._zIndex + offset;
                }
        },

        _animateZoom: function
(opt) {
                var pos =
this._map._latLngToNewLayerPoi
nt(this._latlng, opt.zoom,
opt.center).round();


this._setPos(pos);
        },

        _initInteraction:
function () {

                if
(!this.options.interactive) {
return; }


addClass(this._icon, 'leaflet-
interactive');


this.addInteractiveTarget(this
._icon);

                if
(MarkerDrag) {
                        var
draggable =
this.options.draggable;
                        if
(this.dragging) {

draggable =
this.dragging.enabled();

this.dragging.disable();
                        }
```

```
this.dragging = new
MarkerDrag(this);

                        if
(draggable) {

this.dragging.enable();
                        }
                }
        },

        // @method
setOpacity(opacity: Number):
this
        // Changes the opacity
of the marker.
        setOpacity: function
(opacity) {

this.options.opacity =
opacity;
                if (this._map)
{

this._updateOpacity();
                }

                return this;
        },

        _updateOpacity:
function () {
                var opacity =
this.options.opacity;

                if
(this._icon) {

setOpacity(this._icon,
opacity);
                }

                if
(this._shadow) {
```

```
setOpacity(this._shadow,
opacity);
                }
        },

        _bringToFront:
function () {

this._updateZIndex(this.option
s.riseOffset);
        },

        _resetZIndex: function
() {

this._updateZIndex(0);
        },

        _panOnFocus: function
() {
                var map =
this._map;
                if (!map) {
return; }

                var iconOpts =
this.options.icon.options;
                var size =
iconOpts.iconSize ?
toPoint(iconOpts.iconSize) :
toPoint(0, 0);
                var anchor =
iconOpts.iconAnchor ?
toPoint(iconOpts.iconAnchor) :
toPoint(0, 0);


map.panInside(this._latlng, {

paddingTopLeft: anchor,

paddingBottomRight:
size.subtract(anchor)
                });
        },

        _getPopupAnchor:
```

```
function () {
                return
this.options.icon.options.popu
pAnchor;
        },

        _getTooltipAnchor:
function () {
                return
this.options.icon.options.tool
tipAnchor;
        }
  });


  // factory L.marker(latlng:
LatLng, options? : Marker
options)

  // @factory L.marker(latlng:
LatLng, options? : Marker
options)
  // Instantiates a Marker
object given a geographical
point and optionally an
options object.
  function marker(latlng,
options) {
        return new
Marker(latlng, options);
  }

  /*
   * @class Path
   * @aka L.Path
   * @inherits Interactive
layer
   *
   * An abstract class that
contains options and constants
shared between vector
   * overlays (Polygon,
Polyline, Circle). Do not use
it directly. Extends `Layer`.
   */

  var Path = Layer.extend({
```

```
        // @section
        // @aka Path options
        options: {
                // @option
stroke: Boolean = true
                // Whether to
draw stroke along the path.
Set it to `false` to disable
borders on polygons or
circles.
                stroke: true,

                // @option
color: String = '#3388ff'
                // Stroke
color
                color:
'#3388ff',

                // @option
weight: Number = 3
                // Stroke
width in pixels
                weight: 3,

                // @option
opacity: Number = 1.0
                // Stroke
opacity
                opacity: 1,

                // @option
lineCap: String= 'round'
                // A string
that defines [shape to be used
at the end]
(https://developer.mozilla.org
/docs/Web/SVG/Attribute/stroke
-linecap) of the stroke.
                lineCap:
'round',

                // @option
lineJoin: String = 'round'
                // A string
that defines [shape to be used
```

```
at the corners]
(https://developer.mozilla.org
/docs/Web/SVG/Attribute/stroke
-linejoin) of the stroke.
                lineJoin:
'round',

                // @option
dashArray: String = null
                // A string
that defines the stroke [dash
pattern]
(https://developer.mozilla.org
/docs/Web/SVG/Attribute/stroke
-dasharray). Doesn't work on
`Canvas`-powered layers in
[some old browsers]
(https://developer.mozilla.org
/docs/Web/API/CanvasRendering
ontext2D/setLineDash#Browser_c
ompatibility).
                dashArray:
null,

                // @option
dashOffset: String = null
                // A string
that defines the [distance
into the dash pattern to start
the dash]
(https://developer.mozilla.org
/docs/Web/SVG/Attribute/stroke
-dashoffset). Doesn't work on
`Canvas`-powered layers in
[some old browsers]
(https://developer.mozilla.org
/docs/Web/API/CanvasRendering
ontext2D/setLineDash#Browser_c
ompatibility).
                dashOffset:
null,

                // @option
fill: Boolean = depends
                // Whether to
fill the path with color. Set
it to `false` to disable
```

```
filling on polygons or
circles.
                fill: false,

                // @option
fillColor: String = *
                // Fill color.
Defaults to the value of the
[`color`](#path-color) option
                fillColor:
null,

                // @option
fillOpacity: Number = 0.2
                // Fill
opacity.
                fillOpacity:
0.2,

                // @option
fillRule: String = 'evenodd'
                // A string
that defines [how the inside
of a shape]
(https://developer.mozilla.org
/docs/Web/SVG/Attribute/fill-
rule) is determined.
                fillRule:
'evenodd',

                // className:
'',

                // Option
inherited from "Interactive
layer" abstract class
                interactive:
true,

                // @option
bubblingMouseEvents: Boolean =
true
                // When
`true`, a mouse event on this
path will trigger the same
event on the map
                // (unless
```

```
[`L.DomEvent.stopPropagation`]
(#domevent-stoppropagation) is
used).

bubblingMouseEvents: true
        },

        beforeAdd: function
(map) {
                // Renderer is
set here because we need to
call renderer.getEvents
                // before
this.getEvents.
                this._renderer
= map.getRenderer(this);
        },

        onAdd: function () {

this._renderer._initPath(this)
;
                this._reset();

this._renderer._addPath(this);
        },

        onRemove: function ()
{

this._renderer._removePath(thi
s);
        },

        // @method redraw():
this
        // Redraws the layer.
Sometimes useful after you
changed the coordinates that
the path uses.
        redraw: function () {
                if (this._map)
{

this._renderer._updatePath(thi
s);
                }
```

```
                return this;
        },

        // @method
setStyle(style: Path options):
this
        // Changes the
appearance of a Path based on
the options in the `Path
options` object.
        setStyle: function
(style) {

setOptions(this, style);
                if
(this._renderer) {

this._renderer._updateStyle(th
is);
                        if
(this.options.stroke && style
&&
Object.prototype.hasOwnPropert
y.call(style, 'weight')) {

this._updateBounds();
                        }
                }
                return this;
        },

        // @method
bringToFront(): this
        // Brings the layer to
the top of all path layers.
        bringToFront: function
() {
                if
(this._renderer) {

this._renderer._bringToFront(t
his);
                }
                return this;
        },

        // @method
```

```
bringToBack(): this
        // Brings the layer to
the bottom of all path layers.
        bringToBack: function
() {
                if
(this._renderer) {

this._renderer._bringToBack(th
is);
                }
                return this;
        },

        getElement: function
() {
                return
this._path;
        },

        _reset: function () {
                // defined in
child classes

this._project();

this._update();
        },

        _clickTolerance:
function () {
                // used when
doing hit detection for Canvas
layers
                return
(this.options.stroke ?
this.options.weight / 2 : 0) +

(this._renderer.options.tolera
nce || 0);
        }
  });

  /*
   * @class CircleMarker
   * @aka L.CircleMarker
   * @inherits Path
```

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

  var CircleMarker =
Path.extend({

        // @section
        // @aka CircleMarker
options
        options: {
                fill: true,

                // @option
radius: Number = 10
                // Radius of
the circle marker, in pixels
                radius: 10
        },

        initialize: function
(latlng, options) {

setOptions(this, options);
                this._latlng =
toLatLng(latlng);
                this._radius =
this.options.radius;
        },

        // @method
setLatLng(latLng: LatLng):
this
        // Sets the position
of a circle marker to a new
location.
        setLatLng: function
(latlng) {
                var oldLatLng
= this._latlng;
                this._latlng =
toLatLng(latlng);
                this.redraw();

                // @event
```

```
move: Event
                // Fired when
the marker is moved via
[`setLatLng`](#circlemarker-
setlatlng). Old and new
coordinates are included in
event arguments as
`oldLatLng`, `latlng`.
                return
this.fire('move', {oldLatLng:
oldLatLng, latlng:
this._latlng});
        },

        // @method
getLatLng(): LatLng
        // Returns the current
geographical position of the
circle marker
        getLatLng: function ()
{
                return
this._latlng;
        },

        // @method
setRadius(radius: Number):
this
        // Sets the radius of
a circle marker. Units are in
pixels.
        setRadius: function
(radius) {

this.options.radius =
this._radius = radius;
                return
this.redraw();
        },

        // @method
getRadius(): Number
        // Returns the current
radius of the circle
        getRadius: function ()
{
                return
```

```
this._radius;
        },

        setStyle : function
(options) {
                var radius =
options && options.radius ||
this._radius;

Path.prototype.setStyle.call(t
his, options);

this.setRadius(radius);
                return this;
        },

        _project: function ()
{
                this._point =
this._map.latLngToLayerPoint(t
his._latlng);

this._updateBounds();
        },

        _updateBounds:
function () {
                var r =
this._radius,
                    r2 =
this._radiusY || r,
                    w =
this._clickTolerance(),
                    p = [r +
w, r2 + w];
                this._pxBounds
= new
Bounds(this._point.subtract(p)
, this._point.add(p));
        },

        _update: function () {
                if (this._map)
{

this._updatePath();
                }
```

```
        },

        _updatePath: function
() {

this._renderer._updateCircle(t
his);
        },

        _empty: function () {
                return
this._radius &&
!this._renderer._bounds.inters
ects(this._pxBounds);
        },

        // Needed by the
`Canvas` renderer for
interactivity
        _containsPoint:
function (p) {
                return
p.distanceTo(this._point) <=
this._radius +
this._clickTolerance();
        }
  });


  // @factory
L.circleMarker(latlng: LatLng,
options?: CircleMarker
options)
  // Instantiates a circle
marker object given a
geographical point, and an
optional options object.
  function
circleMarker(latlng, options)
{
        return new
CircleMarker(latlng, options);
  }

  /*
   * @class Circle
   * @aka L.Circle
```

```
   * @inherits CircleMarker
   *
   * A class for drawing
circle overlays on a map.
Extends `CircleMarker`.
   *
   * It's an approximation and
starts to diverge from a real
circle closer to poles (due to
projection distortion).
   *
   * @example
   *
   * ```js
   * L.circle([50.5, 30.5],
{radius: 200}).addTo(map);
   * ```
   */

  var Circle =
CircleMarker.extend({

        initialize: function
(latlng, options,
legacyOptions) {
                if (typeof
options === 'number') {
                        //
Backwards compatibility with
0.7.x factory (latlng, radius,
options?)

options = extend({},
legacyOptions, {radius:
options});
                }

setOptions(this, options);
                this._latlng =
toLatLng(latlng);

                if
(isNaN(this.options.radius)) {
throw new Error('Circle radius
cannot be NaN'); }

                // @section
```

```
                // @aka Circle
options
                // @option
radius: Number; Radius of the
circle, in meters.
                this._mRadius
= this.options.radius;
        },

        // @method
setRadius(radius: Number):
this
        // Sets the radius of
a circle. Units are in meters.
        setRadius: function
(radius) {
                this._mRadius
= radius;
                return
this.redraw();
        },

        // @method
getRadius(): Number
        // Returns the current
radius of a circle. Units are
in meters.
        getRadius: function ()
{
                return
this._mRadius;
        },

        // @method
getBounds(): LatLngBounds
        // Returns the
`LatLngBounds` of the path.
        getBounds: function ()
{
                var half =
[this._radius, this._radiusY
|| this._radius];

                return new
LatLngBounds(

this._map.layerPointToLatLng(t
```

```
his._point.subtract(half)),

this._map.layerPointToLatLng(t
his._point.add(half)));
        },

        setStyle:
Path.prototype.setStyle,

        _project: function ()
{

                var lng =
this._latlng.lng,
                    lat =
this._latlng.lat,
                    map =
this._map,
                    crs =
map.options.crs;

                if
(crs.distance ===
Earth.distance) {
                        var d
= Math.PI / 180,

latR = (this._mRadius /
Earth.R) / d,

top = map.project([lat + latR,
lng]),

bottom = map.project([lat -
latR, lng]),
                            p
= top.add(bottom).divideBy(2),

lat2 = map.unproject(p).lat,

lngR =
Math.acos((Math.cos(latR * d)
- Math.sin(lat * d) *
Math.sin(lat2 * d)) /

(Math.cos(lat * d) *
Math.cos(lat2 * d))) / d;
```

```
                        if
(isNaN(lngR) || lngR === 0) {

lngR = latR / Math.cos(Math.PI
/ 180 * lat); // Fallback for
edge case, #2425
                        }


this._point =
p.subtract(map.getPixelOrigin(
));

this._radius = isNaN(lngR) ? 0
: p.x - map.project([lat2, lng
- lngR]).x;

this._radiusY = p.y - top.y;

                } else {
                        var
latlng2 =
crs.unproject(crs.project(this
._latlng).subtract([this._mRad
ius, 0]));


this._point =
map.latLngToLayerPoint(this._l
atlng);

this._radius = this._point.x -
map.latLngToLayerPoint(latlng2
).x;
                }


this._updateBounds();
        }
  });

  // @factory L.circle(latlng:
LatLng, options?: Circle
options)
  // Instantiates a circle
object given a geographical
```

```
point, and an options object
  // which contains the circle
radius.
  // @alternative
  // @factory L.circle(latlng:
LatLng, radius: Number,
options?: Circle options)
  // Obsolete way of
instantiating a circle, for
compatibility with 0.7.x code.
  // Do not use in new
applications or plugins.
  function circle(latlng,
options, legacyOptions) {
        return new
Circle(latlng, options,
legacyOptions);
  }

  /*
   * @class Polyline
   * @aka L.Polyline
   * @inherits Path
   *
   * A class for drawing
polyline overlays on a map.
Extends `Path`.
   *
   * @example
   *
   * ```js
   * // create a red polyline
from an array of LatLng points
   * var latlngs = [
   *    [45.51, -122.68],
   *    [37.77, -122.43],
   *    [34.04, -118.2]
   * ];
   *
   * var polyline =
L.polyline(latlngs, {color:
'red'}).addTo(map);
   *
   * // zoom the map to the
polyline
   *
map.fitBounds(polyline.getBoun
```

```
ds());
   * ```
   *
   * You can also pass a
multi-dimensional array to
represent a `MultiPolyline`
shape:
   *
   * ```js
   * // create a red polyline
from an array of arrays of
LatLng points
   * var latlngs = [
   *    [[45.51, -122.68],
   *     [37.77, -122.43],
   *     [34.04, -118.2]],
   *    [[40.78, -73.91],
   *     [41.83, -87.62],
   *     [32.76, -96.72]]
   * ];
   * ```
   */


  var Polyline = Path.extend({

        // @section
        // @aka Polyline
options
        options: {
                // @option
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.
                smoothFactor:
1.0,

                // @option
noClip: Boolean = false
                // Disable
polyline clipping.
                noClip: false
```

```
        },

        initialize: function
(latlngs, options) {

setOptions(this, options);

this._setLatLngs(latlngs);
        },

        // @method
getLatLngs(): LatLng[]
        // Returns an array of
the points in the path, or
nested arrays of points in
case of multi-polyline.
        getLatLngs: function
() {
                return
this._latlngs;
        },

        // @method
setLatLngs(latlngs: LatLng[]):
this
        // Replaces all the
points in the polyline with
the given array of
geographical points.
        setLatLngs: function
(latlngs) {

this._setLatLngs(latlngs);
                return
this.redraw();
        },

        // @method isEmpty():
Boolean
        // Returns `true` if
the Polyline has no LatLngs.
        isEmpty: function () {
                return
!this._latlngs.length;
        },

        // @method
```

```
closestLayerPoint(p: Point):
Point
        // Returns the point
closest to `p` on the
Polyline.
        closestLayerPoint:
function (p) {
                var
minDistance = Infinity,
                    minPoint =
null,
                    closest =
_sqClosestPointOnSegment,
                    p1, p2;

                for (var j =
0, jLen = this._parts.length;
j < jLen; j++) {
                        var
points = this._parts[j];

                        for
(var i = 1, len =
points.length; i < len; i++) {

p1 = points[i - 1];

p2 = points[i];


var sqDist = closest(p, p1,
p2, true);


if (sqDist < minDistance) {

minDistance = sqDist;

minPoint = closest(p, p1, p2);

}
                        }
                }
                if (minPoint)
{

minPoint.distance =
```

```
Math.sqrt(minDistance);
                }
                return
minPoint;
        },

        // @method
getCenter(): LatLng
        // Returns the center
([centroid]
(https://en.wikipedia.org/wiki
/Centroid)) of the polyline.
        getCenter: function ()
{
                // throws
error when not yet added to
map as this center calculation
requires projected coordinates
                if
(!this._map) {
                        throw
new Error('Must add layer to
map before using
getCenter()');
                }
                return
polylineCenter(this._defaultSh
ape(), this._map.options.crs);
        },

        // @method
getBounds(): LatLngBounds
        // Returns the
`LatLngBounds` of the path.
        getBounds: function ()
{
                return
this._bounds;
        },

        // @method
addLatLng(latlng: LatLng,
latlngs?: LatLng[]): this
        // Adds a given point
to the polyline. By default,
adds to the first ring of
        // the polyline in
```

```
case of a multi-polyline, but
can be overridden by passing
        // a specific ring as
a LatLng array (that you can
earlier access with
[`getLatLngs`](#polyline-
getlatlngs)).
        addLatLng: function
(latlng, latlngs) {
                latlngs =
latlngs ||
this._defaultShape();
                latlng =
toLatLng(latlng);

latlngs.push(latlng);

this._bounds.extend(latlng);
                return
this.redraw();
        },

        _setLatLngs: function
(latlngs) {
                this._bounds =
new LatLngBounds();
                this._latlngs
=
this._convertLatLngs(latlngs);
        },

        _defaultShape:
function () {
                return
isFlat(this._latlngs) ?
this._latlngs :
this._latlngs[0];
        },

        // recursively convert
latlngs input into actual
LatLng instances; calculate
bounds along the way
        _convertLatLngs:
function (latlngs) {
                var result =
[],
```

```
                    flat =
isFlat(latlngs);

                for (var i =
0, len = latlngs.length; i <
len; i++) {
                        if
(flat) {

result[i] =
toLatLng(latlngs[i]);

this._bounds.extend(result[i])
;
                        } else
{

result[i] =
this._convertLatLngs(latlngs[i
]);
                        }
                }

                return result;
        },

        _project: function ()
{
                var pxBounds =
new Bounds();
                this._rings =
[];

this._projectLatlngs(this._lat
lngs, this._rings, pxBounds);

                if
(this._bounds.isValid() &&
pxBounds.isValid()) {

this._rawPxBounds = pxBounds;

this._updateBounds();
                }
        },

        _updateBounds:
```

```
function () {
                var w =
this._clickTolerance(),
                    p = new
Point(w, w);

                if
(!this._rawPxBounds) {

return;
                }

                this._pxBounds
= new Bounds([

this._rawPxBounds.min.subtract
(p),

this._rawPxBounds.max.add(p)
                ]);
        },

        // recursively turns
latlngs into a set of rings
with projected coordinates
        _projectLatlngs:
function (latlngs, result,
projectedBounds) {
                var flat =
latlngs[0] instanceof LatLng,
                    len =
latlngs.length,
                    i, ring;

                if (flat) {
                        ring =
[];
                        for (i
= 0; i < len; i++) {

ring[i] =
this._map.latLngToLayerPoint(l
atlngs[i]);

projectedBounds.extend(ring[i]
);
                        }
```

```
result.push(ring);
                } else {
                        for (i
= 0; i < len; i++) {

this._projectLatlngs(latlngs[i
], result, projectedBounds);
                        }
                }
        },

        // clip polyline by
renderer bounds so that we
have less to render for
performance
        _clipPoints: function
() {
                var bounds =
this._renderer._bounds;

                this._parts =
[];
                if
(!this._pxBounds ||
!this._pxBounds.intersects(bou
nds)) {

return;
                }

                if
(this.options.noClip) {

this._parts = this._rings;

return;
                }

                var parts =
this._parts,
                    i, j, k,
len, len2, segment, points;

                for (i = 0, k
= 0, len = this._rings.length;
i < len; i++) {
```

```
                        points
= this._rings[i];

                        for (j
= 0, len2 = points.length; j <
len2 - 1; j++) {

segment =
clipSegment(points[j],
points[j + 1], bounds, j,
true);


if (!segment) { continue; }


parts[k] = parts[k] || [];

parts[k].push(segment[0]);


// if segment goes out of
screen, or it's the last one,
it's the end of the line part

if ((segment[1] !== points[j +
1]) || (j === len2 - 2)) {

parts[k].push(segment[1]);

k++;

}
                        }
                }
        },

        // simplify each
clipped part of the polyline
for performance
        _simplifyPoints:
function () {
                var parts =
this._parts,
                    tolerance
= this.options.smoothFactor;
```

```
                for (var i =
0, len = parts.length; i <
len; i++) {

parts[i] = simplify(parts[i],
tolerance);
                }
        },

        _update: function () {
                if
(!this._map) { return; }


this._clipPoints();

this._simplifyPoints();

this._updatePath();
        },

        _updatePath: function
() {

this._renderer._updatePoly(thi
s);
        },

        // Needed by the
`Canvas` renderer for
interactivity
        _containsPoint:
function (p, closed) {
                var i, j, k,
len, len2, part,
                    w =
this._clickTolerance();

                if
(!this._pxBounds ||
!this._pxBounds.contains(p)) {
return false; }

                // hit
detection for polylines
                for (i = 0,
len = this._parts.length; i <
```

```
len; i++) {
                        part =
this._parts[i];

                        for (j
= 0, len2 = part.length, k =
len2 - 1; j < len2; k = j++) {

if (!closed && (j === 0)) {
continue; }


if (pointToSegmentDistance(p,
part[k], part[j]) <= w) {

return true;

}
                        }
                }
                return false;
        }
  });

  // @factory
L.polyline(latlngs: LatLng[],
options?: Polyline options)
  // Instantiates a polyline
object given an array of
geographical points and
  // optionally an options
object. You can create a
`Polyline` object with
  // multiple separate lines
(`MultiPolyline`) by passing
an array of arrays
  // of geographic points.
  function polyline(latlngs,
options) {
        return new
Polyline(latlngs, options);
  }

  // Retrocompat. Allow
plugins to support Leaflet
versions before and after 1.1.
  Polyline._flat = _flat;
```

```
  /*
   * @class Polygon
   * @aka L.Polygon
   * @inherits Polyline
   *
   * A class for drawing
polygon overlays on a map.
Extends `Polyline`.
   *
   * Note that points you pass
when creating a polygon
shouldn't have an additional
last point equal to the first
one — it's better to filter
out such points.
   *
   *
   * @example
   *
   * ```js
   * // create a red polygon
from an array of LatLng points
   * var latlngs = [[37,
-109.05],[41, -109.03],[41,
-102.05],[37, -102.04]];
   *
   * var polygon =
L.polygon(latlngs, {color:
'red'}).addTo(map);
   *
   * // zoom the map to the
polygon
   *
map.fitBounds(polygon.getBound
s());
   * ```
   *
   * You can also pass an
array of arrays of latlngs,
with the first array
representing the outer shape
and the other arrays
representing holes in the
outer shape:
   *
   * ```js
```

