Leaflet

Dokumentation

Mentz Datenverarbeitung GmbH
Grillparzerstraße 18
81675 München
Tel: +49 89 41868-0
Fax: +49 89 41868-160
mdv@mentzdv.de
www.mentzdv.de

Versionsgeschichte				
Dokument Version	Software Version	Datum	Name	Änderungsgrund
0.9		28.07.14	AN	Erstellung mit TODOs

Freigabe			
	Datum	Name	Unterschrift
geprüft:			
freigegeben:			

Inhaltsverzeichnis

1 Allgemeines 4

2 Einrichten von Leaflet 5

3 Plugin-Übersicht 6

3.1 Anpassungen von Leaflet an die EFA 6

3.1.1 mdvLeafletTools 6

3.1.2 mdvLeafletMap 6

3.1.3 mdvLeafletExtension 6

3.2 Request-spezifische Plugins 7

3.2.1 mdvLeafletCoordInfo 7

3.2.2 mdvLeafletGeoObject 7

3.2.3 mdvLeafletTrip 7

3.3 Sonstige 7

3.3.1 mdvLeafletNetworkmap 7

4 Verwendung der Plugins 8

4.1 Grundsätzliche Konfiguration 8

4.1.1 Einbinden der Plugins 8

4.1.2 Initiieren der Plugins 8

4.2 Plugin-Spezifische Parameter 10

4.2.1 mdvLeafletMap 10

4.2.2 mdvLeafletCoordInfo 12

4.2.3 mdvLeafletGeoObject 15

4.2.4 mdvLeafletNetworkmap 16

4.2.5 mdvLeafletTrip 16

1 Allgemeines

Leaflet ist eine Open-Source JavaScript Bibliothek für interaktive Karten, die sowohl für Desktop als auch für mobile Endgeräte geeignet ist.

Seit Mai 2011 entwickelt Vladimir Agafonkin mit einem kleinen Entwicklerteam die Bibliothek und erfreut sich einer immer größeren Community, die Leaflet durch Plugins erweitert.

Vorteile von Leaflet liegen neben der freien Verfügbarkeit und der wachsenden Community und der dadurch immer größer werdenden Vielfalt an Plugins bei der einfachen Verwendung und der extrem einfachen Anpassungsfähigkeit des Plugins. Auch die geringe Größe von (derzeit) 33KB sowie die fehlerfreie unterstützung verschiedenster Gesten (Wisch-, Zoom, aber auch Hover, Maus, Keyboard, …) sprechen sehr für das Plugin.

Nachteile können höchstens in der unsicheren Zukunft des Plugins gesehen werden, da es den Anschein macht, dass Vladimir Agafonkin die einzige Person ist, die das komplette Konzept von Leaflet gestaltet und definiert. Da er jedoch derzeit für MapBox arbeitet, dessen Software sich auf Leaflet stützt, kann auch in der Zukunft von einer Unterstützung ausgegangen werden. Auch ist das Plugin inzwischen soweit ausgereift, dass es auch ohne zukünftigen Veränderungen bereits sehr stabil funktioniert und eine große Vielfalt an Funktionen bietet.

Diese Dokumentation legt seinen Fokus auf die Verwendung der MDV-internen entwickelten Plugins, deren Funktionsweise, sowie deren Verwendung.

2 Einrichten von Leaflet

Derzeit wird die Version 0.73 verwendet, da dies (seit November 2013) die letzte stabile Version des Plugins ist. Diese istüber die Leaflet-Homepage (www.leafletjs.com) frei verfügbar.

Da auf der Homepage von Leaflet (leafletjs.com) ausführliche Tutorials vorhanden sind (www.leafletjs.com/examples/quick-start.html), wird hier nicht auf die Installation von Leaflet eingegangen.

Eine ausführliche API findet sich auf www.leafletjs.com/reference.html.

Offiziell verfügbare Plugins werden auf der Seite ebenfalls referenziert: www.leafletjs.com/plugins.html

Die MDV internen Plugins sind in Mercurial eingecheckt und finden sich unter:

http://source.m.mdv/scm/hg/efa.Layouts/mdv_mdvLeaflet

3 Plugin-Übersicht

Die Plugins wurden anhand des neuen MVV – Layouts (MVV_NG) entwickelt und sind daher noch in der Anfangsphase. Ziel ist es, die Plugins so zu gestalten, dass diese für alle Kunden verwendet werden können und lediglich durch Konfigurationsobjekte angepasst werden (siehe Kapitel 4).

