Working-Paper: Google Maps API

- Eine Schnittstellenbeschreibung von mu21.de -

1 DRAFT: GOOGLE MAPS API (RAW INFORMATIONS) 2
2 VERWENDUNG VON GOOGLE MAPS API 4
3 STRAßENKARTEN, SATELLITENBILDER: EXAMPLES MU21-GOOGLE-MAPS-API 6
4 ROUTENPLANER (DIRECTIONS ELEMENTS): MU21-ROUTENPLANER-GOOGLE-MAPS 9
5 GOOGLE MAPS API MIT MYSQL 11
6 POST: GOOGLE MAPS API AUF MU21.DE 15
7 POST: GOOGLE MAPS API TEMPLATES 17
8 POST: GOOGLE MAPS API MYSQL TEMPLATES 19
9 CODE 1: MU21-GOOGLE-MAPS-API.HTML 20
10 CODE 2: MU21-ROUTENPLANER-GOOGLE-MAPS.HTML 23
11 CODE 3: MU21-GOOGLE-MAPS-API-MYSQL-EXAMPLES.PHP 25
12 QUERVERWEISE 30

Google Maps API mySQL Templates (Post)
Google Maps API Templates by mu21.de (Post)
Google Maps API auf mu21.de (Post)
Verwendung von Google Maps API (Code Archive)
mu21-google-maps-API.html…(Code Archive)
mu21-routenplaner-google-maps.html… (Code Archive)
mu21-google-maps-API-mySQL.php… (Code Archive)
Schwedenreise (praktische Anwendung der Google Maps API)

mu21.de (whois): Michael Uhl · Blumenstraße 28 · 91489 Wilhelmsdorf

1 Draft: Google Maps API (Raw Informations)

- Google Maps API (Online Documentation: http://www.google.com/apis/maps/documentation/)
- Erklärung API (Application Programming Interface, deutsch (etwa): Schnittstelle zur Anwendungsprogrammierung)
- Erläuterung Möglichkeiten / Copyright google maps
  Insbesondere Cobyright (http://www.google.com/apis/maps/signup.html)

  Here are some highlights from the terms for those of you who aren't lawyers:

  - There is no limit on the number of page views you may generate per day using the Maps API. However, if you expect more than 500,000 page views per day, please contact us in advance so we can provision additional capacity to handle your traffic. Otherwise your quality of service may be degraded.
  - There is a limit of 50,000 geocode requests per day per Maps API key. This translates to roughly one geocode request every 1.73 seconds. If you go over this 24-hour limit, the Maps API geocoder may stop working for you temporarily. If you continue to abuse this limit, your access to the Maps API geocoder may be blocked permanently.
  - The Maps API does not include advertising. If we ever decide to change this policy, we will give you at least 90 days notice via the Google Maps API Blog.
  - Your service must be freely accessible to end users. To use Google mapping technology in other types of applications, please use Google Maps for Enterprise.
  - You may not alter or obscure the logos or attribution on the map.
  - Google will upgrade the API periodically, and you must update your site to use the new versions of the API. The Maps team will notify you of updates on the Google Maps API Blog. If we make a non-backwards compatible change, we will give you at least a month's notice to make the transition, during which both versions of the API will be available.
  - There are some uses of the API that we just don't want to see. For instance, we do not want to see maps that identify the places to buy illegal drugs in a city, or any similar illegal activity. We also want to respect people's privacy, so the API should not be used to identify private information about private individuals. Remember that we reserve the right to suspend or terminate your use of the service at any time, so feel free to contact us before you do all the integration work if you are unsure of whether your implementation will meet our Terms of Use.

- Erstellung einfacher Straßenkarten, Satellitenbilder, Marker und Polylines (http://www.mu21.de/index.php/code-archive/mu21-google-maps-api/):

  - xmlns:v="urn:schemas-microsoft-com:vml" für IE
  - Key bitte nicht kopieren!
  - <style type=”text/css”> zur einheitlichen Darstellung (siehe mu21.de)
  - Street View (map1):^
    - map.addControl (Navigationselemente hinzufügen)
    - map.setCenter (erwartet GLatLng(Latitude und Longitude Angabe dezimal N , E) und „Höhenangabe“)
    - Polyline (Start- und Endpunkt, Farbe, Linienstärke) auch umfangreichere Bereichskennzeichnung möglich: http://www.mu21.de/schwedenreise/Karte.htm
    - Marker: Funktion createMarker(point, number) erklä<ren, anschl. Funktionsaufruf „map.addOverlay“ beschreiben
  - Satellite-View (map2):
    - Hinweis: var map = new GMap2(document.getElementById(”map2”));
    - Hinweis: map.setCenter(new GLatLng(49.56583, +10.733), 10, G_SATELLITE_MAP);
    - Polyline und Marker wie map1
    - Hinweis Farbangabe: hexacodes

- Body:
  - Funktionen “onload” angeben: onload=”load()” onunload=”GUnload()
  - Karte darstellen mittels “ <div id=”map1” style= width: 500px; height: 300px”> “ → Angabe „id“, wie im Header deklariert. → Darstellung von mehreren Karten!

- Latitude und Longitude erklären
- Kartengröße (style=”width: XXXpx; height: XXXpx”) und map.setCenter
- Routenplaner (hier: Straßenkarte und Streckenbeschreibung) mit den Directions Elements (http://www.mu21.de/index.php/code-archive/mu21-routenplaner-google-maps/):

  - Wichtig: Key
  - Error Codes
  - function onGDirectionsLoad()
  - …

2 Verwendung von Google Maps API

(→ Verwendung von Google Maps API)

„Abmahnungen: 600 Euro für ein Brötchen-Bild" unter diesem Titel veröffentlichte Spiegel Online[1] Ende Mai 2007 eine Gesichte, die Wochen zuvor bereites in der Blog-Szene hitzig diskutiert wurde. Dieser Artikel zeigt nicht zuletzt die Copyright- und (bei Verletzungen die damit verbundene) Abmahn-Problematik auf, die sich gegenwärtig jeder seriöse Webmaster stellen muss. Besonders kritisch erscheint die Verwendung von Kartenmaterial (Straßenkarten, Luftbilder, etc.), da die Rechte an den Landkarten, welche digital mit Programmen wie Map Point angezeigt werden, in den meisten Fällen gar nicht bei den Herstellern der Programme liegen. Karten aus Microsoft Map Point dürfen lt. Microsoft nicht mehr max. 1000 Mal weitervertrieben werden (siehe: Urheberrecht - Bilder[2]). Diese Einschränkung gilt ebenfalls für Karten, die auf Webseiten gezeigt werden (also, Counter einfügen! Nach 1000 Page Impressions ist Schluss! Und wehe der 1001te Besucher bekommt die Karte zu Gesicht!). Die Einbindung solcher Karten auf der eigenen Webseite stellt also keine wirkliche Alternative dar! Aber keine Sorge:

Der Freund eines jeden Webmasters kennt Abhilfe: Google Maps API[3]

Mit einigen Kenntnissen in Java Script und objektorientierter Programmierung (keine Angst, Grundverständnis von Klassen in C++ reicht aus!) ist es ein leichtes ansprechende Karten zu erstellen und diese auf der Webseite zu veröffentlichen! Und das gute dabei, um sämtliche Urheberrechtsbelange kümmert sich Google. Um Google Maps API nutzten zu können muss lediglich ein "Maps API key"[4] beantragt werden. Und zwar für jedes Verzeichnis auf dem Webserver, in dem sich Seiten mit Kartenmaterial befinden. Neben ein paar grundlegenden Regeln zur Nutzung, die sich sowieso von selbst verstehen, ergibt sich die einzige "Einschränkung" in der Nutzung von Geocode Daten (Routenplanung). Pro 24 Stunden sind pro Key und Tag "nur" 50.000 Anfragen erlaubt (also Vorsicht! Nur alle 1,728 Sekunden eine neue Route berechnen lassen ;-)! Sonst ist für den Rest des Tages Schicht im Schacht mit Routenplanung!).