```
   * var latlngs = [
   *   [[37, -109.05],[41,
-109.03],[41, -102.05],[37,
-102.04]], // outer ring
   *   [[37.29, -108.58],
[40.71, -108.58],[40.71,
-102.50],[37.29, -102.50]] //
hole
   * ];
   * ```
   *
   * Additionally, you can
pass a multi-dimensional array
to represent a MultiPolygon
shape.
   *
   * ```js
   * var latlngs = [
   *   [ // first polygon
   *     [[37, -109.05],[41,
-109.03],[41, -102.05],[37,
-102.04]], // outer ring
   *     [[37.29, -108.58],
[40.71, -108.58],[40.71,
-102.50],[37.29, -102.50]] //
hole
   *   ],
   *   [ // second polygon
   *     [[41, -111.03],[45,
-111.04],[45, -104.05],[41,
-104.05]]
   *   ]
   * ];
   * ```
   */

  var Polygon =
Polyline.extend({

        options: {
                fill: true
        },

        isEmpty: function () {
                return
!this._latlngs.length ||
!this._latlngs[0].length;
```

```
        },

        // @method
getCenter(): LatLng
        // Returns the center
([centroid]
(http://en.wikipedia.org/wiki/
Centroid)) of the Polygon.
        getCenter: function ()
{
                // throws
error when not yet added to
map as this center calculation
requires projected coordinates
                if
(!this._map) {
                        throw
new Error('Must add layer to
map before using
getCenter()');
                }
                return
polygonCenter(this._defaultSha
pe(), this._map.options.crs);
        },

        _convertLatLngs:
function (latlngs) {
                var result =
Polyline.prototype._convertLat
Lngs.call(this, latlngs),
                    len =
result.length;

                // remove last
point if it equals first one
                if (len >= 2
&& result[0] instanceof LatLng
&& result[0].equals(result[len
- 1])) {

result.pop();
                }
                return result;
        },

        _setLatLngs: function
```

```
(latlngs) {

Polyline.prototype._setLatLngs
.call(this, latlngs);
                if
(isFlat(this._latlngs)) {

this._latlngs =
[this._latlngs];
                }
        },

        _defaultShape:
function () {
                return
isFlat(this._latlngs[0]) ?
this._latlngs[0] :
this._latlngs[0][0];
        },

        _clipPoints: function
() {
                // polygons
need a different clipping
algorithm so we redefine that

                var bounds =
this._renderer._bounds,
                    w =
this.options.weight,
                    p = new
Point(w, w);

                // increase
clip padding by stroke width
to avoid stroke on clip edges
                bounds = new
Bounds(bounds.min.subtract(p),
bounds.max.add(p));

                this._parts =
[];
                if
(!this._pxBounds ||
!this._pxBounds.intersects(bou
nds)) {
```

```
return;
                }

                if
(this.options.noClip) {

this._parts = this._rings;

return;
                }

                for (var i =
0, len = this._rings.length,
clipped; i < len; i++) {

clipped =
clipPolygon(this._rings[i],
bounds, true);
                        if
(clipped.length) {

this._parts.push(clipped);
                        }
                }
        },

        _updatePath: function
() {

this._renderer._updatePoly(thi
s, true);
        },

        // Needed by the
`Canvas` renderer for
interactivity
        _containsPoint:
function (p) {
                var inside =
false,
                    part, p1,
p2, i, j, k, len, len2;

                if
(!this._pxBounds ||
!this._pxBounds.contains(p)) {
return false; }
```

```
                // ray casting
algorithm for detecting if
point is in polygon
                for (i = 0,
len = this._parts.length; i <
len; i++) {
                        part =
this._parts[i];

                        for (j
= 0, len2 = part.length, k =
len2 - 1; j < len2; k = j++) {

p1 = part[j];

p2 = part[k];


if (((p1.y > p.y) !== (p2.y >
p.y)) && (p.x < (p2.x - p1.x)
* (p.y - p1.y) / (p2.y - p1.y)
+ p1.x)) {

inside = !inside;

}
                        }
                }

                // also check
if it's on polygon stroke
                return inside
||
Polyline.prototype._containsPo
int.call(this, p, true);
        }

  });


  // @factory
L.polygon(latlngs: LatLng[],
options?: Polyline options)
  function polygon(latlngs,
options) {
        return new
```

```
Polygon(latlngs, options);
  }

  /*
   * @class GeoJSON
   * @aka L.GeoJSON
   * @inherits FeatureGroup
   *
   * Represents a GeoJSON
object or an array of GeoJSON
objects. Allows you to parse
   * GeoJSON data and display
it on the map. Extends
`FeatureGroup`.
   *
   * @example
   *
   * ```js
   * L.geoJSON(data, {
   *    style: function
(feature) {
   *            return {color:
feature.properties.color};
   *    }
   * }).bindPopup(function
(layer) {
   *    return
layer.feature.properties.descr
iption;
   * }).addTo(map);
   * ```
   */

  var GeoJSON =
FeatureGroup.extend({

        /* @section
         * @aka GeoJSON
options
         *
         * @option
pointToLayer: Function = *
         * A `Function`
defining how GeoJSON points
spawn Leaflet layers. It is
internally
         * called when data is
```

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

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

		initialize: function
(geojson, options) {

setOptions(this, options);

			this._layers =
```

```
{};

                if (geojson) {

this.addData(geojson);
                }
        },

        // @method addData(
<GeoJSON> data ): this
        // Adds a GeoJSON
object to the layer.
        addData: function
(geojson) {
                var features =
isArray(geojson) ? geojson :
geojson.features,
                    i, len,
feature;

                if (features)
{
                        for (i
= 0, len = features.length; i
< len; i++) {

// only add this if geometry
or geometries are set and not
null

feature = features[i];

if (feature.geometries ||
feature.geometry ||
feature.features ||
feature.coordinates) {

this.addData(feature);

}
                        }
                        return
this;
                }

                var options =
this.options;
```

```
                if
(options.filter &&
!options.filter(geojson)) {
return this; }

                var layer =
geometryToLayer(geojson,
options);
                if (!layer) {
                        return
this;
                }
                layer.feature
= asFeature(geojson);


layer.defaultOptions =
layer.options;

this.resetStyle(layer);

                if
(options.onEachFeature) {

options.onEachFeature(geojson,
layer);
                }

                return
this.addLayer(layer);
        },

        // @method resetStyle(
<Path> layer? ): this
        // Resets the given
vector layer's style to the
original GeoJSON style, useful
for resetting style after
hover events.
        // If `layer` is
omitted, the style of all
features in the current layer
is reset.
        resetStyle: function
(layer) {
                if (layer ===
```

```
undefined) {
                        return
this.eachLayer(this.resetStyle
, this);
                }
                // reset any
custom styles
                layer.options
= extend({},
layer.defaultOptions);

this._setLayerStyle(layer,
this.options.style);
                return this;
        },

        // @method setStyle(
<Function> style ): this
        // Changes styles of
GeoJSON vector layers with the
given style function.
        setStyle: function
(style) {
                return
this.eachLayer(function
(layer) {

this._setLayerStyle(layer,
style);
                }, this);
        },

        _setLayerStyle:
function (layer, style) {
                if
(layer.setStyle) {
                        if
(typeof style === 'function')
{

style = style(layer.feature);
                        }

layer.setStyle(style);
                }
        }
  });
```

```
  // @section
  // There are several static
functions which can be called
without instantiating
L.GeoJSON:

  // @function
geometryToLayer(featureData:
Object, options?: GeoJSON
options): Layer
  // Creates a `Layer` from a
given GeoJSON feature. Can use
a custom
  // [`pointToLayer`]
(#geojson-pointtolayer) and/or
[`coordsToLatLng`](#geojson-
coordstolatlng)
  // functions if provided as
options.
  function
geometryToLayer(geojson,
options) {

        var geometry =
geojson.type === 'Feature' ?
geojson.geometry : geojson,
            coords = geometry
? geometry.coordinates : null,
            layers = [],
            pointToLayer =
options &&
options.pointToLayer,
            _coordsToLatLng =
options &&
options.coordsToLatLng ||
coordsToLatLng,
            latlng, latlngs,
i, len;

        if (!coords &&
!geometry) {
                return null;
        }

        switch (geometry.type)
{
```

```
        case 'Point':
                latlng =
_coordsToLatLng(coords);
                return
_pointToLayer(pointToLayer,
geojson, latlng, options);

        case 'MultiPoint':
                for (i = 0,
len = coords.length; i < len;
i++) {
                        latlng
= _coordsToLatLng(coords[i]);

layers.push(_pointToLayer(poin
tToLayer, geojson, latlng,
options));
                }
                return new
FeatureGroup(layers);

        case 'LineString':
        case
'MultiLineString':
                latlngs =
coordsToLatLngs(coords,
geometry.type === 'LineString'
? 0 : 1, _coordsToLatLng);
                return new
Polyline(latlngs, options);

        case 'Polygon':
        case 'MultiPolygon':
                latlngs =
coordsToLatLngs(coords,
geometry.type === 'Polygon' ?
1 : 2, _coordsToLatLng);
                return new
Polygon(latlngs, options);

        case
'GeometryCollection':
                for (i = 0,
len =
geometry.geometries.length; i
< len; i++) {
                        var
```

```
geoLayer = geometryToLayer({

geometry:
geometry.geometries[i],

type: 'Feature',

properties: geojson.properties
                        },
options);

                        if
(geoLayer) {

layers.push(geoLayer);
                        }
                }
                return new
FeatureGroup(layers);

        case
'FeatureCollection':
                for (i = 0,
len =
geometry.features.length; i <
len; i++) {
                        var
featureLayer =
geometryToLayer(geometry.featu
res[i], options);

                        if
(featureLayer) {

layers.push(featureLayer);
                        }
                }
                return new
FeatureGroup(layers);

        default:
                throw new
Error('Invalid GeoJSON
object.');
        }
  }
```

```
  function
_pointToLayer(pointToLayerFn,
geojson, latlng, options) {
        return pointToLayerFn
?

pointToLayerFn(geojson,
latlng) :
                  new
Marker(latlng, options &&
options.markersInheritOptions
&& options);
  }

  // @function
coordsToLatLng(coords: Array):
LatLng
  // Creates a `LatLng` object
from an array of 2 numbers
(longitude, latitude)
  // or 3 numbers (longitude,
latitude, altitude) used in
GeoJSON for points.
  function
coordsToLatLng(coords) {
        return new
LatLng(coords[1], coords[0],
coords[2]);
  }

  // @function
coordsToLatLngs(coords: Array,
levelsDeep?: Number,
coordsToLatLng?: Function):
Array
  // Creates a
multidimensional array of
`LatLng`s from a GeoJSON
coordinates array.
  // `levelsDeep` specifies
the nesting level (0 is for an
array of points, 1 for an
array of arrays of points,
etc., 0 by default).
  // Can use a custom
[`coordsToLatLng`](#geojson-
coordstolatlng) function.
```

```
  function
coordsToLatLngs(coords,
levelsDeep, _coordsToLatLng) {
        var latlngs = [];

        for (var i = 0, len =
coords.length, latlng; i <
len; i++) {
                latlng =
levelsDeep ?

coordsToLatLngs(coords[i],
levelsDeep - 1,
_coordsToLatLng) :

(_coordsToLatLng ||
coordsToLatLng)(coords[i]);


latlngs.push(latlng);
        }

        return latlngs;
  }

  // @function
latLngToCoords(latlng: LatLng,
precision?: Number|false):
Array
  // Reverse of
[`coordsToLatLng`](#geojson-
coordstolatlng)
  // Coordinates values are
rounded with [`formatNum`]
(#util-formatnum) function.
  function
latLngToCoords(latlng,
precision) {
        latlng =
toLatLng(latlng);
        return latlng.alt !==
undefined ?

[formatNum(latlng.lng,
precision),
formatNum(latlng.lat,
precision),
```

```
formatNum(latlng.alt,
precision)] :

[formatNum(latlng.lng,
precision),
formatNum(latlng.lat,
precision)];
  }

  // @function
latLngsToCoords(latlngs:
Array, levelsDeep?: Number,
closed?: Boolean, precision?:
Number|false): Array
  // Reverse of
[`coordsToLatLngs`](#geojson-
coordstolatlngs)
  // `closed` determines
whether the first point should
be appended to the end of the
array to close the feature,
only used when `levelsDeep` is
0. False by default.
  // Coordinates values are
rounded with [`formatNum`]
(#util-formatnum) function.
  function
latLngsToCoords(latlngs,
levelsDeep, closed, precision)
{
        var coords = [];

        for (var i = 0, len =
latlngs.length; i < len; i++)
{
                // Check for
flat arrays required to ensure
unbalanced arrays are
correctly converted in
recursion

coords.push(levelsDeep ?

latLngsToCoords(latlngs[i],
isFlat(latlngs[i]) ? 0 :
levelsDeep - 1, closed,
precision) :
```

```
latLngToCoords(latlngs[i],
precision));
        }

        if (!levelsDeep &&
closed) {

coords.push(coords[0]);
        }

        return coords;
  }

  function getFeature(layer,
newGeometry) {
        return layer.feature ?
                extend({},
layer.feature, {geometry:
newGeometry}) :

asFeature(newGeometry);
  }

  // @function
asFeature(geojson: Object):
Object
  // Normalize GeoJSON
geometries/features into
GeoJSON features.
  function asFeature(geojson)
{
        if (geojson.type ===
'Feature' || geojson.type ===
'FeatureCollection') {
                return
geojson;
        }

        return {
                type:
'Feature',
                properties:
{},
                geometry:
geojson
        };
```

```
  }

  var PointToGeoJSON = {
        toGeoJSON: function
(precision) {
                  return
getFeature(this, {
                        type:
'Point',

coordinates:
latLngToCoords(this.getLatLng(
), precision)
                  });
        }
  };

  // @namespace Marker
  // @section Other methods
  // @method
toGeoJSON(precision?:
Number|false): Object
  // Coordinates values are
rounded with [`formatNum`]
(#util-formatnum) function
with given `precision`.
  // Returns a [`GeoJSON`]
(https://en.wikipedia.org/wiki
/GeoJSON) representation of
the marker (as a GeoJSON
`Point` Feature).

Marker.include(PointToGeoJSON)
;

  // @namespace CircleMarker
  // @method
toGeoJSON(precision?:
Number|false): Object
  // Coordinates values are
rounded with [`formatNum`]
(#util-formatnum) function
with given `precision`.
  // Returns a [`GeoJSON`]
(https://en.wikipedia.org/wiki
/GeoJSON) representation of
the circle marker (as a
```

```
GeoJSON `Point` Feature).

Circle.include(PointToGeoJSON)
;

CircleMarker.include(PointToGe
oJSON);


  // @namespace Polyline
  // @method
toGeoJSON(precision?:
Number|false): Object
  // Coordinates values are
rounded with [`formatNum`]
(#util-formatnum) function
with given `precision`.
  // Returns a [`GeoJSON`]
(https://en.wikipedia.org/wiki
/GeoJSON) representation of
the polyline (as a GeoJSON
`LineString` or
`MultiLineString` Feature).
  Polyline.include({
        toGeoJSON: function
(precision) {
                var multi =
!isFlat(this._latlngs);

                var coords =
latLngsToCoords(this._latlngs,
multi ? 1 : 0, false,
precision);

                return
getFeature(this, {
                        type:
(multi ? 'Multi' : '') +
'LineString',

coordinates: coords
                });
        }
  });

  // @namespace Polygon
  // @method
```

```
toGeoJSON(precision?:
Number|false): Object
  // Coordinates values are
rounded with [`formatNum`]
(#util-formatnum) function
with given `precision`.
  // Returns a [`GeoJSON`]
(https://en.wikipedia.org/wiki
/GeoJSON) representation of
the polygon (as a GeoJSON
`Polygon` or `MultiPolygon`
Feature).
  Polygon.include({
        toGeoJSON: function
(precision) {
                  var holes =
!isFlat(this._latlngs),
                      multi =
holes &&
!isFlat(this._latlngs[0]);

                  var coords =
latLngsToCoords(this._latlngs,
multi ? 2 : holes ? 1 : 0,
true, precision);

                  if (!holes) {
                        coords
= [coords];
                  }

                  return
getFeature(this, {
                        type:
(multi ? 'Multi' : '') +
'Polygon',

coordinates: coords
                  });
        }
  });


  // @namespace LayerGroup
  LayerGroup.include({
        toMultiPoint: function
(precision) {
```

```
                var coords =
[];


this.eachLayer(function
(layer) {

coords.push(layer.toGeoJSON(pr
ecision).geometry.coordinates)
;
                });

                return
getFeature(this, {
                        type:
'MultiPoint',

coordinates: coords
                });
        },

        // @method
toGeoJSON(precision?:
Number|false): Object
        // Coordinates values
are rounded with [`formatNum`]
(#util-formatnum) function
with given `precision`.
        // Returns a
[`GeoJSON`]
(https://en.wikipedia.org/wiki
/GeoJSON) representation of
the layer group (as a GeoJSON
`FeatureCollection`,
`GeometryCollection`, or
`MultiPoint`).
        toGeoJSON: function
(precision) {

                var type =
this.feature &&
this.feature.geometry &&
this.feature.geometry.type;

                if (type ===
'MultiPoint') {
                        return
```

```
this.toMultiPoint(precision);
                }

                var
isGeometryCollection = type
=== 'GeometryCollection',
                    jsons =
[];


this.eachLayer(function
(layer) {
                        if
(layer.toGeoJSON) {

var json =
layer.toGeoJSON(precision);

if (isGeometryCollection) {

jsons.push(json.geometry);

} else {

var feature = asFeature(json);

// Squash nested feature
collections

if (feature.type ===
'FeatureCollection') {

jsons.push.apply(jsons,
feature.features);

} else {

jsons.push(feature);

}

}
                        }
                });

                if
(isGeometryCollection) {
```

```
                        return
getFeature(this, {

geometries: jsons,

type: 'GeometryCollection'
                        });
                }

                return {
                        type:
'FeatureCollection',

features: jsons
                };
        }
  });

  // @namespace GeoJSON
  // @factory
L.geoJSON(geojson?: Object,
options?: GeoJSON options)
  // Creates a GeoJSON layer.
Optionally accepts an object
in
  // [GeoJSON format]
(https://tools.ietf.org/html/r
fc7946) to display on the map
  // (you can alternatively
add it later with `addData`
method) and an `options`
object.
  function geoJSON(geojson,
options) {
        return new
GeoJSON(geojson, options);
  }

  // Backward compatibility.
  var geoJson = geoJSON;

  /*
   * @class ImageOverlay
   * @aka L.ImageOverlay
   * @inherits Interactive
layer
   *
```

```
   * Used to load and display
a single image over specific
bounds of the map. Extends
`Layer`.
   *
   * @example
   *
   * ```js
   * var imageUrl =
'https://maps.lib.utexas.edu/m
aps/historical/newark_nj_1922.
jpg',
   *    imageBounds =
[[40.712216, -74.22655],
[40.773941, -74.12544]];
   * L.imageOverlay(imageUrl,
imageBounds).addTo(map);
   * ```
   */

  var ImageOverlay =
Layer.extend({

        // @section
        // @aka ImageOverlay
options
        options: {
                // @option
opacity: Number = 1.0
                // The opacity
of the image overlay.
                opacity: 1,

                // @option
alt: String = ''
                // Text for
the `alt` attribute of the
image (useful for
accessibility).
                alt: '',

                // @option
interactive: Boolean = false
                // If `true`,
the image overlay will emit
[mouse events](#interactive-
layer) when clicked or
```

```
hovered.
		interactive:
false,

		// @option
crossOrigin: Boolean|String =
false
		// Whether the
crossOrigin attribute will be
added to the image.
		// If a String
is provided, the image will
have its crossOrigin attribute
set to the String provided.
This is needed if you want to
access image pixel data.
		// Refer to
[CORS Settings]
(https://developer.mozilla.org
/en-
US/docs/Web/HTML/CORS_settings
_attributes) for valid String
values.
		crossOrigin:
false,

		// @option
errorOverlayUrl: String = ''
		// URL to the
overlay image to show in place
of the overlay that failed to
load.

errorOverlayUrl: '',

		// @option
zIndex: Number = 1
		// The
explicit [zIndex]
(https://developer.mozilla.org
/docs/Web/CSS/CSS_Positioning/
Understanding_z_index) of the
overlay layer.
		zIndex: 1,

		// @option
className: String = ''
```

```
                // A custom
class name to assign to the
image. Empty by default.
                className: ''
        },

        initialize: function
(url, bounds, options) { //
(String, LatLngBounds, Object)
                this._url =
url;
                this._bounds =
toLatLngBounds(bounds);


setOptions(this, options);
        },

        onAdd: function () {
                if
(!this._image) {

this._initImage();

                        if
(this.options.opacity < 1) {

this._updateOpacity();
                        }
                }

                if
(this.options.interactive) {

addClass(this._image,
'leaflet-interactive');

this.addInteractiveTarget(this
._image);
                }


this.getPane().appendChild(thi
s._image);
                this._reset();
        },
```

```
        onRemove: function ()
{

remove(this._image);
                if
(this.options.interactive) {

this.removeInteractiveTarget(t
his._image);
                }
        },

        // @method
setOpacity(opacity: Number):
this
        // Sets the opacity of
the overlay.
        setOpacity: function
(opacity) {

this.options.opacity =
opacity;

                if
(this._image) {

this._updateOpacity();
                }
                return this;
        },

        setStyle: function
(styleOpts) {
                if
(styleOpts.opacity) {

this.setOpacity(styleOpts.opac
ity);
                }
                return this;
        },

        // @method
bringToFront(): this
        // Brings the layer to
the top of all overlays.
        bringToFront: function
```

```
() {
                if (this._map)
{

toFront(this._image);
                }
                return this;
        },

        // @method
bringToBack(): this
        // Brings the layer to
the bottom of all overlays.
        bringToBack: function
() {
                if (this._map)
{

toBack(this._image);
                }
                return this;
        },

        // @method setUrl(url:
String): this
        // Changes the URL of
the image.
        setUrl: function (url)
{
                this._url =
url;

                if
(this._image) {

this._image.src = url;
                }
                return this;
        },

        // @method
setBounds(bounds:
LatLngBounds): this
        // Update the bounds
that this ImageOverlay covers
        setBounds: function
(bounds) {
```

```
                this._bounds =
toLatLngBounds(bounds);

                if (this._map)
{

this._reset();
                }
                return this;
        },

        getEvents: function ()
{
                var events = {
                        zoom:
this._reset,

viewreset: this._reset
                };

                if
(this._zoomAnimated) {

events.zoomanim =
this._animateZoom;
                }

                return events;
        },

        // @method
setZIndex(value: Number): this
        // Changes the
[zIndex](#imageoverlay-zindex)
of the image overlay.
        setZIndex: function
(value) {

this.options.zIndex = value;

this._updateZIndex();
                return this;
        },

        // @method
getBounds(): LatLngBounds
        // Get the bounds that
```

```
this ImageOverlay covers
        getBounds: function ()
{
                return
this._bounds;
        },

        // @method
getElement(): HTMLElement
        // Returns the
instance of
[`HTMLImageElement`]
(https://developer.mozilla.org
/docs/Web/API/HTMLImageElement
)
        // used by this
overlay.
        getElement: function
() {
                return
this._image;
        },

        _initImage: function
() {
                var
wasElementSupplied =
this._url.tagName === 'IMG';
                var img =
this._image =
wasElementSupplied ? this._url
: create$1('img');

                addClass(img,
'leaflet-image-layer');
                if
(this._zoomAnimated) {
addClass(img, 'leaflet-zoom-
animated'); }
                if
(this.options.className) {
addClass(img,
this.options.className); }


img.onselectstart = falseFn;
```

```
img.onmousemove = falseFn;

                // @event
load: Event
                // Fired when
the ImageOverlay layer has
loaded its image
                img.onload =
bind(this.fire, this, 'load');
                img.onerror =
bind(this._overlayOnError,
this, 'error');

                if
(this.options.crossOrigin ||
this.options.crossOrigin ===
'') {

img.crossOrigin =
this.options.crossOrigin ===
true ? '' :
this.options.crossOrigin;
                }

                if
(this.options.zIndex) {

this._updateZIndex();
                }

                if
(wasElementSupplied) {

this._url = img.src;

return;
                }

                img.src =
this._url;
                img.alt =
this.options.alt;
        },

        _animateZoom: function
(e) {
                var scale =
```

```
this._map.getZoomScale(e.zoom)
,
                    offset =
this._map._latLngBoundsToNewLa
yerBounds(this._bounds,
e.zoom, e.center).min;


setTransform(this._image,
offset, scale);
        },

        _reset: function () {
                var image =
this._image,
                    bounds =
new Bounds(

this._map.latLngToLayerPoint(t
his._bounds.getNorthWest()),

this._map.latLngToLayerPoint(t
his._bounds.getSouthEast())),
                    size =
bounds.getSize();


setPosition(image,
bounds.min);


image.style.width  = size.x +
'px';

image.style.height = size.y +
'px';
        },

        _updateOpacity:
function () {

setOpacity(this._image,
this.options.opacity);
        },

        _updateZIndex:
function () {
```

```
                if
(this._image &&
this.options.zIndex !==
undefined &&
this.options.zIndex !== null)
{

this._image.style.zIndex =
this.options.zIndex;
                }
        },

        _overlayOnError:
function () {
                // @event
error: Event
                // Fired when
the ImageOverlay layer fails
to load its image

this.fire('error');

                var errorUrl =
this.options.errorOverlayUrl;
                if (errorUrl
&& this._url !== errorUrl) {

this._url = errorUrl;

this._image.src = errorUrl;
                }
        },

        // @method
getCenter(): LatLng
        // Returns the center
of the ImageOverlay.
        getCenter: function ()
{
                return
this._bounds.getCenter();
        }
  });

  // @factory
L.imageOverlay(imageUrl:
String, bounds: LatLngBounds,
```

```
options?: ImageOverlay
options)
  // Instantiates an image
overlay object given the URL
of the image and the
  // geographical bounds it is
tied to.
  var imageOverlay = function
(url, bounds, options) {
        return new
ImageOverlay(url, bounds,
options);
  };

  /*
   * @class VideoOverlay
   * @aka L.VideoOverlay
   * @inherits ImageOverlay
   *
   * Used to load and display
a video player over specific
bounds of the map. Extends
`ImageOverlay`.
   *
   * A video overlay uses the
[`<video>`]
(https://developer.mozilla.org
/docs/Web/HTML/Element/video)
   * HTML5 element.
   *
   * @example
   *
   * ```js
   * var videoUrl =
'https://www.mapbox.com/bites/
00188/patricia_nasa.webm',
   *    videoBounds = [[ 32,
-130], [ 13, -100]];
   * L.videoOverlay(videoUrl,
videoBounds ).addTo(map);
   * ```
   */

  var VideoOverlay =
ImageOverlay.extend({

        // @section
```

```
        // @aka VideoOverlay
options
        options: {
                // @option
autoplay: Boolean = true
                // Whether the
video starts playing
automatically when loaded.
                // On some
browsers autoplay will only
work with `muted: true`
                autoplay:
true,

