Web-based mapping
An evaluation of four JavaScript APIs

Final thesis in Computer and Information Science at Linköping Institute of Technology
by

Magnus Näslund

LITH-IDA-EX--07/067--SE
Linköping 2007
Web-based mapping
An evaluation of four JavaScript APIs

Final thesis in Computer and Information Science at
Linköping Institute of Technology
by

Magnus Näslund

LITH-IDA-EX--07/067--SE
Linköping 2007

Examiner: Erik Berglund
IDA, Linköping University

Supervisors: Anders Larsson
IDA, Linköping University
Jean-Noel Heyraud
Amadeus IT Group SA
### Title
Web-based mapping - An evaluation of four JavaScript APIs

### Author
Magnus Näslund

### Abstract
As a result of Web 2.0 technologies such as Asynchronous JavaScript and XML (Ajax) web-based applications with rich contents are evolving to be more and more like normal applications in aspects, such as interactivity, functionality, and usability. This evolution makes it possible to create web-based services, providing maps for users to search and browse geographic information. This thesis is an evaluation of functionality, usability and accuracy for the four web-based map APIs: Google Maps, Microsoft Virtual Earth, Multimap and ViaMichelin.

The thesis explains how web-based mapping works, common functionality provided, and evaluates the functionality provided by each map service provider as well as the offered usability. In addition to this, it also includes the results of several tests, illustrating the APIs’ browser compatibility, performance and accuracy.

After testing and evaluation of the four APIs, the conclusion is that none of them can be appointed as the winner. They all have benefits and drawbacks; differences in terms of functionality, compatibility, usability, geocoding and development support, and the choice of API is consequently dependent of the type of application. As a result of this, and the fact that the APIs are constantly changing in terms of functionality and coverage, it is important to create applications independent of the map service provider. This was successfully done during the internship at Amadeus by creating a map abstraction layer in-between the applications and the maps, creating the possibility to switch API, or map service provider, without changed the code.

### Keywords
Web mapping, map API, map evaluation, API Abstraction layer, geocoding
Abstract
As a result of Web 2.0 technologies such as Asynchronous JavaScript and XML (Ajax) web-based applications with rich contents are evolving to be more and more like normal applications in aspects, such as interactivity, functionality, and usability. This evolvement makes it possible to create web-based services, providing maps for users to search and browse geographic information. This thesis is an evaluation of functionality, usability and accuracy for the four web-based map APIs: Google Maps, Microsoft Virtual Earth, Multimap and ViaMichelin.

The thesis explains how web-based mapping works, common functionality provided, and evaluates the functionality provided by each map service provider as well as the offered usability. In addition to this, it also includes the results of several tests, illustrating the APIs’ browser compatibility, performance and accuracy.

After testing and evaluation of the four APIs, the conclusion is that none of them can be appointed as the winner. They all have benefits and drawbacks; differences in terms of functionality, compatibility, usability, geocoding and development support, and the choice of API is consequently dependent of the type of application. As a result of this, and the fact that the APIs are constantly changing in terms of functionality and coverage, it is important to create applications independent of the map service provider. This was successfully done during the internship at Amadeus by creating a map abstraction layer in-between the applications and the maps, creating the possibility to switch API, or map service provider, without changed the code.
# Table of contents

**ABSTRACT** ................................................................................................................................. VI

**TABLE OF CONTENTS** ................................................................................................................... VIII

**LIST OF FIGURES** ......................................................................................................................... X

## 1 INTRODUCTION................................................................. 1

1.1 PURPOSE................................................................................................................................. 1
1.2 BACKGROUND ........................................................................................................................ 1
1.3 METHOD ................................................................................................................................. 1
1.4 STRUCTURE OF THESIS ........................................................................................................ 2

## 2 WEB-BASED MAPPING......................................................... 3

2.1 HISTORY ................................................................................................................................... 3
2.2 MAP TYPES ........................................................................................................................... 3
2.3 COMMON WEB MAP FUNCTIONALITY .................................................................................. 4
2.4 WEB-BASED MAPPING AND USABILITY ............................................................................. 9
2.5 TECHNOLOGY ........................................................................................................................ 10
2.6 GEOGRAPHICAL DATA .......................................................................................................... 13

## 3 GOOGLE MAPS API..................................................................... 14

3.1 VIEWING MODES .................................................................................................................. 14
3.2 NAVIGATION AND CONTROLS .......................................................................................... 14
3.3 OVERLAYS ............................................................................................................................ 16
3.4 ROUTE AND DRIVING DIRECTIONS .................................................................................... 18
3.5 GEOXML OVERLAY ............................................................................................................. 19
3.6 EVENTS ................................................................................................................................... 19
3.7 LANGUAGE SUPPORT .......................................................................................................... 19
3.8 GEOCODING ......................................................................................................................... 20
3.9 LIMITATIONS ........................................................................................................................ 20
3.10 GOOGLE MAPS FOR ENTERPRISES ................................................................................ 21
3.11 DOCUMENTATION AND SUPPORT ................................................................................. 21

## 4 MULTIMAP API.................................................................... 23

4.1 VIEWING MODES .................................................................................................................. 23
4.2 NAVIGATION AND CONTROLS .......................................................................................... 23
4.3 OVERLAYS ............................................................................................................................ 25
4.4 MARKER .................................................................................................................................. 25
4.5 EVENTS ................................................................................................................................... 27
4.6 LANGUAGE SUPPORT .......................................................................................................... 27
4.7 DRIVING DIRECTIONS .......................................................................................................... 27
4.8 GEOCODING ......................................................................................................................... 29
4.9 LOCATION INFORMATION AND SEARCHING ..................................................................... 29
4.10 LIMITATION ......................................................................................................................... 30
4.11 WEB SERVICES .................................................................................................................... 30
4.12 DOCUMENTATION AND SUPPORT ................................................................................. 30

## 5 VIAMICHELIN MAPS AND DRIVE API........................................ 32

5.1 VIEWING MODES .................................................................................................................. 32
5.2 NAVIGATION AND CONTROLS .......................................................................................... 32
5.3 SHAPES ................................................................................................................................... 33
5.4 DRIVING DIRECTIONS .......................................................................................................... 34
5.5 EVENTS ................................................................................................................................... 35
List of figures

FIGURE 1, MAP TYPE CLASSIFICATION .................................................................................. 3
FIGURE 2, X IS INDICATING THE GEOCODING INTERPOLATION ERROR. THE GECODER IS
ASSUMING NUMBER 3 TO BE HALF WAY UP THE STREET, CAUSING THE RETURN
LOCATION TO BE PLACED WRONG .............................................................................. 8
FIGURE 3, INDIVIDUAL ADDRESS INTERPOLATION. EACH ADDRESS IS MATCHED TO A
LATITUDE AND LONGITUDE STORED IN A LAND PARCEL DATABASE ...................... 8
FIGURE 4, THE TRADITIONAL WEB APPLICATION MODEL, TO THE LEFT, COMPARED TO
THE AJAX-BASED MODEL, TO THE RIGHT .................................................................... 11
FIGURE 5, WEB-BASED MAP SYSTEM CONSTRUCTED BY THREE PARTS: CLIENT BROWSER,
SERVER SYSTEM HOSTING SITE WITH MAP AND MAP API SYSTEM .......................... 12
FIGURE 6, GOOGLE MAPS WITH TILES OUTLINED. IMAGE TAKEN FROM [12] ............ 12
FIGURE 7, GOOGLE MAPS API HYBRID VIEW ................................................................. 14
FIGURE 8, MAP CONTROL IN THREE SIZES ..................................................................... 15
FIGURE 9, MAP TYPE CONTROL ..................................................................................... 15
FIGURE 10, SCALE CONTROL ......................................................................................... 15
FIGURE 11, OVERVIEW MAP ......................................................................................... 15
FIGURE 12, LOCAL SEARCH CONTROL .......................................................................... 15
FIGURE 13, A GOOGLE MAPS MARKER DISPLAYED WITH A CUSTOM ICON ............ 16
FIGURE 14, STANDARD MARKERS WITH AN INFO WINDOW ......................................... 16
FIGURE 15, POLYLINE DISPLAYED ON TOP OF GOOGLE MAPS' MAP MODE ............. 17
FIGURE 16, DRIVING DIRECTIONS ON MAP AND TEXTUAL DESCRIPTIONS .......... 18
FIGURE 17, THE CURRENT TRAFFIC SITUATION IN JERSEY CITY DISPLAYED ON A MAP
FROM GOOGLE'S MAP API ......................................................................................... 18
FIGURE 18, MULTIMAP’S WEB-BASED MAP API USING THE HYBRID VIEW, NOTICE THAT
ONLY A PART OF THE MAP IS OVERLAPPED WITH STREET MAPS ........................... 23
FIGURE 19, PAN AND ZOOM WIDGET IN THREE SIZES .................................................... 24
FIGURE 20, MULTIMAP’S MAP TYPE WIDGET ................................................................. 24
FIGURE 21, MAP TOOLS WIDGET .................................................................................... 24
FIGURE 22, LOCATION WIDGET ...................................................................................... 24
FIGURE 23, RIGHT CLICK CONTEXT MENU ..................................................................... 25
FIGURE 24, MARKERS WITH INFOBOXES ....................................................................... 25
FIGURE 25, MULTIMAP DECLUTTERING MARKER FUNCTIONALITY ARRANGING MARKERS
WITH AGGREGATE AND GRID METHODS ................................................................ 26
FIGURE 26, A POLYLINE AND A POLYGON DISPLAYED ON TOP OF MAP DISPLAYED IN
AERIAL/SATELLITE MODE .......................................................................................... 27
FIGURE 27, A MAP FROM VIAMICHELIN’S MAPS AND DRIVE API SHOWING BIARRITZ,
FRANCE ....................................................................................................................... 32
FIGURE 28, THREE VIEWS OF MAP TOOLS: FULL, SCALE + NAVIGATION AND SIMPLE ... 32
FIGURE 29, DEFAULT AND CUSTOM ICON DISPLAYED IN MAP .................................. 33
FIGURE 30, A DEFAULT VIAMICHELIN MAPS AND DRIVE TOOLTIP .............................. 33
FIGURE 31, POLYGON AND CIRCLE OBJECTS ............................................................... 34
FIGURE 32, DAILY WEATHER REPORT DISPLAYED WITH PREFORMATTED HTML IN A TOOL
TIP .................................................................................................................................. 35
FIGURE 33, VIRTUAL EARTH’S ROAD, AERIAL AND HYBRID VIEWS .............................. 39
FIGURE 34, NEGRESO, NICE WITH BIRD’S EYE VIEW .................................................. 40
FIGURE 35, MANHATTAN IN 3D VIEWING MODE ............................................................... 40
FIGURE 36, VIRTUAL EARTH’S DASHBOARD CONTROL IN THREE SIZES: NORMAL, SMALL
AND TINY ...................................................................................................................... 41
FIGURE 37, FIND CONTROL PROVIDED BY THE VIRTUAL EARTH API ............................ 41
FIGURE 38, VIRTUAL EARTH’S OVERVIEW MAP, CALLED MINIMAP .......................... 41
FIGURE 39, PUSHPIN WITH INFOBOX .......................................................................... 42
FIGURE 40, VIRTUAL EARTH API DRIVING DIRECTIONS DISPLAYED WITH POLYLINES,
PUSHPINS AND AN INFOBOX ..................................................................................... 44
1 Introduction

This chapter presents the background, purpose and realisation of the thesis.

1.1 Purpose

The purpose of this thesis and the placement at Amadeus was to evaluate and compare several web-based map APIs, from several providers, to gather knowledge about features and options available. Another key purpose was to investigate how web-based maps can be integrated into web applications and to see if it is possible to create an abstraction layer through which, all the APIs can be integrated.

1.2 Background

As a result of Web 2.0 technologies such as Asynchronous JavaScript and XML (Ajax) web-based applications with rich contents are evolving to be more and more like normal applications in aspects such as interactivity, functionality, and usability. This evolvement makes it possible to create web-based services, providing maps for users to search and browse geographic information, such as places and routes. Today there are several web-based maps available on the market. Companies like Google, Microsoft, Multimap and ViaMichelin all provide their own application programming interface (API) for integration of web-based maps in websites.

Amadeus was founded in the late eighties by four airline companies: Air France, Lufthansa, Iberia, and SAS, to provide a Global Distribution System (GDS). Today Amadeus is a global company, with more than 7000 employees and offices situated around the world.

The focus is still on the GDS and airline booking systems but the company is now working as a complete technology provider for the entire travel industry; providing technology solutions for airlines companies, car rental companies and hotel chains.

Since several of Amadeus’s booking systems are web-based and the market is the travel industry, web-based maps may be a good way to provide customers with geographic information such as hotel locations and driving directions to and from airports. Therefore, Amadeus wished for several web-based map APIs to be evaluated in terms of functionality, compatibility and accuracy; and to investigate how maps could be used in the company’s products.