Wie bereits erwähnt, stellt Google dem Webmaster die Kartendaten via API [5](Application Programming Interface, deutsch (etwa): Schnittstelle zur Anwendungsprogrammierung) zur Verfügung. Anhand dieser API ist es dem Anwender nun möglich individuelle Straßenkarten, Satellitenbilder und sogar Daten zur Routenplanung (Wegbeschreibungen) zu erstellen und anzubieten. Das wichtigste bei der Gestaltung (Programmierung) der Karten ist die genaue Kenntnis der Koordinaten des darzustellenden Gebietes (Ortes). Anzugeben sind dabei die Longitude und die Latitude in dezimaler Form (Norden und Osten als positive Zählrichtung). Die beiden Winkelmaße lassen sich am einfachsten mit dem Location Sensor aus Map Point gewinnen (Umrechnung nicht vergessen!). Alternativ können diese Angaben auch mit dem Online Service von Multimap[6] ermittelt werden (für interessierte Leser, die Freude an dieser Thematik haben, empfehle ich meine Abhandlung zum Thema „Realisierung eines Ortungssystems mit GPS"). Ebenfalls sollte man sich bereits vor der Erstellung darüber im Klaren sein, wie viel Raum man der Karte auf der Internetseite zubilligen möchte. Schließlich hängt davon die erforderliche Höhenangabe im Code ab.

Also: Für den gewünschten Kartenausschnitt sind für dessen Mittelpunkt die beiden Winkelmaße (map.setCenter) und anhand der Größe der Karte (width: XXXpx, height: XXXpx) der "Maßstab" (Übergabeparameter der Funktion GLatLng: 1 (Blick aus dem Weltall) bis 17 (Flughöhe ca. 50 m)) zu ermitteln.

So, los geht's!

Erstellung einfacher Straßenkarten, Satellitenbilder, Marker und Polylines:

mu21.de Google Maps API (Examples)[7]

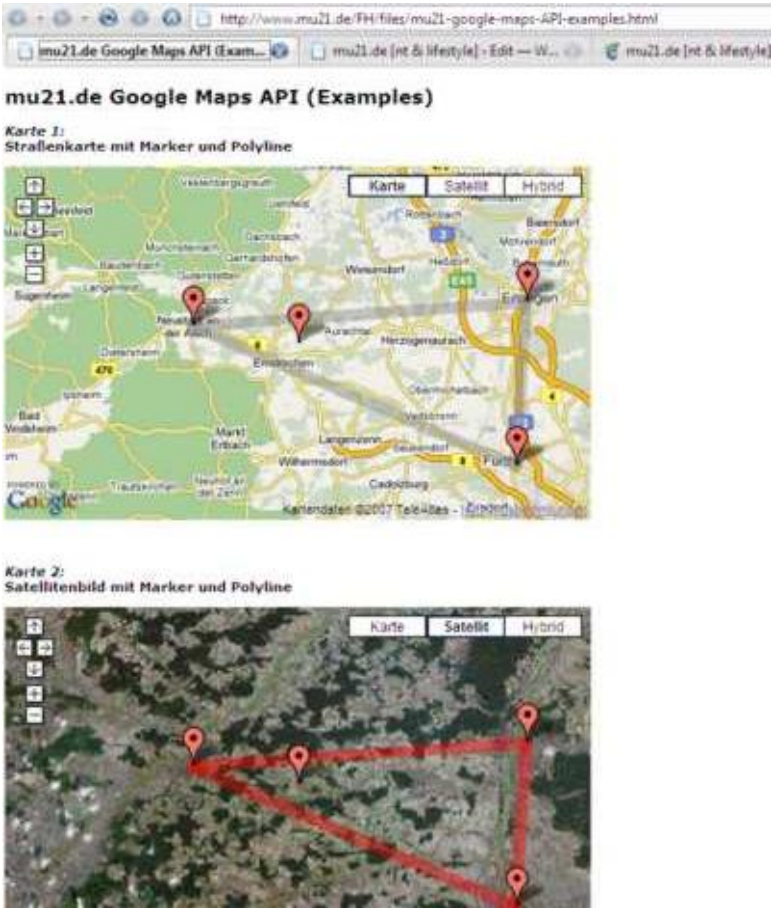
Nutzung als Routenplaner (hier: Straßenkarte und Streckenbeschreibung) mit den Directions Elements:

mu21.de Google Maps API Directions Elements (Example)[8]

3 Straßenkarten, Satellitenbilder: Examples mu21-google-maps-API

(→ mu21-google-maps-API.html…)

…Straßenkarte, Satellitenbild, Marker und Polyline mit Google Maps API - oder: Codeschnipsel 1 des mu21.de-web-2.0-Standards:[9]



- Abbildung 1: mu21.de Google Maps API (Examples) -

Vorab ein paar Erläuterungen zum nachfolgende Code:

Neben der korrekten Nennung des Coding Schemas *(xmlns="http://www.w3.org/1999/xhtml")* nach W3C ist es für die einwandfreie Darstellung im Internet Explorer erforderlich ebenfalls das Microsoft Coding Schema mit anzugeben *(xmlns:v="urn:schemas-microsoft-com:vml")*. Damit kommt es leider zu einem Fehler bei der W3C-Validierung. Aber ohne diese Angabe: Keine Anzeige im IE!

Im Header der Datei wird nun Google der Nutzer mit dem "Maps API key" mitgeteilt:
<script src="http://maps.google.com/maps?file=api&v=2.x&key=[invalid - please sign up]" type="text/javascript"></script>)

Hier habe ich „meinen" Key gelöscht. Auf den Original-Seiten ist er natürlich vorhanden. Aber diesen bitte nicht verwenden, sondern selbst einen Key beantragen (siehe: Verwendung von Google Maps API[10]). Anschließend habe ich der noch etwas css für die Optik spendiert (muss nicht unbedingt sein).

Die eigentliche Arbeit beginnt mit *function load()*: Zuerst eine Abfrage, ob der aufrufende Browser die Seiten anzeigen kann. Hierzu stellt Google die Funktion *GBrowserIsCompatible()* zur Verfügung. Diese stellt bereits eine Fehlerbehandlung zur Verfügung. Darum müssen wir uns also an dieser Stelle nicht kümmern.

Zunächst erzeuge ich eine Straßenkarte (map1): *var map = new GMap2(document.getElementById("map1"));*. Den Benutzer möchte ich die Navigation (Verschieben, Zoom) ermöglichen: *map.addControl(new GSmallMapControl()); map.addControl(new GMapTypeControl());*. Anschließend noch *map.setCenter(new GLatLng(49.56583, +10.733), 10);* - und schon ist die erste Straßenkarte fertig! Wie bereits auf der Seite „Verwendung von Google Maps API" erwähnt, erwartet die Funktion GLatLng als Übergabewerte die nördliche Breite, die östliche Länge und die Höhenangabe (1 (grob) - 17 (detailliert)).