                // @option
loop: Boolean = true
                // Whether the
video will loop back to the
beginning when played.
                loop: true,

                // @option
keepAspectRatio: Boolean =
true
                // Whether the
video will save aspect ratio
after the projection.
                // Relevant
for supported browsers. See
[browser compatibility]
(https://developer.mozilla.org
/en-US/docs/Web/CSS/object-
fit)

keepAspectRatio: true,

                // @option
muted: Boolean = false
                // Whether the
video starts on mute when
loaded.
                muted: false,

                // @option
playsInline: Boolean = true
                // Mobile
browsers will play the video
```

```
right where it is instead of
open it up in fullscreen mode.
                playsInline:
true
        },

        _initImage: function
() {
                var
wasElementSupplied =
this._url.tagName === 'VIDEO';
                var vid =
this._image =
wasElementSupplied ? this._url
: create$1('video');

                addClass(vid,
'leaflet-image-layer');
                if
(this._zoomAnimated) {
addClass(vid, 'leaflet-zoom-
animated'); }
                if
(this.options.className) {
addClass(vid,
this.options.className); }


vid.onselectstart = falseFn;

vid.onmousemove = falseFn;

                // @event
load: Event
                // Fired when
the video has finished loading
the first frame

vid.onloadeddata =
bind(this.fire, this, 'load');

                if
(wasElementSupplied) {
                        var
sourceElements =
vid.getElementsByTagName('sour
ce');
```

```
                        var
sources = [];
                        for
(var j = 0; j <
sourceElements.length; j++) {

sources.push(sourceElements[j]
.src);
                        }


this._url =
(sourceElements.length > 0) ?
sources : [vid.src];

return;
                }

                if
(!isArray(this._url)) {
this._url = [this._url]; }

                if
(!this.options.keepAspectRatio
&&
Object.prototype.hasOwnPropert
y.call(vid.style,
'objectFit')) {

vid.style['objectFit'] =
'fill';
                }
                vid.autoplay =
!!this.options.autoplay;
                vid.loop =
!!this.options.loop;
                vid.muted =
!!this.options.muted;

vid.playsInline =
!!this.options.playsInline;
                for (var i =
0; i < this._url.length; i++)
{
                        var
source = create$1('source');
```

```
source.src = this._url[i];

vid.appendChild(source);
                }
        }

        // @method
getElement(): HTMLVideoElement
        // Returns the
instance of
[`HTMLVideoElement`]
(https://developer.mozilla.org
/docs/Web/API/HTMLVideoElement
)
        // used by this
overlay.
  });


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

  function videoOverlay(video,
bounds, options) {
        return new
VideoOverlay(video, bounds,
options);
  }

  /*
   * @class SVGOverlay
   * @aka L.SVGOverlay
   * @inherits ImageOverlay
   *
   * Used to load, display and
provide DOM access to an SVG
```

```
file over specific bounds of
the map. Extends
`ImageOverlay`.
   *
   * An SVG overlay uses the
[`<svg>`]
(https://developer.mozilla.org
/docs/Web/SVG/Element/svg)
element.
   *
   * @example
   *
   * ```js
   * var svgElement =
document.createElementNS("http
://www.w3.org/2000/svg",
"svg");
   *
svgElement.setAttribute('xmlns
',
"http://www.w3.org/2000/svg");
   *
svgElement.setAttribute('viewB
ox', "0 0 200 200");
   * svgElement.innerHTML =
'<rect width="200"
height="200"/><rect x="75"
y="23" width="50" height="50"
style="fill:red"/><rect x="75"
y="123" width="50" height="50"
style="fill:#0013ff"/>';
   * var svgElementBounds = [
[ 32, -130 ], [ 13, -100 ] ];
   * L.svgOverlay(svgElement,
svgElementBounds).addTo(map);
   * ```
   */

  var SVGOverlay =
ImageOverlay.extend({
        _initImage: function
() {
                var el =
this._image = this._url;

                addClass(el,
'leaflet-image-layer');
```

```
                if
(this._zoomAnimated) {
addClass(el, 'leaflet-zoom-
animated'); }
                if
(this.options.className) {
addClass(el,
this.options.className); }


el.onselectstart = falseFn;
                el.onmousemove
= falseFn;
        }

        // @method
getElement(): SVGElement
        // Returns the
instance of [`SVGElement`]
(https://developer.mozilla.org
/docs/Web/API/SVGElement)
        // used by this
overlay.
  });


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

  function svgOverlay(el,
bounds, options) {
        return new
SVGOverlay(el, bounds,
options);
  }

  /*
```

```
   * @class DivOverlay
   * @inherits Interactive
layer
   * @aka L.DivOverlay
   * Base model for L.Popup
and L.Tooltip. Inherit from it
for custom overlays like
plugins.
   */

  // @namespace DivOverlay
  var DivOverlay =
Layer.extend({

        // @section
        // @aka DivOverlay
options
        options: {
                // @option
interactive: Boolean = false
                // If true,
the popup/tooltip will listen
to the mouse events.
                interactive:
false,

                // @option
offset: Point = Point(0, 0)
                // The offset
of the overlay position.
                offset: [0,
0],

                // @option
className: String = ''
                // A custom
CSS class name to assign to
the overlay.
                className: '',

                // @option
pane: String = undefined
                // `Map pane`
where the overlay will be
added.
                pane:
undefined,
```

```
                // @option
content:
String|HTMLElement|Function =
''
                // Sets the
HTML content of the overlay
while initializing. If a
function is passed the source
layer will be
                // passed to
the function. The function
should return a `String` or
`HTMLElement` to be used in
the overlay.
                content: ''
        },

        initialize: function
(options, source) {
                if (options &&
(options instanceof L.LatLng
|| isArray(options))) {

this._latlng =
toLatLng(options);

setOptions(this, source);
                } else {

setOptions(this, options);

this._source = source;
                }
                if
(this.options.content) {

this._content =
this.options.content;
                }
        },

        // @method openOn(map:
Map): this
        // Adds the overlay to
the map.
        // Alternative to
```

```
`map.openPopup(popup)`/`.openT
ooltip(tooltip)`.
        openOn: function (map)
{
                map =
arguments.length ? map :
this._source._map; //
experimental, not the part of
public api
                if
(!map.hasLayer(this)) {

map.addLayer(this);
                }
                return this;
        },

        // @method close():
this
        // Closes the overlay.
        // Alternative to
`map.closePopup(popup)`/`.clos
eTooltip(tooltip)`
        // and
`layer.closePopup()`/`.closeTo
oltip()`.
        close: function () {
                if (this._map)
{

this._map.removeLayer(this);
                }
                return this;
        },

        // @method
toggle(layer?: Layer): this
        // Opens or closes the
overlay bound to layer
depending on its current
state.
        // Argument may be
omitted only for overlay bound
to layer.
        // Alternative to
`layer.togglePopup()`/`.toggle
Tooltip()`.
```

```
        toggle: function
(layer) {
                if (this._map)
{

this.close();
                } else {
                        if
(arguments.length) {

this._source = layer;
                        } else
{

layer = this._source;
                        }

this._prepareOpen();

                        //
open the overlay on the map

this.openOn(layer._map);
                }
                return this;
        },

        onAdd: function (map)
{

this._zoomAnimated =
map._zoomAnimated;

                if
(!this._container) {

this._initLayout();
                }

                if
(map._fadeAnimated) {

setOpacity(this._container,
0);
                }
```

```
clearTimeout(this._removeTimeo
ut);

this.getPane().appendChild(thi
s._container);
                this.update();

                if
(map._fadeAnimated) {

setOpacity(this._container,
1);
                }


this.bringToFront();

                if
(this.options.interactive) {

addClass(this._container,
'leaflet-interactive');

this.addInteractiveTarget(this
._container);
                }
        },

        onRemove: function
(map) {
                if
(map._fadeAnimated) {

setOpacity(this._container,
0);

this._removeTimeout =
setTimeout(bind(remove,
undefined, this._container),
200);
                } else {

remove(this._container);
                }

                if
(this.options.interactive) {
```

```
removeClass(this._container,
'leaflet-interactive');

this.removeInteractiveTarget(t
his._container);
                }
        },

        // @namespace
DivOverlay
        // @method getLatLng:
LatLng
        // Returns the
geographical point of the
overlay.
        getLatLng: function ()
{
                return
this._latlng;
        },

        // @method
setLatLng(latlng: LatLng):
this
        // Sets the
geographical point where the
overlay will open.
        setLatLng: function
(latlng) {
                this._latlng =
toLatLng(latlng);
                if (this._map)
{

this._updatePosition();

this._adjustPan();
                }
                return this;
        },

        // @method getContent:
String|HTMLElement
        // Returns the content
of the overlay.
        getContent: function
```

```
() {
                return
this._content;
        },

        // @method
setContent(htmlContent:
String|HTMLElement|Function):
this
        // Sets the HTML
content of the overlay. If a
function is passed the source
layer will be passed to the
function.
        // The function should
return a `String` or
`HTMLElement` to be used in
the overlay.
        setContent: function
(content) {
                this._content
= content;
                this.update();
                return this;
        },

        // @method getElement:
String|HTMLElement
        // Returns the HTML
container of the overlay.
        getElement: function
() {
                return
this._container;
        },

        // @method update:
null
        // Updates the overlay
content, layout and position.
Useful for updating the
overlay after something inside
changed, e.g. image loaded.
        update: function () {
                if
(!this._map) { return; }
```

```
this._container.style.visibili
ty = 'hidden';


this._updateContent();

this._updateLayout();

this._updatePosition();


this._container.style.visibili
ty = '';


this._adjustPan();
        },

        getEvents: function ()
{
                var events = {
                        zoom:
this._updatePosition,

viewreset:
this._updatePosition
                };

                if
(this._zoomAnimated) {

events.zoomanim =
this._animateZoom;
                }
                return events;
        },

        // @method isOpen:
Boolean
        // Returns `true` when
the overlay is visible on the
map.
        isOpen: function () {
                return
!!this._map &&
this._map.hasLayer(this);
```

```
        },

        // @method
bringToFront: this
        // Brings this overlay
in front of other overlays (in
the same map pane).
        bringToFront: function
() {
                if (this._map)
{

toFront(this._container);
                }
                return this;
        },

        // @method
bringToBack: this
        // Brings this overlay
to the back of other overlays
(in the same map pane).
        bringToBack: function
() {
                if (this._map)
{

toBack(this._container);
                }
                return this;
        },

        // prepare bound
overlay to open: update latlng
pos / content source (for
FeatureGroup)
        _prepareOpen: function
(latlng) {
                var source =
this._source;
                if
(!source._map) { return false;
}

                if (source
instanceof FeatureGroup) {
                        source
```

```
= null;
                        var
layers = this._source._layers;
                        for
(var id in layers) {

if (layers[id]._map) {

source = layers[id];

break;

}
                        }
                        if
(!source) { return false; } //
Unable to get source layer.

                        // set
overlay source to this layer

this._source = source;
                }

                if (!latlng) {
                        if
(source.getCenter) {

latlng = source.getCenter();
                        } else
if (source.getLatLng) {

latlng = source.getLatLng();
                        } else
if (source.getBounds) {

latlng =
source.getBounds().getCenter()
;
                        } else
{

throw new Error('Unable to get
source layer LatLng.');
                        }
                }
```

```
this.setLatLng(latlng);

                if (this._map)
{
                        //
update the overlay (content,
layout, etc...)

this.update();
                }

                return true;
        },

        _updateContent:
function () {
                if
(!this._content) { return; }

                var node =
this._contentNode;
                var content =
(typeof this._content ===
'function') ?
this._content(this._source ||
this) : this._content;

                if (typeof
content === 'string') {

node.innerHTML = content;
                } else {
                        while
(node.hasChildNodes()) {

node.removeChild(node.firstChi
ld);
                        }

node.appendChild(content);
                }

                // @namespace
DivOverlay
                // @section
DivOverlay events
                // @event
```

```
contentupdate: Event
                // Fired when
the content of the overlay is
updated

this.fire('contentupdate');
        },

        _updatePosition:
function () {
                if
(!this._map) { return; }

                var pos =
this._map.latLngToLayerPoint(t
his._latlng),
                    offset =
toPoint(this.options.offset),
                    anchor =
this._getAnchor();

                if
(this._zoomAnimated) {

setPosition(this._container,
pos.add(anchor));
                } else {
                        offset
= offset.add(pos).add(anchor);
                }

                var bottom =
this._containerBottom = -
offset.y,
                    left =
this._containerLeft = -
Math.round(this._containerWidt
h / 2) + offset.x;

                // bottom
position the overlay in case
the height of the overlay
changes (images loading etc)

this._container.style.bottom =
bottom + 'px';
```

```
this._container.style.left =
left + 'px';
        },

        _getAnchor: function
() {
                return [0, 0];
        }

  });

  Map.include({
        _initOverlay: function
(OverlayClass, content,
latlng, options) {
                var overlay =
content;
                if (!(overlay
instanceof OverlayClass)) {

overlay = new
OverlayClass(options).setConte
nt(content);
                }
                if (latlng) {

overlay.setLatLng(latlng);
                }
                return
overlay;
        }
  });


  Layer.include({
        _initOverlay: function
(OverlayClass, old, content,
options) {
                var overlay =
content;
                if (overlay
instanceof OverlayClass) {

setOptions(overlay, options);

overlay._source = this;
                } else {
```

```
overlay = (old && !options) ?
old : new
OverlayClass(options, this);

overlay.setContent(content);
                }
                return
overlay;
        }
  });