Es folgt eine Übersicht über alle verfügbaren Plugins, die im Mercurial eingecheckt sind (http://source.m.mdv/scm/hg/efa.Layouts/mdv_mdvLeaflet).

Um Kollisionen zu vermeiden, wurden alle Plugins in den Leaflet-Namensraum gefügt.

3.1 Anpassungen von Leaflet an die EFA

Um Leaflet für die EFA zu verwenden und auch die bereits vorhandenen Kacheln in Leaflet einbinden zu können, sind ein paar Anpassungen nötig. Damit bei jedem Kunden nicht jedesmal das Rad neu erfunden werden muss, erledigen dies bereits die ersten 3 Plugins.

3.1.1 mdvLeafletTools

mdvLeafletTools stellt eine Sammlung von Hilfsmethoden zur Verfügung, die von den anderen Plugins verwendet werden. Dies stellt sicher, dass dieselbe Funktionalität immer über eine zentrale Stelle (dieses Plugin) bearbeitet werden können und nicht überall verteilt geändert werden mssen.

Beispiel für eine der Funktionen ist die to_L_latlng(mdv_coord_str)-Methode, die von der EFA erhaltene Koordinaten („4468526.00000,826650.00000“) in das Leaflet-Koordinaten-Format (new L.LatLng(826650, 4468526);) umwandelt.

3.1.2 mdvLeafletMap

Dieses Plugin enthält Erweiterung der Leaflet-Map für die EFA, indem es die Karte instantiiert und wrappt. Zusätzlich bindet sich das Map-Plugin an das window Objekt und ist somit – sobald es eingebunden wurde – über window.LLMap zu erreichen.

Die Leaflet-Karte an sich ist über window.LLMap.map zu erreichen.

3.1.3 mdvLeafletExtension

Das Extension-Plugin enthält ebenfalls Erweiterungen der Leaflet-Map, allerdings fügt es diese direkt der Leaflet-Map hinzu anstatt sie zu wrappen. Im Unterschied zur mdvLeafletMap sorgt dieses Plugin dafür, dass Leaflet-Objekte durch mdv-spezifische Elemente erweitert werden.

Beispiel hierfür ist das Marker-Objekt (L.Marker), das zu einem L.MdvMarker erweitert wurde, um an jedem Marker-Objekt die dazugehörige Stateless-ID anzubinden und ein Popup-on-hover zu ermöglichen.

3.2 Request-spezifische Plugins

Sobald die Leaflet-Map über die im vorherigen Kapitel beschriebenen Plugins erweitert wurde, können request-spezifische Plugins verwendet werden.

3.2.1 mdvLeafletCoordInfo

Das größte Plugin kümmert sich um das Einfügen und Anzeigen von Haltestellen, POIs und sämtlichen anderen durch den COORD_REQUEST verfügbaren Typen.

Dies geschieht, indem sich das Plugin nach dem initieren auf das ‚moveend'-Event der Leaflet-Map horcht. Dadurch wird jedes Mal, wenn sich die Map verschoben, oder sich die Zoomstufe geändert hat ein COORD-Request an die EFA geschickt, um die jeweiligen Marker auf der Karte zu aktualisieren.

Hierbei kümmert es sich auch um das Löschen der Marker, die aus dem sichtbaren Bereich gelangen, um die Karte nicht unnötig langsam zu machen.

Welche Marker, ab welcher Zoom-Stufe und welche Typen generell angezeigt werden sollen wird über das Konfigurationsobjekt übergeben (siehe Kapitel 4).

3.2.2 mdvLeafletGeoObject

Dieses Plugin kümmert sich um das einzeichnen von Linienverläufen. Hierzu bietet es eine Schnittstelle an, um mit gezielten Methodenaufrufen eine Linie durch die übergebene Linien-Stateless-Id anzuzeigen bzw. wieder zu entfernen. Um die Haltestellen auf dem Linienverlauf zu zeigen, verwendet dieses Plugin das mdvLeafletCoordinfo-Plugin.