1.3 Method

This thesis was written during a placement at Amadeus IT Group SA in Sophia Antipolis, France during fall 2007. The company supervisor was Jean-Noël Heyraud, university supervisor was Anders Larsson and examiner at the university was Erik Berglund.

The placement consisted of three main activities:

- A research part wherein the four web-based APIs: Google Maps API, Multimap API, ViaMichelin Maps and Drive API and Microsoft Virtual Earth Map Control API were evaluated and tested in terms of functionality, accuracy and coverage. These four APIs were chosen since: Google Maps is the most commonly used...
API [1], Multimap and Virtual Earth are currently used by Amadeus and ViaMichelin is a French company with well known printed maps. During the evaluation the following versions of the APIs were used: Google Maps 2.x, Multimap 1.2, ViaMichelin Maps and Drive 1.0 and Microsoft Virtual Earth 5.

- A specification part in which currently available map abstraction layers were investigated, and a specification for a new JavaScript abstraction layer was created.

- A development part wherein the earlier specified abstraction layer was developed, and later tested in the Amadeus product HotelsPlus.

### 1.4 Structure of thesis

The thesis starts with the first chapter containing a description of the final thesis project including background, purpose and realisation. Chapter two is describing how web-based map APIs works, common functionality and how the geographical data is collected. After this, chapter three to six are reviews and information about the four map APIs; Google Maps, Virtual Earth, ViaMichelin and Multimap. Chapter seven holds several API tests to determine the map accuracy, browser support and usability as well as an API comparison. The next chapter, eight, is describing an abstraction layer created to eliminate the problems with service provider dependability. Chapter nine contains a discussion and conclusions drawn from the API reviews and tests. Finally, the last two chapters, ten and eleven, include explanations of terms and abbreviations and a list of references.
2 Web-based mapping
Web-based mapping is the process of generating and providing maps on the World Wide Web for users to search and browse spatial information, such as locations and routes. Previously cartography was expensive and restricted to a few companies and institutes, but due to the change in technologies, with high bandwidth internet connections and advanced web development techniques, geographical data can be provided and transferred across Internet at a low cost, making it possible for everyone to integrate and display a map in a website.

2.1 History
Web-based mapping has existed since the mid nineties, Mapquest launched their first online address mapping and driving directions with a mapping output in 1996 [2]. The same year UK-based Multimap launched their website offering web-based mapping, driving directions and location based services [3]; a website which quick became one of the most popular sites in the United Kingdom.

On February 8th 2005 Google announced a web-based map system, Google Maps, on their blog [4]. In July the same year Microsoft released Virtual Earth, today called Live Search Maps and after this, both Google and Microsoft soon released their Application Programming Interfaces (APIs) to provide easy integration of web-based maps in websites.

2.2 Map types
Web-based maps were first classified into groups by Kraak, Menno-Jan and Brown, Allan [5]. They divided the maps into static and dynamic maps and further distinguished interactive and view-only maps.

![Figure 1, map type classification.](image)

2.2.1 Static maps
Static maps are usually view-only maps, created once and often manual. The maps have normally no interactivity and are consisting of an image. However, there are interactive static maps, also called clickable maps, which are containing clickable geographic object or regions. A click could lead the user to other information as another map, images or other web pages.

A frequent way to use non interactive static maps is on companies “contact us” page, to show the location of an office or a store.
2.2.2 Dynamic maps
Dynamic maps are generated and delivered on demand when a web page is viewed providing the possibility to display realtime information, such as the current weather conditions, the current traffic situation or the results from an election. The maps are normally generated from data stored in a dynamic data source, e.g. a database, and created and delivered using technologies such as server-side languages, JavaScript, Ajax or Macromedia Flash.

2.2.3 View-only
View-only maps are maps where no interaction is possible. The maps always display the same area and it is not possible to zoom, pan or click the map.

2.2.4 Interactive
Interactive maps are more advanced than view-only maps. The interactive maps let the users explore the map, navigate, add and remove additional information, change map parameters and much more.

Previously most maps integrated into web sites were view-only single image maps without possibility to interact; but due to enhancements of web technologies maps are getting more interactive and map service providers today offer web-based map APIs for an easy integration of dynamic and interactive maps. This thesis will therefore focus on the modern dynamic and interactive web-based map APIs built upon Ajax technologies.

2.3 Common web map functionality
A web-based map is a simplified Geographic Information System (GIS) for non-expert users. A normal GIS provides tools for users to create interactive queries, analyze the spatial information, edit data, maps, and present the results of all these operations. For example a GIS can be used by meteorologists to generate a map with isopleths, or contour lines, which indicate amounts of rainfall in different regions.

In contrast to this, the main purpose of web mapping is to present and to let users, without experience, explore geographic information such as driving directions or available hotels and restaurants in a city; therefore, usability is an important aspect.

2.3.1 Viewing modes
Web-based map API service providers offer several viewing modes. This chapter will explain the most common used: road map, satellite and hybrid. Some providers offer unique modes such as Virtual Earth’s Bird’s eye mode, these viewing modes are explained in the chapter reviewing the API that is providing it.

Road Map mode
The road map view consists of topographic and street maps. The maps are in general available in several zoom levels; displaying everything in-between the entire earth and detailed street maps in cities.
Satellite/aerial mode
The satellite/aerial mode is imagery photographed from satellites or airplanes. The images are taken at a ninety-degree angle and show landscapes and rooftops. Normally a map provider’s imagery covers the entire world; however, there is a big difference in resolution depending on the location. Major cities are normally covered with high-resolution imagery showing detailed street views and for less populated areas, only low-resolution images showing the landscape is provided.

Hybrid mode
The hybrid view is a combination of the road map view and the satellite/aerial mode; it consists of satellite/aerial imagery overlaid by a road map. This makes it possible to view real photos of landscapes and cities and at the same time see countries, highways and streets with names.

2.3.2 Navigation – zooming and panning
Web maps normally provide two ways of navigation: zooming and panning.

By the European Commission zooming is defined as “the process of magnifying or reducing the scale of a map or image displayed of the monitor” [6].

Zooming may take place in the following ways: original-centre: the new view is centred at the same location as the previous one, re-centre: the new view is centred at a point selected by the user, by marquee: the user can magnify a sub-region by selecting the opposite corners of the rectangle encompassing an area of interest, by number: the user can assign the exact scale to the map visual display by typing the scale denominator and zoom-reset: the user can reset the map and redisplay the initial view.

Panning is defined as “the process of changing the position at which the view is displayed, without modifying the scale” [6].

Panning may take place by discrete movements of the view point by using hot keys or buttons; or continuously through traditional scroll bars, by clicking the map in the wanted direction or by clicking and dragging the map with the mouse cursor.

2.3.3 Controls
The user interface of a web-based map consists of a map and an operation interface, the controls. The controls are a set of objects, of various sizes and types, that users can manipulate with the mouse or other input devices to interact with the map. In general map APIs provide controls for panning, zooming and change of map viewing mode. Pan buttons can either be grouped together or distributed around the map frame and a slider to make zooming faster and easier is also a common object.

2.3.4 Overlays
To be able to display additional information, e.g. points of interest or routes, on top of map most APIs provide several overlay objects. Following are examples of available overlays: markers, information windows, polylines, polygons and circles.
Markers
Markers, also called pushpins, are the most commonly used overlay on maps. Markers are used to display a geographical location or a point of interest. To make it possible for a user to distinguish between several different types of points, e.g. hotels and restaurants, displayed at the same time, map APIs provide functionality to customise the markers’ looks, e.g. icon, size and text. Usually more information than just a marker has to be displayed and therefore it is possible to attach more information to a location or point of interest. The information can be added to markers and be shown in an information window, also called infowindow and infobox, when the marker is clicked or mouse-overed.

Information windows
An information window, called infowindow or infobox, is a box containing supplementary information displayed on top of the map. Most commonly, the windows are attached to a marker and shown when the marker is clicked, although, some APIs also provide possibility to add information windows without adding a marker to map.

The content in information windows is normally HTML code; making it possible to create windows containing tables, images and links.

To adapt the map and the information windows to the containing site the APIs provide functionality to customize the map look and feel through Cascading Style Sheets (CSS).

Polylines, polygons and circles
Polylines, polygons and circles are objects drawn on top of the map. A polyline consists of lines, a polygon of sides and a body and a circle of a side and a body. Normally both the sides and the body are customizable by allowing the colour, width and opacity to be changed. A common type of use is, for example, to draw a route returned by the driving directions functionality using polylines.

2.3.5 Driving directions
Web-based map systems frequently offer functionality to provide users with driving directions. The directions can be two-point or multipoint, and sometimes optimisation is offered for multipoint directions to provide users with the optimal order to visit the supplied locations. The directions returned by the API are consisting of two parts: textual descriptions including road names, numbers and occasionally street signs; and overlays, i.e. polylines, drawn on the map to mark the route. In addition to this, information such as the trip duration, length and cost may also be returned.

All web-based map service providers offer several options for the directions. This may be options such as:

- directions for going: by car, by foot or by bike;
- possibility to choose the quickest, most economic or fastest route;
- possibility to exclude highways from the route.
2.3.6 Geocoding

Geocoding is one of the most important services provided by web-based map APIs and geographical information systems. This procedure can be defined as the process of locating points, or geographical coordinates, on the surface of the Earth from alphanumeric addressing data. The geocoding course of action can be divided into three phases: parsing, matching and locating.

During the parsing phase the address information sent to the geocoder is analyzed and translated into a structured and standardised database tuple which is later used in the matching phase. Since this is the geocoding face where the human input is interpreted it is of importance to deal with various ways to write an address; common mistakes, different separators and unofficial place names all have to be dealt with.

The second phase is the matching phase. This is where the geocoder tries to find the most precise reference that can be associated with the structured address returned by the parsing phase. Since the address information sent to the geocoder, from an application or user, is not always a complete address, the information retrieved from the parsing phase might be incomplete. Since some parts of the address may be missing a decision about which reference that will generate the most precise results has to be taken.

Finally, the location phase retrieves the result from the matching phase and determines the actual coordinates to assign to the given address. The process of assigning the address coordinates can be performed in a variety of ways [7]: individual point address, centreline interpolation, thoroughfare interpolation, reference area, street crossing, neighbourhood centroid, postal area and municipality. The choice of strategy is depending on the available geospatial data and the addressing system used in the requested area.

Address systems is varying for the world’s different parts. The western numbering system, sequentially increasing numbering along the street with odd numbers on one side and even on the other, is widely accepted but is far from dominant. Many cities implement special numbering which is often inherited from ancient history, and has therefore not been adapted to the needs of a modern city [8].

The most common technique used is centreline range interpolation geocoding, or also called address interpolation. This technique uses data from a geographic information system where street segments already have been mapped within the geographic coordinate space and attributed with address ranges. The geocoder takes an address, matches it to a street and specific segment, such as a block. The geocoder then interpolates the position of the address, within the range along the segment. For example if a street has numbers from 1 to 6 and number 3 is sent to the geocoder, it will return a location half way up the block (Figure 2). The problem with address interpolation is that address numbers are not evenly spaced along the street and geocoders often assume that odd numbers are on one side of the street and even numbers on the others, which as mentioned before is not always the case. This issues makes address interpolation geocoding quite inaccurate for some locations.
Another technique used is individual point address, also called roof-top geocoding by Microsoft and Multimap. The individual point address technique uses a database with land parcels stored together with geographical coordinates (Figure 3). The geocoder takes an address, matches it to the correct land parcel and returns the stored coordinates, creating a geocoding process more precise than interpolation based. The drawback is that geospatial data has to be available and at the moment this is only the case for some parts of the United States.

Some APIs, for example Microsoft Virtual Earth, use a combination of the two techniques; Individual point address geocoding is used for locations where geospatial data is available, i.e. for United States, and otherwise interpolation based geocoding is performed.

According to [7], web-based map APIs and geographic information systems should provide a Geocoding Quality Indicator (GQI) as a compliment to the returned coordinates. The indicator could be used as an assessment of reliability and accuracy as well as an indication of the range of situations in which the geocoded data may or may not be used.

In addition to normal geocoding some APIs also provide reverse geocoding which is the process of returning an estimated street address for a given coordinate. For example, a user clicks on a map causing an event to call a method which is sending the coordinate to
the geocoder, the geocoder then returns the estimated street address. The address is normally interpolated from an address range assigned to the road segment in the same way as interpolation based geocoding, e.g. if a user clicks at the midpoint of a segment that starts with address 1 and ends with 6, the returned value will be 3, but reverse geocoding could also be performed by using individual point addresses, if geospatial data is available.

Chapter 7.1, Geocoder, shows the results from a test where several web-based map API geocoders were tested in the terms of accuracy.