  /*
   * @class Popup
   * @inherits DivOverlay
   * @aka L.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.
   *
   * @example
   *
   * If you want to just bind
a popup to marker click and
then open it, it's really
easy:
   *
   * ```js
   *
marker.bindPopup(popupContent)
.openPopup();
   * ```
   * Path overlays like
polylines also have a
`bindPopup` method.
   *
   * A popup can be also
standalone:
   *
```

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


  // @namespace Popup
  var Popup =
DivOverlay.extend({

        // @section
        // @aka Popup options
        options: {
                // @option
pane: String = 'popupPane'
                // `Map pane`
where the popup will be added.
                pane:
'popupPane',

                // @option
offset: Point = Point(0, 7)
                // The offset
of the popup position.
                offset: [0,
7],

                // @option
maxWidth: Number = 300
                // Max width
of the popup, in pixels.
                maxWidth: 300,

                // @option
```

```
minWidth: Number = 50
                // Min width
of the popup, in pixels.
                minWidth: 50,

                // @option
maxHeight: Number = null
                // If set,
creates a scrollable container
of the given height
                // inside a
popup if its content exceeds
it.
                // The
scrollable container can be
styled using the
                // `leaflet-
popup-scrolled` CSS class
selector.
                maxHeight:
null,

                // @option
autoPan: Boolean = true
                // Set it to
`false` if you don't want the
map to do panning animation
                // to fit the
opened popup.
                autoPan: true,

                // @option
autoPanPaddingTopLeft: Point =
null
                // The margin
between the popup and the top
left corner of the map
                // view after
autopanning was performed.

autoPanPaddingTopLeft: null,

                // @option
autoPanPaddingBottomRight:
Point = null
                // The margin
between the popup and the
```

```
bottom right corner of the map
                // view after
autopanning was performed.

autoPanPaddingBottomRight:
null,

                // @option
autoPanPadding: Point =
Point(5, 5)
                // Equivalent
of setting both top left and
bottom right autopan padding
to the same value.

autoPanPadding: [5, 5],

                // @option
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.
                keepInView:
false,

                // @option
closeButton: Boolean = true
                // Controls
the presence of a close button
in the popup.
                closeButton:
true,

                // @option
autoClose: Boolean = true
                // Set it to
`false` if you want to
override the default behavior
of
                // the popup
closing when another popup is
opened.
                autoClose:
true,
```

```
                // @option
closeOnEscapeKey: Boolean =
true
                // Set it to
`false` if you want to
override the default behavior
of
                // the ESC key
for closing of the popup.

closeOnEscapeKey: true,

                // @option
closeOnClick: Boolean = *
                // Set it if
you want to override the
default behavior of the popup
closing when user clicks
                // on the map.
Defaults to the map's
[`closePopupOnClick`](#map-
closepopuponclick) option.

                // @option
className: String = ''
                // A custom
CSS class name to assign to
the popup.
                className: ''
        },

        // @namespace Popup
        // @method openOn(map:
Map): this
        // Alternative to
`map.openPopup(popup)`.
        // Adds the popup to
the map and closes the
previous one.
        openOn: function (map)
{
                map =
arguments.length ? map :
this._source._map; //
experimental, not the part of
public api
```

```
                if
(!map.hasLayer(this) &&
map._popup &&
map._popup.options.autoClose)
{

map.removeLayer(map._popup);
                }
                map._popup =
this;

                return
DivOverlay.prototype.openOn.ca
ll(this, map);
        },

        onAdd: function (map)
{

DivOverlay.prototype.onAdd.cal
l(this, map);

                // @namespace
Map
                // @section
Popup events
                // @event
popupopen: PopupEvent
                // Fired when
a popup is opened in the map

map.fire('popupopen', {popup:
this});

                if
(this._source) {
                        //
@namespace Layer
                        //
@section Popup events
                        //
@event popupopen: PopupEvent
                        //
Fired when a popup bound to
this layer is opened

this._source.fire('popupopen',
```

```
{popup: this}, true);
                        // For
non-path layers, we toggle the
popup when clicking
                        //
again the layer, so prevent
the map to reopen it.
                        if (!
(this._source instanceof
Path)) {

this._source.on('preclick',
stopPropagation);
                        }
                }
        },

        onRemove: function
(map) {

DivOverlay.prototype.onRemove.
call(this, map);

                // @namespace
Map
                // @section
Popup events
                // @event
popupclose: PopupEvent
                // Fired when
a popup in the map is closed

map.fire('popupclose', {popup:
this});

                if
(this._source) {
                        //
@namespace Layer
                        //
@section Popup events
                        //
@event popupclose: PopupEvent
                        //
Fired when a popup bound to
this layer is closed
```

```
this._source.fire('popupclose'
, {popup: this}, true);
                        if (!
(this._source instanceof
Path)) {

this._source.off('preclick',
stopPropagation);
                        }
                }
        },

        getEvents: function ()
{
                var events =
DivOverlay.prototype.getEvents
.call(this);

                if
(this.options.closeOnClick !==
undefined ?
this.options.closeOnClick :
this._map.options.closePopupOn
Click) {

events.preclick = this.close;
                }

                if
(this.options.keepInView) {

events.moveend =
this._adjustPan;
                }

                return events;
        },

        _initLayout: function
() {
                var prefix =
'leaflet-popup',
                        container
= this._container =
create$1('div',
                        prefix
+ ' ' +
```

```
(this.options.className || '')
+
                          '
leaflet-zoom-animated');

                var wrapper =
this._wrapper =
create$1('div', prefix + '-
content-wrapper', container);

this._contentNode =
create$1('div', prefix + '-
content', wrapper);


disableClickPropagation(contai
ner);

disableScrollPropagation(this.
_contentNode);
                on(container,
'contextmenu',
stopPropagation);


this._tipContainer =
create$1('div', prefix + '-
tip-container', container);
                this._tip =
create$1('div', prefix + '-
tip', this._tipContainer);

                if
(this.options.closeButton) {
                        var
closeButton =
this._closeButton =
create$1('a', prefix + '-
close-button', container);

closeButton.setAttribute('role
', 'button'); // overrides the
implicit role=link of <a>
elements #7399

closeButton.setAttribute('aria
-label', 'Close popup');
```

```
closeButton.href = '#close';

closeButton.innerHTML = '<span
aria-hidden="true">&#215;
</span>';


on(closeButton, 'click',
function (ev) {

preventDefault(ev);

this.close();
                        },
this);
                }
        },

        _updateLayout:
function () {
                var container
= this._contentNode,
                    style =
container.style;

                style.width =
'';

style.whiteSpace = 'nowrap';

                var width =
container.offsetWidth;
                width =
Math.min(width,
this.options.maxWidth);
                width =
Math.max(width,
this.options.minWidth);

                style.width =
(width + 1) + 'px';

style.whiteSpace = '';

                style.height =
'';
```

```
                var height =
container.offsetHeight,
                    maxHeight
= this.options.maxHeight,

scrolledClass = 'leaflet-
popup-scrolled';

                if (maxHeight
&& height > maxHeight) {

style.height = maxHeight +
'px';

addClass(container,
scrolledClass);
                } else {

removeClass(container,
scrolledClass);
                }


this._containerWidth =
this._container.offsetWidth;
        },

        _animateZoom: function
(e) {
                var pos =
this._map._latLngToNewLayerPoi
nt(this._latlng, e.zoom,
e.center),
                    anchor =
this._getAnchor();

setPosition(this._container,
pos.add(anchor));
        },

        _adjustPan: function
(e) {
                if
(!this.options.autoPan) {
return; }
                if
```

```
(this._map._panAnim) {
this._map._panAnim.stop(); }

                var map =
this._map,

marginBottom =
parseInt(getStyle(this._contai
ner, 'marginBottom'), 10) ||
0,

containerHeight =
this._container.offsetHeight +
marginBottom,

containerWidth =
this._containerWidth,
                    layerPos =
new Point(this._containerLeft,
-containerHeight -
this._containerBottom);


layerPos._add(getPosition(this
._container));

                var
containerPos =
map.layerPointToContainerPoint
(layerPos),
                    padding =
toPoint(this.options.autoPanPa
dding),
                    paddingTL
=
toPoint(this.options.autoPanPa
ddingTopLeft || padding),
                    paddingBR
=
toPoint(this.options.autoPanPa
ddingBottomRight || padding),
                    size =
map.getSize(),
                    dx = 0,
                    dy = 0;

                if
```

```
(containerPos.x +
containerWidth + paddingBR.x >
size.x) { // right
                        dx =
containerPos.x +
containerWidth - size.x +
paddingBR.x;
                }
                if
(containerPos.x - dx -
paddingTL.x < 0) { // left
                        dx =
containerPos.x - paddingTL.x;
                }
                if
(containerPos.y +
containerHeight + paddingBR.y
> size.y) { // bottom
                        dy =
containerPos.y +
containerHeight - size.y +
paddingBR.y;
                }
                if
(containerPos.y - dy -
paddingTL.y < 0) { // top
                        dy =
containerPos.y - paddingTL.y;
                }

                // @namespace
Map
                // @section
Popup events
                // @event
autopanstart: Event
                // Fired when
the map starts autopanning
when opening a popup.
                if (dx || dy)
{
                        map

.fire('autopanstart')

.panBy([dx, dy], {animate: e
&& e.type === 'moveend'});
```

```
                }
        },

        _getAnchor: function
() {
                // Where
should we anchor the popup on
the source layer?
                return
toPoint(this._source &&
this._source._getPopupAnchor ?
this._source._getPopupAnchor()
: [0, 0]);
        }

  });

  // @namespace Popup
  // @factory
L.popup(options?: Popup
options, source?: Layer)
  // Instantiates a `Popup`
object given an optional
`options` object that
describes its appearance and
location and an optional
`source` object that is used
to tag the popup with a
reference to the Layer to
which it refers.
  // @alternative
  // @factory L.popup(latlng:
LatLng, options?: Popup
options)
  // Instantiates a `Popup`
object given `latlng` where
the popup will open and an
optional `options` object that
describes its appearance and
location.
  var popup = function
(options, source) {
        return new
Popup(options, source);
  };
```

```
  /* @namespace Map
   * @section Interaction
Options
   * @option
closePopupOnClick: Boolean =
true
   * Set it to `false` if you
don't want popups to close
when user clicks the map.
   */
  Map.mergeOptions({
        closePopupOnClick:
true
  });


  // @namespace Map
  // @section Methods for
Layers and Controls
  Map.include({
        // @method
openPopup(popup: Popup): this
        // Opens the specified
popup while closing the
previously opened (to make
sure only one is opened at one
time for usability).
        // @alternative
        // @method
openPopup(content:
String|HTMLElement, latlng:
LatLng, options?: Popup
options): this
        // Creates a popup
with the specified content and
options and opens it in the
given point on a map.
        openPopup: function
(popup, latlng, options) {

this._initOverlay(Popup,
popup, latlng, options)

.openOn(this);

                return this;
        },
```

```
        // @method
closePopup(popup?: Popup):
this
        // Closes the popup
previously opened with
[openPopup](#map-openpopup)
(or the given one).
        closePopup: function
(popup) {
                popup =
arguments.length ? popup :
this._popup;
                if (popup) {

popup.close();
                }
                return this;
        }
  });

  /*
   * @namespace Layer
   * @section Popup methods
example
   *
   * All layers share a set of
methods convenient for binding
popups to it.
   *
   * ```js
   * var layer =
L.Polygon(latlngs).bindPopup('
Hi There!').addTo(map);
   * layer.openPopup();
   * layer.closePopup();
   * ```
   *
   * Popups will also be
automatically opened when the
layer is clicked on and closed
when the layer is removed from
the map or another popup is
opened.
   */

  // @section Popup methods
```

```
  Layer.include({

        // @method
bindPopup(content:
String|HTMLElement|Function|Po
pup, options?: Popup options):
this
        // Binds a popup to
the layer with the passed
`content` and sets up the
        // necessary event
listeners. If a `Function` is
passed it will receive
        // the layer as the
first argument and should
return a `String` or
`HTMLElement`.
        bindPopup: function
(content, options) {
                this._popup =
this._initOverlay(Popup,
this._popup, content,
options);
                if
(!this._popupHandlersAdded) {

this.on({

click: this._openPopup,

keypress: this._onKeyPress,

remove: this.closePopup,

move: this._movePopup
                        });

this._popupHandlersAdded =
true;
                }

                return this;
        },

        // @method
unbindPopup(): this
        // Removes the popup
```

```
previously bound with
`bindPopup`.
        unbindPopup: function
() {
                if
(this._popup) {

this.off({

click: this._openPopup,

keypress: this._onKeyPress,

remove: this.closePopup,

move: this._movePopup
                        });

this._popupHandlersAdded =
false;

this._popup = null;
                }
                return this;
        },

        // @method
openPopup(latlng?: LatLng):
this
        // Opens the bound
popup at the specified
`latlng` or at the default
popup anchor if no `latlng` is
passed.
        openPopup: function
(latlng) {
                if
(this._popup &&
this._popup._prepareOpen(latln
g)) {
                        //
open the popup on the map

this._popup.openOn(this._map);
                }
                return this;
        },
```

```
        // @method
closePopup(): this
        // Closes the popup
bound to this layer if it is
open.
        closePopup: function
() {
                if
(this._popup) {

this._popup.close();
                }
                return this;
        },

        // @method
togglePopup(): this
        // Opens or closes the
popup bound to this layer
depending on its current
state.
        togglePopup: function
() {
                if
(this._popup) {

this._popup.toggle(this);
                }
                return this;
        },

        // @method
isPopupOpen(): boolean
        // Returns `true` if
the popup bound to this layer
is currently open.
        isPopupOpen: function
() {
                return
(this._popup ?
this._popup.isOpen() : false);
        },

        // @method
setPopupContent(content:
String|HTMLElement|Popup):
```

```
this
        // Sets the content of
the popup bound to this layer.
        setPopupContent:
function (content) {
                if
(this._popup) {

this._popup.setContent(content
);
                }
                return this;
        },

        // @method getPopup():
Popup
        // Returns the popup
bound to this layer.
        getPopup: function ()
{
                return
this._popup;
        },

        _openPopup: function
(e) {
                if
(!this._popup || !this._map) {

return;
                }
                // prevent map
click
                stop(e);

                var target =
e.layer || e.target;
                if
(this._popup._source ===
target && !(target instanceof
Path)) {
                        //
treat it like a marker and
figure out
                        // if
we should toggle it
open/closed
```

```
                        if
(this._map.hasLayer(this._popu
p)) {

this.closePopup();
                        } else
{

this.openPopup(e.latlng);
                        }

return;
                }

this._popup._source = target;

this.openPopup(e.latlng);
        },

        _movePopup: function
(e) {

this._popup.setLatLng(e.latlng
);
        },

        _onKeyPress: function
(e) {
                if
(e.originalEvent.keyCode ===
13) {

this._openPopup(e);
                }
        }
  });

  /*
   * @class Tooltip
   * @inherits DivOverlay
   * @aka L.Tooltip
   * Used to display small
texts on top of map layers.
   *
   * @example
   * If you want to just bind
a tooltip to marker:
```

```
   *
   * ```js
   * marker.bindTooltip("my
tooltip text").openTooltip();
   * ```
   * Path overlays like
polylines also have a
`bindTooltip` method.
   *
   * A tooltip can be also
standalone:
   *
   * ```js
   * var tooltip = L.tooltip()
   *     .setLatLng(latlng)
   *     .setContent('Hello
world!<br />This is a nice
tooltip.')
   *     .addTo(map);
   * ```
   * or
   * ```js
   * var tooltip =
L.tooltip(latlng, {content:
'Hello world!<br />This is a
nice tooltip.'})
   *     .addTo(map);
   * ```
   *
   *
   * Note about tooltip
offset. Leaflet takes two
options in consideration
   * for computing tooltip
offsetting:
   * - the `offset` Tooltip
option: it defaults to [0, 0],
and it's specific to one
tooltip.
   *   Add a positive x offset
to move the tooltip to the
right, and a positive y offset
to
   *   move it to the bottom.
Negatives will move to the
left and top.
   * - the `tooltipAnchor`
```

```
Icon option: this will only be
considered for Marker. You
   *   should adapt this value
if you use a custom icon.
   */


  // @namespace Tooltip
  var Tooltip =
DivOverlay.extend({

        // @section
        // @aka Tooltip
options
        options: {
                // @option
pane: String = 'tooltipPane'
                // `Map pane`
where the tooltip will be
added.
                pane:
'tooltipPane',

                // @option
offset: Point = Point(0, 0)
                // Optional
offset of the tooltip
position.
                offset: [0,
0],

                // @option
direction: String = 'auto'
                // Direction
where to open the tooltip.
Possible values are: `right`,
`left`,
                // `top`,
`bottom`, `center`, `auto`.
                // `auto` will
dynamically switch between
`right` and `left` according
to the tooltip
                // position on
the map.
                direction:
'auto',
```

```
                // @option
permanent: Boolean = false
                // Whether to
open the tooltip permanently
or only on mouseover.
                permanent:
false,

                // @option
sticky: Boolean = false
                // If true,
the tooltip will follow the
mouse instead of being fixed
at the feature center.
                sticky: false,

                // @option
opacity: Number = 0.9
                // Tooltip
container opacity.
                opacity: 0.9
        },

        onAdd: function (map)
{

DivOverlay.prototype.onAdd.cal
l(this, map);

this.setOpacity(this.options.o
pacity);

                // @namespace
Map
                // @section
Tooltip events
                // @event
tooltipopen: TooltipEvent
                // Fired when
a tooltip is opened in the
map.

map.fire('tooltipopen',
{tooltip: this});

                if
```

```
(this._source) {

this.addEventParent(this._sour
ce);

                        //
@namespace Layer
                        //
@section Tooltip events
                        //
@event tooltipopen:
TooltipEvent
                        //
Fired when a tooltip bound to
this layer is opened.

this._source.fire('tooltipopen
', {tooltip: this}, true);
                }
        },

        onRemove: function
(map) {

DivOverlay.prototype.onRemove.
call(this, map);

                // @namespace
Map
                // @section
Tooltip events
                // @event
tooltipclose: TooltipEvent
                // Fired when
a tooltip in the map is
closed.

map.fire('tooltipclose',
{tooltip: this});

                if
(this._source) {

this.removeEventParent(this._s
ource);

                        //
```

```
@namespace Layer
                        //
@section Tooltip events
                        //
@event tooltipclose:
TooltipEvent
                        //
Fired when a tooltip bound to
this layer is closed.

this._source.fire('tooltipclos
e', {tooltip: this}, true);
                }
        },

        getEvents: function ()
{
                var events =
DivOverlay.prototype.getEvents
.call(this);

                if
(!this.options.permanent) {

events.preclick = this.close;
                }

                return events;
        },

        _initLayout: function
() {
                var prefix =
'leaflet-tooltip',
                    className
= prefix + ' ' +
(this.options.className || '')
+ ' leaflet-zoom-' +
(this._zoomAnimated ?
'animated' : 'hide');


this._contentNode =
this._container =
create$1('div', className);
```

```
this._container.setAttribute('
role', 'tooltip');

this._container.setAttribute('
id', 'leaflet-tooltip-' +
stamp(this));
        },

        _updateLayout:
function () {},

        _adjustPan: function
() {},

        _setPosition: function
(pos) {
                var subX,
subY,
                    map =
this._map,
                    container
= this._container,

centerPoint =
map.latLngToContainerPoint(map
.getCenter()),

tooltipPoint =
map.layerPointToContainerPoint
(pos),
                    direction
= this.options.direction,

tooltipWidth =
container.offsetWidth,

tooltipHeight =
container.offsetHeight,
                    offset =
toPoint(this.options.offset),
                    anchor =
this._getAnchor();

                if (direction
=== 'top') {
                        subX =
tooltipWidth / 2;
```

```
                        subY =
tooltipHeight;
                } else if
(direction === 'bottom') {
                        subX =
tooltipWidth / 2;
                        subY =
0;
                } else if
(direction === 'center') {
                        subX =
tooltipWidth / 2;
                        subY =
tooltipHeight / 2;
                } else if
(direction === 'right') {
                        subX =
0;
                        subY =
tooltipHeight / 2;
                } else if
(direction === 'left') {
                        subX =
tooltipWidth;
                        subY =
tooltipHeight / 2;
                } else if
(tooltipPoint.x <
centerPoint.x) {

direction = 'right';
                        subX =
0;
                        subY =
tooltipHeight / 2;
                } else {

direction = 'left';
                        subX =
tooltipWidth + (offset.x +
anchor.x) * 2;
                        subY =
tooltipHeight / 2;
                }

                pos =
pos.subtract(toPoint(subX,
```

```
subY,
true)).add(offset).add(anchor)
;


removeClass(container,
'leaflet-tooltip-right');

removeClass(container,
'leaflet-tooltip-left');

removeClass(container,
'leaflet-tooltip-top');

removeClass(container,
'leaflet-tooltip-bottom');

addClass(container, 'leaflet-
tooltip-' + direction);

setPosition(container, pos);
        },

        _updatePosition:
function () {
                var pos =
this._map.latLngToLayerPoint(t
his._latlng);

this._setPosition(pos);
        },

        setOpacity: function
(opacity) {

this.options.opacity =
opacity;

                if
(this._container) {

setOpacity(this._container,
opacity);
                }
        },

        _animateZoom: function
```

```
(e) {
                var pos =
this._map._latLngToNewLayerPoi
nt(this._latlng, e.zoom,
e.center);

this._setPosition(pos);
        },

        _getAnchor: function
() {
                // Where
should we anchor the tooltip
on the source layer?
                return
toPoint(this._source &&
this._source._getTooltipAnchor
&& !this.options.sticky ?
this._source._getTooltipAnchor
() : [0, 0]);
        }

  });

  // @namespace Tooltip
  // @factory
L.tooltip(options?: Tooltip
options, source?: Layer)
  // Instantiates a `Tooltip`
object given an optional
`options` object that
describes its appearance and
location and an optional
`source` object that is used
to tag the tooltip with a
reference to the Layer to
which it refers.
  // @alternative
  // @factory
L.tooltip(latlng: LatLng,
options?: Tooltip options)
  // Instantiates a `Tooltip`
object given `latlng` where
the tooltip will open and an
optional `options` object that
describes its appearance and
location.
```

```
  var tooltip = function
(options, source) {
        return new
Tooltip(options, source);
  };

  // @namespace Map
  // @section Methods for
Layers and Controls
  Map.include({

        // @method
openTooltip(tooltip: Tooltip):
this
        // Opens the specified
tooltip.
        // @alternative
        // @method
openTooltip(content:
String|HTMLElement, latlng:
LatLng, options?: Tooltip
options): this
        // Creates a tooltip
with the specified content and
options and open it.
        openTooltip: function
(tooltip, latlng, options) {

this._initOverlay(Tooltip,
tooltip, latlng, options)

.openOn(this);

                return this;
        },

        // @method
closeTooltip(tooltip:
Tooltip): this
        // Closes the tooltip
given as parameter.
        closeTooltip: function
(tooltip) {

tooltip.close();
                return this;
        }
```

```
  });

  /*
   * @namespace Layer
   * @section Tooltip methods
example
   *
   * All layers share a set of
methods convenient for binding
tooltips to it.
   *
   * ```js
   * var layer =
L.Polygon(latlngs).bindTooltip
('Hi There!').addTo(map);
   * layer.openTooltip();
   * layer.closeTooltip();
   * ```
   */

  // @section Tooltip methods
  Layer.include({

        // @method
bindTooltip(content:
String|HTMLElement|Function|To
oltip, options?: Tooltip
options): this
        // Binds a tooltip to
the layer with the passed
`content` and sets up the
        // necessary event
listeners. If a `Function` is
passed it will receive
        // the layer as the
first argument and should
return a `String` or
`HTMLElement`.
        bindTooltip: function
(content, options) {

                if
(this._tooltip &&
this.isTooltipOpen()) {

this.unbindTooltip();
```

```
                }

                this._tooltip
= this._initOverlay(Tooltip,
this._tooltip, content,
options);

this._initTooltipInteractions(
);

                if
(this._tooltip.options.permane
nt && this._map &&
this._map.hasLayer(this)) {

this.openTooltip();
                }

                return this;
        },

        // @method
unbindTooltip(): this
        // Removes the tooltip
previously bound with
`bindTooltip`.
        unbindTooltip:
function () {
                if
(this._tooltip) {

this._initTooltipInteractions(
true);

this.closeTooltip();

this._tooltip = null;
                }
                return this;
        },


_initTooltipInteractions:
function (remove) {
                if (!remove &&
this._tooltipHandlersAdded) {
return; }
```

```
                var onOff =
remove ? 'off' : 'on',
                    events = {

remove: this.closeTooltip,
                        move:
this._moveTooltip
                    };
                if
(!this._tooltip.options.perman
ent) {

events.mouseover =
this._openTooltip;

events.mouseout =
this.closeTooltip;

events.click =
this._openTooltip;
                        if
(this._map) {

this._addFocusListeners();
                        } else
{

events.add =
this._addFocusListeners;
                        }
                } else {

events.add =
this._openTooltip;
                }
                if
(this._tooltip.options.sticky)
{

events.mousemove =
this._moveTooltip;
                }
                this[onOff]
(events);

this._tooltipHandlersAdded =
!remove;
```

```
        },

        // @method
openTooltip(latlng?: LatLng):
this
        // Opens the bound
tooltip at the specified
`latlng` or at the default
tooltip anchor if no `latlng`
is passed.
        openTooltip: function
(latlng) {
                if
(this._tooltip &&
this._tooltip._prepareOpen(lat
lng)) {
                        //
open the tooltip on the map

this._tooltip.openOn(this._map
);

                        if
(this.getElement) {

this._setAriaDescribedByOnLaye
r(this);
                        } else
if (this.eachLayer) {

this.eachLayer(this._setAriaDe
scribedByOnLayer, this);
                        }
                }
                return this;
        },

        // @method
closeTooltip(): this
        // Closes the tooltip
bound to this layer if it is
open.
        closeTooltip: function
() {
                if
(this._tooltip) {
                        return
```

```
this._tooltip.close();
                }
        },

        // @method
toggleTooltip(): this
        // Opens or closes the
tooltip bound to this layer
depending on its current
state.
        toggleTooltip:
function () {
                if
(this._tooltip) {

this._tooltip.toggle(this);
                }
                return this;
        },

        // @method
isTooltipOpen(): boolean
        // Returns `true` if
the tooltip bound to this
layer is currently open.
        isTooltipOpen:
function () {
                return
this._tooltip.isOpen();
        },

        // @method
setTooltipContent(content:
String|HTMLElement|Tooltip):
this
        // Sets the content of
the tooltip bound to this
layer.
        setTooltipContent:
function (content) {
                if
(this._tooltip) {

this._tooltip.setContent(conte
nt);
                }
                return this;
```

```
        },

        // @method
getTooltip(): Tooltip
        // Returns the tooltip
bound to this layer.
        getTooltip: function
() {
                return
this._tooltip;
        },

        _addFocusListeners:
function () {
                if
(this.getElement) {

this._addFocusListenersOnLayer
(this);
                } else if
(this.eachLayer) {

this.eachLayer(this._addFocusL
istenersOnLayer, this);
                }
        },


_addFocusListenersOnLayer:
function (layer) {

on(layer.getElement(),
'focus', function () {

this._tooltip._source = layer;

this.openTooltip();
                }, this);

on(layer.getElement(), 'blur',
this.closeTooltip, this);
        },


_setAriaDescribedByOnLayer:
function (layer) {
```

```
layer.getElement().setAttribut
e('aria-describedby',
this._tooltip._container.id);
        },


        _openTooltip: function
(e) {
                if
(!this._tooltip || !this._map
|| (this._map.dragging &&
this._map.dragging.moving()))
{

return;
                }

this._tooltip._source =
e.layer || e.target;


this.openTooltip(this._tooltip
.options.sticky ? e.latlng :
undefined);
        },

        _moveTooltip: function
(e) {
                var latlng =
e.latlng, containerPoint,
layerPoint;
                if
(this._tooltip.options.sticky
&& e.originalEvent) {

containerPoint =
this._map.mouseEventToContaine
rPoint(e.originalEvent);

layerPoint =
this._map.containerPointToLaye
rPoint(containerPoint);
                        latlng
=
this._map.layerPointToLatLng(l
ayerPoint);
                }
```

```
this._tooltip.setLatLng(latlng
);
        }
  });

  /*
   * @class DivIcon
   * @aka L.DivIcon
   * @inherits Icon
   *
   * Represents a lightweight
icon for markers that uses a
simple `<div>`
   * element instead of an
image. Inherits from `Icon`
but ignores the `iconUrl` and
shadow options.
   *
   * @example
   * ```js
   * var myIcon =
L.divIcon({className: 'my-div-
icon'});
   * // you can set .my-div-
icon styles in CSS
   *
   * L.marker([50.505, 30.57],
{icon: myIcon}).addTo(map);
   * ```
   *
   * By default, it has a
'leaflet-div-icon' CSS class
and is styled as a little
white square with a shadow.
   */

  var DivIcon = Icon.extend({
        options: {
                // @section
                // @aka
DivIcon options
                iconSize: [12,
12], // also can be set
through CSS

                // iconAnchor:
```

```
(Point),
                //
popupAnchor: (Point),

                // @option
html: String|HTMLElement = ''
                // Custom HTML
code to put inside the div
element, empty by default.
Alternatively,
                // an instance
of `HTMLElement`.
                html: false,

                // @option
bgPos: Point = [0, 0]
                // Optional
relative position of the
background, in pixels
                bgPos: null,

                className:
'leaflet-div-icon'
        },

        createIcon: function
(oldIcon) {
                var div =
(oldIcon && oldIcon.tagName
=== 'DIV') ? oldIcon :
document.createElement('div'),
                    options =
this.options;

                if
(options.html instanceof
Element) {

empty(div);

div.appendChild(options.html);
                } else {

div.innerHTML = options.html
!== false ? options.html : '';
                }
```

```
                if
(options.bgPos) {
                        var
bgPos =
toPoint(options.bgPos);

div.style.backgroundPosition =
(-bgPos.x) + 'px ' + (-
bgPos.y) + 'px';
                }

this._setIconStyles(div,
'icon');

                return div;
        },

        createShadow: function
() {
                return null;
        }
  });

  // @factory
L.divIcon(options: DivIcon
options)
  // Creates a `DivIcon`
instance with the given
options.
  function divIcon(options) {
        return new
DivIcon(options);
  }

  Icon.Default = IconDefault;

  /*
   * @class GridLayer
   * @inherits Layer
   * @aka L.GridLayer
   *
   * Generic class for
handling a tiled grid of HTML
elements. This is the base
class for all tile layers and
replaces `TileLayer.Canvas`.
   * GridLayer can be extended
```

```
to create a tiled grid of HTML
elements like `<canvas>`,
`<img>` or `<div>`. GridLayer
will handle creating and
animating these DOM elements
for you.
   *
   *
   * @section Synchronous
usage
   * @example
   *
   * To create a custom layer,
extend GridLayer and implement
the `createTile()` method,
which will be passed a `Point`
object with the `x`, `y`, and
`z` (zoom level) coordinates
to draw your tile.
   *
   * ```js
   * var CanvasLayer =
L.GridLayer.extend({
   *     createTile:
function(coords){
   *         // create a
<canvas> element for drawing
   *         var tile =
L.DomUtil.create('canvas',
'leaflet-tile');
   *
   *         // setup tile
width and height according to
the options
   *         var size =
this.getTileSize();
   *         tile.width =
size.x;
   *         tile.height =
size.y;
   *
   *         // get a canvas
context and draw something on
it using coords.x, coords.y
and coords.z
   *         var ctx =
tile.getContext('2d');
```

```
   *
   *         // return the
tile so it can be rendered on
screen
   *         return tile;
   *     }
   * });
   * ```
   *
   * @section Asynchronous
usage
   * @example
   *
   * Tile creation can also be
asynchronous, this is useful
when using a third-party
drawing library. Once the tile
is finished drawing it can be
passed to the `done()`
callback.
   *
   * ```js
   * var CanvasLayer =
L.GridLayer.extend({
   *     createTile:
function(coords, done){
   *         var error;
   *
   *         // create a
<canvas> element for drawing
   *         var tile =
L.DomUtil.create('canvas',
'leaflet-tile');
   *
   *         // setup tile
width and height according to
the options
   *         var size =
this.getTileSize();
   *         tile.width =
size.x;
   *         tile.height =
size.y;
   *
   *         // draw something
asynchronously and pass the
tile to the done() callback
```