3.2.3 mdvLeafletTrip

Um eine bereits gerechnete Fahrt anzeigen zu können, verwendet man das mdvLeafletTrip-Plugin. Dieses lässt sich ebenfalls wie das GeoObject-Plugin durch public-Methoden ansteuern um einen von der EFA erhaltenen Trip in der Karte anzuzeigen.

3.3 Sonstige

3.3.1 mdvLeafletNetworkmap

Einen Ausnahmefall bietet die mdvLeafletNetworkmap. Diese stellt unabhängig zu der vorher verwendeten Karte eine eigene Konfiguration von Leaflet dar, die es ermöglicht, einen Netzwerkplan darzustellen und aus diesem eine Haltestelle zu wählen, die in ein ODV-Input-Feld eingefügt werden soll. Hierfür sind neben dem Netzwerkplan auch die Koordinaten der Haltestelle nötig.

4 Verwendung der Plugins

Wie bereits im vorherigen Kapitel beschrieben, stellen mdvLeafletTools, mdvLeafletMap und mdvLeafletExtension die Grundlage für die Verwendung von Leaflet für EFA-Layouts dar. Zusätzlich können die in Kapitel 3.2 und 3.3 beschriebenen Plugins eingebunden und verwendet werden.

Die Konfiguration der Plugins orientiert sich an dem grundsätzlichen Leaflet-Design. Dieses schreibt vor, dass beim Instantiieren eines Objekts die nötigen Parameter im Aufruf übergeben werden und zusätzlich ein optionales Optionen-Objekt, das mit den Keys die jeweiligen Konfigurationen und mit den Values die dazugehörigen Werte festlegt.

4.1 Grundsätzliche Konfiguration

4.1.1 Einbinden der Plugins

Um ein Plugin zu verwenden, muss der Quellcode natürlich in das HTML-Markup eingebunden werden.

```
<script src="/dev/js/vendor/leaflet-src.js"></script>
<script src="/mdv/mdvLeaflet/3rdParty/oms.min.js"></script>
<script src="/mdv/mdvLeaflet/mdvLeafletExtension-src.js"></script>
<script src="/mdv/mdvLeaflet/mdvLeafletMap-src.js"></script>
<script src="/mdv/mdvLeaflet/mdvLeafletTools-src.js"></script>
<script src="/mdv/mdvLeaflet/mdvLeafletNetworkmap-src.js"></script>
<script src="/mdv/mdvLeaflet/mdvLeafletCoordInfo-src.js"></script>
<script src="/mdv/mdvLeaflet/mdvLeafletGeoObject-src.js"></script>
<script src="/mdv/mdvLeaflet/mdvLeafletTrip-src.js"></script>
```

Die folgende Reihenfolge der Einbindung ist hierbei zu beachten:

1. Das Basis Leaflet-Plugin
2. mdvLeafletExtension
3. mdvLeafletMap
4. mdvLeafletTools
5. wahlweise die anderen Plugins in unabhängiger Reihenfolge, wobei darauf zu achten ist, dass mdvLeafletGeoObject und mdvLeafletTrip auf mdvLeafletCoordInfo aufbauen, wodurch letzteres als erstes eingebunden werden muss. CoordInfo wiederum verwendet das Spiderfier-Plugin, das ebenfalls davor eingebunden werden muss.

4.1.2 Initiieren der Plugins

Zunächst muss die mdvLeafletMap instantiiert werden, danach folgen die anderen Plugins in unabhängiger Reihenfolge (auch mdvLeafletGeoObject/Trip).