2.3.7 Points of Interest
A Point of Interest (POI) is a specific point location that someone may find useful or interesting. In web mapping a point of interest may be, for example, a restaurant, a hotel or a train station.

Web-map API providers offers two types of POI databases. The most common is a POI database where content of the database is managed by the company providing the API. The company cannot be controlled and such a database can be used for searches as, for example, a restaurant in New York. The other is a self controlled POI database, hosted by the API provider, containing any type of POIs. In this case the POI data stored in the database is fully controllable, and in general it is managed through a web interface.

2.3.8 Limitations
Web-based map service providers commonly provide two versions of their APIs: a free version for non-commercial use and an enterprise version to be used by companies in commercial products. The free version normally comes with transaction and geocoding limitations, and enterprise customers are typically provided with support, enhanced functionality and transactions limits based on the type of contract and price paid.

2.4 Web-based mapping and usability
Since web-based maps are integrated into websites used by people without experience of web-based maps or Geographical Information Systems, usability is an important aspect. Although the maps may be based on the same technologies there exist no standards for web-based map operations and appearance; each map service provider has developed its own interface style and layout. Therefore web-based maps do not have a common Graphical User Interface (GUI), vocabulary and syntax [9].

As mentioned before navigation in terms of zooming and panning can be done in several ways and according to [9], an original center zoom: the new rendition is centered at the same location as the previous one, together with a grouped pan button design arranged according to the directions, is the most efficient solution in terms of time and user errors. They also state that moving the map by dragging it with the mouse, usually with a hand cursor, is not only an intuitive way to navigate, it is also compatible with most other map operations, making it easy to use such a function together with others. For this reason most modern web-based maps use this as the default way to pan the map.

In addition to this study of panning and zooming usability Maurits van der Vlugt and Ian Stanley is in [10] proposing six guidelines for web-based mapping:
Terminology should be clear and unambiguous and designers should avoid jargon.

Professional designers should be used to improve the graphic design of the site. The design should enhance and complement the text and the maps, focusing the user’s attention on the content.

A meaningful legend should be presented as part of the default view of the map so that the map is self-explanatory.

A locator, or context map, that shows where the map is being viewed in relation to a larger geographic area should be provided.

Buttons should have text or icons with tooltips showing the name and describing the purpose or action. They should be large enough for users to accurately identify the text or image and to click with the mouse.

Help should be provided and there must be a range of appropriate error messages.

2.5 Technology

To create or implement a web-based map, several techniques, any programming environment, programming language and server side framework, can be used. In any case of technology choice, both server and client side technologies have to be used to be able to provide and display dynamic and interactive maps.

2.5.1 Ajax - Asynchronous JavaScript and XML

Ajax or Asynchronous JavaScript and XML, which it is shorthand for, allows web applications to be loaded and updated through several micro requests instead of just one large macro request, which is the traditional model for web applications. Jesse James Garrett, President and founder of Adaptive Path, states that Ajax is not a single technology, it is several technologies, each flourishing in its own right, coming together in powerful new ways [11]. He declares that Ajax incorporates:

- standards-based presentation using XHTML and CSS;
- dynamic display and interaction using the Document Object Model (DOM);
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- and JavaScript binding everything together.

A traditional web application works like this: the user sends, through a browser, an HTTP request; the web server processes the request, and after this returns the page to the browser as HTML/XHTML and CSS (Figure 4).
Figure 4, the traditional web application model, to the left, compared to the Ajax-based model, to the right.

When the Ajax model is used a new layer, an Ajax engine, is introduced in-between the browser and the web server. The browser is now sending JavaScript calls to the Ajax engine which is sending HTTP request to the server using XMLHttpRequest objects calls, establishing an independent and asynchronous communication channel between the client-side and the server-side (Figure 4).

As a result of this asynchronous communication, web applications are more interactive and responsive; providing the user with information on demand instead of during the initial page load.

2.5.2 Web-based maps

When an integrated map is displayed in a user’s browser window three parts are involved: the user’s browser, the web server hosting the site with the map and the API system (Figure 5).

A map system consists of several sets of servers, one set hosting static files, such as JavaScripts and images, one set used to calculate routes and other geographical information and two sets of servers hosting the road map and satellite tile images.
Figure 5, web-based map system constructed by three parts: client browser, server system hosting site with map and map API system.

On the client side Dynamic HTML (DHTML) and Ajax are used to display the map in the browser window. DHTML is the combination of using the JavaScript event model together with CSS positioning. This is what creates the possibility to drag objects, in this case the map, displayed in web pages. The map is broken up into a grip of images called tiles, which are absolutely positioned square formed images, normally 256 x 256 pixels, displaying the different parts of the map (Figure 6). When the map is dragged new images are fetched from the tile servers as soon as the area is visible and the no longer visible tiles are removed. This removing and adding process creates the map’s infinite scrolling effect [12].

Figure 6, Google maps with tiles outlined. Image taken from [12].

To create the map’s interactivity; the possibility to add map overlays, open infowindows and display information on top of the map Ajax is used. For example, when a marker is clicked, an Ajax request is made, behind the scenes, back to the server to fetch the data to
be displayed in the information window. When the data is returned to the browser the page’s DOM is dynamically updated without refreshing the entire page, displaying the spatial information on the map seamlessly like in a normal desktop application.

2.6 Geographical data

The geographical data used and displayed by the different web map APIs is in general coming from the same data vendors. The majority of the APIs use a combination of data from companies as NAVTEQ and TeleAtlas for the road maps.

The satellite/aerial imagery is coming from companies like DigitalGlobe which is photographing the Earth’s surface with their QuickBird satellite. The satellite makes fifteen orbits around the Earth per day and photographs strips which measure 16.5 by 330 kilometers. The average resolution is 60 square centimeters per pixel but because of the huge competition for the camera most of the planet has only been photographed with a low resolution. In addition to the QuickBird satellite, low resolution imagery is also captured by others satellites as the Landsat-7, which has been taking images of the entire planet at a resolution of 15 meters [13].

After the images have been taken they are sent to ground stations for post-processing. The differences in photographic angle are corrected and the images are resampled so that their pixels will be aligned with a latitude-longitude grid.

Many areas of high interest, as cities, are also photographed by aircraft to get a higher resolution; clearly visible in the resulting photos are car sunroofs, lampposts, and even people.
3 Google Maps API

This chapter is a review of the functionality provided by the Google Maps API. The information and images presented are based on the Google Maps API documentation [14] and implementations using the API.

3.1 Viewing modes

The Google Maps API offers three viewing modes: the Map mode which is topographic and street maps, the Satellite mode which is satellite and high-resolution aerial photographs and the Hybrid mode which is street maps overlaid on top of satellite/aerial pictures.

Google provides high-resolution satellite images for most urban areas and the image data is at most approximately one to three years old [15].

![Google Maps API Hybrid view.](image)

3.2 Navigation and controls

The Google Maps API map can be manipulated in several ways. A user can change the map’s position by dragging the map with the mouse or by using the map control’s pan buttons; four grouped buttons arranged according to the directions letting the user do discrete moves. The API provides the following types of zooming: original-centre zoom with zoom buttons, mouse wheel and slider; and re-centre zoom by double clicking the left mouse button. By adding extensions from the Google Maps utility library controls can be added allowing users to perform zoom by marquee.

The API offers a set of controls to navigate and interact with the map. The following controls can be added to the map: a map control (Figure 8), map-type control (Figure 9), scale control (Figure 10), overview map control (Figure 11) and a local search control (Figure 12). All control buttons are provided with localised tooltips displayed on a mouse-over action.

The map control is available in three sizes: GLargeMapControl with pan buttons and a zoom slider, GSmallMapControl with pan and zoom buttons and small version called GSmallZoomControl which only contains buttons for zooming.
The map type control, GMapTypeControl, contains buttons to choose between the three viewing modes: Map, Satellite and Hybrid.

The scale control, GScaleControl, is a scale indicator displaying the map’s current scale, like the scale indicator on traditionally maps. The scale control shows the current scale in both miles and kilometres.

The overview map control, GOverviewMapControl, shows a small map in the corner of the main map, showing the user the current viewport in relation to a larger geographic area. The overview map is fully interactive – if it is dragged, the main map is panned to the new position as well.

The local search control is a location search field showed directly on top of the map. To incorporate a local search into a Google Maps API mashup, references to the Google Ajax Search API and the Local Search control must be added.
3.3 Overlays

3.3.1 Markers
A marker is a way of indicating a geographical location, longitude and latitude, on a map. With the Google Maps API this is done by using the GMarker object. There is no limitations of the number of markers that can be placed on the map but anecdotal evidence suggests that the performance starts to slow down after a hundred or so markers [16].

The marker object constructor takes a point, latitude and longitude, and an optional object, containing marker options such as an icon object, as arguments. The icon object creates the possibility to create several types of markers, with custom images or icons (Figure 13). An icon consists of several images: a foreground image, a shadow image and images to use if a map is printed.

The API has a set-image method to make it possible to change the icon image after the marker has been added to the map; however, neither the print image nor the shadow image can be adjusted. Therefore, this method is primarily intended to implement highlighting or dimming effects, rather than drastic changes in markers appearances.

Figure 13, a Google Maps marker displayed with a custom icon.

3.3.2 Infowindows
To display additional information such as, text, images or links, on top of the map an info widows can be added to the map. The API’s infowindow class, called GInfoWindow (Figure 14), is a floating window with HTML content displayed above the map. An info window can be attached directly to a position on the map or to a marker object and it can contain several tabs. Only one window can be open on the map at the same time and if an infowindow is opened outside the current viewport, the map will pan smoothly until the entire window is inside.

Figure 14, standard markers with an info window.

3.3.3 Polylines and polygons
Polyline and polygon objects are used to create custom objects (Figure 15), such as lines showing a route or a polygon highlighting a region, on top of the map. Both polylines and polygons objects are created from an array containing latitude and longitude points and a set of option arguments. The objects’ colour, width and opacity can be customised to adapt the object to the specific use.
The polyline and polygon objects are displayed using Vector Markup Language (VML) and for this to work in Internet Explorer it is necessary to include a specific line of HTML code:

```html
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-microsoft-com:vml" />
```

Figure 15, polyline displayed on top of Google Maps' Map mode.

### 3.3.4 Custom controls

With the Google Maps API a subclass of the `GControl` object can be created; providing the possibility to add custom controls and elements to the map. When a custom control is created two handlers for two class methods are required: `initialize()` method and `getDefaultPosition()`. The `initialize()` method must return a DOM element, while the `getDefaultPosition()` method must return a position object. Custom controls can be used to create controls with a custom look and feel or controls with functionality different from the standard functionality provided by the API.

### 3.3.5 Marker manager

Adding a large number of markers to a Google map may both slow down rendering of the map and introduce too much visual clutter, especially at certain zoom levels. The marker manager utility provides a solution to both of these issues, allowing efficient display of hundreds of markers on the same map and the ability to specify at which zoom levels markers should appear.

The marker manager offloads management of markers registered with the utility, keeping track of which markers are visible at certain zoom levels within the current view, and passing only these markers to the map for drawing purposes. The manager monitors the map's current viewport and zoom level, dynamically adding or removing markers from the map as they become active.

A problem with the marker manager is that there is no possibility to remove markers provided. However, the Google Maps utility library, `http://gmaps-utility-library.googlecode.com/`, offers a new version of the marker manager with marker remove support.
3.4 Route and driving directions

The Google Maps API can provide users multiple-points driving directions as a series of textual descriptions, the route marked on a map with polylines or both at the same time (Figure 16).

![Figure 16, driving directions on map and textual descriptions.](image)

The API can translate the textual instructions into several languages by setting the locale parameter. Currently, it has translations for: Japanese, French, German, Italian, Spanish, Catalan, Basque, Dutch, and Galician.

Instead of using the preformatted textual descriptions an event listener can call a method used for presenting the directions. This call-back method can use the routes and step information, with distance and duration, stored in the direction object returned by the API to create textual descriptions.

In addition to the directions the API offers the possibility to add information about the current traffic situation to the maps (Figure 17). The traffic information is displayed with polylines and is only available for some supported cities.

![Figure 17, the current traffic situation in Jersey City displayed on a map from Google’s map API.](image)
3.5 GeoXML overlay
The Google Maps API supports the Keyhole Markup Language (KML) and the GeoRSS data formats for displaying geographic information. KML is an XML-based language used to describe geospatial data and GeoRSS is an extension to RSS containing information about geographical points, lines, and polygons.

Geographic information provided in these data formats are added to the map by using the \texttt{GGeoXml} object, whose constructor takes the URL of a publicly accessible XML file. \texttt{GGeoXml} placemarks are rendered as markers, while \texttt{GGeoXml} polylines and polygons are rendered as Google Maps API polylines and polygons.