```
   *
setTimeout(function() {
   *              done(error,
tile);
   *          }, 1000);
   *
   *          return tile;
   *      }
   * });
   * ```
   *
   * @section
   */


  var GridLayer =
Layer.extend({

        // @section
        // @aka GridLayer
options
        options: {
                // @option
tileSize: Number|Point = 256
                // Width and
height of tiles in the grid.
Use a number if width and
height are equal, or
`L.point(width, height)`
otherwise.
                tileSize: 256,

                // @option
opacity: Number = 1.0
                // Opacity of
the tiles. Can be used in the
`createTile()` function.
                opacity: 1,

                // @option
updateWhenIdle: Boolean =
(depends)
                // Load new
tiles only when panning ends.
                // `true` by
default on mobile browsers, in
order to avoid too many
```

```
requests and keep smooth
navigation.
                // `false`
otherwise in order to display
new tiles _during_ panning,
since it is easy to pan
outside the
                //
[`keepBuffer`](#gridlayer-
keepbuffer) option in desktop
browsers.

updateWhenIdle:
Browser.mobile,

                // @option
updateWhenZooming: Boolean =
true
                // By default,
a smooth zoom animation
(during a [touch zoom](#map-
touchzoom) or a [`flyTo()`]
(#map-flyto)) will update grid
layers every integer zoom
level. Setting this option to
`false` will update the grid
layer only when the smooth
animation ends.

updateWhenZooming: true,

                // @option
updateInterval: Number = 200
                // Tiles will
not update more than once
every `updateInterval`
milliseconds when panning.

updateInterval: 200,

                // @option
zIndex: Number = 1
                // The
explicit zIndex of the tile
layer.
                zIndex: 1,
```

```
                // @option
bounds: LatLngBounds =
undefined
                // If set,
tiles will only be loaded
inside the set `LatLngBounds`.
                bounds: null,

                // @option
minZoom: Number = 0
                // The minimum
zoom level down to which this
layer will be displayed
(inclusive).
                minZoom: 0,

                // @option
maxZoom: Number = undefined
                // The maximum
zoom level up to which this
layer will be displayed
(inclusive).
                maxZoom:
undefined,

                // @option
maxNativeZoom: Number =
undefined
                // Maximum
zoom number the tile source
has available. If it is
specified,
                // the tiles
on all zoom levels higher than
`maxNativeZoom` will be loaded
                // from
`maxNativeZoom` level and
auto-scaled.
                maxNativeZoom:
undefined,

                // @option
minNativeZoom: Number =
undefined
                // Minimum
zoom number the tile source
has available. If it is
```

```
specified,
                // the tiles
on all zoom levels lower than
`minNativeZoom` will be loaded
                // from
`minNativeZoom` level and
auto-scaled.
                minNativeZoom:
undefined,

                // @option
noWrap: Boolean = false
                // Whether the
layer is wrapped around the
antimeridian. If `true`, the
                // GridLayer
will only be displayed once at
low zoom levels. Has no
                // effect when
the [map CRS](#map-crs)
doesn't wrap around. Can be
used
                // in
combination with [`bounds`]
(#gridlayer-bounds) to prevent
requesting
                // tiles
outside the CRS limits.
                noWrap: false,

                // @option
pane: String = 'tilePane'
                // `Map pane`
where the grid layer will be
added.
                pane:
'tilePane',

                // @option
className: String = ''
                // A custom
class name to assign to the
tile layer. Empty by default.
                className: '',

                // @option
keepBuffer: Number = 2
```

```
                // When
panning the map, keep this
many rows and columns of tiles
before unloading them.
                keepBuffer: 2
        },

        initialize: function
(options) {

setOptions(this, options);
        },

        onAdd: function () {

this._initContainer();

                this._levels =
{};
                this._tiles =
{};


this._resetView(); // implicit
_update() call
        },

        beforeAdd: function
(map) {

map._addZoomLimit(this);
        },

        onRemove: function
(map) {

this._removeAllTiles();

remove(this._container);

map._removeZoomLimit(this);

this._container = null;
                this._tileZoom
= undefined;
        },
```

```
        // @method
bringToFront: this
        // Brings the tile
layer to the top of all tile
layers.
        bringToFront: function
() {
                if (this._map)
{

toFront(this._container);

this._setAutoZIndex(Math.max);
                }
                return this;
        },

        // @method
bringToBack: this
        // Brings the tile
layer to the bottom of all
tile layers.
        bringToBack: function
() {
                if (this._map)
{

toBack(this._container);

this._setAutoZIndex(Math.min);
                }
                return this;
        },

        // @method
getContainer: HTMLElement
        // Returns the HTML
element that contains the
tiles for this layer.
        getContainer: function
() {
                return
this._container;
        },

        // @method
setOpacity(opacity: Number):
```

```
this
        // Changes the
[opacity](#gridlayer-opacity)
of the grid layer.
        setOpacity: function
(opacity) {

this.options.opacity =
opacity;

this._updateOpacity();
                return this;
        },

        // @method
setZIndex(zIndex: Number):
this
        // Changes the
[zIndex](#gridlayer-zindex) of
the grid layer.
        setZIndex: function
(zIndex) {

this.options.zIndex = zIndex;

this._updateZIndex();

                return this;
        },

        // @method isLoading:
Boolean
        // Returns `true` if
any tile in the grid layer has
not finished loading.
        isLoading: function ()
{
                return
this._loading;
        },

        // @method redraw:
this
        // Causes the layer to
clear all the tiles and
request them again.
        redraw: function () {
```

```
                if (this._map)
{

this._removeAllTiles();
                        var
tileZoom =
this._clampZoom(this._map.getZ
oom());
                        if
(tileZoom !== this._tileZoom)
{

this._tileZoom = tileZoom;

this._updateLevels();
                        }

this._update();
                }
                return this;
        },

        getEvents: function ()
{
                var events = {

viewprereset:
this._invalidateAll,

viewreset: this._resetView,
                        zoom:
this._resetView,

moveend: this._onMoveEnd
                };

                if
(!this.options.updateWhenIdle)
{
                        //
update tiles on move, but not
more often than once per given
interval
                        if
(!this._onMove) {

this._onMove =
```

```
throttle(this._onMoveEnd,
this.options.updateInterval,
this);
                        }


events.move = this._onMove;
                }

                if
(this._zoomAnimated) {

events.zoomanim =
this._animateZoom;
                }

                return events;
        },

        // @section Extension
methods
        // Layers extending
`GridLayer` shall reimplement
the following method.
        // @method
createTile(coords: Object,
done?: Function): HTMLElement
        // Called only
internally, must be overridden
by classes extending
`GridLayer`.
        // Returns the
`HTMLElement` corresponding to
the given `coords`. If the
`done` callback
        // is specified, it
must be called when the tile
has finished loading and
drawing.
        createTile: function
() {
                return
document.createElement('div');
        },

        // @section
        // @method
```

```
getTileSize: Point
        // Normalizes the
[tileSize option](#gridlayer-
tilesize) into a point. Used
by the `createTile()` method.
        getTileSize: function
() {
                var s =
this.options.tileSize;
                return s
instanceof Point ? s : new
Point(s, s);
        },

        _updateZIndex:
function () {
                if
(this._container &&
this.options.zIndex !==
undefined &&
this.options.zIndex !== null)
{

this._container.style.zIndex =
this.options.zIndex;
                }
        },

        _setAutoZIndex:
function (compare) {
                // go through
all other layers of the same
pane, set zIndex to max + 1
(front) or min - 1 (back)

                var layers =
this.getPane().children,
                        edgeZIndex
= -compare(-Infinity,
Infinity); // -Infinity for
max, Infinity for min

                for (var i =
0, len = layers.length,
zIndex; i < len; i++) {

                        zIndex
```

```
= layers[i].style.zIndex;

                        if
(layers[i] !== this._container
&& zIndex) {

edgeZIndex =
compare(edgeZIndex, +zIndex);
                        }
                }

                if
(isFinite(edgeZIndex)) {

this.options.zIndex =
edgeZIndex + compare(-1, 1);

this._updateZIndex();
                }
        },

        _updateOpacity:
function () {
                if
(!this._map) { return; }

                // IE doesn't
inherit filter opacity
properly, so we're forced to
set it on tiles
                if
(Browser.ielt9) { return; }


setOpacity(this._container,
this.options.opacity);

                var now = +new
Date(),
                    nextFrame
= false,
                    willPrune
= false;

                for (var key
in this._tiles) {
                        var
```

```
tile = this._tiles[key];
                        if
(!tile.current ||
!tile.loaded) { continue; }

                        var
fade = Math.min(1, (now -
tile.loaded) / 200);


setOpacity(tile.el, fade);
                        if
(fade < 1) {

nextFrame = true;
                        } else
{

if (tile.active) {

willPrune = true;

} else {

this._onOpaqueTile(tile);

}

tile.active = true;
                        }
                }

                if (willPrune
&& !this._noPrune) {
this._pruneTiles(); }

                if (nextFrame)
{

cancelAnimFrame(this._fadeFram
e);

this._fadeFrame =
requestAnimFrame(this._updateO
pacity, this);
                }
        },
```

```
        _onOpaqueTile:
falseFn,

        _initContainer:
function () {
                if
(this._container) { return; }


this._container =
create$1('div', 'leaflet-layer
' + (this.options.className ||
''));

this._updateZIndex();

                if
(this.options.opacity < 1) {

this._updateOpacity();
                }


this.getPane().appendChild(thi
s._container);
        },

        _updateLevels:
function () {

                var zoom =
this._tileZoom,
                    maxZoom =
this.options.maxZoom;

                if (zoom ===
undefined) { return undefined;
}

                for (var z in
this._levels) {
                        z =
Number(z);
                        if
(this._levels[z].el.children.l
ength || z === zoom) {
```

```
this._levels[z].el.style.zInde
x = maxZoom - Math.abs(zoom -
z);

this._onUpdateLevel(z);
                        } else
{

remove(this._levels[z].el);

this._removeTilesAtZoom(z);

this._onRemoveLevel(z);

delete this._levels[z];
                        }
                }

                var level =
this._levels[zoom],
                    map =
this._map;

                if (!level) {
                        level
= this._levels[zoom] = {};


level.el = create$1('div',
'leaflet-tile-container
leaflet-zoom-animated',
this._container);

level.el.style.zIndex =
maxZoom;


level.origin =
map.project(map.unproject(map.
getPixelOrigin()),
zoom).round();

level.zoom = zoom;


this._setZoomTransform(level,
```

```
map.getCenter(),
map.getZoom());

                        //
force the browser to consider
the newly added element for
transition

falseFn(level.el.offsetWidth);


this._onCreateLevel(level);
                }

                this._level =
level;

                return level;
        },

        _onUpdateLevel:
falseFn,

        _onRemoveLevel:
falseFn,

        _onCreateLevel:
falseFn,

        _pruneTiles: function
() {
                if
(!this._map) {

return;
                }

                var key, tile;

                var zoom =
this._map.getZoom();
                if (zoom >
this.options.maxZoom ||
                        zoom <
this.options.minZoom) {

this._removeAllTiles();
```

```
return;
                }

                for (key in
this._tiles) {
                        tile =
this._tiles[key];

tile.retain = tile.current;
                }

                for (key in
this._tiles) {
                        tile =
this._tiles[key];
                        if
(tile.current && !tile.active)
{

var coords = tile.coords;

if
(!this._retainParent(coords.x,
coords.y, coords.z, coords.z -
5)) {

this._retainChildren(coords.x,
coords.y, coords.z, coords.z +
2);

}
                        }
                }

                for (key in
this._tiles) {
                        if
(!this._tiles[key].retain) {

this._removeTile(key);
                        }
                }
        },

        _removeTilesAtZoom:
function (zoom) {
```

```
                for (var key
in this._tiles) {
                        if
(this._tiles[key].coords.z !==
zoom) {

continue;
                        }

this._removeTile(key);
                }
        },

        _removeAllTiles:
function () {
                for (var key
in this._tiles) {

this._removeTile(key);
                }
        },

        _invalidateAll:
function () {
                for (var z in
this._levels) {

remove(this._levels[z].el);

this._onRemoveLevel(Number(z))
;
                        delete
this._levels[z];
                }

this._removeAllTiles();

                this._tileZoom
= undefined;
        },

        _retainParent:
function (x, y, z, minZoom) {
                var x2 =
Math.floor(x / 2),
                    y2 =
Math.floor(y / 2),
```

```
                    z2 = z -
1,
                    coords2 =
new Point(+x2, +y2);
                coords2.z =
+z2;

                var key =
this._tileCoordsToKey(coords2)
,
                    tile =
this._tiles[key];

                if (tile &&
tile.active) {

tile.retain = true;
                        return
true;

                } else if
(tile && tile.loaded) {

tile.retain = true;
                }

                if (z2 >
minZoom) {
                        return
this._retainParent(x2, y2, z2,
minZoom);
                }

                return false;
        },

        _retainChildren:
function (x, y, z, maxZoom) {

                for (var i = 2
* x; i < 2 * x + 2; i++) {
                        for
(var j = 2 * y; j < 2 * y + 2;
j++) {


var coords = new Point(i, j);
```

```
coords.z = z + 1;


var key =
this._tileCoordsToKey(coords),

tile = this._tiles[key];


if (tile && tile.active) {

tile.retain = true;

continue;


} else if (tile &&
tile.loaded) {

tile.retain = true;

}


if (z + 1 < maxZoom) {

this._retainChildren(i, j, z +
1, maxZoom);

}
                        }
                    }
        },

        _resetView: function
(e) {
                var animating
= e && (e.pinch || e.flyTo);

this._setView(this._map.getCen
ter(), this._map.getZoom(),
animating, animating);
        },

        _animateZoom: function
(e) {
```

```
this._setView(e.center,
e.zoom, true, e.noUpdate);
        },

        _clampZoom: function
(zoom) {
                var options =
this.options;

                if (undefined
!== options.minNativeZoom &&
zoom < options.minNativeZoom)
{
                        return
options.minNativeZoom;
                }

                if (undefined
!== options.maxNativeZoom &&
options.maxNativeZoom < zoom)
{
                        return
options.maxNativeZoom;
                }

                return zoom;
        },

        _setView: function
(center, zoom, noPrune,
noUpdate) {
                var tileZoom =
Math.round(zoom);
                if
((this.options.maxZoom !==
undefined && tileZoom >
this.options.maxZoom) ||

(this.options.minZoom !==
undefined && tileZoom <
this.options.minZoom)) {

tileZoom = undefined;
                } else {

tileZoom =
```

```
this._clampZoom(tileZoom);
                }

                var
tileZoomChanged =
this.options.updateWhenZooming
&& (tileZoom !==
this._tileZoom);

                if (!noUpdate
|| tileZoomChanged) {


this._tileZoom = tileZoom;

                        if
(this._abortLoading) {

this._abortLoading();
                        }


this._updateLevels();

this._resetGrid();

                        if
(tileZoom !== undefined) {

this._update(center);
                        }

                        if
(!noPrune) {

this._pruneTiles();
                        }

                        //
Flag to prevent _updateOpacity
from pruning tiles during
                        // a
zoom anim or a pinch gesture

this._noPrune = !!noPrune;
                }
```

```
this._setZoomTransforms(center
, zoom);
        },

        _setZoomTransforms:
function (center, zoom) {
                for (var i in
this._levels) {

this._setZoomTransform(this._l
evels[i], center, zoom);
                }
        },

        _setZoomTransform:
function (level, center, zoom)
{
                var scale =
this._map.getZoomScale(zoom,
level.zoom),
                    translate
=
level.origin.multiplyBy(scale)

.subtract(this._map._getNewPix
elOrigin(center,
zoom)).round();

                if
(Browser.any3d) {

setTransform(level.el,
translate, scale);
                } else {

setPosition(level.el,
translate);
                }
        },

        _resetGrid: function
() {
                var map =
this._map,
                    crs =
map.options.crs,
```

```
                    tileSize =
this._tileSize =
this.getTileSize(),
                    tileZoom =
this._tileZoom;

                var bounds =
this._map.getPixelWorldBounds(
this._tileZoom);
                if (bounds) {

this._globalTileRange =
this._pxBoundsToTileRange(boun
ds);
                }

                this._wrapX =
crs.wrapLng &&
!this.options.noWrap && [

Math.floor(map.project([0,
crs.wrapLng[0]], tileZoom).x /
tileSize.x),

Math.ceil(map.project([0,
crs.wrapLng[1]], tileZoom).x /
tileSize.y)
                ];
                this._wrapY =
crs.wrapLat &&
!this.options.noWrap && [

Math.floor(map.project([crs.wr
apLat[0], 0], tileZoom).y /
tileSize.x),

Math.ceil(map.project([crs.wra
pLat[1], 0], tileZoom).y /
tileSize.y)
                ];
        },

        _onMoveEnd: function
() {
                if (!this._map
|| this._map._animatingZoom) {
return; }
```

```
this._update();
        },

        _getTiledPixelBounds:
function (center) {
                var map =
this._map,
                    mapZoom =
map._animatingZoom ?
Math.max(map._animateToZoom,
map.getZoom()) :
map.getZoom(),
                    scale =
map.getZoomScale(mapZoom,
this._tileZoom),

pixelCenter =
map.project(center,
this._tileZoom).floor(),
                    halfSize =
map.getSize().divideBy(scale *
2);

                return new
Bounds(pixelCenter.subtract(ha
lfSize),
pixelCenter.add(halfSize));
        },

        // Private method to
load tiles in the grid's
active zoom level according to
map bounds
        _update: function
(center) {
                var map =
this._map;
                if (!map) {
return; }
                var zoom =
this._clampZoom(map.getZoom())
;

                if (center ===
undefined) { center =
```

```
map.getCenter(); }
                if
(this._tileZoom === undefined)
{ return; }     // if out of
minzoom/maxzoom

                var
pixelBounds =
this._getTiledPixelBounds(cent
er),
                    tileRange
=
this._pxBoundsToTileRange(pixe
lBounds),
                    tileCenter
= tileRange.getCenter(),
                    queue =
[],
                    margin =
this.options.keepBuffer,

noPruneRange = new
Bounds(tileRange.getBottomLeft
().subtract([margin, -
margin]),

tileRange.getTopRight().add([m
argin, -margin]));

                // Sanity
check: panic if the tile range
contains Infinity somewhere.
                if (!
(isFinite(tileRange.min.x) &&

isFinite(tileRange.min.y) &&

isFinite(tileRange.max.x) &&

isFinite(tileRange.max.y))) {
throw new Error('Attempted to
load an infinite number of
tiles'); }

                for (var key
in this._tiles) {
                        var c
```

```
= this._tiles[key].coords;
                        if
(c.z !== this._tileZoom ||
!noPruneRange.contains(new
Point(c.x, c.y))) {

this._tiles[key].current =
false;
                        }
                }

                // _update
just loads more tiles. If the
tile zoom level differs too
much
                // from the
map's, let _setView reset
levels and prune old tiles.
                if
(Math.abs(zoom -
this._tileZoom) > 1) {
this._setView(center, zoom);
return; }

                // create a
queue of coordinates to load
tiles from
                for (var j =
tileRange.min.y; j <=
tileRange.max.y; j++) {
                        for
(var i = tileRange.min.x; i <=
tileRange.max.x; i++) {

var coords = new Point(i, j);

coords.z = this._tileZoom;


if
(!this._isValidTile(coords)) {
continue; }


var tile =
this._tiles[this._tileCoordsTo
Key(coords)];
```

```
if (tile) {

tile.current = true;

} else {

queue.push(coords);

}
                        }
                }

                // sort tile
queue to load tiles in order
of their distance to center

queue.sort(function (a, b) {
                        return
a.distanceTo(tileCenter) -
b.distanceTo(tileCenter);
                });

                if
(queue.length !== 0) {
                        // if
it's the first batch of tiles
to load
                        if
(!this._loading) {

this._loading = true;

// @event loading: Event

// Fired when the grid layer
starts loading tiles.

this.fire('loading');
                        }

                        //
create DOM fragment to append
tiles in one batch
                        var
fragment =
document.createDocumentFragmen
```

```
t();

                        for (i
= 0; i < queue.length; i++) {

this._addTile(queue[i],
fragment);
                        }


this._level.el.appendChild(fra
gment);
                }
        },

        _isValidTile: function
(coords) {
                var crs =
this._map.options.crs;

                if
(!crs.infinite) {
                        //
don't load tile if it's out of
bounds and not wrapped
                        var
bounds =
this._globalTileRange;
                        if
((!crs.wrapLng && (coords.x <
bounds.min.x || coords.x >
bounds.max.x)) ||

(!crs.wrapLat && (coords.y <
bounds.min.y || coords.y >
bounds.max.y))) { return
false; }
                }

                if
(!this.options.bounds) {
return true; }

                // don't load
tile if it doesn't intersect
the bounds in options
                var tileBounds
```

```
=
this._tileCoordsToBounds(coord
s);
                return
toLatLngBounds(this.options.bo
unds).overlaps(tileBounds);
        },

        _keyToBounds: function
(key) {
                return
this._tileCoordsToBounds(this.
_keyToTileCoords(key));
        },

        _tileCoordsToNwSe:
function (coords) {
                var map =
this._map,
                    tileSize =
this.getTileSize(),
                    nwPoint =
coords.scaleBy(tileSize),
                    sePoint =
nwPoint.add(tileSize),
                    nw =
map.unproject(nwPoint,
coords.z),
                    se =
map.unproject(sePoint,
coords.z);
                return [nw,
se];
        },

        // converts tile
coordinates to its
geographical bounds
        _tileCoordsToBounds:
function (coords) {
                var bp =
this._tileCoordsToNwSe(coords)
,
                    bounds =
new LatLngBounds(bp[0],
bp[1]);
```

```
                if
(!this.options.noWrap) {
                        bounds
=
this._map.wrapLatLngBounds(bou
nds);
                }
                return bounds;
        },
        // converts tile
coordinates to key for the
tile cache
        _tileCoordsToKey:
function (coords) {
                return
coords.x + ':' + coords.y +
':' + coords.z;
        },

        // converts tile cache
key to coordinates
        _keyToTileCoords:
function (key) {
                var k =
key.split(':'),
                    coords =
new Point(+k[0], +k[1]);
                coords.z =
+k[2];
                return coords;
        },

        _removeTile: function
(key) {
                var tile =
this._tiles[key];
                if (!tile) {
return; }


remove(tile.el);

                delete
this._tiles[key];

                // @event
tileunload: TileEvent
```

```
                // Fired when
a tile is removed (e.g. when a
tile goes off the screen).

this.fire('tileunload', {
                        tile:
tile.el,

coords:
this._keyToTileCoords(key)
                });
        },

        _initTile: function
(tile) {
                addClass(tile,
'leaflet-tile');

                var tileSize =
this.getTileSize();

tile.style.width = tileSize.x
+ 'px';

tile.style.height = tileSize.y
+ 'px';


tile.onselectstart = falseFn;

tile.onmousemove = falseFn;

                // update
opacity on tiles in IE7-8
because of filter inheritance
problems
                if
(Browser.ielt9 &&
this.options.opacity < 1) {

setOpacity(tile,
this.options.opacity);
                }
        },

        _addTile: function
(coords, container) {
```

```
                var tilePos =
this._getTilePos(coords),
                    key =
this._tileCoordsToKey(coords);

                var tile =
this.createTile(this._wrapCoor
ds(coords),
bind(this._tileReady, this,
coords));


this._initTile(tile);

                // if
createTile is defined with a
second argument ("done"
callback),
                // we know
that tile is async and will be
ready later; otherwise
                if
(this.createTile.length < 2) {
                    //
mark tile as ready, but delay
one frame for opacity
animation to happen

requestAnimFrame(bind(this._ti
leReady, this, coords, null,
tile));
                }


setPosition(tile, tilePos);

                // save tile
in cache

this._tiles[key] = {
                        el:
tile,

coords: coords,

current: true
                };
```

```
container.appendChild(tile);
                // @event
tileloadstart: TileEvent
                // Fired when
a tile is requested and starts
loading.

this.fire('tileloadstart', {
                        tile:
tile,

coords: coords
                });
        },

        _tileReady: function
(coords, err, tile) {
                if (err) {
                        //
@event tileerror:
TileErrorEvent
                        //
Fired when there is an error
loading a tile.

this.fire('tileerror', {

error: err,

tile: tile,

coords: coords
                        });
                }

                var key =
this._tileCoordsToKey(coords);

                tile =
this._tiles[key];
                if (!tile) {
return; }

                tile.loaded =
+new Date();
```

```
                if
(this._map._fadeAnimated) {

setOpacity(tile.el, 0);

cancelAnimFrame(this._fadeFram
e);

this._fadeFrame =
requestAnimFrame(this._updateO
pacity, this);
                } else {

tile.active = true;

this._pruneTiles();
                }

                if (!err) {

addClass(tile.el, 'leaflet-
tile-loaded');

                        //
@event tileload: TileEvent
                        //
Fired when a tile loads.

this.fire('tileload', {

tile: tile.el,

coords: coords
                        });
                }

                if
(this._noTilesToLoad()) {

this._loading = false;
                        //
@event load: Event
                        //
Fired when the grid layer
loaded all visible tiles.

this.fire('load');
```

```
                        if
(Browser.ielt9 ||