Nach dem Instantiieren kann man sich auf die von den Plugins bereitgestellten Events subscriben um kundenspezifisch darauf zu reagieren. So kann zb. ein eigenes Map-Kontextmenü gefüllt, oder Kontroll-Elemente modifiziert werden, sobald ein Linienverlauf in die Karte eingezeichnet wurde.

```
mvv.init_map = function () {
    // #if DEBUG is True
    console.log('mvv.init_map');
    // #endif
    // MAP GENERAL
    this.llmap = new LLMap('map', mvv.conf.map);

    this.llmap.map
            // pass through click event to suggestion list
            .on('mousedown', mvv.on_map_mousedown)
            // map context menu
            .on('contextmenu', mvv.mapcb.context_menu.bind(this));

    // COORDINFO
    this.llmap.llcoordinfo = new L.LLCoordInfo(this.llmap.map, mvv.conf.coordinfo);
    this.llmap.llcoordinfo
            .on('marker_changed', mvv.mapcb.marker_changed.bind(this))
            .on('popup', mvv.mapcb.coord_popup.bind(this))
            .reset_markers();  // manually trigger inital rendering
    // LVP
    this.llmap.llgeoobjects = new L.LLGeoObject(this.llmap.map, mvv.conf.geoobject);
    this.llmap.llgeoobjects
            .on('geo_render_complete', mvv.mapcb.geo_render_complete.bind(this))
            .on('geo_remove_complete', mvv.mapcb.geo_remove_complete.bind(this))
            .on('markerpopup', mvv.mapcb.geo_marker_popup.bind(this))
            .on('linepopup', mvv.mapcb.geo_line_popup.bind(this));

    // TRIP VISUALISATION
    this.llmap.lltrip = new L.LLTrip(this.llmap.map, mvv.conf.lltrip);
    this.llmap.lltrip
            .on('trip_render_complete', mvv.mapcb.trip_render_complete.bind(this))
            .on('trip_remove_complete', mvv.mapcb.trip_remove_complete.bind(this))
            .on('popup', mvv.mapcb.trip_popup.bind(this));
};
```

In dieser init_map Methode werden beim MVV alle nötigen Funktionalitäten an die Plugins gebunden. Die Konfiguration der Plugins wurde in ein eigenes Config-File verlagert.

Diese Konfigurationen sind normale Key-Value Objekte, die das Verhalten des Plugins spezifizieren.

```
mvv.conf.coordinfo = {
    marker : {
        POI_POINT : {
            min_zoom : 0,
            icon_func : mvv.conf.get_poi_icon,
            filter : mvv.conf.poi_filter_function
        },
        POI_AREA : {
            min_zoom : 0,
            icon_func : mvv.conf.get_poi_icon
        },
        STOP : {
            min_zoom : 12,
            icon_func : mvv.conf.get_stop_icon,
            icon_change_lvl : 16  // not used by plugin, but in get_stop_icon func
        }
    },
    showPopupOnMouseOver : false,
    requestPrefix : '/mvv',
    initial_rendering : false
    // #if DEBUG is True
    , showPolygons : false
    // #endif
};
```

In diesem Beispiel wird das CoordInfo-Plugin so angepasst, dass es sowohl Haltestellen, als auch POIs anzeigt, jeweils mit MVV-spezifischen Marker-Icons und ab gewissen Zoomstufen.

4.2 Plugin-Spezifische Parameter

Es folgt eine Übersicht über alle Parameter der jeweiligen Plugins und eine Beschreibung deren Auswirkungen auf die Funktionalität. mdvLeafletTools und mdvLeafletExtension können nicht konfiguriert werden, da diese nur Funktionen bereitstellen.

4.2.1 mdvLeafletMap

TODO Parameterliste als Tabelle
TODO parameterliste

Die MVV-spezifische Map-Konfiguration sieht folgendermaßen aus:

```
mvv.conf.map = {
    // latlng -> [y, x]
    center : [825574, 4460333],
    // initial zoom level
    zoom : 6,
    // tile layers : base and overlay
    layers : {
        // base layers : mutually exclusive
        base : [
            {
                attribution : i18n.map_attr,
                // used as description, if L.control.layers is used with another base layer
                // e.g ortho
                caption : 'Standard',
                // make this a  L.MdvTileLayer
                type : 'mdv',
                tilepath : 'http://cockpit.mvv-muenchen.de:8888/Maps/tiles/{q}',
                minZoom : 0,
                maxZoom : 17,
                // realOffsetY from mdv config
                tilingOffsetY : 600000,
                // realOffsetX from mdv config
                tilingOffsetX : 4000000,
                // coverage of the map layer - bottomleft, topright
                // [(realOffsetY + realHeight), realOffsetX]
                southWest : [1124287, 4000000],
                // [(realOffsetY, (realOffsetX + realWidth)]
                northEast : [600000, 5048575]
            }
        ],
        overlay : []
    },
    // coordinate reference system
    // see http://leafletjs.com/reference.html#icrs
    crs : {
        // instantiate L.MdvCRS
        mdv : true,
        // unused so far
        code : 'MVTT',
        // realWidth / (numTilesX * tileSize), starting from zoomlevel 0
        resolutions : [
            204.7998046875,
            144.8153935185,
            102.3999399038,
            72.4076967593,
            51.1999797078,
            36.2038483796,
            25.5999795752,
            18.1019241898,
            12.7999897876,
            9.0509620949,
            6.3999974427,
            4.5254810475,
            3.1999993602,
            2.2627411154,
            1.5999980805,
            1.1313724026,
            0.8000004800,
            0.5656862013
        ]
    },
    zoomControl : false
};
```