The API currently supports the following KML features:

- Placemarks as GMarkers
- Icons with a size of 32x32 px
- Descriptive HTML displayed inside infowindows
- Polylines/Polygons
- Styles for polylines/polygons
- Ground Overlays without rotation/opacity

The API also currently supports the display of markers, using the default icon, based on latitudes/longitudes provided in a feed tagged with GeoRSS elements.

3.6 Events
The API offers a possibility to add event listeners. Events can be triggered on mouse movement, mouse clicks and keystrokes.

3.7 Language support
The Google Maps API supports the following languages [17]:

- Japanese
- French
- German
- Italian
- Spanish
- Catalan
- Basque
- Dutch
- Galician

A change of the locale parameter will affect the language used on the map’s controls, tool tips and copyrights.
3.8 Geocoding

The Google maps API has a geocoder which can be accessed directly by sending HTTP requests to Google or by using the geocoder object to send requests within a JavaScript. The geocoder can be modified to prefer results within a given viewport or country and by default it has a client-side based cache to make geocoding faster if information for the same location is requested several times.

The Google Maps API provides two methods for geocoding: getLatLng that returns the latitude and longitude for a location and getLocations that returns objects containing structured information about the locations. The get latitude and longitude function only returns the best hit while the get locations method returns all the locations matching the search string.

The get location method returns a JavaScript Object Notation (JSON) object containing the following information:

- Status, response type and response code
- Placemark(s) containing:
  - An address
  - Address details
    - The address formatted as eXtensible Address Language (xAL)
    - Accuracy - An attribute indicating how accurately the given address was geocoded
  - Point - A point in 3D space with coordinates: the address’ longitude, latitude and altitude

The geocoder can currently perform geocoding in [18]: Austria, Australia, Belgium, Brazil, Canada, The Czech Republic, France, Germany, Hungary, Italy, Japan, Luxembourg, New Zealand, The Netherlands, Poland, Portugal, Spain, Sweden, Switzerland, the United Kingdom and the United States.

3.9 Limitations

Since version 2, there are no longer any page view limits with the free version of the API. If, however, the site gets more than 500,000 page views per day, Google asks to tell them before launching the site so that they can prepare in advance to handle the traffic [19].

The Google Maps API geocoder has a rate limit of the equivalent of 50,000 requests per day (1.725 seconds per request). If a site reaches this limit for several minutes it will be blocked by Google a day. If the limit continues to be abused, the access to the Maps API geocoder may be blocked permanently [20].
3.10 Google maps for enterprises

Google Maps is available with full enterprise licensing and support. The price is based on the number of page views and geocode requests handled by the Google Maps for Enterprise API and starts at $10,000 per year.

Google Maps for Enterprise offers several features, including implementation guidance, telephone support and the ability to use Google Maps for internal and external applications.

Another reason to use the enterprise edition is if Google chooses to start displaying advertisements on the maps it will be optional to include it.

More information about the enterprise API is available at:
http://www.google.com/enterprise/maps/

3.11 Documentation and support

The Google API documentation is divided into two parts; one part containing concepts and examples and one part containing the API class reference. The concepts and example part includes an introduction to the API, map and geocoder examples and an API overview.

Since the Google Maps API is widely used the availability of code examples and support is good. To get information, examples, news and help Google provides a discussion group (19464 items 05/07/2007), a FAQ and a developer blog. These sites can be found at the following addresses:

API FAQ: http://code.google.com/support/bin/topic.py?topic=10028

API Discussion Group: http://groups.google.com/group/Google-Maps-API

Among other things the discussion group includes:

- A bug tracker available at:
  http://groups.google.com/group/Google-Maps-API/web/known-possible-api-bugs

- Feature requests available at:
  http://groups.google.com/group/Google-Maps-API/web/api-feature-requests

- Links to examples and tutorials available at:
  http://groups.google.com/group/Google-Maps-API/web/links-to-examples-tutorials

API Blog: http://googlemapsapi.blogspot.com/

In addition to the Google provided resources there is also a wiki, http://mapki.com, which holds a getting started guide, a list of Google Maps projects, code snippets, a knowledge base, a collection of developer tools and much more.
3.11.1 Extensions

In addition to the functionality provided by Google, the Google Maps developer community provides several third-party extensions to the API. Among other useful information a list of third-party extensions and some tutorials are available at Mike Williams Google Maps tutorial site available at the following address: http://www.econym.demon.co.uk/googlemaps/.

To keep the file size down Google has excluded some functionality to get a reliable and quick-loading API, however there is an open source project called Google Maps utility library with useful extensions. The JavaScript files can be included from their server or downloaded for local use.

The utility library contains extensions such as: zoom by marquee, an advanced marker manager with marker delete support and labelled markers.

The library is available at: http://code.google.com/p/gmaps-utility-library/
4 Multimap API

This chapter contains a review of the functionality and services provided by Multimap’s web-based map API. The information and images are based on the API documentation [21] and implementations using the API.

4.1 Viewing modes

The Multimap API offers three viewing modes: Map mode with topographic and street maps, Aerial mode with satellite photographs and Hybrid with street maps overlaid on top of satellite/aerial photographs.

Different from most other map APIs Multimap provides maps from several data vendors through a partnership program. If there is more than one map available for the currently view location, the user can choose vendor by placing the mouse cursor over the map type widget’s map button.

The Hybrid viewing (Figure 18) mode in the Multimap API differs from other map API providers’ hybrid viewing modes; instead of overlaying the entire map with street maps Multimap only overlays the map inside a square controlled by the user.

![Hybrid view](image)

**Figure 18, Multimap’s web-based map API using the Hybrid view, notice that only a part of the map is overlaid with street maps.**

4.2 Navigation and controls

The Multimap API map’s viewport can be manipulated in several ways. By default the API provides the following types of zooming: original-centre by using control buttons or slider; or re-centre by double clicking the left mouse button. The map tools widget also provides: re-centre zoom by single left click and zoom by marquee. Panning can be performed: by using the grouped pan buttons, dragging the mouse or by a single left click on the map, which is provided by the map tools widget.

In addition to the mouse map navigation, the API also provides the following controls: pan and zoom widget (Figure 19), map type widget (Figure 20), map tools widget (Figure 21), location widget (Figure 22), overview widget and a context menu (Figure 23). All
control buttons without text are provided with tool tips in English displayed when the mouse cursor is placed over the button.

The pan and zoom widget is available in three sizes: `MMPanZoomWidget`, `MMSmallPanZoomWidget` and `MMSmallZoomWidget`. This widget provides users with buttons for panning the map, returning to the initial start location and zooming using a slider or zoom buttons. The pan buttons, causing the map to make a discrete move, are displayed in a group and arranged according to the direction the frame around the map will move.

![Pan and zoom widget in three sizes.](image)

The map type widget, `MMMapTypeWidget`, lets the user change between the three available map viewing modes. If there are road maps available from several data providers the user can choose provider by placing the mouse cursor over the map mode button.

![Multimap's map type widget.](image)

The API’s tools widget allows users choose how the map behaves when it is dragged or clicked with the mouse. The user can choose between five modes: drag map, drag to zoom (zoom by marquee), drag to navigate, click to zoom in (re-centre zoom in) and click to zoom out (re-centre zoom out).

![Map tools widget.](image)

The API provides a widget, `MMLocationWidget`, which displays the currently viewed location.

![Location widget.](image)

The overview widget, `MMOverviewWidget`, displays a small map in the corner of the main map showing the user the current viewport in relation to a larger geographic area.
The overview map is fully interactive – if it is dragged, the main map is panned to the new position as well.

When a user right clicks on the Multimap API’s map a context menu is presented. By default the menu contains items to zoom in/out at, move the map to and add a marker at the clicked location. The menu’s contents can be changed; items can be added and removed at any time.

![Figure 23, right click context menu.](image)

All widgets standard appearance can be customised; the Multimap API offers a possibility to override the style classes and this creates the possibility add control widgets with a look and feel adapted to web application where the map is used.

### 4.3 Overlays

#### 4.4 Marker

Markers are used to display locations or points of interest on top the map, to provide users with information additional to the information provided by the actual map. Multimap’s API provides a marker object, `MMMarkerOverlay`, which is created by sending a position, longitude and latitude, and a marker option object to the marker constructor. The marker option object contains information such as: the icon to use, if the marker should be draggable and the label to display on top of the icon image. The marker options, including icon settings, can be changed at any time, by calling a reset method after the marker has been added to the map. This can be used to create marker highlight and dimming effects as well as for updating the label.

![Figure 24, markers with infoboxes.](image)

The marker object can be assigned information which will be displayed in an infobox (Figure 24, to the left), also called info window, opened from the marker when the maker is clicked by a user. The API supports tabbed info boxes with HTML contents and if an infobox outside the current viewport is opened the map will be panned until the entire box inside the frame.
To provide customisability of the infoboxes the API offers possibility to override the default style sheet; creating the possibility to add infoboxes with a look and feel adapted to the application where the map is used (Figure 24, to the right).

### 4.4.1 Decluttering markers

When markers for two or more separate locations are added to a map there is a chance that the icons that mark them may overlap, especially when the map in question is on a large scale or has been zoomed out. To avoid this, Multimap's API offers decluttering options for markers displayed close to each other.

Markers can be arranged into groups and there are two methods for arranging the groups:

- **Aggregate** (Figure 25, to the left): works by amalgamating all markers that overlap into a single marker based on the average mean position of all the overlapping markers. Any infoboxes that are attached to markers that are aggregated become grouped together automatically.

- **Grid** (Figure 25, to the right): works by calculating which markers overlap and arranging them on a grid.

### 4.4.2 Polyline and polygons

Polylines and polygons are the same type of objects in the Multimap API; they are drawn on the map the same way and the only difference is an argument telling the API to draw a line between the first and the last point in the array of points sent to the drawing method.

The API allows the change of line colour, opacity, line width and fill colour, to be able to customize and adapt the objects. The polyline and polygon objects are, as with Google Maps, displayed using VML and for this to work in Internet Explorer it is necessary to include the flowing line of HTML code:

```html
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-microsoft-com:vml"/>
```
4.5 Events
The API offers a possibility to add event listeners. The API can catch any mouse events or events sent by the map.

4.6 Language support
The textual driving instructions are available in the following languages [22]:
- English - United States
- English - United Kingdom
- Danish
- German
- Spanish
- French
- Italian
- Dutch
- Norwegian
- Portuguese
- Swedish.

4.7 Driving directions
The API includes functionality to provide users with multiple-point driving directions. This is done by creating a `MMRouteRequester` object and request a route. The requester object will, when the results are placed in the route object sent to the request method, call a specified callback method. In difference to other web-based map APIs the Multimap API's driving directions methods do not plot the returned route on the map automatically; the API return an object, `MMRoute`, containing the total distance and duration as well as an array of polylines which are used to plot the route. In addition to this the route object also contains an array of stages which route parts in-between two waypoints. Each stage consists of several steps; the textual descriptions.

Driving instructions provided by Multimap may look like the following:
Table 1, driving directions example from http://www.multimap.com/share/documentation/api/1.2/demos/basic_routing.htm.

Directions can be created for the quickest or shortest way and the estimated time can be based on walking or driving. It is also possible to exclude highways, create multi-modal directions, drive here then walk here, and to optimize the location visiting order, to make the route as short as possible.
4.8 Geocoding
Multimap provides functionality for translating postal addresses into geographical coordinates, geocoding. The Multimap API geocoder can be accessed either within a JavaScript, with the MMGeocoder object, or through a Representational State Transfer (REST) web service; a software architecture for distributed hypermedia systems. When a geocoding operation is performed, address calculations and look-up is done with interpolation; however, Multimap has a partnership with Microsoft and will in the future provide Microsoft MapPoint and Virtual Earth’s roof-top geocoding service.

The geocoder takes an address object as an argument and returns a set of location objects, one for each matching location. The address object, MMAddress, contains information such as area, city, zip code, country code and a quick-search string, and the returned location objects contain information such as the returned location’s coordinates, address and the ideal zoom factor to use when displaying it, but no information about how accurate the results are.

When a geocoding operation is performed not all address information has to be provided; the easiest way is to provide the quick-search string parameter which is an unparsed text containing all address information. However, one thing to take into consideration is that it may be easier for the geocoder to parse the address information if it is sent separated in parts rather than as a single string containing all information, and there may also be differences in parsing accuracy depending on how the parts in the string are separated, e.g. with spaces or commas.

Chapter 7.1, Geocoder, shows the results from a test where several web-based map API geocoders were tested in terms of accuracy. Multimap was tested both with addresses sent separated, using city, zip code and street, and with the quick-search string using both space and comma separation.

4.9 Location information and searching
The Multimap API provides functionality to display points of interest on the map. The points of interest can be fetched either from Multimap’s location information database or from a self-controlled POI database, stored by Multimap.

The location information database contains POIs, such as airports, petrol stations or restaurants, in 32 countries [23], but there is no information about the contents available.