!this._map._fadeAnimated) {

requestAnimFrame(this._pruneTi
les, this);
                        } else
{

// Wait a bit more than 0.2
secs (the duration of the tile
fade-in)

// to trigger a pruning.

setTimeout(bind(this._pruneTil
es, this), 250);
                        }
                }
        },

        _getTilePos: function
(coords) {
                return
coords.scaleBy(this.getTileSiz
e()).subtract(this._level.orig
in);
        },

        _wrapCoords: function
(coords) {
                var newCoords
= new Point(

this._wrapX ?
wrapNum(coords.x, this._wrapX)
: coords.x,

this._wrapY ?
wrapNum(coords.y, this._wrapY)
: coords.y);
                newCoords.z =
coords.z;
                return
newCoords;
        },
```

```
        _pxBoundsToTileRange:
function (bounds) {
                var tileSize =
this.getTileSize();
                return new
Bounds(

bounds.min.unscaleBy(tileSize)
.floor(),

bounds.max.unscaleBy(tileSize)
.ceil().subtract([1, 1]));
        },

        _noTilesToLoad:
function () {
                for (var key
in this._tiles) {
                        if
(!this._tiles[key].loaded) {
return false; }
                }
                return true;
        }
  });

  // @factory
L.gridLayer(options?:
GridLayer options)
  // Creates a new instance of
GridLayer with the supplied
options.
  function gridLayer(options)
{
        return new
GridLayer(options);
  }

  /*
   * @class TileLayer
   * @inherits GridLayer
   * @aka L.TileLayer
   * Used to load and display
tile layers on the map. Note
that most tile servers require
attribution, which you can set
```

```
under `Layer`. Extends
`GridLayer`.
   *
   * @example
   *
   * ```js
   *
L.tileLayer('https://tile.open
streetmap.org/{z}/{x}/{y}.png?
{foo}', {foo: 'bar',
attribution: '&copy; <a
href="https://www.openstreetma
p.org/copyright">OpenStreetMap
</a>
contributors'}).addTo(map);
   * ```
   *
   * @section URL template
   * @example
   *
   * A string of the following
form:
   *
   * ```
   *
'https://{s}.somedomain.com/bl
abla/{z}/{x}/{y}{r}.png'
   * ```
   *
   * `{s}` means one of the
available subdomains (used
sequentially to help with
browser parallel requests per
domain limitation; subdomain
values are specified in
options; `a`, `b` or `c` by
default, can be omitted),
`{z}` — zoom level, `{x}` and
`{y}` — tile coordinates.
`{r}` can be used to add
"&commat;2x" to the URL to
load retina tiles.
   *
   * You can use custom keys
in the template, which will be
[evaluated](#util-template)
from TileLayer options, like
```

```
this:
	*
	* ```
	*
L.tileLayer('https://{s}.somed
omain.com/{foo}/{z}/{x}/{y}.pn
g', {foo: 'bar'});
	* ```
	*/


  var TileLayer =
GridLayer.extend({

		// @section
		// @aka TileLayer
options
		options: {
			// @option
minZoom: Number = 0
			// The minimum
zoom level down to which this
layer will be displayed
(inclusive).
			minZoom: 0,

			// @option
maxZoom: Number = 18
			// The maximum
zoom level up to which this
layer will be displayed
(inclusive).
			maxZoom: 18,

			// @option
subdomains: String|String[] =
'abc'
			// Subdomains
of the tile service. Can be
passed in the form of one
string (where each letter is a
subdomain name) or an array of
strings.
			subdomains:
'abc',

			// @option
```

```
errorTileUrl: String = ''
                // URL to the
tile image to show in place of
the tile that failed to load.
                errorTileUrl:
'',

                // @option
zoomOffset: Number = 0
                // The zoom
number used in tile URLs will
be offset with this value.
                zoomOffset: 0,

                // @option
tms: Boolean = false
                // If `true`,
inverses Y axis numbering for
tiles (turn this on for [TMS]
(https://en.wikipedia.org/wiki
/Tile_Map_Service) services).
                tms: false,

                // @option
zoomReverse: Boolean = false
                // If set to
true, the zoom number used in
tile URLs will be reversed
(`maxZoom - zoom` instead of
`zoom`)
                zoomReverse:
false,

                // @option
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.
                detectRetina:
false,

                // @option
crossOrigin: Boolean|String =
```

```
false
                // Whether the
crossOrigin attribute will be
added to the tiles.
                // If a String
is provided, all tiles will
have their crossOrigin
attribute set to the String
provided. This is needed if
you want to access tile pixel
data.
                // Refer to
[CORS Settings]
(https://developer.mozilla.org
/en-
US/docs/Web/HTML/CORS_settings
_attributes) for valid String
values.
                crossOrigin:
false,

                // @option
referrerPolicy: Boolean|String
= false
                // Whether the
referrerPolicy attribute will
be added to the tiles.
                // If a String
is provided, all tiles will
have their referrerPolicy
attribute set to the String
provided.
                // This may be
needed if your map's rendering
context has a strict default
but your tile provider expects
a valid referrer
                // (e.g. to
validate an API token).
                // Refer to
[HTMLImageElement.referrerPoli
cy]
(https://developer.mozilla.org
/en-
US/docs/Web/API/HTMLImageEleme
nt/referrerPolicy) for valid
String values.
```

```
referrerPolicy: false
        },

        initialize: function
(url, options) {

                this._url =
url;

                options =
setOptions(this, options);

                // detecting
retina displays, adjusting
tileSize and zoom levels
                if
(options.detectRetina &&
Browser.retina &&
options.maxZoom > 0) {


options.tileSize =
Math.floor(options.tileSize /
2);

                        if
(!options.zoomReverse) {

options.zoomOffset++;

options.maxZoom =
Math.max(options.minZoom,
options.maxZoom - 1);
                        } else
{

options.zoomOffset--;

options.minZoom =
Math.min(options.maxZoom,
options.minZoom + 1);
                        }


options.minZoom = Math.max(0,
options.minZoom);
```

```
                } else if
(!options.zoomReverse) {
                        //
make sure maxZoom is gte
minZoom

options.maxZoom =
Math.max(options.minZoom,
options.maxZoom);
                } else {
                        //
make sure minZoom is lte
maxZoom

options.minZoom =
Math.min(options.maxZoom,
options.minZoom);
                }

                if (typeof
options.subdomains ===
'string') {

options.subdomains =
options.subdomains.split('');
                }


this.on('tileunload',
this._onTileRemove);
        },

        // @method setUrl(url:
String, noRedraw?: Boolean):
this
        // Updates the layer's
URL template and redraws it
(unless `noRedraw` is set to
`true`).
        // If the URL does not
change, the layer will not be
redrawn unless
        // the noRedraw
parameter is set to false.
        setUrl: function (url,
noRedraw) {
                if (this._url
```

```
=== url && noRedraw ===
undefined) {

noRedraw = true;
                }

                this._url =
url;

                if (!noRedraw)
{

this.redraw();
                }
                return this;
        },

        // @method
createTile(coords: Object,
done?: Function): HTMLElement
        // Called only
internally, overrides
GridLayer's [`createTile()`]
(#gridlayer-createtile)
        // to return an
`<img>` HTML element with the
appropriate image URL given
`coords`. The `done`
        // callback is called
when the tile has been loaded.
        createTile: function
(coords, done) {
                var tile =
document.createElement('img');

                on(tile,
'load', bind(this._tileOnLoad,
this, done, tile));
                on(tile,
'error',
bind(this._tileOnError, this,
done, tile));

                if
(this.options.crossOrigin ||
this.options.crossOrigin ===
'') {
```

```
tile.crossOrigin =
this.options.crossOrigin ===
true ? '' :
this.options.crossOrigin;
                }

                // for this
new option we follow the
documented behavior
                // more
closely by only setting the
property when string
                if (typeof
this.options.referrerPolicy
=== 'string') {

tile.referrerPolicy =
this.options.referrerPolicy;
                }

                // The alt
attribute is set to the empty
string,
                // allowing
screen readers to ignore the
decorative image tiles.
                //
https://www.w3.org/WAI/tutoria
ls/images/decorative/
                //
https://www.w3.org/TR/html-
aria/#el-img-empty-alt
                tile.alt = '';

                tile.src =
this.getTileUrl(coords);

                return tile;
        },

        // @section Extension
methods
        // @uninheritable
        // Layers extending
`TileLayer` might reimplement
the following method.
```

```
        // @method
getTileUrl(coords: Object):
String
        // Called only
internally, returns the URL
for a tile given its
coordinates.
        // Classes extending
`TileLayer` can override this
function to provide custom
tile URL naming schemes.
        getTileUrl: function
(coords) {
                var data = {
                        r:
Browser.retina ? '@2x' : '',
                        s:
this._getSubdomain(coords),
                        x:
coords.x,
                        y:
coords.y,
                        z:
this._getZoomForUrl()
                };
                if (this._map
&&
!this._map.options.crs.infinit
e) {
                        var
invertedY =
this._globalTileRange.max.y -
coords.y;
                        if
(this.options.tms) {

data['y'] = invertedY;
                        }

data['-y'] = invertedY;
                }

                return
template(this._url,
extend(data, this.options));
        },
```

```
        _tileOnLoad: function
(done, tile) {
                // For
https://github.com/Leaflet/Lea
flet/issues/3332
                if
(Browser.ielt9) {

setTimeout(bind(done, this,
null, tile), 0);
                } else {

done(null, tile);
                }
        },

        _tileOnError: function
(done, tile, e) {
                var errorUrl =
this.options.errorTileUrl;
                if (errorUrl
&& tile.getAttribute('src')
!== errorUrl) {

tile.src = errorUrl;
                }
                done(e, tile);
        },

        _onTileRemove:
function (e) {
                e.tile.onload
= null;
        },

        _getZoomForUrl:
function () {
                var zoom =
this._tileZoom,
                maxZoom =
this.options.maxZoom,
                zoomReverse =
this.options.zoomReverse,
                zoomOffset =
this.options.zoomOffset;

                if
```

```
(zoomReverse) {
                        zoom =
maxZoom - zoom;
                }

                return zoom +
zoomOffset;
        },

        _getSubdomain:
function (tilePoint) {
                var index =
Math.abs(tilePoint.x +
tilePoint.y) %
this.options.subdomains.length
;
                return
this.options.subdomains[index]
;
        },

        // stops loading all
tiles in the background layer
        _abortLoading:
function () {
                var i, tile;
                for (i in
this._tiles) {
                        if
(this._tiles[i].coords.z !==
this._tileZoom) {

tile = this._tiles[i].el;


tile.onload = falseFn;

tile.onerror = falseFn;


if (!tile.complete) {

tile.src = emptyImageUrl;

var coords =
this._tiles[i].coords;
```

```
remove(tile);

delete this._tiles[i];

// @event tileabort: TileEvent

// Fired when a tile was
loading but is now not wanted.

this.fire('tileabort', {

tile: tile,

coords: coords

});

}
                        }
                }
        },

        _removeTile: function
(key) {
                var tile =
this._tiles[key];
                if (!tile) {
return; }

                // Cancels any
pending http requests
associated with the tile

tile.el.setAttribute('src',
emptyImageUrl);

                return
GridLayer.prototype._removeTil
e.call(this, key);
        },

        _tileReady: function
(coords, err, tile) {
                if (!this._map
|| (tile &&
tile.getAttribute('src') ===
emptyImageUrl)) {
```

```
return;
                }

                return
GridLayer.prototype._tileReady
.call(this, coords, err,
tile);
        }
  });


  // @factory
L.tilelayer(urlTemplate:
String, options?: TileLayer
options)
  // Instantiates a tile layer
object given a `URL template`
and optionally an options
object.

  function tileLayer(url,
options) {
        return new
TileLayer(url, options);
  }

  /*
   * @class TileLayer.WMS
   * @inherits TileLayer
   * @aka L.TileLayer.WMS
   * Used to display [WMS]
(https://en.wikipedia.org/wiki
/Web_Map_Service) services as
tile layers on the map.
Extends `TileLayer`.
   *
   * @example
   *
   * ```js
   * var nexrad =
L.tileLayer.wms("http://mesone
t.agron.iastate.edu/cgi-
bin/wms/nexrad/n0r.cgi", {
   *    layers: 'nexrad-n0r-
900913',
   *    format: 'image/png',
```

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

  var TileLayerWMS =
TileLayer.extend({

        // @section
        // @aka TileLayer.WMS
options
        // If any custom
options not documented here
are used, they will be sent to
the
        // WMS server as extra
parameters in each request
URL. This can be useful for
        // [non-standard
vendor WMS parameters]
(https://docs.geoserver.org/st
able/en/user/services/wms/vend
or.html).
        defaultWmsParams: {
                service:
'WMS',
                request:
'GetMap',

                // @option
layers: String = ''
                // **
(required)** Comma-separated
list of WMS layers to show.
                layers: '',

                // @option
styles: String = ''
                // Comma-
separated list of WMS styles.
                styles: '',

                // @option
format: String = 'image/jpeg'
                // WMS image
```

```
format (use `'image/png'` for
layers with transparency).
                format:
'image/jpeg',

                // @option
transparent: Boolean = false
                // If `true`,
the WMS service will return
images with transparency.
                transparent:
false,

                // @option
version: String = '1.1.1'
                // Version of
the WMS service to use
                version:
'1.1.1'
        },

        options: {
                // @option
crs: CRS = null
                // Coordinate
Reference System to use for
the WMS requests, defaults to
                // map CRS.
Don't change this if you're
not sure what it means.
                crs: null,

                // @option
uppercase: Boolean = false
                // If `true`,
WMS request parameter keys
will be uppercase.
                uppercase:
false
        },

        initialize: function
(url, options) {

                this._url =
url;
```

```
                var wmsParams
= extend({},
this.defaultWmsParams);

                // all keys
that are not TileLayer options
go to WMS params
                for (var i in
options) {
                        if (!
(i in this.options)) {

wmsParams[i] = options[i];
                        }
                }

                options =
setOptions(this, options);

                var realRetina
= options.detectRetina &&
Browser.retina ? 2 : 1;
                var tileSize =
this.getTileSize();

wmsParams.width = tileSize.x *
realRetina;

wmsParams.height = tileSize.y
* realRetina;

                this.wmsParams
= wmsParams;
        },

        onAdd: function (map)
{

                this._crs =
this.options.crs ||
map.options.crs;

this._wmsVersion =
parseFloat(this.wmsParams.vers
ion);

                var
```

```
projectionKey =
this._wmsVersion >= 1.3 ?
'crs' : 'srs';

this.wmsParams[projectionKey]
= this._crs.code;


TileLayer.prototype.onAdd.call
(this, map);
        },

        getTileUrl: function
(coords) {

                var tileBounds
=
this._tileCoordsToNwSe(coords)
,
                    crs =
this._crs,
                    bounds =
toBounds(crs.project(tileBound
s[0]),
crs.project(tileBounds[1])),
                    min =
bounds.min,
                    max =
bounds.max,
                    bbox =
(this._wmsVersion >= 1.3 &&
this._crs === EPSG4326 ?
                    [min.y,
min.x, max.y, max.x] :
                    [min.x,
min.y, max.x,
max.y]).join(','),
                    url =
TileLayer.prototype.getTileUrl
.call(this, coords);
                return url +

getParamString(this.wmsParams,
url, this.options.uppercase) +

(this.options.uppercase ?
'&BBOX=' : '&bbox=') + bbox;
```

```
        },

        // @method
setParams(params: Object,
noRedraw?: Boolean): this
        // Merges an object
with the new parameters and
re-requests tiles on the
current screen (unless
`noRedraw` was set to true).
        setParams: function
(params, noRedraw) {


extend(this.wmsParams,
params);

                if (!noRedraw)
{

this.redraw();
                }

                return this;
        }
  });


  // @factory
L.tileLayer.wms(baseUrl:
String, options: TileLayer.WMS
options)
  // Instantiates a WMS tile
layer object given a base URL
of the WMS service and a WMS
parameters/options object.
  function tileLayerWMS(url,
options) {
        return new
TileLayerWMS(url, options);
  }

  TileLayer.WMS =
TileLayerWMS;
  tileLayer.wms =
tileLayerWMS;
```

```
  /*
   * @class Renderer
   * @inherits Layer
   * @aka L.Renderer
   *
   * Base class for vector
renderer implementations
(`SVG`, `Canvas`). Handles the
   * DOM container of the
renderer, its bounds, and its
zoom animation.
   *
   * A `Renderer` works as an
implicit layer group for all
`Path`s - the renderer
   * itself can be added or
removed to the map. All paths
use a renderer, which can
   * be implicit (the map will
decide the type of renderer
and use it automatically)
   * or explicit (using the
[`renderer`](#path-renderer)
option of the path).
   *
   * Do not use this class
directly, use `SVG` and
`Canvas` instead.
   *
   * @event update: Event
   * Fired when the renderer
updates its bounds, center and
zoom, for example when
   * its map has moved
   */

  var Renderer =
Layer.extend({

        // @section
        // @aka Renderer
options
        options: {
                // @option
padding: Number = 0.1
                // How much to
extend the clip area around
```

```
the map view (relative to its
size)
                // e.g. 0.1
would be 10% of map view in
each direction
                padding: 0.1
        },

        initialize: function
(options) {

setOptions(this, options);
                stamp(this);
                this._layers =
this._layers || {};
        },

        onAdd: function () {
                if
(!this._container) {

this._initContainer(); //
defined by renderer
implementations

                        if
(this._zoomAnimated) {

addClass(this._container,
'leaflet-zoom-animated');
                        }
                }


this.getPane().appendChild(thi
s._container);

this._update();

this.on('update',
this._updatePaths, this);
        },

        onRemove: function ()
{

this.off('update',
```

```
this._updatePaths, this);

this._destroyContainer();
        },

        getEvents: function ()
{
                var events = {

viewreset: this._reset,
                        zoom:
this._onZoom,

moveend: this._update,

zoomend: this._onZoomEnd
                };
                if
(this._zoomAnimated) {

events.zoomanim =
this._onAnimZoom;
                }
                return events;
        },

        _onAnimZoom: function
(ev) {

this._updateTransform(ev.cente
r, ev.zoom);
        },

        _onZoom: function () {

this._updateTransform(this._ma
p.getCenter(),
this._map.getZoom());
        },

        _updateTransform:
function (center, zoom) {
                var scale =
this._map.getZoomScale(zoom,
this._zoom),
                    viewHalf =
this._map.getSize().multiplyBy
```

```
(0.5 + this.options.padding),

currentCenterPoint =
this._map.project(this._center
, zoom),


topLeftOffset =
viewHalf.multiplyBy(-
scale).add(currentCenterPoint)

.subtract(this._map._getNewPix
elOrigin(center, zoom));

                if
(Browser.any3d) {

setTransform(this._container,
topLeftOffset, scale);
                } else {

setPosition(this._container,
topLeftOffset);
                }
        },

        _reset: function () {

this._update();

this._updateTransform(this._ce
nter, this._zoom);

                for (var id in
this._layers) {

this._layers[id]._reset();
                }
        },

        _onZoomEnd: function
() {
                for (var id in
this._layers) {

this._layers[id]._project();
                }
```

```
        },

        _updatePaths: function
() {
                for (var id in
this._layers) {

this._layers[id]._update();
                }
        },

        _update: function () {
                // Update
pixel bounds of renderer
container (for
positioning/sizing/clipping
later)
                // Subclasses
are responsible of firing the
'update' event.
                var p =
this.options.padding,
                    size =
this._map.getSize(),
                    min =
this._map.containerPointToLaye
rPoint(size.multiplyBy(-
p)).round();

                this._bounds =
new Bounds(min,
min.add(size.multiplyBy(1 + p
* 2)).round());

                this._center =
this._map.getCenter();
                this._zoom =
this._map.getZoom();
        }
  });

  /*
   * @class Canvas
   * @inherits Renderer
   * @aka L.Canvas
   *
   * Allows vector layers to
```

```
be displayed with [`<canvas>`]
(https://developer.mozilla.org
/docs/Web/API/Canvas_API).
   * Inherits `Renderer`.
   *
   * Due to [technical
limitations]
(https://caniuse.com/canvas),
Canvas is not
   * available in all web
browsers, notably IE8, and
overlapping geometries might
   * not display properly in
some edge cases.
   *
   * @example
   *
   * Use Canvas by default for
all paths in the map:
   *
   * ```js
   * var map = L.map('map', {
   *    renderer: L.canvas()
   * });
   * ```
   *
   * Use a Canvas renderer
with extra padding for
specific vector geometries:
   *
   * ```js
   * var map = L.map('map');
   * var myRenderer =
L.canvas({ padding: 0.5 });
   * var line = L.polyline(
coordinates, { renderer:
myRenderer } );
   * var circle = L.circle(
center, { renderer: myRenderer
} );
   * ```
   */

  var Canvas =
Renderer.extend({

        // @section
```

```
        // @aka Canvas options
        options: {
                // @option
tolerance: Number = 0
                // How much to
extend the click tolerance
around a path/object on the
map.
                tolerance: 0
        },

        getEvents: function ()
{
                var events =
Renderer.prototype.getEvents.c
all(this);

events.viewprereset =
this._onViewPreReset;
                return events;
        },

        _onViewPreReset:
function () {
                // Set a flag
so that a
viewprereset+moveend+viewreset
only updates&redraws once

this._postponeUpdatePaths =
true;
        },

        onAdd: function () {

Renderer.prototype.onAdd.call(
this);

                // Redraw
vectors since canvas is
cleared upon removal,
                // in case of
removing the renderer itself
from the map.
                this._draw();
        },
```

```
        _initContainer:
function () {
                var container
= this._container =
document.createElement('canvas
');

                on(container,
'mousemove',
this._onMouseMove, this);
                on(container,
'click dblclick mousedown
mouseup contextmenu',
this._onClick, this);
                on(container,
'mouseout',
this._handleMouseOut, this);

container['_leaflet_disable_ev
ents'] = true;

                this._ctx =
container.getContext('2d');
        },

        _destroyContainer:
function () {

cancelAnimFrame(this._redrawRe
quest);
                delete
this._ctx;

remove(this._container);

off(this._container);
                delete
this._container;
        },

        _updatePaths: function
() {
                if
(this._postponeUpdatePaths) {
return; }

                var layer;
```

```
this._redrawBounds = null;
                for (var id in
this._layers) {
                        layer
= this._layers[id];

layer._update();
                }

this._redraw();
        },

        _update: function () {
                if
(this._map._animatingZoom &&
this._bounds) { return; }


Renderer.prototype._update.cal
l(this);

                var b =
this._bounds,
                    container
= this._container,
                    size =
b.getSize(),
                    m =
Browser.retina ? 2 : 1;


setPosition(container, b.min);

                // set canvas
size (also clearing it); use
double size on retina

container.width = m * size.x;

container.height = m * size.y;

container.style.width = size.x
+ 'px';

container.style.height =
size.y + 'px';
```

```
                if
(Browser.retina) {

this._ctx.scale(2, 2);
                }

                // translate
so we use the same path
coordinates after canvas
element moves

this._ctx.translate(-b.min.x,
-b.min.y);

                // Tell paths
to redraw themselves

this.fire('update');
        },

        _reset: function () {

Renderer.prototype._reset.call
(this);

                if
(this._postponeUpdatePaths) {

this._postponeUpdatePaths =
false;

this._updatePaths();
                }
        },

        _initPath: function
(layer) {

this._updateDashArray(layer);

this._layers[stamp(layer)] =
layer;

                var order =
layer._order = {
                        layer:
```

```
layer,
                        prev:
this._drawLast,
                        next:
null
                };
                if
(this._drawLast) {
this._drawLast.next = order; }
                this._drawLast
= order;

this._drawFirst =
this._drawFirst ||
this._drawLast;
        },

        _addPath: function
(layer) {

this._requestRedraw(layer);
        },

        _removePath: function
(layer) {
                var order =
layer._order;
                var next =
order.next;
                var prev =
order.prev;

                if (next) {

next.prev = prev;
                } else {

this._drawLast = prev;
                }
                if (prev) {

prev.next = next;
                } else {

this._drawFirst = next;
                }
```

```
                delete
layer._order;

                delete
this._layers[stamp(layer)];


this._requestRedraw(layer);
        },

        _updatePath: function
(layer) {
                // Redraw the
union of the layer's old pixel
                // bounds and
the new pixel bounds.

this._extendRedrawBounds(layer
);

layer._project();

layer._update();
                // The redraw
will extend the redraw bounds
                // with the
new pixel bounds.

this._requestRedraw(layer);
        },

        _updateStyle: function
(layer) {

this._updateDashArray(layer);

this._requestRedraw(layer);
        },

        _updateDashArray:
function (layer) {
                if (typeof
layer.options.dashArray ===
'string') {
                        var
parts =
layer.options.dashArray.split(
```

```
/[, ]+/),

dashArray = [],

dashValue,
                                i;
                            for (i
= 0; i < parts.length; i++) {

dashValue = Number(parts[i]);

// Ignore dash array
containing invalid lengths

if (isNaN(dashValue)) {
return; }

dashArray.push(dashValue);
                            }

layer.options._dashArray =
dashArray;
                } else {

layer.options._dashArray =
layer.options.dashArray;
                }
        },

        _requestRedraw:
function (layer) {
                if
(!this._map) { return; }


this._extendRedrawBounds(layer
);

this._redrawRequest =
this._redrawRequest ||
requestAnimFrame(this._redraw,
this);
        },

        _extendRedrawBounds:
function (layer) {
                if
```

```
(layer._pxBounds) {
                        var
padding =
(layer.options.weight || 0) +
1;

this._redrawBounds =
this._redrawBounds || new
Bounds();

this._redrawBounds.extend(laye
r._pxBounds.min.subtract([padd
ing, padding]));

this._redrawBounds.extend(laye