Damit die Karte nun nicht so langweilig daher kommt, zeichnen wir ein bisschen darin herum. Um ein bestimmtes Gebiet hervorzuheben bieten sich Polylines an. Für eine Gerade zwischen zwei Punkten benötigen wir die Koordinaten des Start- und des Endpunktes. Daneben erwartet die Funktion *new GPolyline* noch die Angabe der Farbe (mittels Hexacode) und der Linienstärke. Durch Angabe mehrere Punkte lassen sich somit auch komplexere Gebilde hervorheben (siehe Karte Schwedenreise).

Immer noch zu langweilig? Stimmt! Orte wollen wir schließlich ebenfalls kennzeichnen. Hiefür bieten sich Marker exzellent an. Falls die Karte mehrere Marker enthalten soll, bietet es sich (schon alleine aus Bequemlichkeit) an eine Funktion hierfür zu schreiben: *function createMarker(point, number)*. Als Übergabewerte werden dabei die Koordinaten und die Bezeichnung, die erscheinen soll, erwartet. Der Aufruf erfolgt beispielsweise mit: *map.addOverlay(createMarker(new GLatLng(49.5797, +10.6103), 'NEA'));*.

Somit haben wir nun eine durchaus informative Straßenkarte erstellt. Damit der Browser diese auch darstellen kann, muss das Objekt noch in den body der HTML-Datei eingebunden werden. Zu beachten ist dabei der Tag *<body onload="load()" onunload="GUnload()">* anstatt einfach nur *<body>*. Nun * * einfügen und HTML im Browser laden.

Das ganze funktioniert natürlich auch mit Satellitenfotos. Zu Demonstrationszwecken habe ich einfach eine neue Karte mit *var map = new GMap2(document.getElementById("map2";)); map.addControl(new GSmallMapControl()); map.addControl(new GMapTypeControl()); map.setCenter(new GLatLng(49.56583, +10.733), 10, G_SATELLITE_MAP);* generiert. Wichtig ist dabei, die Angabe G_SATELLITE_MAP. Bei mehreren Karten auf einer Seite ist dabei zu beachten, dass unterschiedliche IDs für die Karten vergeben werden *(vgl. map1, map2)*.

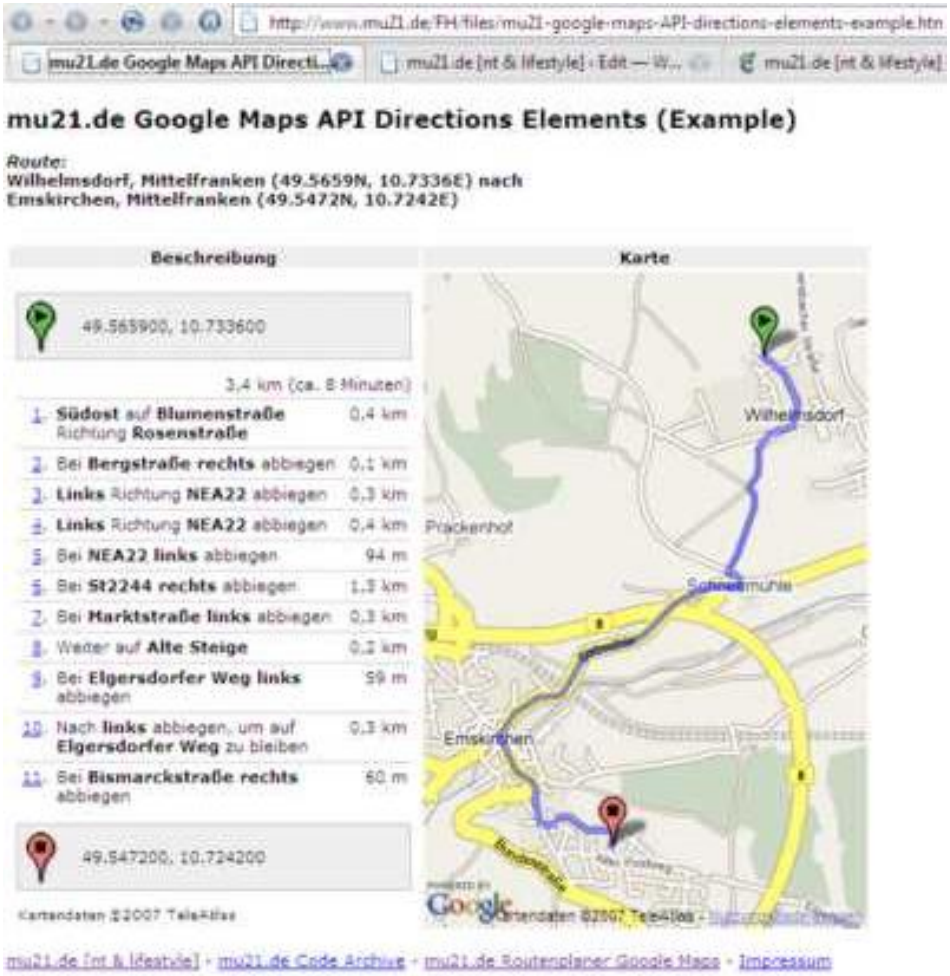
[→ Code 1]

Die Anzeige des oben genannten Codes ist auf der Seite mu21.de Google Maps API (Examples) zu bewundern. Wie bereits mehrfach erwähnt, den „Maps API key" von dieser Seite unter keinen Umständen kopieren! Falls doch: Eins ist sicher, Google kommt euch auf die Schliche (bei der Nutzung der API wird die aufrufende Domain an Google übertragen)! Also, bitte selbst beantragen!

4 Routenplaner (Directions Elements): mu21-routenplaner-google-maps

(→ mu21-routenplaner-google-maps.html…)

…Routenplaner (hier: Straßenkarte und Streckenbeschreibung) mit den Directions Elements von Google Maps API - oder: Codeschnipsel 2 des mu21.de-web-2.0-Standards:[11]



- Abbildung 2: mu21.de Google Maps API Directions Elements (Example) -

Wie gehabt, vorab einige Hinweise zum nachfolgenden Code:

Bis zur Zeile *//<![CDATA[* bitte ich die Hinweise zum Code auf der Seite mu21-google-maps-API.html zu beachten (insbesondere zum „Maps API key").

Am Anfang erfolg die Deklaration der Elemente *map* (die Straßenkarte), *gdir* (die Route), *geocoder* (die (berechneten) Geocoder Daten, gleichzeitige Initialisierung) und die Marker für den Start und Zielpunkt (*var addressMarker;*).

Mit der Funktion *load()* wird die Browser-Kompatibilität geprüft, die Karte (*map = new GMap2(document.getElementById("map"));*) und die Beschreibung *gdir = new GDirections* erzeugt sowie der Start- und der Zielpunkt sowie die Sprache der Navigation eingestellt. Laut Spezifikation der Funktion *setDirections* würde es

sogar reichen die postalischen Adressen der Navigationspunkte anzugeben. Erfahrungsgemäß hat sich allerdings erwiesen, diese für Europa im Winkelmaß anzugeben (siehe hierzu: Verwendung von Google Maps API).