The self-controlled POI database is managed through a web interface called the Client Zone portal and the API provides several methods to perform searches with the stored records. Below are some examples of available search methods:

Spatial search
Spatial searches are based on locations and can be used to get results such as “Where is the nearest …?” or “What is within the area …?”.

Non-spatial search
Non-spatial searches are based on non-location criteria such as name, opening hours or categories. Performing searches of this kind is also known as ‘filtering’.
**Combined search**
A combined search can be used to find results of the certain type, filtering, within a certain area, spatial search.

**Searching along route**
A search along a route is used to find points of interest along a route between two or more locations within a specified distance from the route.

**4.10 Limitation**
The Open API, the free version, of the API has a transaction limit set to 5% of the traffic of Multimap’s public site, www.multimap.com, and one individual site is allowed to use up to 1% of community’s total traffic [24].

**4.11 Web services**
In addition the JavaScript API Multimap also provides web services, in this case REST, for geocoding, search and routing. Web services start with a request being created. The source can be a browser or an application. The request is sent as a query string formatted into an XML document and transmitted across the Internet to the web server on a given port, usually port 80 which is the HTTP port. When a request arrives, the query is parsed to determine which components need to be instantiated and what methods to call. Finally, the result is bundled back up into an XML document and sent back to the calling application.

By default, the Multimap API web services return an XML response. The XML format that is returned depends on the module that has been called, such as the Geocoding, Searching or Routing module, but common elements between each element are defined in the same way across all modules.

**4.12 Documentation and support**
The Multimap API is documented with an online software development kit (SDK). The SDK includes explanations and basic and advanced examples with source code for all the functionality provided by the JavaScript and the web services APIs. In addition to this a full class reference is available.

- The Javascript API documentation is available at: http://www.multimap.com/apidocs/1.2/demos/index.htm
- The web services API documentation is available at: http://www.multimap.com/apidocs/1.2/web_service/index.htm
- The full class reference documentation is available at: http://www.multimap.com/apidocs/1.2/classes/

For business customers Multimap provides telephone and e-mail support by their support teams across the globe.
In addition to this support Multimaps also provides a developer forum for the Open API and a blog. However, the latter two support options are new and are not helpful at the moment.

The blog was opened in late may 2007 and has at the moment 4 posted items (23/07/2007). The blog is available at: http://blogs.multimap.com/

The developer forum for the Multimap Open API has 7 topics at the moment (23/07/2007). The forum is available at: http://forums.multimap.com/
5 ViaMichelin Maps and Drive API

This chapter is a review of the functionality provided by ViaMichelin’s Maps and Drive API. The information and images are based on the API documentation [25] and implementations using the map API.

5.1 Viewing modes

The ViaMichelin Maps and Drive API only provides one viewing mode, the Map mode with topographic and street maps (Figure 27). The maps have the same colours and symbols as ViaMichelin’s normal printed maps, which are common in Europe.

![Figure 27, a map from ViaMichelin's Maps and Drive API showing Biarritz, France.](image)

5.2 Navigation and controls

The ViaMichelin Maps and Drive API map can be navigated by clicking or dragging the map with the mouse. Original-centre zooming is made by clicking zoom buttons or by using zoom slider. Panning can be made by: using map tools control’s pan buttons, by clicking the map or by dragging the map with the mouse.

The API offers one all-in-one control called Map Tools (Figure 28). This control contains: a pan control, a zoom level control, a zoom box control and a point of interest (POI) control. The control can be viewed in three modes: simple, scale + navigation and full with POI.

![Figure 28, three views of map tools: full, scale + navigation and simple.](image)
5.3 Shapes

5.3.1 Icons
Icons, also called markers or pushpins, are used to display a location on the map (Figure 29). With the ViaMichelin API an icon is added to the map by first creating an icon layer object, `VMIconLayer`. The icon layer’s constructor takes three arguments geographical coordinates, an icon object and HTML contents to display the icon's tool tip. The first argument, a `VMLonLat`, is mandatory but the latter two are optional; the icon object is only used if a custom icon image is to be used and if no HTML content is passed to the constructor the tooltip will be disabled. A custom icon is specified and sent to the icon layer object during creation and cannot be changed after the layer has been created and added to the map.

![Figure 29, default and custom icon displayed in map.](image)

5.3.2 Tooltips
To display information, text, links, and images, on the map tooltips are used (Figure 30). A tooltip, also called infowindow or infobox, is always attached to an icon layer object and is displayed when a user clicks the icon object with the mouse. The content of the tooltip is HTML code; which makes it possible to customise the text, links and images displayed inside the tooltip. When an icon is added and a new tooltip displayed, the map will pan until the tooltip is centered in the current viewport.

![Figure 30, a default ViaMichelin Maps and Drive tooltip.](image)

5.3.3 Polylines, polygons and circles
The Polyline, polygon and circle objects (Figure 31) are, as explained by their names, used to create lines, custom shaped objects and circles displayed on top of the map; and for this the API provides three classes: `VMPolyLine`, `VMPolygonShape` and `VMCircleShape`.

Polylines and polygons are created from an array of coordinate objects, `VMLatLon`, and circles are created from a coordinate object and a radius in meters. The objects can be
customized by setting: stroke colour, fill colour, stroke width in pixels and opacity as a value between 0 and 1.

Figure 31, polygon and circle objects.

5.3.4 Complex layers
When large number of shapes are added to the map it can be hard to manage them, and therefore ViaMichelin provides a complex layer object to which all types of layers, icons, polylines, polygons and circles, can be added at the same time. A layer can be used to remove a group of shape objects or to calculate the boundaries of all shape objects added to a layer.

5.4 Driving directions
The API's driving directions, in the API called itineraries, are used to calculate an itinerary between two map points, possibly including intermediate steps, and according to various parameters. In Europe and North America it is possible to choose among customized itineraries: recommended by Michelin, shortest, quickest, economical, discovery, on foot or by bike. Toll costs are automatically calculated for automobile, motorcycle or caravan. Petrol costs are also computed based on the consumption of a hatchback, compact, family car, sedan or luxury car.

This route plan takes into consideration major road works and planned road closures for the date specified on main roads in France, Germany and Italy. In France fixed speed cameras are displayed.

To perform a driving directions request an itinerary object, VMItinerary, has to be created. The start, stop as well as the intermediate locations are then added to the itinerary object. One thing to notice is that the driving directions method not performs geocoding and, therefore, all the locations have to be geocoded before sent to the itinerary object. When the route is calculated a specified callback method is called and the results can be fetched in the itinerary object. The route is not plotted on the map automatically but a complex layer containing all shapes is stored in the itinerary object and the directions can for that reason easily be displayed.
5.5 Events
The API offers a possibility to add event listeners. Events can be triggered on, mouse click, mouse move start, mouse move stop, pan start, pan stop, zoom in and zoom out.

5.6 Weather
The Maps and Drive API enterprise edition provides weather information (Figure 32). A daily weather report or a five day forecast can be received for a location. The information is returned in a weather object, \textit{VMWeather}, containing the report as a \textit{VMWeatherInformation} object and the forecast as an array of \textit{VMWeatherInformation} objects. The meteorological measurements can be retrieved as either as preformatted HTML, as a string, or by accessing the information variables in the weather information objects.

Figure 32, daily weather report displayed with preformatted HTML in a tool tip.

A weather report contains the following information, or meteorological measurements [26]: measured temperature, apparent temperature, minimum temperature as forecast, maximum temperature as forecast, URL to an image symbolising the indicated weather, description of the image, wind speed, wind direction, relative humidity information in percent, visibility, day when the measurement was done or when forecast was predicted and hour of the report.

5.7 Language support
The ViaMichelin Maps and Drive API supports for the following languages:

- French
- English
- German
- Italian
- Spanish
- Dutch.

A change of language will affect the roadmap, the weather reports and forecasts and POI searches.
5.8 Geocoding

The ViaMichelin Maps and Drive API offers geocoding functionality for translation of postal addresses into geographical coordinates. For this process the API provides two objects: VMGeocoder and VMGeoSearch. Both objects have a search method taking an address object, VMAddress, as argument but the VMGeocoder returns the best hit as longitude and latitude, VMLonLat, and VMGeoSearch object’s search method returns a list of all matching locations as full address objects containing the information below.

An address object contains the following information:

- Coordinates – longitude and latitude
- Address – Address of the map point in text format (number, street, avenue, etc.)
- Zip code – Zip code of the map point
- City – City of the map point
- Country – Country of the map point
- Country ISO code – Country code of the map point in ISO format (3 characters)
- Country VM code – Country code of the map point in ViaMichelin format
- VM address line – Address preformatted by ViaMichelin
- VM city line – Zip code and city preformatted by ViaMichelin
- VM ambiguity line – Complete address preformatted by ViaMichelin to remove ambiguities

It is not mandatory to provide all of this information to perform a geocoding search; however, if the ISO country code is not specified the geocoder tends to choose European locations even though the location is a major city. For example a search for New York without a country code gives Lincolnshire - Skegness - New York in the United Kingdom as result.

Two things to notice are that it is neither possible to send an address as an unparsed text; it has to be separated into several parts, nor any information about the accuracy of the geocoding results is given.

In addition to the normal geocoding the Maps and Drive API enterprise edition also offers the possibility for reverse geocoding, which means that a postal address can be calculated from geographic coordinates. This is done by creating a VMReverseGeo object and calling the search method with a latitude/longitude object as input. The results will be returned as an address object.
5.9 Points of Interest

5.9.1 Proximity search
For professional customers, using the enterprise edition, ViaMichelin provides possibility to create a point of interest database hosted on their servers. The database is used as source for proximity searches, which means that a search can be used to find, for example, restaurants in Paris or car rental companies in New York, depending on the information stored. The database is managed through a web-based interface available at: http://www.admin.viamichelin.com.

5.9.2 Hotel search
Based on the hotel reservations service available at www.viamichelin.com; the search engine provides search methods for hotels proximity for a specified location. The search results include quality ratings from the Michelin Guide and the hotel database consists of more than 40000 hotels in Europe [26].

5.10 Limitations
The free version of the Maps and Drive API has a request limit set to [27]:

- 10,000 Requests in a 24-hour period from midnight to midnight for requests to view maps and/or search for hotels.
- 1,000 Requests in a 24-hour period from midnight to midnight for requests to calculate routes and/or for geocoding.

If this number of requests is exceeded, ViaMichelin reserves the right to suspend access to Maps & Drive API Services by users of Client’s Site until the expiration of the 24-hour period during which said limits were exceeded.

5.11 ViaMichelin Maps and Drive for enterprises
ViaMichelin offers an enterprise edition of the API. This edition has more features, such as: reverse geocoding, weather information and self-controlled POI database, no advertisement and customer support services in five European countries.

Support can be given in the following languages:

- French
- English
- Spanish
- Italian
- German.
5.12 Documentation and support

ViaMichelin provides a developer network site available in French and English. The site contains code examples and API documentation and is available at the following addresses:

- English: http://dev.viamichelin.fr/wswebsite/gbr/jsp/vmdn/VMDN-Api-Maps-Drive.jsp
- French: http://dev.viamichelin.fr/wswebsite/fra/jsp/vmdn/VMDN-Api-Maps-Drive.jsp

ViaMichelin also provides a developer blog where news is posted. The blog does not contain a lot of news concerning the API; it is mostly containing publicity for the API and the professional version.

The blog is available in English and French at the following addresses:

- French: http://dev-blog.viamichelin.com/fr/
6 Virtual Earth Map Control API

This chapter is presenting the functionality provided by Microsoft’s Virtual Earth Map Control API. The information is based on development using the API and documentation available at the Virtual Earth Development site [28].

6.1 Viewing modes

The Virtual Earth API, called Map Control, offers five viewing modes: Road with topographic and street maps (Figure 33, top left), aerial with satellite/aerial photographs (Figure 33, top right), hybrid with street maps overlaid on top of satellite/aerial photographs (Figure 33, bottom), Bird’s eye (Figure 34) view with high-resolution aerial photographs and 3D viewing mode (Figure 35) with three dimensional models of cities and constructions.

Figure 33, Virtual Earth’s Road, Aerial and Hybrid views.

The bird’s eye view is a feature that lets the users see high-resolution pictures of cities in four different angels. The images are high-quality, taking in 45 degree angle and available in two zoom levels. In this viewing mode the users can see signs on tops of buildings, facades and advertisements. The bird’s eye view is currently available for all major cities and according to the news site five.tv (http://www.five.tv) by 2008 they aim to have photographed 900 European cities, all cities with a population greater than 50000 [29].
The 3D mode is a way to see 3D maps instead of normal flat maps. In this mode the user can fly around, rotate and tilt and to make the 3D model look realistic the buildings are textured using images taken from different angles.