r._pxBounds.max.add([padding,
padding]));
                }
        },

        _redraw: function () {

this._redrawRequest = null;

                if
(this._redrawBounds) {

this._redrawBounds.min._floor(
);

this._redrawBounds.max._ceil()
;
                }

                this._clear();
// clear layers in redraw
bounds
                this._draw();
// draw layers


this._redrawBounds = null;
        },

        _clear: function () {
                var bounds =
this._redrawBounds;
```

```
                if (bounds) {
                        var
size = bounds.getSize();

this._ctx.clearRect(bounds.min
.x, bounds.min.y, size.x,
size.y);
                } else {

this._ctx.save();

this._ctx.setTransform(1, 0,
0, 1, 0, 0);

this._ctx.clearRect(0, 0,
this._container.width,
this._container.height);

this._ctx.restore();
                }
        },

        _draw: function () {
                var layer,
bounds = this._redrawBounds;

this._ctx.save();
                if (bounds) {
                        var
size = bounds.getSize();

this._ctx.beginPath();

this._ctx.rect(bounds.min.x,
bounds.min.y, size.x, size.y);

this._ctx.clip();
                }

                this._drawing
= true;

                for (var order
= this._drawFirst; order;
order = order.next) {
                        layer
= order.layer;
```

```
                        if
(!bounds || (layer._pxBounds
&&
layer._pxBounds.intersects(bou
nds))) {

layer._updatePath();
                        }
                }

                this._drawing
= false;


this._ctx.restore();  //
Restore state before clipping.
        },

        _updatePoly: function
(layer, closed) {
                if
(!this._drawing) { return; }

                var i, j,
len2, p,
                    parts =
layer._parts,
                    len =
parts.length,
                    ctx =
this._ctx;

                if (!len) {
return; }


ctx.beginPath();

                for (i = 0; i
< len; i++) {
                        for (j
= 0, len2 = parts[i].length; j
< len2; j++) {

p = parts[i][j];

ctx[j ? 'lineTo' : 'moveTo']
```

```
(p.x, p.y);
                        }
                        if
(closed) {

ctx.closePath();
                        }
                }


this._fillStroke(ctx, layer);

                // TODO
optimization: 1 fill/stroke
for all features with equal
style instead of 1 for each
feature
        },

        _updateCircle:
function (layer) {

                if
(!this._drawing ||
layer._empty()) { return; }

                var p =
layer._point,
                    ctx =
this._ctx,
                    r =
Math.max(Math.round(layer._rad
ius), 1),
                    s =
(Math.max(Math.round(layer._ra
diusY), 1) || r) / r;

                if (s !== 1) {

ctx.save();

ctx.scale(1, s);
                }


ctx.beginPath();
                ctx.arc(p.x,
```

```
p.y / s, r, 0, Math.PI * 2,
false);

                if (s !== 1) {

ctx.restore();
                }


this._fillStroke(ctx, layer);
        },

        _fillStroke: function
(ctx, layer) {
                var options =
layer.options;

                if
(options.fill) {

ctx.globalAlpha =
options.fillOpacity;

ctx.fillStyle =
options.fillColor ||
options.color;

ctx.fill(options.fillRule ||
'evenodd');
                }

                if
(options.stroke &&
options.weight !== 0) {
                        if
(ctx.setLineDash) {

ctx.setLineDash(layer.options
&& layer.options._dashArray ||
[]);
                        }

ctx.globalAlpha =
options.opacity;

ctx.lineWidth =
options.weight;
```

```
ctx.strokeStyle =
options.color;

ctx.lineCap = options.lineCap;

ctx.lineJoin =
options.lineJoin;

ctx.stroke();
                }
        },

        // Canvas obviously
doesn't have mouse events for
individual drawn objects,
        // so we emulate that
by calculating what's under
the mouse on mousemove/click
manually

        _onClick: function (e)
{
                var point =
this._map.mouseEventToLayerPoi
nt(e), layer, clickedLayer;

                for (var order
= this._drawFirst; order;
order = order.next) {
                        layer
= order.layer;
                        if
(layer.options.interactive &&
layer._containsPoint(point)) {

if (!(e.type === 'click' ||
e.type === 'preclick') ||
!this._map._draggableMoved(lay
er)) {

clickedLayer = layer;

}
                        }
                }
```

```
this._fireEvent(clickedLayer ?
[clickedLayer] : false, e);
        },

        _onMouseMove: function
(e) {
                if (!this._map
|| this._map.dragging.moving()
|| this._map._animatingZoom) {
return; }

                var point =
this._map.mouseEventToLayerPoi
nt(e);

this._handleMouseHover(e,
point);
        },


        _handleMouseOut:
function (e) {
                var layer =
this._hoveredLayer;
                if (layer) {
                        // if
we're leaving the layer, fire
mouseout

removeClass(this._container,
'leaflet-interactive');

this._fireEvent([layer], e,
'mouseout');

this._hoveredLayer = null;

this._mouseHoverThrottled =
false;
                }
        },

        _handleMouseHover:
function (e, point) {
                if
(this._mouseHoverThrottled) {
```

```
return;
                }

                var layer,
candidateHoveredLayer;

                for (var order
= this._drawFirst; order;
order = order.next) {
                        layer
= order.layer;
                        if
(layer.options.interactive &&
layer._containsPoint(point)) {

candidateHoveredLayer = layer;
                        }
                }

                if
(candidateHoveredLayer !==
this._hoveredLayer) {

this._handleMouseOut(e);

                        if
(candidateHoveredLayer) {

addClass(this._container,
'leaflet-interactive'); //
change cursor

this._fireEvent([candidateHove
redLayer], e, 'mouseover');

this._hoveredLayer =
candidateHoveredLayer;
                        }
                }


this._fireEvent(this._hoveredL
ayer ? [this._hoveredLayer] :
false, e);


this._mouseHoverThrottled =
```

```
true;

setTimeout(bind(function () {

this._mouseHoverThrottled =
false;
                }, this), 32);
        },

        _fireEvent: function
(layers, e, type) {

this._map._fireDOMEvent(e,
type || e.type, layers);
        },

        _bringToFront:
function (layer) {
                var order =
layer._order;

                if (!order) {
return; }

                var next =
order.next;
                var prev =
order.prev;

                if (next) {

next.prev = prev;
                } else {
                        //
Already last

return;
                }
                if (prev) {

prev.next = next;
                } else if
(next) {
                        //
Update first entry unless this
is the
                        //
```

```
single entry

this._drawFirst = next;
                }

                order.prev =
this._drawLast;

this._drawLast.next = order;

                order.next =
null;
                this._drawLast
= order;


this._requestRedraw(layer);
        },

        _bringToBack: function
(layer) {
                var order =
layer._order;

                if (!order) {
return; }

                var next =
order.next;
                var prev =
order.prev;

                if (prev) {

prev.next = next;
                } else {
                        //
Already first

return;
                }
                if (next) {

next.prev = prev;
                } else if
(prev) {
                        //
```

```
Update last entry unless this
is the
                          //
single entry

this._drawLast = prev;
                }

                order.prev =
null;

                order.next =
this._drawFirst;

this._drawFirst.prev = order;

this._drawFirst = order;


this._requestRedraw(layer);
        }
  });

  // @factory
L.canvas(options?: Renderer
options)
  // Creates a Canvas renderer
with the given options.
  function canvas(options) {
        return Browser.canvas
? new Canvas(options) : null;
  }

  /*
   * Thanks to Dmitry
Baranovsky and his Raphael
library for inspiration!
   */


  var vmlCreate = (function ()
{
        try {

document.namespaces.add('lvml'
, 'urn:schemas-microsoft-
com:vml');
```

```
                return
function (name) {
                        return
document.createElement('<lvml:
' + name + ' class="lvml">');
                };
        } catch (e) {
                // Do not
return fn from catch block so
`e` can be garbage collected
                // See
https://github.com/Leaflet/Lea
flet/pull/7279
        }
        return function (name)
{
                return
document.createElement('<' +
name + ' xmlns="urn:schemas-
microsoft.com:vml"
class="lvml">');
        };
  })();


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

  // mixin to redefine some
SVG methods to handle VML
syntax which is similar but
with some differences
  var vmlMixin = {

        _initContainer:
function () {

this._container =
```

```
create$1('div', 'leaflet-vml-
container');
        },

        _update: function () {
                if
(this._map._animatingZoom) {
return; }

Renderer.prototype._update.cal
l(this);

this.fire('update');
        },

        _initPath: function
(layer) {
                var container
= layer._container =
vmlCreate('shape');


addClass(container, 'leaflet-
vml-shape ' +
(this.options.className ||
''));


container.coordsize = '1 1';

                layer._path =
vmlCreate('path');

container.appendChild(layer._p
ath);


this._updateStyle(layer);

this._layers[stamp(layer)] =
layer;
        },

        _addPath: function
(layer) {
                var container
= layer._container;
```

```
this._container.appendChild(co
ntainer);

                if
(layer.options.interactive) {

layer.addInteractiveTarget(con
tainer);
                }
        },

        _removePath: function
(layer) {
                var container
= layer._container;

remove(container);

layer.removeInteractiveTarget(
container);
                delete
this._layers[stamp(layer)];
        },

        _updateStyle: function
(layer) {
                var stroke =
layer._stroke,
                    fill =
layer._fill,
                    options =
layer.options,
                    container
= layer._container;


container.stroked =
!!options.stroke;

container.filled =
!!options.fill;

                if
(options.stroke) {
                        if
(!stroke) {
```

```
stroke = layer._stroke =
vmlCreate('stroke');
                        }

container.appendChild(stroke);

stroke.weight = options.weight
+ 'px';

stroke.color = options.color;

stroke.opacity =
options.opacity;

                        if
(options.dashArray) {

stroke.dashStyle =
isArray(options.dashArray) ?

options.dashArray.join(' ') :

options.dashArray.replace(/(
*, *)/g, ' ');
                        } else
{

stroke.dashStyle = '';
                        }

stroke.endcap =
options.lineCap.replace('butt'
, 'flat');

stroke.joinstyle =
options.lineJoin;

                } else if
(stroke) {

container.removeChild(stroke);

layer._stroke = null;
                }

                if
```

```
(options.fill) {
                        if
(!fill) {

fill = layer._fill =
vmlCreate('fill');
                        }

container.appendChild(fill);

fill.color = options.fillColor
|| options.color;

fill.opacity =
options.fillOpacity;

                } else if
(fill) {

container.removeChild(fill);

layer._fill = null;
                }
        },

        _updateCircle:
function (layer) {
                var p =
layer._point.round(),
                    r =
Math.round(layer._radius),
                    r2 =
Math.round(layer._radiusY ||
r);


this._setPath(layer,
layer._empty() ? 'M0 0' :
                        'AL '
+ p.x + ',' + p.y + ' ' + r +
',' + r2 + ' 0,' + (65535 *
360));
        },

        _setPath: function
(layer, path) {
                layer._path.v
```

```
= path;
        },

        _bringToFront:
function (layer) {

toFront(layer._container);
        },

        _bringToBack: function
(layer) {

toBack(layer._container);
        }
  };

  var create = Browser.vml ?
vmlCreate : svgCreate;

  /*
   * @class SVG
   * @inherits Renderer
   * @aka L.SVG
   *
   * Allows vector layers to
be displayed with [SVG]
(https://developer.mozilla.org
/docs/Web/SVG).
   * Inherits `Renderer`.
   *
   * Due to [technical
limitations]
(https://caniuse.com/svg), SVG
is not
   * available in all web
browsers, notably Android 2.x
and 3.x.
   *
   * Although SVG is not
available on IE7 and IE8,
these browsers support
   * [VML]
(https://en.wikipedia.org/wiki
/Vector_Markup_Language)
   * (a now deprecated
technology), and the SVG
renderer will fall back to VML
```

```
in
   * this case.
   *
   * @example
   *
   * Use SVG by default for
all paths in the map:
   *
   * ```js
   * var map = L.map('map', {
   *    renderer: L.svg()
   * });
   * ```
   *
   * Use a SVG renderer with
extra padding for specific
vector geometries:
   *
   * ```js
   * var map = L.map('map');
   * var myRenderer = L.svg({
padding: 0.5 });
   * var line = L.polyline(
coordinates, { renderer:
myRenderer } );
   * var circle = L.circle(
center, { renderer: myRenderer
} );
   * ```
   */

  var SVG = Renderer.extend({

        _initContainer:
function () {

this._container =
create('svg');

                // makes it
possible to click through svg
root; we'll reset it back in
individual paths

this._container.setAttribute('
pointer-events', 'none');
```

```
this._rootGroup = create('g');

this._container.appendChild(th
is._rootGroup);
        },

        _destroyContainer:
function () {

remove(this._container);

off(this._container);
                delete
this._container;
                delete
this._rootGroup;
                delete
this._svgSize;
        },

        _update: function () {
                if
(this._map._animatingZoom &&
this._bounds) { return; }


Renderer.prototype._update.cal
l(this);

                var b =
this._bounds,
                    size =
b.getSize(),
                    container
= this._container;

                // set size of
svg-container if changed
                if
(!this._svgSize ||
!this._svgSize.equals(size)) {

this._svgSize = size;

container.setAttribute('width'
, size.x);
```

```
container.setAttribute('height
', size.y);
                }

                // movement:
update container viewBox so
that we don't have to change
coordinates of individual
layers

setPosition(container, b.min);

container.setAttribute('viewBo
x', [b.min.x, b.min.y, size.x,
size.y].join(' '));


this.fire('update');
        },

        // methods below are
called by vector layers
implementations

        _initPath: function
(layer) {
                var path =
layer._path = create('path');

                // @namespace
Path
                // @option
className: String = null
                // Custom
class name set on an element.
Only for SVG renderer.
                if
(layer.options.className) {

addClass(path,
layer.options.className);
                }

                if
(layer.options.interactive) {
```

```
addClass(path, 'leaflet-
interactive');
                }


this._updateStyle(layer);

this._layers[stamp(layer)] =
layer;
        },

        _addPath: function
(layer) {
                if
(!this._rootGroup) {
this._initContainer(); }

this._rootGroup.appendChild(la
yer._path);

layer.addInteractiveTarget(lay
er._path);
        },

        _removePath: function
(layer) {

remove(layer._path);

layer.removeInteractiveTarget(
layer._path);
                delete
this._layers[stamp(layer)];
        },

        _updatePath: function
(layer) {

layer._project();

layer._update();
        },

        _updateStyle: function
(layer) {
                var path =
layer._path,
```

```
                    options =
layer.options;

                if (!path) {
return; }

                if
(options.stroke) {

path.setAttribute('stroke',
options.color);

path.setAttribute('stroke-
opacity', options.opacity);

path.setAttribute('stroke-
width', options.weight);

path.setAttribute('stroke-
linecap', options.lineCap);

path.setAttribute('stroke-
linejoin', options.lineJoin);

                        if
(options.dashArray) {

path.setAttribute('stroke-
dasharray',
options.dashArray);
                        } else
{

path.removeAttribute('stroke-
dasharray');
                        }

                        if
(options.dashOffset) {

path.setAttribute('stroke-
dashoffset',
options.dashOffset);
                        } else
{

path.removeAttribute('stroke-
```

```
dashoffset');
                        }
                } else {

path.setAttribute('stroke',
'none');
                }

                if
(options.fill) {

path.setAttribute('fill',
options.fillColor ||
options.color);

path.setAttribute('fill-
opacity',
options.fillOpacity);

path.setAttribute('fill-rule',
options.fillRule ||
'evenodd');
                } else {

path.setAttribute('fill',
'none');
                }
        },

        _updatePoly: function
(layer, closed) {

this._setPath(layer,
pointsToPath(layer._parts,
closed));
        },

        _updateCircle:
function (layer) {
                var p =
layer._point,
                    r =
Math.max(Math.round(layer._rad
ius), 1),
                    r2 =
Math.max(Math.round(layer._rad
iusY), 1) || r,
```

```
                    arc = 'a'
+ r + ',' + r2 + ' 0 1,0 ';

                // drawing a
circle with two half-arcs
                var d =
layer._empty() ? 'M0 0' :
                        'M' +
(p.x - r) + ',' + p.y +
                        arc +
(r * 2) + ',0 ' +
                        arc +
(-r * 2) + ',0 ';


this._setPath(layer, d);
        },

        _setPath: function
(layer, path) {

layer._path.setAttribute('d',
path);
        },

        // SVG does not have
the concept of zIndex so we
resort to changing the DOM
order of elements
        _bringToFront:
function (layer) {

toFront(layer._path);
        },

        _bringToBack: function
(layer) {

toBack(layer._path);
        }
  });

  if (Browser.vml) {
        SVG.include(vmlMixin);
  }

  // @namespace SVG
```

```
  // @factory L.svg(options?:
Renderer options)
  // Creates a SVG renderer
with the given options.
  function svg(options) {
        return Browser.svg ||
Browser.vml ? new SVG(options)
: null;
  }

  Map.include({
        // @namespace Map;
@method getRenderer(layer:
Path): Renderer
        // Returns the
instance of `Renderer` that
should be used to render the
given
        // `Path`. It will
ensure that the `renderer`
options of the map and paths
        // are respected, and
that the renderers do exist on
the map.
        getRenderer: function
(layer) {
                // @namespace
Path; @option renderer:
Renderer
                // Use this
specific instance of
`Renderer` for this path.
Takes
                // precedence
over the map's [default
renderer](#map-renderer).
                var renderer =
layer.options.renderer ||
this._getPaneRenderer(layer.op
tions.pane) ||
this.options.renderer ||
this._renderer;

                if (!renderer)
{

renderer = this._renderer =
```

```
this._createRenderer();
                }

                if
(!this.hasLayer(renderer)) {

this.addLayer(renderer);
                }
                return
renderer;
        },

        _getPaneRenderer:
function (name) {
                if (name ===
'overlayPane' || name ===
undefined) {
                        return
false;
                }

                var renderer =
this._paneRenderers[name];
                if (renderer
=== undefined) {

renderer =
this._createRenderer({pane:
name});

this._paneRenderers[name] =
renderer;
                }
                return
renderer;
        },

        _createRenderer:
function (options) {
                // @namespace
Map; @option preferCanvas:
Boolean = false
                // Whether
`Path`s should be rendered on
a `Canvas` renderer.
                // By default,
all `Path`s are rendered in a
```

```
`SVG` renderer.
                return
(this.options.preferCanvas &&
canvas(options)) ||
svg(options);
        }
  });

  /*
   * L.Rectangle extends
Polygon and creates a
rectangle when passed a
LatLngBounds object.
   */

  /*
   * @class Rectangle
   * @aka L.Rectangle
   * @inherits Polygon
   *
   * A class for drawing
rectangle overlays on a map.
Extends `Polygon`.
   *
   * @example
   *
   * ```js
   * // 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);
   * ```
   *
   */
```

```
  var Rectangle =
Polygon.extend({
        initialize: function
(latLngBounds, options) {

Polygon.prototype.initialize.c
all(this,
this._boundsToLatLngs(latLngBo
unds), options);
        },

        // @method
setBounds(latLngBounds:
LatLngBounds): this
        // Redraws the
rectangle with the passed
bounds.
        setBounds: function
(latLngBounds) {
                return
this.setLatLngs(this._boundsTo
LatLngs(latLngBounds));
        },

        _boundsToLatLngs:
function (latLngBounds) {
                latLngBounds =
toLatLngBounds(latLngBounds);
                return [

latLngBounds.getSouthWest(),

latLngBounds.getNorthWest(),

latLngBounds.getNorthEast(),

latLngBounds.getSouthEast()
                ];
        }
  });


  // @factory
L.rectangle(latLngBounds:
LatLngBounds, options?:
Polyline options)
  function
```

```
rectangle(latLngBounds,
options) {
        return new
Rectangle(latLngBounds,
options);
  }

  SVG.create = create;
  SVG.pointsToPath =
pointsToPath;

  GeoJSON.geometryToLayer =
geometryToLayer;
  GeoJSON.coordsToLatLng =
coordsToLatLng;
  GeoJSON.coordsToLatLngs =
coordsToLatLngs;
  GeoJSON.latLngToCoords =
latLngToCoords;
  GeoJSON.latLngsToCoords =
latLngsToCoords;
  GeoJSON.getFeature =
getFeature;
  GeoJSON.asFeature =
asFeature;

  /*
   * L.Handler.BoxZoom is used
to add shift-drag zoom
interaction to the map
   * (zoom to a selected
bounding box), enabled by
default.
   */

  // @namespace Map
  // @section Interaction
Options
  Map.mergeOptions({
        // @option boxZoom:
Boolean = true
        // Whether the map can
be zoomed to a rectangular
area specified by
        // dragging the mouse
while pressing the shift key.
        boxZoom: true
```

```
  });

  var BoxZoom =
Handler.extend({
        initialize: function
(map) {
                this._map =
map;

this._container =
map._container;
                this._pane =
map._panes.overlayPane;

this._resetStateTimeout = 0;

map.on('unload',
this._destroy, this);
        },

        addHooks: function ()
{

on(this._container,
'mousedown',
this._onMouseDown, this);
        },

        removeHooks: function
() {

off(this._container,
'mousedown',
this._onMouseDown, this);
        },

        moved: function () {
                return
this._moved;
        },

        _destroy: function ()
{

remove(this._pane);
                delete
this._pane;
```

```
        },

        _resetState: function
() {

this._resetStateTimeout = 0;
                this._moved =
false;
        },


_clearDeferredResetState:
function () {
                if
(this._resetStateTimeout !==
0) {

clearTimeout(this._resetStateT
imeout);

this._resetStateTimeout = 0;
                }
        },

        _onMouseDown: function
(e) {
                if
(!e.shiftKey || ((e.which !==
1) && (e.button !== 1))) {
return false; }

                // Clear the
deferred resetState if it
hasn't executed yet, otherwise
it
                // will
interrupt the interaction and
orphan a box element in the
container.

this._clearDeferredResetState(
);

this._resetState();


disableTextSelection();
```

```
disableImageDrag();


this._startPoint =
this._map.mouseEventToContaine
rPoint(e);

                on(document, {

contextmenu: stop,

mousemove: this._onMouseMove,

mouseup: this._onMouseUp,

keydown: this._onKeyDown
                }, this);
        },

        _onMouseMove: function
(e) {
                if
(!this._moved) {

this._moved = true;


this._box = create$1('div',
'leaflet-zoom-box',
this._container);

addClass(this._container,
'leaflet-crosshair');


this._map.fire('boxzoomstart')
;
                }

                this._point =
this._map.mouseEventToContaine
rPoint(e);

                var bounds =
new Bounds(this._point,
this._startPoint),
```

```
                    size =
bounds.getSize();


setPosition(this._box,
bounds.min);


this._box.style.width  =
size.x + 'px';

this._box.style.height =
size.y + 'px';
        },

        _finish: function () {
                if
(this._moved) {

remove(this._box);

removeClass(this._container,
'leaflet-crosshair');
                }


enableTextSelection();

enableImageDrag();

                off(document,
{

contextmenu: stop,

mousemove: this._onMouseMove,

mouseup: this._onMouseUp,

keydown: this._onKeyDown
                }, this);
        },

        _onMouseUp: function
(e) {
                if ((e.which
!== 1) && (e.button !== 1)) {
```

```
return; }


this._finish();

                if
(!this._moved) { return; }
                // Postpone to
next JS tick so internal click
event handling
                // still see
it as "moved".

this._clearDeferredResetState(
);

this._resetStateTimeout =
setTimeout(bind(this._resetSta
te, this), 0);

                var bounds =
new LatLngBounds(

this._map.containerPointToLatL
ng(this._startPoint),

this._map.containerPointToLatL
ng(this._point));

                this._map

.fitBounds(bounds)

.fire('boxzoomend',
{boxZoomBounds: bounds});
        },

        _onKeyDown: function
(e) {
                if (e.keyCode
=== 27) {

this._finish();

this._clearDeferredResetState(
);
```

```
this._resetState();
                }
        }
  });

  // @section Handlers
  // @property boxZoom:
Handler
  // Box (shift-drag with
mouse) zoom handler.

Map.addInitHook('addHandler',
'boxZoom', BoxZoom);

  /*
   * L.Handler.DoubleClickZoom
is used to handle double-click
zoom on the map, enabled by
default.
   */

  // @namespace Map
  // @section Interaction
Options

  Map.mergeOptions({
        // @option
doubleClickZoom:
Boolean|String = true
        // Whether the map can
be zoomed in by double
clicking on it and
        // zoomed out by
double clicking while holding
shift. If passed
        // `'center'`, double-
click zoom will zoom to the
center of the
        //  view regardless of
where the mouse was.
        doubleClickZoom: true
  });

  var DoubleClickZoom =
Handler.extend({
        addHooks: function ()
{
```

```
this._map.on('dblclick',
this._onDoubleClick, this);
        },

        removeHooks: function
() {

this._map.off('dblclick',
this._onDoubleClick, this);
        },

        _onDoubleClick:
function (e) {
                var map =
this._map,
                    oldZoom =
map.getZoom(),
                    delta =
map.options.zoomDelta,
                    zoom =
e.originalEvent.shiftKey ?
oldZoom - delta : oldZoom +
delta;

                if
(map.options.doubleClickZoom
=== 'center') {

map.setZoom(zoom);
                } else {

map.setZoomAround(e.containerP
oint, zoom);
                }
        }
  });

  // @section Handlers
  //
  // 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 `Handler` methods). For
example:
  //
  // ```js
  //
map.doubleClickZoom.disable();
  // ```
  //
  // @property
doubleClickZoom: Handler
  // Double click zoom
handler.

Map.addInitHook('addHandler',
'doubleClickZoom',
DoubleClickZoom);

  /*
   * L.Handler.MapDrag is used
to make the map draggable
(with panning inertia),
enabled by default.
   */

  // @namespace Map
  // @section Interaction
Options
  Map.mergeOptions({
        // @option dragging:
Boolean = true
        // Whether the map is
draggable with mouse/touch or
not.
        dragging: true,

        // @section Panning
Inertia Options
        // @option inertia:
Boolean = *
        // If enabled, panning
of the map will have an
inertia effect where
        // the map builds
momentum while dragging and
continues moving in
        // the same direction
for some time. Feels
```

```
especially nice on touch
        // devices. Enabled by
default.
        inertia: true,