4.2.2 mdvLeafletCoordInfo

TODO Parameterliste als Tabelle

// DEBUG PARAMETERS

debugName : ('LLCoordInfo') name that is used for debug-outputs

showPolygons : (false) visualize the new areas that are filled with markers

// GENERAL PARAMETERS

bufferSize : (5000) width of the buffer frame outside of the visible area.
this parameter assures that markers are not instantly removed
from the map whenever they get out of the current window.
reason: whenever a marker is added to the map, its position is
calculated whenever the map is moved, although it can be
outside the visual frame. if the buffer size is too big, the
map can slow down as the position of too many unnecessary
markers are calculated. on the other hand it is useful to keep
a buffered size outside the visual frame to avoid too much
redrawings of markers.

iconChange : (true) tell the plugin whether icons can change on different
zoom levels or not. if they change, the plugin has to check on
every zoomchange whether the randomly chosen reference icon
has changed or not. by setting this parameter to false (-> you
know that your marker-icons do not change on different zoom
levels) this check is disabled and performance is increased.

initialRendering: (true) render markers on initialization of CoordInfo-Plugin.
this can also be handled manually when you set this option to
false and call the public plugin-method 'reset_markers' after
initialization. this should be done if the 'marker-changed'-
events are required on startup.

marker : definition of all markes that should be displayed. the key of
every marker definition equals the type of the marker as it is
requested on the EFA (e.g. 'STOP' or 'POI_POINT'). the value
is an object with the type-specific definitions.
these include:
filter : (null) a filter function that can exclude coord-
infos although they are in the coord-info response

group : (L.markerGroup) a leaflet-group the markers are added to. Modify this parameter only if you are sure what you are doing.

icon_func : (function that returns the default leaflet-marker-icon) function that returns a Leaflet-Icon. This function is called with the current zoom level and the specific marker information, whenever a marker is rendered. Keep sure that this function is performant as it can slow down the whole render-process and though the look and feel of your map!

max_zoom : (this.llmap.getMaxZoom()) maximal zoom-level the markers are added to the map. maximal zoom-level in leaflet describes the detailmost zoom of the map.

min_zoom : (0) minimal zoom-level the markers are added to the map. minimal zoom-level in leaflet describes the farmost zoom of the map.

default marker definition:

```
{
  POI_POINT : {
     max_zoom : this.llmap.getMaxZoom(),
     min_zoom : 0,
     group : L.markerGroup(),
     filter : null,
     icon_func : function () {
        return defaultMarkerIcon;
     }
  },
  STOP : {
     max_zoom : this.llmap.getMaxZoom(),
     min_zoom : 0,
     group : L.markerGroup(),
     filter : null,
     icon_func : function () {
        return defaultMarkerIcon;
```

```
            }
          }
        }
oms          : (new OverlappingMarkerSpiderfier with:
           nearbyDistance : 20, keepSpiderfied :  true)
           OverlappingMarkerSpiderfier is a leaflet-plugin that shifts
           overlapping markers away whenever one of the markers is
           clicked to allow the user an easier selection of the marker.
           Only modify this option if you know that you are doing!
processingTime  : (100) time (in ms) of rendering process before a break is
           taken. this break assures that all other map functionalities
           can be handled after that time
processBreak    : (15) time (in ms) of the break between two rendering
           processes (minimum 15 ms). This time is thought for other map
           functionalities like zooming/moving... .
render_asynchronous : (true) influences the way the given marker-information is
           rendered. if rendered asynchronous, the plugin takes a render-
           break of $processBreak ms after $processingTime ms rendering.
           disable this option whenever you know that only a small amount
           of markers should be rendered. this option assures that other
           map functionalities are still available while the quite long
           and blocking marker rendering process.
requestPrefix   : ('') prefix that should be used for the coord-info-request.
           this prefix is added to the fix '/XSLT_COORD_REQUEST?' string.
showPopupOnMouseOver : (true) show a marker's popup on mouse hover. otherwise
           the popups are shown when the marker gets clicked
staticData      : (false) pass static marker information to the plugin instead
           of firing a coord-request. this information needs to be in the
           same style as the response of the coord-request! if this value
           is falsy, the marker-information is requested normally by the
           coord-info-efa request.
```