6.2 Navigation and controls

The Virtual Earth map’s viewport can be navigated in several ways, zooming can be performed by: original-centre using zoom with buttons, mouse wheel or slider and re-centre zoom by double left mouse click; and panning by using the keyboard arrow keys or my dragging the map with the mouse.

Additionally to the mouse and keyboard navigation the Map Control API provides several controls, a dashboard (Figure 36), a find control (Figure 37) and a mini map (Figure 38), for navigation and map manipulation. All control buttons, without text explanations, are provided with tooltips in English.

The dashboard control is available in three different sizes, normal, small and tiny, and set by calling the $SetDashboardSize$ method with a size argument.

The normal size lets the user to zoom in and out using a slider and buttons and to switch between road, aerial, hybrid, bird’s eye and 3D viewing modes. The small dashboard provides buttons for zooming and to switch viewing mode between road, aerial and hybrid and with the tiny it is only possible to zoom in and out.
With the find control, added by calling the `ShowFindControl` on the `VEMap` object, a user can search for locations and businesses directly on the map. If the result is ambiguous the user can choose the most suitable alternative by selecting it among the group of matching results displayed on the map.

The mini map control, showed by calling the `VEMap` object’s `ShowMiniMap` method, shows a small map which gives the overview of the current viewport in relation to a larger geographic area which on the mini map is indicated with a square. The mini map is fully interactive – if it is dragged, the main map will be panned to the new location as well.

In addition to these controls the API also provides functionality to add custom controls to the map; offering possibility to create controls and elements with a custom look, displaying additional information or providing functionality additional to the default functionality provided by the API standard controls. A custom control is added by calling the `VEMap` object’s `AddControl` which takes a HTML element and a z-index as arguments.
6.3 Overlays/shapes

6.3.1 Pushpin
Pushpins, also called markers, are used to display geographic locations, longitude and latitude, or POIs on the map. With Virtual Earth all shapes are the same object type. When a shape, for example a pushpin, is created a VEShape object is created and a VEShapeType, in this case VEShapeType.Pushpin, is sent as an argument to specify the type of shape to create.

The shape object has SetCustomIcon method, taking a string as argument, which is used to change the pushpins’ icon. The argument string can be either HTML code or an URL to an icon images; if the argument string begins with the < character, it is considered to be HTML and if it begins with something else it is treated as URL to an image. If the URL is invalid or does not point to an image, nothing will be displayed for the pushpin icon.

A pushpin can also be assigned information which will be displayed in an infobox (Figure 39) opened when a user mouse-overs the pushpin. An infobox’s content is HTML and the API’s default infobox style sheet can be overridden. This creates the possibility to display infoboxes with a look and feel adapted to the specific application.

![Figure 39, Pushpin with infobox.](image)

6.3.2 Polylines and polygons
Polyline and polygon objects are used to create lines and custom formed objects on top of the map. Both objects are, as pushpins, VEShape objects and added to the map with the VEMap object’s AddShape method called with either with a VEShapeType.Polyline or VEShapeType.Polygon argument and an array of coordinate objects. If the shape is a polygon the last point in the array will be connected to the first point.

The shape object class provides methods for setting the line width, line color and opacity for polylines and polygons and additionally a polygon method for setting the fill color.
6.3.3 Shape layers
Shape layers are a mechanism to create and manage arbitrary groups of shapes, pushpins, polylines, and polygons. A map shape layer can be created from GeoRSS XML files, custom map tiles, or from any public Live Search Maps collection.

Any type of shape can be added to the layer and the layer class provides methods for hiding/showing the layer and to calculate a best-fit, $VELatLongRectangle$ object, based on the shapes currently present in the layer.

6.3.4 Messages
The Map Control API offers a method, $VEMap.ShowMessage$, to display a specified message in a dialog box on the map. The message dialog box appears for 10 seconds, and then disappears automatically. Neither the look of the dialog box nor the title in a message window is possible to customise, only the actual message itself.

6.4 Route and driving directions
The API provides functionality to give users two-point driving directions. Directions can be requested, by calling $VEMap.GetRoute$, for the shortest or the quickest way and results can be indicated in either miles or kilometres. The get-route method is called with start and end locations, which can be sent as string values of addresses, place names, or as $VELatLong$ objects.

The directions are returned as polylines displayed on the map (Figure 40) and a route object. The route object, a $VERoute$, contains itinerary objects for the starting point, the final destination as well as the itinerary; together creating the textual step-by-step driving directions for the route.

This itinerary object contains the following items:

- **Distance**, the total distance of the route.
- **Distance unit**, defines the units, miles or kilometres, used in the route.
- **Time**, the total estimated driving time of the route.
- **Route type**, defines the type, quickest or shortest, of the route.
- **Segment(s)**, containing:
  - **Segment**, the driving instructions for this segment of the route.
  - **Distance**, the distance of this segment of the route.
  - **Start point**, longitude and latitude for the start point of this segment.
6.5 GeoRSS and VECollection layer
The Map Control API includes mechanisms for importing Live Search Maps collections, VECollection, and GeoRSS feed data for creating shape layers containing geographic information. The layers are created by using the ImportShapeLayerData method and source files or feeds can contain location objects such as pushpins, polylines and polygons.

6.6 Events
The API offers possibility to add event listeners, keyboard and mouse events such as key presses, left/right/middle mouse clicks, mouse-overs on shapes, and mouse wheel scrolls can all be caught. If the event occurs directly on a shape object, access is given to that specific object to perform any action. The default Virtual Earth event actions can be used or overridden by own custom actions [30].

6.7 Language support
The Virtual Earth Map Control API only has localisation support for English.

6.8 Overfetching
Overfetching also knows as tile buffering means that one to three extra rings of tiles outside of the current map view are fetched when the map is loaded. The tradeoff is that tile overfetching improves panning performance but increases the page's initial load time.

The table below shows the amount of data fetched for four levels of overfetching. In this test the map was viewed using road map mode cantered to latitude: 50, longitude 10 with the zoom level set to 5 and the map element size set to 900 x 600 pixels.

<table>
<thead>
<tr>
<th>Overfetching level</th>
<th>Number of image requests</th>
<th>Total size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41</td>
<td>433 KB</td>
</tr>
<tr>
<td>1</td>
<td>83</td>
<td>666 KB</td>
</tr>
<tr>
<td>2</td>
<td>133</td>
<td>913 KB</td>
</tr>
<tr>
<td>3</td>
<td>195</td>
<td>1.18 MB</td>
</tr>
</tbody>
</table>

Table 2, data loaded with Virtual Earth’s four levels of overfetching.
6.9 Geocoding

The Map Control API provides the ability to perform geocoding by passing an unparsed address string to the find service, `VEMap.Find`. The API parses the address string, prompting the user with possible matches for ambiguous queries, can be disabled, and returns the results from the geocoder.

In addition to the location geocoding the find method also takes an argument called `what`, providing the possibility to perform POI searches. The POIs returned by the find method are not specified; however it is possible to check if the search results are sponsored advertisements or not.

When a find request is done the find method sends two results to the specified callback method, a what-result: containing information such as name, description, coordinates and phone number about POIs found and a where-result: containing name and coordinates of matching locations. The API does not supply any information about the results’ accuracy.

In addition to the normal interpolation geocoding provided by most web-based map APIs, the Virtual Earth API also provides roof-top geocoding. Roof-top geocoding is more accurate than a geocoder using interpolation; however, the land parcel data needed to perform roof-top geocoding is at the moment only available for 40 percent of the addresses in the United States.

6.10 Limitations

The free version of the Virtual Earth API has a transaction limit set to 100,000 transactions during a 24-hour period [31].

6.11 Virtual Earth for enterprises

The Virtual Earth platform can be purchased with two different licensing models. The licensing model is determined by the type of application [32]:

- Internet applications with anonymous users has the following components:
  - Platform Access*
  - Usage (per transaction only)
- Known user applications has the following components:
  - Platform Access*
  - Usage (per transaction and per known user)

Standard product support is included with in both licensing models and additional premium product support is available.

---

* Platform Access is available in a Standard or Advanced version, depending on functionality.
6.12 **Documentation and support**

Microsoft provides two software development kits (SDK); one reference SDK and one interactive SDK which offers hands-on, task-based demonstrations of features available, complete with code examples and links back to the reference SDK.

Interactive SDK: http://dev.live.com/virtualearth/sdk/


Microsoft provides several resources where information about Map Control development can be found.

- The Virtual Earth site on Windows Live dev has all the latest news and resources for developing on the Virtual Earth platform, and is available at http://dev.live.com/virtualearth

- The Virtual Earth developer centre available on MSDN holds articles about Virtual Earth development, guidance on using the Virtual Earth API and a gallery with example mashups. The development center site is available at http://msdn2.microsoft.com/en-us/virtualearth/default.aspx

- Virtual Earth Map Control discussion forum on MSDN. At the moment, 05/07/2007, the forum holds 1,440 items. The forum is available at http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=537&SiteID=1

In addition to the Microsoft provided resources there is also an independent site called viavirtualearth, www.viavirtualearth.com, which holds a wiki, a forum with 572 items, 07/05/2007, a Frequently Asked Question (FAQ), blogs and articles about Virtual Earth development.
7 API tests

7.1 Geocoder

All four tested APIs supply geocoding functionality for translation of postal addresses into geographical coordinates. Addresses can be sent either as unparsed texts containing all address information or as multiple address parts such as: street, zip code, city, etc. Google Maps, Virtual Earth and Multimap support unparsed addresses while ViaMichelin only supports parsed addresses; however, in difference to the other three APIs ViaMichelin provides, in their enterprise edition, functionality for performing reverse geocoding – translating a geographical location into a postal address. Below is a table showing the geocoding functionality provided by the four APIs.

<table>
<thead>
<tr>
<th></th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsed geocoding</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Unparsed geocoding</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Reverse geocoding</td>
<td></td>
<td></td>
<td>Yes, in enterprise edition</td>
<td></td>
</tr>
</tbody>
</table>

Table 3, geocoding functionality provided by the four evaluated APIs.

7.1.1 Spherical law of cosines

The distance calculation between two points on a sphere’s surface, using geographic coordinates, can be done in several ways: Haversine formula, the Spherical laws of cosines and the Vincenty formula. When geocoding tests were performed to measure the API geocoders’ accuracy the Spherical law of cosines [33] was used to calculate the distances between points. The formula is defined as:

\[
d = \arccos(\sin(lat_1) \cdot \sin(lat_2) + \cos(lat_1) \cdot \cos(lat_2) \cdot \cos(long_2 - long_1)) \cdot R
\]

where \((lat_1, long_1)\) is one point’s coordinates, \((lat_2, long_2)\) the other point’s coordinates, \(d\) is the distance and \(R\) the sphere’s radius which in this case was the Earth’s average radius, 6374 km.

A latitude, or lat, is the angle between any point on the earth's surface and the plane of the Equator. Each pole is 90 degrees: +90 degrees at the North Pole and -90 degrees at the South Pole and in-between the Equator is placed at the 0-degree parallel of latitude.

A longitude, or long, is the angle east or west between any point on the earth's surface and the plane of an arbitrary north-south line between the two geographical poles. The line passing through Greenwich, UK, is the international zero-longitude reference line, the Prime Meridian.
The Spherical law of cosines is less accurate than the Haversine formula, but according to the company Movable Type Ltd, creator of a JavaScript implementation of the Haversine and Spherical law of cosines formulas, the numeric precision of JavaScript is so good, 15 significant figures using IEEE 754 floating-point numbers [34], that the simple Spherical law of cosines formula gives well-conditioned results down to distances as small as around 1 meter. In view of this it is probably worth, in most situations, to use the simpler Spherical law of cosines in preference to the more complicated and slower Haversine formula [35].

Since JavaScript implementations exist for two of the formulas, Haversine and the Spherical law of cosines the choice was between these two, and since the numeric precision of the Spherical law of cosines JavaScript implementation is sufficient, this formula was chosen.

7.1.2 First geocoding test
To measure the APIs’ geocoding accuracy a number of addresses spread around the world were geocoded by each API’s geocoder and the result locations were compared with reference locations to calculate the distance error; the distance between the location returned by the geocoder and the reference location.

The first test was performed the 10th of August 2007 and 212 hotel addresses spread around the world, Canada, Brazil, Denmark, Finland, France, Germany, Switzerland, Singapore, Spain and United States, were used as test data. Below are graphs showing the test results for the four APIs. The distance errors displayed in the diagrams are sorted after length and addresses with a distance of 15,000,000 meters or more were not found by the geocoder.

Multimap’s geocoder supports both parsed and unparsed addresses. Multimap Quick Search (QS) shows the results when unparsed addresses were used and Multimap when parsed addresses were used. In the tables UP shows that input addresses were unparsed and P that input addresses were sent as parts.

Geocoder test result – Entire world
212 addresses in: Canada, Brazil, Denmark, Finland, France, Germany, Switzerland, Singapore, Spain and United States. See figure 41 and table 4.
Figure 41, geocoding distance error for the entire world.