        // @option
inertiaDeceleration: Number =
3000
        // The rate with which
the inertial movement slows
down, in pixels/second².
        inertiaDeceleration:
3400, // px/s^2

        // @option
inertiaMaxSpeed: Number =
Infinity
        // Max speed of the
inertial movement, in
pixels/second.
        inertiaMaxSpeed:
Infinity, // px/s

        // @option
easeLinearity: Number = 0.2
        easeLinearity: 0.2,

        // TODO refactor, move
to CRS
        // @option
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.
        worldCopyJump: false,

        // @option
maxBoundsViscosity: Number =
0.0
        // If `maxBounds` is
```

```
set, this option will control
how solid the bounds
        // are when dragging
the map around. The default
value of `0.0` allows the
        // user to drag
outside the bounds at normal
speed, higher values will
        // slow down map
dragging outside bounds, and
`1.0` makes the bounds fully
        // solid, preventing
the user from dragging outside
the bounds.
        maxBoundsViscosity:
0.0
  });

  var Drag = Handler.extend({
        addHooks: function ()
{
                if
(!this._draggable) {
                        var
map = this._map;


this._draggable = new
Draggable(map._mapPane,
map._container);


this._draggable.on({

dragstart: this._onDragStart,

drag: this._onDrag,

dragend: this._onDragEnd
                        },
this);


this._draggable.on('predrag',
this._onPreDragLimit, this);
                        if
(map.options.worldCopyJump) {
```

```
this._draggable.on('predrag',
this._onPreDragWrap, this);

map.on('zoomend',
this._onZoomEnd, this);


map.whenReady(this._onZoomEnd,
this);
                        }
                }

addClass(this._map._container,
'leaflet-grab leaflet-touch-
drag');

this._draggable.enable();

this._positions = [];
                this._times =
[];
        },

        removeHooks: function
() {

removeClass(this._map._contain
er, 'leaflet-grab');

removeClass(this._map._contain
er, 'leaflet-touch-drag');

this._draggable.disable();
        },

        moved: function () {
                return
this._draggable &&
this._draggable._moved;
        },

        moving: function () {
                return
this._draggable &&
this._draggable._moving;
        },
```

```
        _onDragStart: function
() {
                var map =
this._map;

                map._stop();
                if
(this._map.options.maxBounds
&&
this._map.options.maxBoundsVis
cosity) {
                        var
bounds =
toLatLngBounds(this._map.optio
ns.maxBounds);


this._offsetLimit = toBounds(

this._map.latLngToContainerPoi
nt(bounds.getNorthWest()).mult
iplyBy(-1),

this._map.latLngToContainerPoi
nt(bounds.getSouthEast()).mult
iplyBy(-1)

.add(this._map.getSize()));


this._viscosity =
Math.min(1.0, Math.max(0.0,
this._map.options.maxBoundsVis
cosity));
                } else {

this._offsetLimit = null;
                }

                map

.fire('movestart')

.fire('dragstart');

                if
```

```
(map.options.inertia) {

this._positions = [];

this._times = [];
                }
        },

        _onDrag: function (e)
{
                if
(this._map.options.inertia) {
                        var
time = this._lastTime = +new
Date(),

pos = this._lastPos =
this._draggable._absPos ||
this._draggable._newPos;


this._positions.push(pos);

this._times.push(time);


this._prunePositions(time);
                }

                this._map

.fire('move', e)

.fire('drag', e);
        },

        _prunePositions:
function (time) {
                while
(this._positions.length > 1 &&
time - this._times[0] > 50) {

this._positions.shift();

this._times.shift();
                }
        },
```

```
        _onZoomEnd: function
() {
                var pxCenter =
this._map.getSize().divideBy(2
),

pxWorldCenter =
this._map.latLngToLayerPoint([
0, 0]);


this._initialWorldOffset =
pxWorldCenter.subtract(pxCente
r).x;

this._worldWidth =
this._map.getPixelWorldBounds(
).getSize().x;
        },

        _viscousLimit:
function (value, threshold) {
                return value -
(value - threshold) *
this._viscosity;
        },

        _onPreDragLimit:
function () {
                if
(!this._viscosity ||
!this._offsetLimit) { return;
}

                var offset =
this._draggable._newPos.subtra
ct(this._draggable._startPos);

                var limit =
this._offsetLimit;
                if (offset.x <
limit.min.x) { offset.x =
this._viscousLimit(offset.x,
limit.min.x); }
                if (offset.y <
limit.min.y) { offset.y =
```

```
this._viscousLimit(offset.y,
limit.min.y); }
                if (offset.x >
limit.max.x) { offset.x =
this._viscousLimit(offset.x,
limit.max.x); }
                if (offset.y >
limit.max.y) { offset.y =
this._viscousLimit(offset.y,
limit.max.y); }


this._draggable._newPos =
this._draggable._startPos.add(
offset);
        },

        _onPreDragWrap:
function () {
                // TODO
refactor to be able to adjust
map pane position after zoom
                var worldWidth
= this._worldWidth,
                    halfWidth
= Math.round(worldWidth / 2),
                    dx =
this._initialWorldOffset,
                    x =
this._draggable._newPos.x,
                    newX1 = (x
- halfWidth + dx) % worldWidth
+ halfWidth - dx,
                    newX2 = (x
+ halfWidth + dx) % worldWidth
- halfWidth - dx,
                    newX =
Math.abs(newX1 + dx) <
Math.abs(newX2 + dx) ? newX1 :
newX2;


this._draggable._absPos =
this._draggable._newPos.clone(
);

this._draggable._newPos.x =
```

```
newX;
        },

        _onDragEnd: function
(e) {
                var map =
this._map,
                    options =
map.options,

                    noInertia
= !options.inertia ||
e.noInertia ||
this._times.length < 2;


map.fire('dragend', e);

                if (noInertia)
{

map.fire('moveend');

                } else {

this._prunePositions(+new
Date());

                        var
direction =
this._lastPos.subtract(this._p
ositions[0]),

duration = (this._lastTime -
this._times[0]) / 1000,

ease = options.easeLinearity,


speedVector =
direction.multiplyBy(ease /
duration),

speed =
speedVector.distanceTo([0,
0]),
```

```
limitedSpeed =
Math.min(options.inertiaMaxSpe
ed, speed),

limitedSpeedVector =
speedVector.multiplyBy(limited
Speed / speed),


decelerationDuration =
limitedSpeed /
(options.inertiaDeceleration *
ease),

offset =
limitedSpeedVector.multiplyBy(
-decelerationDuration /
2).round();

                        if
(!offset.x && !offset.y) {

map.fire('moveend');

                        } else
{

offset =
map._limitOffset(offset,
map.options.maxBounds);


requestAnimFrame(function () {

map.panBy(offset, {

duration:
decelerationDuration,

easeLinearity: ease,

noMoveStart: true,

animate: true

});
```

```
});
                        }
                    }
            }
    });

    // @section Handlers
    // @property dragging:
Handler
    // Map dragging handler (by
both mouse and touch).

Map.addInitHook('addHandler',
'dragging', Drag);

    /*
     * L.Map.Keyboard is
handling keyboard interaction
with the map, enabled by
default.
     */

    // @namespace Map
    // @section Keyboard
Navigation Options
    Map.mergeOptions({
            // @option keyboard:
Boolean = true
            // Makes the map
focusable and allows users to
navigate the map with keyboard
            // arrows and `+`/`-`
keys.
            keyboard: true,

            // @option
keyboardPanDelta: Number = 80
            // Amount of pixels to
pan when pressing an arrow
key.
            keyboardPanDelta: 80
    });

    var Keyboard =
Handler.extend({
```

```
        keyCodes: {
                left:    [37],
                right:   [39],
                down:    [40],
                up:      [38],
                zoomIn:  [187,
107, 61, 171],
                zoomOut: [189,
109, 54, 173]
        },

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


this._setPanDelta(map.options.
keyboardPanDelta);

this._setZoomDelta(map.options
.zoomDelta);
        },

        addHooks: function ()
{
                var container
= this._map._container;

                // make the
container focusable by tabbing
                if
(container.tabIndex <= 0) {

container.tabIndex = '0';
                }

                on(container,
{
                        focus:
this._onFocus,
                        blur:
this._onBlur,

mousedown: this._onMouseDown
                }, this);
```

```
                this._map.on({
                        focus:
this._addHooks,
                        blur:
this._removeHooks
                }, this);
        },

        removeHooks: function
() {

this._removeHooks();


off(this._map._container, {
                        focus:
this._onFocus,
                        blur:
this._onBlur,

mousedown: this._onMouseDown
                }, this);


this._map.off({
                        focus:
this._addHooks,
                        blur:
this._removeHooks
                }, this);
        },

        _onMouseDown: function
() {
                if
(this._focused) { return; }

                var body =
document.body,
                    docEl =
document.documentElement,
                    top =
body.scrollTop ||
docEl.scrollTop,
                    left =
body.scrollLeft ||
docEl.scrollLeft;
```

```
this._map._container.focus();


window.scrollTo(left, top);
        },

        _onFocus: function ()
{
                this._focused
= true;

this._map.fire('focus');
        },

        _onBlur: function () {
                this._focused
= false;

this._map.fire('blur');
        },

        _setPanDelta: function
(panDelta) {
                var keys =
this._panKeys = {},
                    codes =
this.keyCodes,
                    i, len;

                for (i = 0,
len = codes.left.length; i <
len; i++) {

keys[codes.left[i]] = [-1 *
panDelta, 0];
                }
                for (i = 0,
len = codes.right.length; i <
len; i++) {

keys[codes.right[i]] =
[panDelta, 0];
                }
                for (i = 0,
len = codes.down.length; i <
```

```
len; i++) {

keys[codes.down[i]] = [0,
panDelta];
                }
                for (i = 0,
len = codes.up.length; i <
len; i++) {

keys[codes.up[i]] = [0, -1 *
panDelta];
                }
        },

        _setZoomDelta:
function (zoomDelta) {
                var keys =
this._zoomKeys = {},
                    codes =
this.keyCodes,
                    i, len;

                for (i = 0,
len = codes.zoomIn.length; i <
len; i++) {

keys[codes.zoomIn[i]] =
zoomDelta;
                }
                for (i = 0,
len = codes.zoomOut.length; i
< len; i++) {

keys[codes.zoomOut[i]] = -
zoomDelta;
                }
        },

        _addHooks: function ()
{
                on(document,
'keydown', this._onKeyDown,
this);
        },

        _removeHooks: function
() {
```

```
                off(document,
'keydown', this._onKeyDown,
this);
        },

        _onKeyDown: function
(e) {
                if (e.altKey
|| e.ctrlKey || e.metaKey) {
return; }

                var key =
e.keyCode,
                    map =
this._map,
                    offset;

                if (key in
this._panKeys) {
                        if
(!map._panAnim ||
!map._panAnim._inProgress) {

offset = this._panKeys[key];

if (e.shiftKey) {

offset =
toPoint(offset).multiplyBy(3);

}


map.panBy(offset);


if (map.options.maxBounds) {

map.panInsideBounds(map.option
s.maxBounds);

}
                        }
                } else if (key
in this._zoomKeys) {

map.setZoom(map.getZoom() +
```

```
(e.shiftKey ? 3 : 1) *
this._zoomKeys[key]);

                } else if (key
=== 27 && map._popup &&
map._popup.options.closeOnEsca
peKey) {

map.closePopup();

                } else {

return;
                }

                stop(e);
        }
  });

  // @section Handlers
  // @section Handlers
  // @property keyboard:
Handler
  // Keyboard navigation
handler.

Map.addInitHook('addHandler',
'keyboard', Keyboard);

  /*
   * L.Handler.ScrollWheelZoom
is used by L.Map to enable
mouse scroll wheel zoom on the
map.
   */

  // @namespace Map
  // @section Interaction
Options
  Map.mergeOptions({
        // @section Mouse
wheel options
        // @option
scrollWheelZoom:
Boolean|String = true
        // Whether the map can
be zoomed by using the mouse
```

```
wheel. If passed `'center'`,
        // it will zoom to the
center of the view regardless
of where the mouse was.
        scrollWheelZoom: true,

        // @option
wheelDebounceTime: Number = 40
        // Limits the rate at
which a wheel can fire (in
milliseconds). By default
        // user can't zoom via
wheel more often than once per
40 ms.
        wheelDebounceTime: 40,

        // @option
wheelPxPerZoomLevel: Number =
60
        // How many scroll
pixels (as reported by
[L.DomEvent.getWheelDelta]
(#domevent-getwheeldelta))
        // mean a change of
one full zoom level. Smaller
values will make wheel-zooming
        // faster (and vice
versa).
        wheelPxPerZoomLevel:
60
  });

  var ScrollWheelZoom =
Handler.extend({
        addHooks: function ()
{

on(this._map._container,
'wheel', this._onWheelScroll,
this);

                this._delta =
0;
        },

        removeHooks: function
() {
```

```
off(this._map._container,
'wheel', this._onWheelScroll,
this);
        },

        _onWheelScroll:
function (e) {
                var delta =
getWheelDelta(e);

                var debounce =
this._map.options.wheelDebounc
eTime;

                this._delta +=
delta;

this._lastMousePos =
this._map.mouseEventToContaine
rPoint(e);

                if
(!this._startTime) {

this._startTime = +new Date();
                }

                var left =
Math.max(debounce - (+new
Date() - this._startTime), 0);


clearTimeout(this._timer);
                this._timer =
setTimeout(bind(this._performZ
oom, this), left);

                stop(e);
        },

        _performZoom: function
() {
                var map =
this._map,
                    zoom =
map.getZoom(),
```

```
                    snap =
this._map.options.zoomSnap ||
0;

                map._stop();
// stop panning and fly
animations if any

                // map the
delta with a sigmoid function
to -4..4 range leaning on
-1..1
                var d2 =
this._delta /
(this._map.options.wheelPxPerZ
oomLevel * 4),
                    d3 = 4 *
Math.log(2 / (1 + Math.exp(-
Math.abs(d2)))) / Math.LN2,
                    d4 = snap
? Math.ceil(d3 / snap) * snap
: d3,
                    delta =
map._limitZoom(zoom +
(this._delta > 0 ? d4 : -d4))
- zoom;

                this._delta =
0;

this._startTime = null;

                if (!delta) {
return; }

                if
(map.options.scrollWheelZoom
=== 'center') {

map.setZoom(zoom + delta);
                } else {

map.setZoomAround(this._lastMo
usePos, zoom + delta);
                }
        }
  });
```

```
  // @section Handlers
  // @property
scrollWheelZoom: Handler
  // Scroll wheel zoom
handler.

Map.addInitHook('addHandler',
'scrollWheelZoom',
ScrollWheelZoom);

  /*
   * L.Map.TapHold is used to
simulate `contextmenu` event
on long hold,
   * which otherwise is not
fired by mobile Safari.
   */

  var tapHoldDelay = 600;

  // @namespace Map
  // @section Interaction
Options
  Map.mergeOptions({
        // @section Touch
interaction options
        // @option tapHold:
Boolean
        // Enables simulation
of `contextmenu` event,
default is `true` for mobile
Safari.
        tapHold:
Browser.touchNative &&
Browser.safari &&
Browser.mobile,

        // @option
tapTolerance: Number = 15
        // The max number of
pixels a user can shift his
finger during touch
        // for it to be
considered a valid tap.
        tapTolerance: 15
  });
```

```
  var TapHold =
Handler.extend({
        addHooks: function ()
{

on(this._map._container,
'touchstart', this._onDown,
this);
        },

        removeHooks: function
() {

off(this._map._container,
'touchstart', this._onDown,
this);
        },

        _onDown: function (e)
{

clearTimeout(this._holdTimeout
);
                if
(e.touches.length !== 1) {
return; }

                var first =
e.touches[0];
                this._startPos
= this._newPos = new
Point(first.clientX,
first.clientY);


this._holdTimeout =
setTimeout(bind(function () {

this._cancel();
                        if
(!this._isTapValid()) {
return; }

                        //
prevent simulated mouse events
https://w3c.github.io/touch-
```

```
events/#mouse-events

on(document, 'touchend',
preventDefault);

on(document, 'touchend
touchcancel',
this._cancelClickPrevent);

this._simulateEvent('contextme
nu', first);
                }, this),
tapHoldDelay);

                on(document,
'touchend touchcancel
contextmenu', this._cancel,
this);
                on(document,
'touchmove', this._onMove,
this);
        },

        _cancelClickPrevent:
function cancelClickPrevent()
{
                off(document,
'touchend', preventDefault);
                off(document,
'touchend touchcancel',
cancelClickPrevent);
        },

        _cancel: function () {

clearTimeout(this._holdTimeout
);
                off(document,
'touchend touchcancel
contextmenu', this._cancel,
this);
                off(document,
'touchmove', this._onMove,
this);
        },

        _onMove: function (e)
```

```
{
                var first =
e.touches[0];
                this._newPos =
new Point(first.clientX,
first.clientY);
        },

        _isTapValid: function
() {
                return
this._newPos.distanceTo(this._
startPos) <=
this._map.options.tapTolerance
;
        },

        _simulateEvent:
function (type, e) {
                var
simulatedEvent = new
MouseEvent(type, {

bubbles: true,

cancelable: true,
                        view:
window,
                        //
detail: 1,

screenX: e.screenX,

screenY: e.screenY,

clientX: e.clientX,

clientY: e.clientY,
                        //
button: 2,
                        //
buttons: 2
                });


simulatedEvent._simulated =
true;
```

```
e.target.dispatchEvent(simulat
edEvent);
        }
  });

  // @section Handlers
  // @property tapHold:
Handler
  // Long tap handler to
simulate `contextmenu` event
(useful in mobile Safari).

Map.addInitHook('addHandler',
'tapHold', TapHold);

  /*
   * L.Handler.TouchZoom is
used by L.Map to add pinch
zoom on supported mobile
browsers.
   */

  // @namespace Map
  // @section Interaction
Options
  Map.mergeOptions({
        // @section Touch
interaction options
        // @option touchZoom:
Boolean|String = *
        // Whether the map can
be zoomed by touch-dragging
with two fingers. If
        // passed `'center'`,
it will zoom to the center of
the view regardless of
        // where the touch
events (fingers) were. Enabled
for touch-capable web
        // browsers.
        touchZoom:
Browser.touch,

        // @option
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.
        bounceAtZoomLimits:
true
  });

  var TouchZoom =
Handler.extend({
        addHooks: function ()
{

addClass(this._map._container,
'leaflet-touch-zoom');

on(this._map._container,
'touchstart',
this._onTouchStart, this);
        },

        removeHooks: function
() {

removeClass(this._map._contain
er, 'leaflet-touch-zoom');

off(this._map._container,
'touchstart',
this._onTouchStart, this);
        },

        _onTouchStart:
function (e) {
                var map =
this._map;
                if (!e.touches
|| e.touches.length !== 2 ||
map._animatingZoom ||
this._zooming) { return; }

                var p1 =
map.mouseEventToContainerPoint
(e.touches[0]),
                    p2 =
```

```
map.mouseEventToContainerPoint
(e.touches[1]);


this._centerPoint =
map.getSize()._divideBy(2);

this._startLatLng =
map.containerPointToLatLng(thi
s._centerPoint);
                if
(map.options.touchZoom !==
'center') {

this._pinchStartLatLng =
map.containerPointToLatLng(p1.
add(p2)._divideBy(2));
                }


this._startDist =
p1.distanceTo(p2);

this._startZoom =
map.getZoom();

                this._moved =
false;
                this._zooming
= true;

                map._stop();

                on(document,
'touchmove',
this._onTouchMove, this);
                on(document,
'touchend touchcancel',
this._onTouchEnd, this);


preventDefault(e);
        },

        _onTouchMove: function
(e) {
                if (!e.touches
```

```
|| e.touches.length !== 2 ||
!this._zooming) { return; }

                var map =
this._map,
                    p1 =
map.mouseEventToContainerPoint
(e.touches[0]),
                    p2 =
map.mouseEventToContainerPoint
(e.touches[1]),
                    scale =
p1.distanceTo(p2) /
this._startDist;

                this._zoom =
map.getScaleZoom(scale,
this._startZoom);

                if
(!map.options.bounceAtZoomLimi
ts && (

(this._zoom < map.getMinZoom()
&& scale < 1) ||

(this._zoom > map.getMaxZoom()
&& scale > 1))) {

this._zoom =
map._limitZoom(this._zoom);
                }

                if
(map.options.touchZoom ===
'center') {

this._center =
this._startLatLng;
                        if
(scale === 1) { return; }
                } else {
                        // Get
delta from pinch to center, so
centerLatLng is delta applied
to initial pinchLatLng
                        var
```

```
delta =
p1._add(p2)._divideBy(2)._subt
ract(this._centerPoint);
                        if
(scale === 1 && delta.x === 0
&& delta.y === 0) { return; }

this._center =
map.unproject(map.project(this
._pinchStartLatLng,
this._zoom).subtract(delta),
this._zoom);
                }

                if
(!this._moved) {

map._moveStart(true, false);

this._moved = true;
                }


cancelAnimFrame(this._animRequ
est);

                var moveFn =
bind(map._move, map,
this._center, this._zoom,
{pinch: true, round: false},
undefined);

this._animRequest =
requestAnimFrame(moveFn, this,
true);


preventDefault(e);
        },

        _onTouchEnd: function
() {
                if
(!this._moved ||
!this._zooming) {

this._zooming = false;
```

```
return;
                }

                this._zooming
= false;

cancelAnimFrame(this._animRequ
est);

                off(document,
'touchmove',
this._onTouchMove, this);
                off(document,
'touchend touchcancel',
this._onTouchEnd, this);

                // Pinch
updates GridLayers' levels
only when zoomSnap is off, so
zoomSnap becomes noUpdate.
                if
(this._map.options.zoomAnimati
on) {

this._map._animateZoom(this._c
enter,
this._map._limitZoom(this._zoo
m), true,
this._map.options.zoomSnap);
                } else {

this._map._resetView(this._cen
ter,
this._map._limitZoom(this._zoo
m));
                }
        }
  });

  // @section Handlers
  // @property touchZoom:
Handler
  // Touch zoom handler.

Map.addInitHook('addHandler',
'touchZoom', TouchZoom);
```

```
  Map.BoxZoom = BoxZoom;
  Map.DoubleClickZoom =
DoubleClickZoom;
  Map.Drag = Drag;
  Map.Keyboard = Keyboard;
  Map.ScrollWheelZoom =
ScrollWheelZoom;
  Map.TapHold = TapHold;
  Map.TouchZoom = TouchZoom;

  var L$1 = {
    __proto__: null,
    version: version,
    Control: Control,
    control: control,
    Class: Class,
    Handler: Handler,
    extend: extend,
    bind: bind,
    stamp: stamp,
    setOptions: setOptions,
    Browser: Browser,
    Evented: Evented,
    Mixin: Mixin,
    Util: Util,
    PosAnimation:
PosAnimation,
    Draggable: Draggable,
    DomEvent: DomEvent,
    DomUtil: DomUtil,
    Point: Point,
    point: toPoint,
    Bounds: Bounds,
    bounds: toBounds,
    Transformation:
Transformation,
    transformation:
toTransformation,
    LineUtil: LineUtil,
    PolyUtil: PolyUtil,
    LatLng: LatLng,
    latLng: toLatLng,
    LatLngBounds:
LatLngBounds,
    latLngBounds:
toLatLngBounds,
```

```
    CRS: CRS,
    Projection: index,
    Layer: Layer,
    LayerGroup: LayerGroup,
    layerGroup: layerGroup,
    FeatureGroup:
FeatureGroup,
    featureGroup:
featureGroup,
    ImageOverlay:
ImageOverlay,
    imageOverlay:
imageOverlay,
    VideoOverlay:
VideoOverlay,
    videoOverlay:
videoOverlay,
    SVGOverlay: SVGOverlay,
    svgOverlay: svgOverlay,
    DivOverlay: DivOverlay,
    Popup: Popup,
    popup: popup,
    Tooltip: Tooltip,
    tooltip: tooltip,
    icon: icon,
    DivIcon: DivIcon,
    divIcon: divIcon,
    Marker: Marker,
    marker: marker,
    Icon: Icon,
    GridLayer: GridLayer,
    gridLayer: gridLayer,
    TileLayer: TileLayer,
    tileLayer: tileLayer,
    Renderer: Renderer,
    Canvas: Canvas,
    canvas: canvas,
    Path: Path,
    CircleMarker:
CircleMarker,
    circleMarker:
circleMarker,
    Circle: Circle,
    circle: circle,
    Polyline: Polyline,
    polyline: polyline,
    Polygon: Polygon,
```

```
    polygon: polygon,
    Rectangle: Rectangle,
    rectangle: rectangle,
    SVG: SVG,
    svg: svg,
    GeoJSON: GeoJSON,
    geoJSON: geoJSON,
    geoJson: geoJson,
    Map: Map,
    map: createMap
  };

  var globalL = extend(L$1,
{noConflict: noConflict});

  var globalObject =
getGlobalObject();
  var oldL = globalObject.L;

  globalObject.L = globalL;

  function noConflict() {
        globalObject.L = oldL;
        return globalL;
  }

  function getGlobalObject() {
        if (typeof globalThis
!== 'undefined') { return
globalThis; }
        if (typeof self !==
'undefined') { return self; }
        if (typeof window !==
'undefined') { return window;
}
        if (typeof global !==
'undefined') { return global;
}

        throw new
Error('Unable to locate global
object.');
  }

  exports.Bounds = Bounds;
  exports.Browser = Browser;
  exports.CRS = CRS;
```

```
  exports.Canvas = Canvas;
  exports.Circle = Circle;
  exports.CircleMarker =
CircleMarker;
  exports.Class = Class;
  exports.Control = Control;
  exports.DivIcon = DivIcon;
  exports.DivOverlay =
DivOverlay;
  exports.DomEvent = DomEvent;
  exports.DomUtil = DomUtil;
  exports.Draggable =
Draggable;
  exports.Evented = Evented;
  exports.FeatureGroup =
FeatureGroup;
  exports.GeoJSON = GeoJSON;
  exports.GridLayer =
GridLayer;
  exports.Handler = Handler;
  exports.Icon = Icon;
  exports.ImageOverlay =
ImageOverlay;
  exports.LatLng = LatLng;
  exports.LatLngBounds =
LatLngBounds;
  exports.Layer = Layer;
  exports.LayerGroup =
LayerGroup;
  exports.LineUtil = LineUtil;
  exports.Map = Map;
  exports.Marker = Marker;
  exports.Mixin = Mixin;
  exports.Path = Path;
  exports.Point = Point;
  exports.PolyUtil = PolyUtil;
  exports.Polygon = Polygon;
  exports.Polyline = Polyline;
  exports.Popup = Popup;
  exports.PosAnimation =
PosAnimation;
  exports.Projection = index;
  exports.Rectangle =
Rectangle;
  exports.Renderer = Renderer;
  exports.SVG = SVG;
  exports.SVGOverlay =
```

```
SVGOverlay;
  exports.TileLayer =
TileLayer;
  exports.Tooltip = Tooltip;
  exports.Transformation =
Transformation;
  exports.Util = Util;
  exports.VideoOverlay =
VideoOverlay;
  exports.bind = bind;
  exports.bounds = toBounds;
  exports.canvas = canvas;
  exports.circle = circle;
  exports.circleMarker =
circleMarker;
  exports.control = control;
  exports["default"] =
globalL;
  exports.divIcon = divIcon;
  exports.extend = extend;
  exports.featureGroup =
featureGroup;
  exports.geoJSON = geoJSON;
  exports.geoJson = geoJson;
  exports.gridLayer =
gridLayer;
  exports.icon = icon;
  exports.imageOverlay =
imageOverlay;
  exports.latLng = toLatLng;
  exports.latLngBounds =
toLatLngBounds;
  exports.layerGroup =
layerGroup;
  exports.map = createMap;
  exports.marker = marker;
  exports.noConflict =
noConflict;
  exports.point = toPoint;
  exports.polygon = polygon;
  exports.polyline = polyline;
  exports.popup = popup;
  exports.rectangle =
rectangle;
  exports.setOptions =
setOptions;
  exports.stamp = stamp;
```

```
  exports.svg = svg;
  exports.svgOverlay =
svgOverlay;
  exports.tileLayer =
tileLayer;
  exports.tooltip = tooltip;
  exports.transformation =
toTransformation;
  exports.version = version;
  exports.videoOverlay =
videoOverlay;

}));
//# sourceMappingURL=leaflet-
src.js.map
```