Mit *function handleErrors()* wird das Error-Handling realisiert (Erklärung hierzu: siehe jeweilige Fehlerbeschreibung im Code (*alert*)).

Mit der Funktion *function onGDirectionsLoad()* werden die Geocoder Daten geholt. Zusätzlich können zum Debugen darin eigene Funktionen angegeben werden, um z.B. an Informationen der *load()*-Funktion heranzukommen.

Um die Ergebnisse der Navigation zu präsentieren, habe ich das ganze innerhalb des Bodys in eine Tabelle gepackt. Umfangreichere Streckenbeschreibungen können somit immer noch übersichtlich dargestellt werden (siehe Schwedenreise Reiseverlauf).

[→ Code 2]

Die Anzeige des oben genannten Codes ist auf der mu21.de Google Maps API Directions Elements (Example) zu bewundern. Wie bereits mehrfach erwähnt, den „Maps API key" von dieser Seite unter keinen Umständen kopieren! Falls doch: Eins ist sicher, Google kommt euch auf die Schliche (bei der Nutzung der API wird die aufrufende Domain an Google übertragen)! Also, bitte selbst beantragen!

5 Google Maps API mit mySQL

(→ mu21-google-maps-API-mySQL.php…)

…Umgebungskarte mit Marker, Straßenkarte und Satellitenbild mit Marker und Polylines dargestellt unter Nutzung der Google Maps API. Die Übergabewerte für die Funktionen werden mittels mySQL-Abfragen generiert. Codeschnipsel 3 des mu21.de-web-2.0-Standards[12]:

- Abbildung 3: mu21.de Google Maps API with mySQL (Examples) -

Wie gehabt, vorab einige Hinweise zum nachfolgenden Code:

An dieser Stelle weiße ich allerdings ausschließlich auf die Besonderheiten hin, die mit der Benutzung einer mySQL-Datenbank entstehen bzw. auf einige „Stolperfallen" bei der php-Programmierung. Eine ausführliche Beschreibung zur

Google Maps API Programmierung stelle ich auf den Seiten "mu21-google-maps-API.html" und "mu21-routenplaner-google-maps.html" zur Verfügung!

Wir beginnen also ab der Kommentar-Zeile //–MAP1-Single-View-(MySQL)–//:

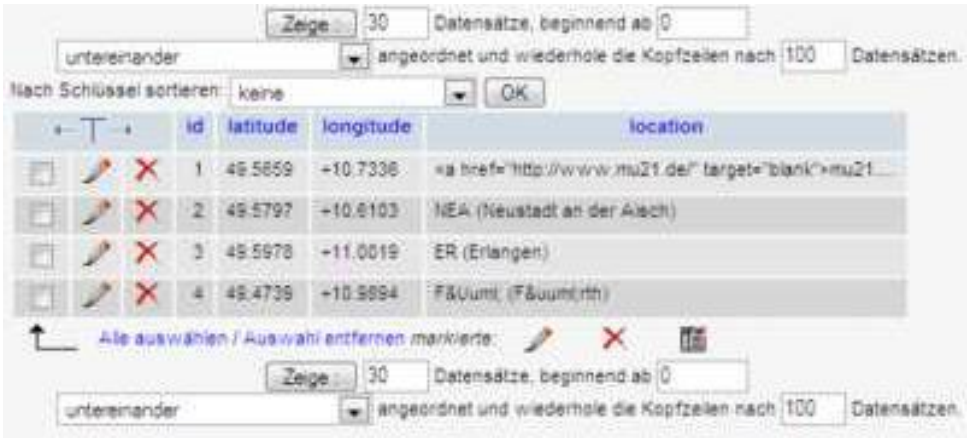
Zunächst muss der mySQL-Server, die Datenbank, der Benutzername und das Passwort mitgeteilt und verbunden werden. Zweckmäßigerweise wird hierzu eine Datei mit diesen Daten inkludiert (hier "mu21-sql-config.php"):

```
//--------------------Datei-mu21-sql-config-------------------//

<?php
$db_server = "db.mySQL-srv.example"; //die Adresse des mySQL-Servers
$db_database = "db.database"; //der Name der Datenbank
$db_account = "db.username"; //der Username zum Verbinden mit der Datenbank
$db_password = "db.password"; //das zugehörige Passwort

mysql_connect($db_server,$db_account,$db_password);
mysql_select_db($db_database);
?>
```

Anschließend wird der SQL-Befehl SELECT in der php-Variablen $query gespeichert. Die zu selektierende Tabelle *maps* ist folgendermaßen aufgebaut[13]:



- Abbildung 4: mySQL Tabelle "maps" (phpMyAdmin) -

Spalte "id" mit dem Primärschlüssel zur eindeutigen Zuordnung der Datensätze, die Spalten "latitude", "longitude" und "location" beschreiben den geographischen Ort mit seinen Koordinaten und den zugehörigen Namen:

```
id int(255) NOT NULL auto_increment,
latitude varchar(8) NOT NULL,
longitude varchar(8) NOT NULL,
location varchar(100) NOT NULL,
PRIMARY KEY (id)
```

Da wir an dieser Stelle eine Umgebungskarte darstellen möchten, benötigen wir auch nur einen Datensatz. Diesen wählen wir in der Anfrage mit "WHERE id=1" (1. Datensatz). Mit dem Befehl mysql_query wird die Anfrage an die SQL-Datenbank geschickt und in $resault gespeichert. Mittels der nachfolgenden while-Schleife wird der Datensatz, der mittels "mysql_fetch_array" zur Verfügung steht, in den php-Variablen $mapid, $lat, $long und $loc gespeichert. Der Datensatz steht nun für den PHP-Interpreter bereit. Um den Speicher der Variablen $resault wieder freizugeben wird der Befehl "mysql_free_result" ausgeführt. Außerdem lässt sich somit die Sicherheit des Skriptes zu erhöhen. Im Anschluss wird die Verbindung zu mySQL geschlossen.

Im Anschluss wird eine Karte mit den entsprechenden Kontrollelementen erzeugt ("echo 'var map = new GMap2…"). Da die Google Maps API mit Java-Script angesprochen wird, wird nur soviel php-Code verwendet wie unbedingt nötig. Dadurch lässt sich die Geschwindigkeit des Skripts steigern. Zweckmäßigerweise wird eine Umgebungskarte nach den im Datensatz hinterlegten Koordinaten zentriert ("echo "map.setCenter…"). Um den Ort zu kennzeichnen wird noch ein Marker in der Karte benötigt (siehe: "mu21-google-maps-API.html"). Hierfür wird die Zeile "<?php echo "map.addOverlay(createMarker(new GLatLng("; echo $lat; echo ", "; echo $long; echo "), '"; echo $loc; echo "'));";?>" benötigt.

Somit ist die Umgebungskarte erstellt!

Es lassen sich jedoch nicht nur einzelne Datensätze darstellen, sondern auch mehrere auf einmal (z.B. alle Datensätze, die einem bestimmten Kriterium entsprechen). In diesem Beispiel werden die Marker wie im Vorbild "mu21.de Google Maps API (Examples)" mit Polylines verbunden. Da die Bereitstellung der Übergabewerte mittels mySQL an dieser Stelle zur Verdeutlichung der Programmiertechnik keinen weitern Nutzen bringt, wird entsprechend darauf verzichtet.