<table>
<thead>
<tr>
<th>[m]</th>
<th>Google Maps (UP)</th>
<th>Multimap QS (UP)</th>
<th>Multimap (P)</th>
<th>ViaMichelin (P)</th>
<th>Virtual Earth (UP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st quartile</td>
<td>16</td>
<td>760</td>
<td>30</td>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>Median</td>
<td>53</td>
<td>831000</td>
<td>72</td>
<td>68</td>
<td>93000</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>178</td>
<td>15000000</td>
<td>740</td>
<td>380</td>
<td>1800000</td>
</tr>
</tbody>
</table>

Table 4, geocoding distance error test results for the entire world.

**Geocoder test result – Europe**

96 addresses in: Denmark, Finland, France, Germany, Switzerland and Spain. See figure 42 and table 5.

Figure 42, geocoding distance error for Europe.

<table>
<thead>
<tr>
<th>[m]</th>
<th>Google Maps (UP)</th>
<th>Multimap QS (UP)</th>
<th>Multimap (P)</th>
<th>ViaMichelin (P)</th>
<th>Virtual Earth (UP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st quartile</td>
<td>16</td>
<td>700</td>
<td>18</td>
<td>10</td>
<td>3400</td>
</tr>
<tr>
<td>Median</td>
<td>53</td>
<td>5000</td>
<td>41</td>
<td>60</td>
<td>320000</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>100</td>
<td>799000</td>
<td>90</td>
<td>130</td>
<td>4100000</td>
</tr>
</tbody>
</table>

Table 5, geocoding distance error test results for Europe.
Geocoder test result – North America
69 addresses in: Canada and United States. See figure 43 and table 6.

![Distance error - North America](image)

Table 6, geocoding distance error test results for North America.

<table>
<thead>
<tr>
<th>[m]</th>
<th>Google Maps (UP)</th>
<th>Multimap QS (UP)</th>
<th>Multimap (P)</th>
<th>ViaMichelin (P)</th>
<th>Virtual Earth (UP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st quartile</td>
<td>8.6</td>
<td>180</td>
<td>35</td>
<td>15</td>
<td>6.0</td>
</tr>
<tr>
<td>Median</td>
<td>32</td>
<td>636000</td>
<td>130</td>
<td>78</td>
<td>510</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>130</td>
<td>15000000</td>
<td>580</td>
<td>520</td>
<td>1200000</td>
</tr>
</tbody>
</table>

Geocoder test result – South America
28 addresses in: Brazil. See figure 44 and table 7.

![Distance error - South America](image)

Table 7, geocoding distance error test results for South America.

<table>
<thead>
<tr>
<th>[m]</th>
<th>Google Maps (UP)</th>
<th>Multimap QS (UP)</th>
<th>Multimap (P)</th>
<th>ViaMichelin (P)</th>
<th>Virtual Earth (UP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st quartile</td>
<td>37</td>
<td>15000000</td>
<td>5500</td>
<td>7.1</td>
<td>2500</td>
</tr>
<tr>
<td>Median</td>
<td>85</td>
<td>15000000</td>
<td>7200</td>
<td>10</td>
<td>6500000</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>180</td>
<td>15000000</td>
<td>7600</td>
<td>9100</td>
<td>16000000</td>
</tr>
</tbody>
</table>
Table 7, geocoding distance error test results for South America.

Geocoder test result – Asia (Singapore)
23 addresses in: Singapore. See figure 45 and table 8.

![Distance error - Asia](image)

**Figure 45, geocoding distance error for Asia.**

<table>
<thead>
<tr>
<th>[m]</th>
<th>Google Maps (UP)</th>
<th>Multimap QS (UP)</th>
<th>Multimap (P)</th>
<th>ViaMichelin (P)</th>
<th>Virtual Earth (UP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st quartile</td>
<td>69</td>
<td>15000000</td>
<td>53</td>
<td>61</td>
<td>0.37</td>
</tr>
<tr>
<td>Median</td>
<td>160</td>
<td>15000000</td>
<td>120</td>
<td>190</td>
<td>0.49</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>230</td>
<td>15000000</td>
<td>620</td>
<td>450</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Table 8, geocoding distance error test results for Asia.

### 7.1.3 Second geocoding test

When the first geocoding test was performed the unparsed addresses were sent with the address parts separated with spaces as the following address: 29 RUE DE BEAUNE 75007 PARIS FR. After the test was performed the results were discussed with the Multimap support team and they stated that the parser functionality used in their API is optimised for addresses with parts separated with commas. Therefore, after the discussion the same test was performed one more time, this time using commas, to see if this was the case for any of the other API providers as well. The test was conducted the 24th of September 2007 and the same list of addresses was used in the second test, but this time with the address parts separated by commas like the following address: 29 RUE DE BEAUNE, 75007, PARIS, FR. Results displayed in figure 46 and table 9.
Figure 46, geocoding distance error for the entire world with unparsed addresses sent with comma separation.

<table>
<thead>
<tr>
<th>[m]</th>
<th>Google Maps (UP)</th>
<th>Multimap QS (UP)</th>
<th>Multimap (P)</th>
<th>ViaMichelin (P)</th>
<th>Virtual Earth (UP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st quartile</td>
<td>16 (16)</td>
<td>9 (760)</td>
<td>7 (30)</td>
<td>10 (11)</td>
<td>1 (28)</td>
</tr>
<tr>
<td>Median</td>
<td>51 (53)</td>
<td>48 (831000)</td>
<td>40 (72)</td>
<td>51 (68)</td>
<td>6 (93000)</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>136 (178)</td>
<td>620 (15000000)</td>
<td>230 (740)</td>
<td>288 (380)</td>
<td>585 (1800000)</td>
</tr>
</tbody>
</table>

Table 9, geocoding distance error results from the second geocoding test. When the test was performed addresses from the entire world were used and the unparsed addresses were sent with the parts separated by commas. Values from the first test are given inside the parenthesis.

7.1.4 Geocoding test results

The geocoding distance error tests show that the APIs’ geocoding accuracy is highly dependent of how the input data is sent, with Google Maps as an exception. When the first test, with space separation of address parts, was performed, both Multimap and Virtual Earth performed inadequately in terms of geocoding accuracy. Both APIs’ distance error medians are far-off a sufficient result, and even Multimap’s 1st quartile is unsatisfactory. Google maps’ performance was, using unparsed addresses, on the same level as Multimap’s and ViaMichelin’s, with parsed addresses as input.

The second test, with comma separation of address parts, confirms the geocoders’ input data sensibility. With comma separated address parts Multimap’s and especially Virtual Earth’s performance improved significantly. Google maps performed equivalent, both using space and comma separation, demonstrating excellent stability.

The two tests also show that both Multimap and ViaMichelin, using parsed addresses, provided an enhanced geocoding when the second test was performed, even though the addresses were sent, exactly, the same way. This shows that the web-based map service providers improved the geocoding functionality or the geographical coverage between the two tests.
7.2 File load test

The amount of data needed to be downloaded differs between the APIs; there are differences in JavaScripts’ file sizes, tile image sizes and the number of servers used to provide users with map and satellite tile images.

To test the amount of data download when a web-based map is viewed, the same area was displayed with the road map mode, at the same zoom level and with the same map size. The downloaded files were recorded to see sizes and hosting servers and below table 10 shows the results, the amount of data downloaded. A complete list of downloaded files, hosting servers and test information can be found in Appendix A – load tests.

All file sizes are in kilobytes (KB) and Google Maps and Virtual Earth provide their API JavaScript files with compression.

<table>
<thead>
<tr>
<th>[KB]</th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript size</td>
<td>73 (190(^1))</td>
<td>322</td>
<td>192</td>
<td>146 (582(^1))</td>
</tr>
<tr>
<td>Tile image size</td>
<td>435</td>
<td>216</td>
<td>119</td>
<td>434</td>
</tr>
<tr>
<td>Total size</td>
<td>516</td>
<td>553</td>
<td>337</td>
<td>606</td>
</tr>
<tr>
<td>Tile servers</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 10, map API load test results.

7.2.1 File load test results

The load test shows that there are variations in JavaScript file sizes, tile image sizes and the number of servers providing the static content. Google Maps and Virtual Earth provide the smallest JavaScript files; however, this is due to compression provided by the web servers. The actual file sizes of the files loaded by the browser are 190 an 582 kilobytes, respectively.

ViaMichelin and Multimap total tile image size is less then half of Google Maps’ and Virtual Earth’s; however, ViaMichelin and Multimap provide 15 and 16 tile images while Google Maps and Virtual Earth provide 24 and 36 tile images respectively.

Google Maps, Multimap and Virtual Earth all provide 4 servers hosting the tile images, while ViaMichelin supplies one tile image server.

7.3 Browser compatibility

When a web-based map is used in a commercial product cross browser compatibility is an important aspect. The table below shows the browsers currently supported by the APIs according to the providers [36][37][38], however, in addition to the officially supported browsers the APIs may work with other browsers as well, and therefore the browser compatibility was tested.

\(^1\) Uncompressed
To test if the APIs were working with supported browser as well as not officially supported browsers they were tested. During the test the following set of browsers was used:

- Internet Explorer 7
- Firefox 2.0.0.8
- Netscape 9.0.0.1
- Opera 9.24
- Safari 3.0.2

The test was performed by loading a web site containing a map, to see if the map was displayed or not. Not all API functionality was tested and therefore it is not sure that the browsers reported as working are supporting 100% of the API functionality. Below is a table showing for which browsers the maps were loaded and displayed correctly.

<table>
<thead>
<tr>
<th>Browser</th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Explorer</td>
<td>6.0+</td>
<td>5.5+</td>
<td>6.0+</td>
<td>5.5+</td>
</tr>
<tr>
<td>Firefox</td>
<td>0.8+</td>
<td>1.0+</td>
<td>1.5+</td>
<td>0.9+</td>
</tr>
<tr>
<td>Mozilla</td>
<td>1.4+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netscape</td>
<td>7.1+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opera</td>
<td>8.02+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safari</td>
<td>1.2.4+</td>
<td>1.3+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 11, API browser compatibility according to map API service providers.

Table 12, browser compatibility test results. Working browsers are marked with X.

* Route Highlighting not supported on Firefox 1.0.x
7.3.1 Browser compatibility results

The browser compatibility test shows that Google Maps offers the best browser compatibility by officially supporting all tested browsers. Multimap, on the other hand, works with all tested browsers, though, only three of them are officially supported. Virtual Earth provides the least satisfying compatibility, by officially supporting two browsers, Internet Explorer and Firefox and function with three.

7.4 API comparison

This chapter presents the APIs’ differences in terms of functionality, customisability and usability.

7.4.1 Viewing modes

<table>
<thead>
<tr>
<th>Viewing mode</th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map/road</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Satellite/Aerial</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hybrid</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bird’s eye</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3D</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 13, API viewing mode table. Supported modes are marked with X.

7.4.2 Controls

7.4.2.1 Google Maps

<table>
<thead>
<tr>
<th>Control</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map control in three sizes</td>
<td>Pan, return to start location and zoom</td>
</tr>
<tr>
<td>Map type control</td>
<td>Change map viewing mode</td>
</tr>
<tr>
<td>Scale control</td>
<td>Scale indicator</td>
</tr>
<tr>
<td>Overview map control</td>
<td>Small overview of the map</td>
</tr>
<tr>
<td>Local search control</td>
<td>Location search control (Google Ajax search API)</td>
</tr>
<tr>
<td>Custom controls</td>
<td>Yes, possibility to add HTML elements to map</td>
</tr>
</tbody>
</table>

Table 14, available controls with Google Maps.
7.4.2.2 Multimap

<table>
<thead>
<tr>
<th>Control</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pan and zoom widget in three sizes</td>
<td>Pan, return to start location and zoom</td>
</tr>
<tr>
<td>Map type widget</td>
<td>Change map viewing mode</td>
</tr>
<tr>
<td>Tools widget</td>
<td>Mouse drag and click behavior control with five modes: drag map, drag to zoom, drag to navigate, click to zoom in and click to zoom out.</td>
</tr>
<tr>
<td>Scale</td>
<td>Fixed Scale indicator, not removable</td>
</tr>
<tr>
<td>Location widget</td>
<td>Displays the currently displayed location</td>
</tr>
<tr>
<td>Overview widget</td>
<td>Small overview of the map</td>
</tr>
<tr>
<td>Context menu</td>
<td>Mouse right click menu</td>
</tr>
</tbody>
</table>

Table 15, available controls with Multimap.

7.4.2.3 ViaMichelin

<table>
<thead>
<tr>
<th>Control</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map tools in three sizes</td>
<td>Pan, zoom, zoom box and points of interest control</td>
</tr>
<tr>
<td>Scale</td>
<td>Fixed Scale indicator, not removable</td>
</tr>
</tbody>
</table>

Table 16, available controls with ViaMichelin.