4.2.3 mdvLeafletGeoObject

TODO Parameterliste als Tabelle

```
// DEBUG PARAMETERS
debugName       : ('LLGeoObject') name that is used for debug-outputs

// GENERAL PARAMETERS
colors          : (null) function that returns the color code (format:'#3DBBE4')
                  for a specific mot-id.
defaultColor    : ('#000000') color that is used if $colors is not defined or
                  $colors returns a falsy value
dir_escape      : (null) object to determines how the direction information can
                  be filtered out of a stateless id. requires the following
                  format:
                  {
                    regex: regular expresion to find the
                           direction definition in a
                           given stateless-id,
                    replace : string the direction def.
                              should be replaced with
                  }
drawMarkers     : (true) draw stop-icons for each line or not
markerOptions   : dictionary with settings for the LLCoordInfo-plugin that cares
                  about the stop-marker render-process.
                  default value:
                  {
                    showPopupOnMouseOver : true,
                    iconChange      : false,
                    bufferOnHide    : 'STOP',
                    marker          : {
                       STOP    : {
                          icon_func: function () {
                             return new L.Marker().options.icon;
                          }
                       }
                    }
```

```
                // #if DEBUG is True
                , debugName      : 'LLCoordInfoLineStops'
                // #endif
              }
requestPrefix   : ('') prefix that should be used for the geoobject-request.
              this prefix is added to the fix '/XSLT_GEOOBJECT_REQUEST?'
```

4.2.4 mdvLeafletNetworkmap

TODO Paramterliste als Tabelle

TODO liste

4.2.5 mdvLeafletTrip

TODO Parameterliste als Tabelle

```
// DEBUG PARAMETERS
debugName       : ('LLTrip') name that is used for debug-outputs


// GENERAL PARAMETERS
colors          : (null) function that returns the color code (format:'#3DBBE4')
              for a specific mot-id.
defaultColor    : ('#000000') color that is used if $colors is not defined or
              $colors returns a falsy value
drawMarkers     : (true) draw stop-icons
interchangeDashValue: (null), set this parameter to 'x, y' when interchanges
              should be displayed in dashed lines. x and y describe the
              length of the dashes and the distances between them (see
              leaflet doku for dashed polylines)
markDestination : (true) mark the finishing-point/stop of the whole trip by an
              additional marker
markerOptions   : dictionary with settings for the LLCoordInfo-plugin that cares
              about the stop-marker render-process.
              default value:
              {
                showPopupOnMouseOver : true,
                iconChange      : false,
                bufferOnHide    : 'STOP',
                marker          : {
```

```
                STOP    : {
                    icon_func: function () {
                        return new L.Marker().options.icon;
                    }
                }
            }
            // #if DEBUG is True
            , debugName       : 'LLCoordInfoTripChanges'
            // #endif
        };
markOrigin      : (true) mark the starting-point/stop of the whole trip by an
                  additional marker
mind_stopovers  : (true) mind the stopover-information in the given data. this
                  allows to display the stopovers of the trip afterwards. if you
                  disable this functionality, the performance is increased as
                  the stopover-marker are not rendered
requestPrefix   : ('') prefix that should be used for the coord-info-request.
                  this prefix is added to the fix '/XSLT_TRIP_REQUEST2?' string.
                  WARNING! This plugin is not able to send requests up to now.
                        This parameter has no functionality though.
showDestStops   : (true) display the final-stop of every partial trip
showOrigStops   : (true) display the start-stop of every partial trip
showPolyLineNumbers : (false) display the corresponding mot number next to the
                  drawn polyline.
splitUp         : (true) split up the whole trip in its partial trips. if set
                        false, the whole trip is shown in one
```