Konzentrieren wir uns also auf die Darstellung mehrer Marker:

Wie gehabt müssen auch hier die Daten des mySQL-Servers bekannt gemacht werden: "include ("mu21-sql-config.php");". Es sollen alle Datensätzen dargestellt werden, deren Primärschlüssel kleiner als 5 ist. Folglich ist in der Abfrage die where_definition notwendig: "WHERE id < 5". Nun wird auf die Speicherung in php-Variablen verzichtet. Der Java-Script-Code wird somit direkt mittels der while-Schleife generiert.

Das Prinzip zur Markererstellung ist im Beispiel MAP2-Street-View und MAP3-Satellite-View kongruent.

Im HTML-Teil muss noch die mapID übergeben werden: "<?php echo 'id="map'; echo $mapid; echo '";?>".

[→ Code 3]

Die Anzeige des oben genannten Codes ist auf der Seite mu21.de Google Maps API with mySQL (Examples) zu bewundern. Wie bereits mehrfach erwähnt, den "Maps API key" von dieser Seite unter keinen Umständen kopieren! Falls doch:

Eins ist sicher, Google kommt euch auf die Schliche (bei der Nutzung der API wird die aufrufende Domain an Google übertragen)! Also, bitte selbst beantragen!

6 Post: Google Maps API auf mu21.de

(→Google Maps API auf mu21.de)

Die Programmierschnittstelle Google Maps API verwende ich im Bereich der Schwedenreise v2.0. Hiermit wurde diese Web-2.0-fähig. Der größte Vorteil besteht jedoch ohne Zweifel im urheberrechtlich unbedenklichen Kartenmaterial, welches Google den Webmastern auf dieser Weise zur Verfügung stellt. Damit sind neben den Autotouren (Småland, Halland, Vänern, Vättern) und den Städtetouren (Stockholm, Göteborg) auch der Reiseverlauf und die Übersichtskarten illustriert.

„Abmahnungen: 600 Euro für ein Brötchen-Bild" unter diesem Titel veröffentlichte Spiegel Online[14] Ende Mai eine Geschichte, die bereites Wochen zuvor hitzig in der Blog-Szene (kein Rechtschreibfehler! Nicht mal nach Heiligendamm! Außer dem (schwarzen) Block gibt es trotzdem noch das Kunstwort mit dem „g" am Ende) diskutiert wurde. Solche Ereignisse (die Brötchen-Geschichte) sensibilisieren! Also habe ich mich ein bisschen mit dem Thema Urheberrecht in Bezug auf Kartenmaterial beschäftigt.

Die Quintessenz: Kartenprodukte dürfen (insofern man ein lizenziertes Exemplar besitzt) nicht mehr als 1000 Mal weitervertrieben werden. Dies gilt auch für Karten auf Webseiten! Es gibt also zwei Alternativen: Echtzeit Auswertung der Page Impressions und die damit verbundene Entfernung des Kartenmaterials, falls sich der 1001 Besucher ankündigt; Oder: Verwendung von Google Maps API. Die erforderlichen Programmierkenntnisse besitzt der anständige Webmaster, oder man eignet sie sich an (weiterführende Informationen, insbesondere zur Winkelmaßermittlung (-berechnung) habe ich unter Verwendung von Google Maps API zusammengestellt)!

Nachdem ich mich also mit der Google API Technik vertraut gemacht habe, verwende ich diese im Bereich der Schwedenreise 2.0 nicht zuletzt um diese aufzuwerten und „Web-2.0-fähig" zu machen (siehe Relaunch Schwedenreise…[15] und Schwedenreise, die unendliche (Versions-) Geschichte[16]):

Damit ist es mir gelungen sowohl die Autotouren als auch die Reisen innerhalb Schwedens zu illustrieren. Gerade der Einsatz von Ortsmarken bietet die Möglichkeiten, einerseits die schönsten Sehenswürdigkeiten in der Karte aufzuzeigen, und andererseits auch die Geheimtipps bzw. die schwer zu findenden Orte kartographisch zu nennen. Zu einerseits siehe: Bildergalerie: Stockholm - Venedig des Nordens[17]. Die erste Karte auf dieser Seite zeigt eine Straßenkarte des Großraums mit einigen Markern als Hinweis (bzw. Link zur jeweiligen Beschreibung) zu Ausflugszielen im Umland von Stockholm. Die zweite Karte zeigt als Satellitenbild der Innenstadt die absoluten Hot Spots Stockholms (die Marker sind ebenfalls als Link zum jeweiligen Reisebericht ausgeführt). Zu Andererseits: z.B. Bildergalerie: Vimmbery - Unterwegs im Herzen Smålands[18]. Hier nutze ich die Möglichkeiten der API-Programmiertechnik um auf schwer zu findende Orte wie z.B. auf Skurugata, Gibberyd (Katthult) und Svedestorp (Bullerbü) aufmerksam zu machen.

Selbst die Übersichtskarten[19] ließen sich unter Verwendung von Polylines und Markern (siehe insbesondere Südschweden[20] - hier gelangt man durch anklicken

der verschiedenen Ortsmarken zur jeweiligen Bildergalerie des Ausflugs) ansprechend gestalten.

Damit aber noch nicht Genug! Nein! Auch Routenplanung ist möglich! In diesem Fall ehr "Routenpräsentation". Endlich konnte ich auch den Reiseverlauf[21] veranschaulichen. Ergebnis: u.A. Reiseverlauf Schwedenreise 09/2002 - Tag 1[22].

Unter dem Motto schwedenreise-2.0-at-mu21.de-goes-web-2.0 wünsche ich euch viel Spaß auf den überarbeiteten Seiten!

7 Post: Google Maps API Templates

(→ Google Maps API Templates)

Im Code-Archive (http://code.mu21.de) stelle ich einige Google Maps API Templates mit ausführlicher Beschreibung zur Verfügung. Neben einer generellen Erklärung zur Verwendung der Programmierschnittstelle (Verwendung von Google Maps API) erläutere ich im Besonderen die Erstellung von Straßenkarten und Satellitenbilder (mit Markern und Polylines) und die Bereitstellung eines Routenplaners (Directions Elements) auf der eigenen Webseite.

Neu im Code Archive von mu21.de: Beispiele zur Anwendung von Google Maps API. Hierzu stelle ich exemplarisch den Code verschiedener Funktionen vor, mit denen sich von Straßenkarten und Satellitenbilder (inkl. Marker und Polylines) bis hin zur perfekten Navigation (vgl. Routenplaner) urheberrechtlich (für den Nicht-Kommerziellen Gebrauch) unbedenkliche Karten erstellen lassen.

Voraussetzung für Google Maps API ist die Beantragung eines „Maps API key" (siehe Verwendung von Google Maps API). Hiermit wird nicht zuletzt sichergestellt, dass die Urherberrechte (insbesondere an den Geocode Daten - nominell darf nur alle 1,728 Sekunden eine neue Route berechnet werden) nicht verletzt werden.

Neben einigen Programmierkenntnissen (Java Script, Objektorientierte Programmierung) sollte der potentielle Anwender der Google Maps Programmierschnittstelle Erfahrungen in der Gewinnung von Longitude und Latitude mitbringen (für interessierte Leser, die Freude an dieser Thematik haben, empfehle ich meine Abhandlung zum Thema „Realisierung eines Ortungssystems mit GPS"[23]).

Auf der Seite mu21-google-maps-API.html stelle ich den erforderlichen Code zur Generierung von fast fehlerfreien XHTML-Code (1 Fehler - derjenige der diesen abstellt, darf mir die Lösung nennen und den Fehler (*xmlns:v="urn:schemas-microsoft-com:vml*" - leider nötig um den IE zu genügen) behalten!) vor um Straßenkarten und Satellitenbilder für Webseiten erstellen zu können. Damit die Karten von gewisser Professionalität zeugen, dürfen an dieser Stelle natürlich weder Navigationselemente noch die von google.maps gewohnte Auswahlmöglichkeit zwischen Straßenkarte, Satellitenbild bzw. Hybridansicht fehlen. Um den Code besser nachvollziehen zu können, ist dieser auf der Seite mu21.de Google Maps API (Examples) zu Demonstrationszwecken komplett XHTML codiert.

Ein weiteres interessantes Feature von Google Maps API ist die Nutzung der Geocode Daten. Damit lässt sich problemlos ein Routenplaner zu Navigationszwecken realisieren. Die für den Benutzer zugängliche Routenplanerfunktion stelle ich an dieser Stelle aufgrund der latenten Missbrauchsgefahr nicht vor (falls ihr für eure Seite trotzdem diese Funktion nutzen wollt: Google ist euer Freund!)! Trotzdem: Es lassen sich viele Erkenntnisse aus meinen Code-Beispiel (mu21-routenplaner-google-maps.html) ziehen. Vollständig ist das Ergebnis der Kodierung auf der Seite mu21.de Google Maps API Directions Elements (Example) zu betrachten.

Wie gehabt stelle ich an dieser Stelle für interessierte Leser wieder das Working Paper mit weiteren Hintergrundinformationen (inkl. Code) zur Verfügung: working-paper-mu21-api-description[24].

8 Post: Google Maps API mySQL Templates

(→ Google Maps API mySQL Templates)

Mit den Google Maps API mySQL Templates beschreibe ich den Einsatz von mySQL-Datenbanken (codiert in php) zur Realisierung größerer Visualisierungsprojekte mit der Google Maps API Programmierung. Darin wird erklärt wie Umgebungskarten (Darstellung eines SQL-Datensatzes), Straßenkarten und Satellitenbilder mit Markern und Polylines (Darstellung mehrerer SQL-Datensätze) erzeugt werden.

Bereits im Juni 2007 veröffentlichte ich mit den Google Maps API Templates einige grundlegende Beispiele zur Google Maps API Programmierung. Anhand dessen lassen sich sämtliche gängige Funktionen für die private Webseite problemlos realisieren. Aber auch kleinere kommerzielle Projekte sind damit mühelos virtuell geographisch zu visualisieren.

Seit der Veröffentlichung erhalte ich reichlich Feedback zum Code und den dazugehörigen Beschreibungen. Des Öfteren werden (Komplett-) Lösungen für größere Projekte angefragt. Als Lösungsansatz biete ich diesbezüglich die Integration einer mySQL-Datenbank zur Speicherung der Datensätze an.

Dass es sich mit den in php beschriebenen mySQL-Funktionen ausgezeichnet auf mySQL-Datenbanken zugreifen lässt ist nicht wirklich ein Geheimnis! Die Codierung in php bietet sich also förmlich an! Allerdings wird die Google Maps API mit Java-Script angesprochen… Es stellt sich nunmehr die Frage: Wie sind die Programmiersprachen php, mySQL und Java-Script zu kombinieren damit W3C-konformes HTML entsteht?

Antworten auf diese Frage gibt das Code Beispiel "mu21.de Google Maps API with mySQL (Examples)" und die Dokumentation hierzu im Code Archive: mu21-google-maps-API-mySQL.php (Explanation) und vor allem das Tutorial Google Maps API (eine Schnittstellenbeschreibung). Damit ist eine kompakte Anleitung zu allen relevanten Anwendungsgebieten der Google Maps API vorhanden. Jede Anwendung wird dort didaktisch mit Code-Beispielen illustriert.

Viel Spaß damit!

P.S.: Wie gehabt ist Feedback jederzeit über das Kontakt-Formular willkommen!

9 Code 1: mu21-google-maps-API.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-microsoft-
com:vml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>mu21.de Google Maps API (Examples)</title>

<script src=”http://maps.google.com/maps?file=api&v=2.x&key=[invalid -
please sign up]” type=”text/javascript”></script>
<!–Key bitte nicht kopieren! Kann auf
http://www.google.com/apis/maps/signup.html beantragt werden!–>

<style type=”text/css”>
v\:* {
behavior:url(#default#VML);
}
body {
font-family: Verdana, Arial, sans serif;
font-size: 11px;
margin: 2px;
}
table.directions th {
background-color:#EEEEEE;
}
img {
color: #000000;
}
</style>

<script type=”text/javascript”>

//<![CDATA[

function load() {
if (GBrowserIsCompatible()) {

//————————————-MAP1-Street-View————————————-//

var map = new GMap2(document.getElementById(”map1”));

map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.setCenter(new GLatLng(49.56583, +10.733), 10);

// Polyline

var polyline = new GPolyline( [
new GLatLng(49.5797, +10.6103), //NEA
new GLatLng(49.5978, +11.0019) //ER
], “#888888”, 10);
map.addOverlay(polyline);

var polyline = new GPolyline( [
new GLatLng(49.5978, +11.0019), //ER
new GLatLng(49.4739, +10.9894) //FÜ
], “#888888”, 10);
map.addOverlay(polyline);
```

```
var polyline = new GPolyline( [
new GLatLng(49.4739, +10.9894), //FÜ
new GLatLng(49.5797, +10.6103) //NEA
], "#888888", 10);
map.addOverlay(polyline);

// Marker

function createMarker(point, number) {
var marker = new GMarker(point);
GEvent.addListener(marker, "click", function() {
marker.openInfoWindowHtml("Marker <b>" + number + "</b>");
});
return marker;
}

map.addOverlay(createMarker(new GLatLng(49.56583, +10.733), '<a
href="http://www.mu21.de/" target="blank">mu21.de!</a>'));
map.addOverlay(createMarker(new GLatLng(49.5797, +10.6103), 'NEA'));
map.addOverlay(createMarker(new GLatLng(49.5978, +11.0019), 'ER'));
map.addOverlay(createMarker(new GLatLng(49.4739, +10.9894), 'FÜ'));

//————————————-MAP2-Satellite-View————————————-//

var map = new GMap2(document.getElementById("map2"));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.setCenter(new GLatLng(49.56583, +10.733), 10, G_SATELLITE_MAP);

// Polyline

var polyline = new GPolyline( [
new GLatLng(49.5797, +10.6103), //NEA
new GLatLng(49.5978, +11.0019) //ER
], "#ff0000", 10);
map.addOverlay(polyline);

var polyline = new GPolyline( [
new GLatLng(49.5978, +11.0019), //ER
new GLatLng(49.4739, +10.9894) //FÜ
], "#ff0000", 10);
map.addOverlay(polyline);

var polyline = new GPolyline( [
new GLatLng(49.4739, +10.9894), //FÜ
new GLatLng(49.5797, +10.6103) //NEA
], "#ff0000", 10);
map.addOverlay(polyline);

// Marker

function createMarker(point, number) {
var marker = new GMarker(point);
GEvent.addListener(marker, "click", function() {
marker.openInfoWindowHtml("Marker <b>" + number + "</b>");
});
return marker;
}

map.addOverlay(createMarker(new GLatLng(49.56583, +10.733), '<a
href="http://www.mu21.de/" target="blank">mu21.de!</a>'));
map.addOverlay(createMarker(new GLatLng(49.5797, +10.6103), 'NEA'));
```

```
map.addOverlay(createMarker(new GLatLng(49.5978, +11.0019), 'ER'));
map.addOverlay(createMarker(new GLatLng(49.4739, +10.9894), 'FÜ'));
}
}

//]]>

</script>
</head>

<body onload="load()" onunload="GUnload()">

<h2>mu21.de Google Maps API (Examples)</h2>

<p><b><em>Karte 1:</em>
<br/>Straßenkarte mit Marker und Polyline</b><br/></p>
<div id="map1" style="width: 500px; height: 300px"></div>
<p><br/><br/><b><em>Karte 2:</em>
<br/>Satellitenbild mit Marker und Polyline</b><br/></p>
<div id="map2" style="width: 500px; height: 300px"></div>

<p><br/><br/><a href="http://www.mu21.de/">mu21.de [nt & lifestyle]</a> -
<a href="http://www.mu21.de/index.php/code-archive/">mu21.de Code
Archive</a> - <a
href="http://www.mu21.de/index.php/about/">Impressum</a></p>

</body>

</html>
```

10 Code 2: mu21-routenplaner-google-maps.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-microsoft-
com:vml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>mu21.de Google Maps API Directions Elements (Example)</title>

<script src=” http://maps.google.com/?file=api&amp;v=2.x&amp;key=[invalid -
please sign up]” type=”text/javascript”></script>
<!–Key bitte nicht kopieren! Kann auf
http://www.google.com/apis/maps/signup.html beantragt werden!–>

<style type=”text/css”>
v\:* {
behavior:url(#default#VML);
}
body {
font-family: Verdana, Arial, sans serif;
font-size: 11px;
margin: 2px;
}
table.directions th {
background-color:#EEEEEE;
}
img {
color: #000000;
}
</style>

<script type=”text/javascript”>

//<![CDATA[

var map;
var gdir;
var geocoder = null;
var addressMarker;

function load() {
if (GBrowserIsCompatible()) {
map = new GMap2(document.getElementById(”map”));

gdir = new GDirections(map, document.getElementById(”directions”));
GEvent.addListener(gdir, “load”, onGDirectionsLoad);
GEvent.addListener(gdir, “error”, handleErrors);

setDirections(”49.5659N, 10.7336E”, “49.5472, 10.7242”, “de_DE”);
}
}

function setDirections(fromAddress, toAddress, locale) {
gdir.load(”from: ” + fromAddress + ” to: ” + toAddress, { “locale”: locale
});
}

function handleErrors() {
if (gdir.getStatus().code == G_GEO_UNKNOWN_ADDRESS)
alert(”Adresse existiert nicht (evtl. mit Latitude und Longitude
```

```
versuchen)!\n Fehler: ” + gdir.getStatus().code);
else if (gdir.getStatus().code == G_GEO_SERVER_ERROR)
alert(”Adresse wurde nicht gefungen (evtl. mit Latitude und Longitude
versuchen)!\n Fehler: ” + gdir.getStatus().code);

else if (gdir.getStatus().code == G_GEO_MISSING_QUERY)
alert(”Adresse vollständig angeben!\n Fehler: ” + gdir.getStatus().code);

else if (gdir.getStatus().code == G_GEO_BAD_KEY)
alert(”Google Maps API Key nicht gültig! Bitte nicht kopieren! Key kann auf
http://www.google.com/apis/maps/signup.html beantragt werden! \n Fehler: ”
+ gdir.getStatus().code);

else if (gdir.getStatus().code == G_GEO_BAD_REQUEST)
alert(”Fehler bei der Berechnung. Bitte nochmal versuchen!\n Fehler: ” +
gdir.getStatus().code);

else alert(”Unbekannter Fehler!\n Fehler: ” + gdir.getStatus().code);

}

function onGDirectionsLoad() {
//Um an Informationen der load()-Funktion heranzukommen.
//Bei Nichtverwendung nicht löschen!
}

//]]>

</script>

</head>


<body onload=”load()” onunload=”GUnload()”>

<h2>mu21.de Google Maps API Directions Elements (Example)</h2>

<p><b><em>Route:</em>
<br/>Wilhelmsdorf, Mittelfranken (49.5659N, 10.7336E) nach
<br/>Emskirchen, Mittelfranken (49.5472N, 10.7242E)</b>
<br/><br/></p>
<table class=”directions”>

<tr><th>Beschreibung</th><th>Karte</th></tr>

<tr>
<td valign=”top”><div id=”directions” style=”width: 275px”></div></td>
<td valign=”top”><div id=”map” style=”width: 310px; height:
450px”></div></td>
</tr>

</table>

<p><a href=”http://www.mu21.de/”>mu21.de [nt &amp;amp; lifestyle]</a> - <a
href=”http://www.mu21.de/index.php/code-archive/”>mu21.de Code Archive</a>
- <a href=”http://www.mu21.de/index.php/about/”>Impressum</a></p>


</body>

</html>
```

11 Code 3: mu21-google-maps-API-mySQL-examples.php

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml” xmlns:v=”urn:schemas-microsoft-
com:vml”>
<head>
<meta http-equiv=”content-type” content=”text/html; charset=utf-8”/>
<title>mu21.de Google Maps API with MySQL (Examples)</title>
<meta name=”author” content=”mu21.de - Michael Uhl, Wilhelmsdorf, Germany”
/>
<meta name=”copyright” content=”mu21.de - Michael Uhl, Wilhelmsdorf,
Germany” />
<meta name=”robots” content=”index,follow” />
<meta name=”description” content=”[…]” />
<meta name=”keywords” content=”[…]” />

<script src=” http://maps.google.com/?file=api&v=2.x&key=[…]”
type=”text/javascript”></script>
<!–Key bitte nicht kopieren! Kann auf
http://www.google.com/apis/maps/signup.html beantragt werden!–>
<style type=”text/css”>
v\:* {
behavior:url(#default#VML);
}
body {
font-family: Verdana, Arial, sans serif;
font-size: 11px;
margin: 2px;
}
table.directions th {
background-color:#EEEEEE;
}
img {
color: #000000;
}
</style>

<script type=”text/javascript”>

//<![CDATA[

function load() {
if (GBrowserIsCompatible()) {

//————————MAP1-Single-View-(MySQL)————————//

<?php include (”mu21-sql-config.php”);

$query = “SELECT * FROM maps WHERE id=1”;
$result = mysql_query($query);

while ($line = mysql_fetch_array($result))
{
$mapid=$line[id];$lat=$line[latitude]; $long=$line[longitude];
$loc=$line[location];
}

mysql_free_result($result);
mysql_close();
```

```
echo ‘var map = new GMap2(document.getElementById(”‘; echo “map”; echo
$mapid; echo ‘”));’;?>

map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());

<?php
echo “map.setCenter(new GLatLng(”; echo $lat; echo “, “; echo $long; echo
“), 10);”;?>

// Marker

function createMarker(point, number) {
var marker = new GMarker(point);
GEvent.addListener(marker, “click”, function() {
marker.openInfoWindowHtml(”Marker <b>” + number + “</b>”);
});
return marker;
}

<?php echo “map.addOverlay(createMarker(new GLatLng(”; echo $lat; echo “,
“; echo $long; echo “), ‘”; echo $loc; echo “‘));”;?>

//————————-MAP2-Street-View————————-//

var map = new GMap2(document.getElementById(”mapStreet”));

map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());

map.setCenter(new GLatLng(49.55583, +10.833), 10);

// Polyline

var polyline = new GPolyline( [
new GLatLng(49.5797, +10.6103), //NEA
new GLatLng(49.5978, +11.0019) //ER
], “#888888”, 10);
map.addOverlay(polyline);

var polyline = new GPolyline( [
new GLatLng(49.5978, +11.0019), //ER
new GLatLng(49.4739, +10.9894) //FUE
], “#888888”, 10);
map.addOverlay(polyline);

var polyline = new GPolyline( [
new GLatLng(49.4739, +10.9894), //FUE
new GLatLng(49.5797, +10.6103) //NEA
], “#888888”, 10);
map.addOverlay(polyline);

// Marker

function createMarker(point, number)
{
var marker = new GMarker(point);
GEvent.addListener(marker, “click”, function() {
marker.openInfoWindowHtml(”Marker <b>” + number + “</b>”);
});
return marker;
}
```

```
<?php include (”mu21-sql-config.php”);

$query = “SELECT * FROM maps WHERE id < 5”;
$result = mysql_query($query);

while ($line = mysql_fetch_array($result))
{
echo “map.addOverlay(createMarker(new GLatLng(”; echo $line[latitude]; echo
“,”; echo $line[longitude]; echo “), ‘”; echo $line[location]; echo “’));”;
}

mysql_free_result($result);
mysql_close();
?>

//————————-MAP3-Satellite-View————————-//

var map = new GMap2(document.getElementById(”mapSatellite”));
map.addControl(new GSmallMapControl());
map.addControl(new GMapTypeControl());
map.setCenter(new GLatLng(49.55583, +10.833), 10, G_SATELLITE_MAP);

// Polyline

var polyline = new GPolyline( [
new GLatLng(49.5797, +10.6103), //NEA
new GLatLng(49.5978, +11.0019) //ER
], “#ff0000”, 10);
map.addOverlay(polyline);

var polyline = new GPolyline( [
new GLatLng(49.5978, +11.0019), //ER
new GLatLng(49.4739, +10.9894) //FUE
], “#ff0000”, 10);
map.addOverlay(polyline);

var polyline = new GPolyline( [
new GLatLng(49.4739, +10.9894), //FUE
new GLatLng(49.5797, +10.6103) //NEA
], “#ff0000”, 10);
map.addOverlay(polyline);

// Marker

function createMarker(point, number)
{
var marker = new GMarker(point);
GEvent.addListener(marker, “click”, function() {
marker.openInfoWindowHtml(”Marker <b>” + number + “</b>”);
});
return marker;
}

<?php include (”mu21-sql-config.php”);

$query = “SELECT * FROM maps WHERE id < 5”;
$result = mysql_query($query);

while ($line = mysql_fetch_array($result))
{
echo “map.addOverlay(createMarker(new GLatLng(”; echo $line[latitude]; echo
```

```
“,”; echo $line[longitude]; echo “), ‘”; echo $line[location]; echo “‘));”;
}

mysql_free_result($result);
mysql_close();
?>
}
}

//]]>

</script>
</head>

<body onload=”load()” onunload=”GUnload()”>
<p><a href=”http://www.mu21.de/”>mu21.de [nt & lifestyle]</a> - <a
href=”http://www.mu21.de/index.php/code-archive/”>mu21.de Code Archive</a>
- <a href=”http://www.mu21.de/index.php/code-archive/mu21-google-maps-api-
mysql”>mu21.de Google Maps APIwith MySQL (Description)</a> - <a
href=”http://www.mu21.de/index.php/about/”>Impressum</a></p>
<h2>mu21.de Google Maps API with mySQL (Examples)</h2>
<p>Download <a href=”http://www.mu21.de/wp-content/plugins/wp-
downloadMonitor/download.php?id=66”>Google Maps API (eine
Schnittstellenbeschreibung)</a></p>
<p><b><em>Karte <?php echo $mapid;?>:</em>
<br/>Umgebungskarte mit Marker generiert mit MySQL-Abfragen</b><br/></p>
<div <?php echo ‘id=”map’; echo $mapid; echo ‘”‘;?> style=”width: 500px;
height: 300px”></div>
<p><b><em>Karte Street View:</em>
<br/>Straßenkarte mit Marker und Polyline generiert mit MySQL-
Abfragen</b><br/></p>
<div id=”mapStreet” style=”width: 500px; height: 300px”></div>
<p><br/><br/><b><em>Karte Satellite View:</em>
<br/>Satellitenbild mit Marker und Polyline generiert mit MySQL-
Abfragen</b><br/></p>
<div id=”mapSatellite” style=”width: 500px; height: 300px”></div>
<p><br/><br/>Whois <a href=”http://www.mu21.de/”>mu21.de [nt &
lifestyle]</a>: Code by Michael Uhl - <a
href=”http://www.mu21.de/index.php/about/”>Impressum</a></p>
</body>
</html>
```

12 Querverweise

[1] http://www.spiegel.de/netzwelt/web/0,1518,483856,00.html
[2] http://www.microsoft.com/germany/unternehmen/informationen/rechtlichehinweise/bilder.mspx#ETCAC
[3] http://www.google.com/apis/maps/
[4] http://www.google.com/apis/maps/signup.html
[5] http://code.google.com/apis/
[6] http://www.multimap.com/
[7] http://www.mu21.de/index.php/code-archive/mu21-google-maps-api/
[8] http://www.mu21.de/index.php/code-archive/mu21-routenplaner-google-maps/
[9] Abbildung 1: http://www.mu21.de/FH/files/mu21-google-maps-API-examples.html
[10] http://www.mu21.de/index.php/code-archive/verwendung-google-maps-api/
[11] Abbildung 2: http://www.mu21.de/FH/files/mu21-google-maps-API-directions-elements-example.html
[12] Abbildung 3: http://www.mu21.de/index.php/code-archive/mu21-google-maps-api-mysql
[13] Abbildung 4: mySQL Tabelle „maps" (phpMyAdmin)
[14] http://www.spiegel.de/netzwelt/web/0,1518,483856,00.html
[15] http://www.mu21.de/index.php/94/relaunch-schwedenreise/
[16] http://www.mu21.de/index.php/35/schwedenreise/
[17] http://www.mu21.de/schwedenreise/Stockholm.htm
[18] http://www.mu21.de/schwedenreise/Vimmerby.htm
[19] http://www.mu21.de/schwedenreise/Karte.htm
[20] http://www.mu21.de/schwedenreise/Suedschweden.htm
[21] http://www.mu21.de/schwedenreise/Reise.htm
[22] http://www.mu21.de/schwedenreise/reiseverlauf1.html
[23] http://www.mu21.de/wp-content/plugins/wp-downloadMonitor/download.php?id=42
[24] http://www.mu21.de/wp-content/plugins/wp-downloadMonitor/download.php?id=66