7.4.2.4 Virtual Earth

<table>
<thead>
<tr>
<th>Control</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dashboard in three sizes</td>
<td>Zoom and change viewing mode</td>
</tr>
<tr>
<td>Scale</td>
<td>Fixed Scale indicator, not removable</td>
</tr>
<tr>
<td>Find control</td>
<td>Location search control. Where and what search</td>
</tr>
<tr>
<td>Mini map</td>
<td>Small overview of the map</td>
</tr>
<tr>
<td>3D map control</td>
<td>Pan, rotate and tilt the map in 3D mode</td>
</tr>
<tr>
<td>Custom controls</td>
<td>Yes, possibility to add HTML elements to map</td>
</tr>
</tbody>
</table>

Table 17, available controls with Microsoft Virtual Earth.
### 7.4.3 Overlays

<table>
<thead>
<tr>
<th>Overlay</th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pushpin/marker</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Info window/box</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Detached info window/box</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tabbed info window/box</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Polyline</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Polygon</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Circle</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Content grouping</td>
<td>Marker manager</td>
<td>Decluttering markers</td>
<td>Complex layer</td>
<td>Content layer</td>
</tr>
</tbody>
</table>

Table 18, API provided overlays.

### 7.4.4 Driving direction

All four APIs support driving directions, with different options and display functionality.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple point</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No, only two point directions</td>
</tr>
<tr>
<td>Textual descriptions</td>
<td>Collection of plain text directions or preformatted HTML</td>
<td>Collection of plain text directions</td>
<td>Preformatted HTML with road signs and symbols</td>
<td>Collection of plain text directions</td>
</tr>
<tr>
<td>Options</td>
<td>Quickest or shortest route. Durations based on walking or driving.</td>
<td>Routes recommended by Michelin, shortest, quickest, economical, discovery, on foot or by bike.</td>
<td></td>
<td>Shortest or quickest route</td>
</tr>
<tr>
<td>Route information</td>
<td>Distance and duration</td>
<td>Distance and duration</td>
<td>Toll costs for automobile, motorcycle or caravan. Petrol costs</td>
<td>Distance and duration</td>
</tr>
</tbody>
</table>
Traffic information

<table>
<thead>
<tr>
<th>Traffic information</th>
<th>Current traffic situation in some supported cities</th>
<th>based on the type of car</th>
</tr>
</thead>
<tbody>
<tr>
<td>Localization</td>
<td>9 languages</td>
<td>10 languages</td>
</tr>
<tr>
<td></td>
<td>6 languages</td>
<td>1 language</td>
</tr>
</tbody>
</table>

Table 19, API driving direction functionality.

7.4.5 Customisability

<table>
<thead>
<tr>
<th>Element</th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pushpin/marker</td>
<td>Custom icons, changeable at any time</td>
<td>Custom icons, changeable at any time</td>
<td>Custom icons</td>
<td>Fully customisable pushpins with HTML or icon image. Changeable at any time</td>
</tr>
<tr>
<td>Info window/box</td>
<td>Customisable with PdMarker extension</td>
<td>Customisable with CSS</td>
<td>Customisable with CSS</td>
<td>Customisable with CSS</td>
</tr>
<tr>
<td>Controls</td>
<td>Customisable with extensions</td>
<td>Customisable with CSS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 20, customisability provided by APIs.

7.4.6 GeoXML overlays

<table>
<thead>
<tr>
<th>Data format</th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoRSS</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>KML</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VE collection</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 21, GeoXML overlays and layers provided by APIs.
7.4.7 Usability
To create an efficient map solution in terms of time and user errors the following guidelines should be followed:

- Possibility to pan the map by dragging it with the mouse.
- Use centre zoom as default zoom mode.
- Provide grouped pan buttons arranged according to direction.
- Use a clear terminology.
- Provide an overview map showing where the map is being viewed in relation to a larger geographic area.
- Buttons should have text or icons with tooltips showing the name and describing the purpose or action.

All four APIs provide draggable maps and original centre zoom as default, however, there are some differences in other areas. Google offers the best usability with localised tooltips, overview map and grouped pan buttons, in contrast to ViaMichelin API that neither provides tooltips nor an overview map. Below is a table displaying the guidelines provided by the APIs.

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Google Maps</th>
<th>Multimap</th>
<th>ViaMichelin</th>
<th>Virtual Earth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draggable map</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Grouped pan buttons</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No pan buttons</td>
</tr>
<tr>
<td>Pan buttons grouped according to direction</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No pan buttons</td>
</tr>
<tr>
<td>Tooltips</td>
<td>Localised tooltips for all buttons</td>
<td>English tooltips for buttons without text</td>
<td>No</td>
<td>English tooltips for buttons without text</td>
</tr>
<tr>
<td>Default zoom</td>
<td>Original centre</td>
<td>Original centre</td>
<td>Original centre</td>
<td>Original centre</td>
</tr>
<tr>
<td>Overview map</td>
<td>Yes, fully intractable</td>
<td>Yes, fully intractable</td>
<td>No</td>
<td>Yes, fully intractable</td>
</tr>
</tbody>
</table>

Table 22, API usability evaluation results.
7.4.8 API comparison results

A comparison of the four map APIs’ features, customisability and usability shows that there are several similarities, as well as differences. The four APIs differs and have pros and cons in all kind of aspects, viewing modes, controls, driving directions and usability.

The map viewing modes differs from Virtual Earth providing five, including the unique Bird’s eye view, and ViaMichelin only one, the normal road map. In terms of map controls Multimap provides the largest set, in contrast to ViaMichelin that only provides one control; however, this control lets the user handle all map functionality. The driving directions functionality differs significantly. ViaMichelin provides the best driving directions with toll and petrol costs, road signs and seven types of directions as well as localisation; Virtual Earth’s driving directions can only be two-point and textual descriptions are only available in English.

In terms of usability Google Maps shows the best results The API provides a draggable map with grouped pan buttons arranged according to direction, localised tooltips, original centre zoom and an overview map. ViaMichelin does neither provide tooltips nor an overview map.
8 Map abstraction layer

When a web-based map is integrated into an application or web site a lot of time and effort is invested in writing the code. The dilemma is that all web-based map APIs work differently and have their own interfaces, causing your implementation to be dependent on just one vendor. This is not something wanted, and if for example the current used map has to be switched as a result of changes in terms and conditions, new functionality, etc. the code has to be rewritten.

There are at least two ways to eliminate this dependability problem. As Marc Wick, founder of GeoNames, says, the best solution to the problem would be if the map vendors (Google, Microsoft, Multimap, etc.) could agree on a web map API standard, a common interface used by all APIs [39]. However, since they provide their own unique features and dependability might be a part of their business strategy, a standard like this does not exist.

The other way to solve this problem would be to create an abstraction layer placed in-between the web application and the map APIs. The abstraction layer would provide a common interface to all included map APIs at the same time and therefore the map used could easily be switched with a small work effort.

8.1 Existing abstraction layers

At the moment there exits two abstraction layer projects: MyMap [40] and Mapstraction [41].

MyMap is a small test project, providing support for Google Maps, Microsoft Virtual Earth and Yahoo maps and only support for some basic functionality is implemented. MyMap is for the moment offering the following functionality:

- Create maps
- Set map modes. All map providers support at least three display modes: Road map, Satellite/Aerial and Hybrid
- Set a position on the map
- Zoom in/out
- Add/remove markers
- Geocoding
- Display infowindows
- Delete and unload maps

Mapstraction is a bigger project with support for six major mapping vendors: Yahoo Maps, Google Maps, Microsoft Virtual Earth, Map24, Multimap and MapQuest. In addition to these vendors, projects like OpenStreetMap, FreeEarth and OpenLayers are supported. For the moment the Mapstraction project is offering functionality for:
Three viewing modes: Road map, Satellite/Aerial and Hybrid
- Markers and infowindows
- Marker Filters
- Polylines, polygons
- Image Overlay
- GeoRSS
- Geocoding
- Routes

The idea behind Mapstraction is that if the API is only using the lowest common denominator of the features offered by all map APIs. This will create the possibility to switch map vendor without changing to much of the code, but it will also cause the loss of unique features, as the Bird’s eye view. Another problem with both MyMap and Mapstraction is that the implementations are made for old versions of the APIs and not active. For example both abstractions layer are implemented for version 3 of the Virtual Earth API. Based on these two problems the decision to create a new abstraction layer was taken.

### 8.2 JavaScript API abstraction layer

All evaluated APIs have a JavaScript interface, and therefore, the abstraction layer was created as a JavaScript application located in-between web applications and the APIs, providing a similar interface as the APIs (Figure 47). Most of the methods implemented in the layer will be forwarding method calls from the application to the map APIs, creating a common interface; however, since the APIs are constructed differently, with different methods and different object types, some common objects have to be created. A latitude-longitude object has to be created for coordinates as well as objects for locations, markers, polylines and polygons. In addition to this some common objects have to be created, such as routes, parts and steps, to provide driving directions for all APIs.

![Figure 47, JavaScript abstraction layer positioned in-between web applications and map APIs.](image)
Another important aspect is to keep all APIs’ unique features, as Virtual Earth’s bird’s eye view and ViaMichelin’s driving directions. This can be done by letting all API objects have methods for the unique features. If the feature is supported the API class will contain methods and functionality to handle the call; if the feature is not supported, instead of actually doing anything when the method is called a console warning message is sent back informing that the requested method is not supported. With this technique all unique features can be used without causing JavaScript errors when non-supported methods are called.

The abstraction layer was designed to have one abstract base class containing all methods callable from the application where the map is used. In this class all methods return a non-supported method warning message. All methods supported by an API are later overridden in the specific API classes which are inheriting from the abstract main class.

The abstraction layer application was created and tested successfully in the Amadeus product HotelsPlus (Figure 48) as a proof of concept, to show that an abstraction layer like this could be used by web applications to provide a common interface to several API.

Figure 48, Google Maps integrated into a site through the abstraction layer.
9 Conclusion

After testing and evaluation of the four web-based map APIs, the conclusion is that none of them can be appointed as the winner. This thesis shows that APIs differ in terms of functionality, compatibility, usability, geocoding and development support and the choice of API is consequently dependent of the type of application; for which reasons the map is used. All of the four APIs have benefits and drawbacks and as a result of this, the various parts of the requested functionality have to be rated in terms of importance, to be able to choose which API to use.

By studying and testing the APIs functionality, the conclusion can be made that they all have differences in viewing modes provided, driving directions functionality and points of interest support as well as other functionality.

In terms of compatibility, Google and Multimap work with most browsers on the market, while ViaMichelin and Virtual Earth provide less browser compatibility and support. The browser test also shows the dissimilarities between the browsers supported by the map service providers, and the browsers that are actually working together with the API.

The map usability is similar with three of the APIs, Google Maps, Multimap and Virtual Earth, among which, Google Maps provides the best usability with localised tooltips, an overview map and pan buttons; and ViaMichelin the poorest usability by neither providing tooltips nor an overview map. For ViaMichelin to reach the same level of usability as the other three evaluated APIs, these two aspects have to be improved.

During the API evaluations, the geocoding tests show that the APIs’ geocoders support parsed addresses, unparsed addresses or, in Multimap’s case, both at the same time; however, the tests also show that some of the APIs do not support completely unparsed addresses. When unparsed addresses are sent to a geocoder, such as Virtual Earth’s and Multimap’s, it is important to separate the address parts, in the unparsed text, with commas: “street, city, country”. The loss in terms of accuracy is significant if spaces are used to separate the parts instead of commas. On the other hand; in addition to this the tests show that Google Maps provides a stable parser with the same geocoding accuracy regardless of the address part separation.

Another important aspect to keep in mind while choosing API is development support, since there is a significant difference between the APIs. Google is the most popular API and the company provide an active forum, and a bug-tracker and several third-party extensions are available. Multimap’s and ViaMichelin’s APIs are new and the developer support and community is unfortunate.

Since all APIs have benefits and drawbacks, and are constantly changing in terms of functionality and coverage, it is important to create applications independent of the map service provider. This can be done in two ways: either the map service providers can have a common interface or an abstraction layer in between the application and the APIs can be used. Given that a common interface does not exist, the conclusion is that the latter solution, to create an abstraction layer, is the only possible. During the internship at Amadeus where this evaluation was made, such a layer was created with success and integrated into an existing product as a proof-of-concept.
## 10 Explanations and abbreviation

This chapter explains terms and abbreviations used in the thesis.

<table>
<thead>
<tr>
<th>Term or Abbreviation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ajax</td>
<td>Asynchronous JavaScripts and XML is a web development technique used for creating interactive web applications.</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets is a stylesheet language used to describe the presentation of a document written in a markup language, such as HTML.</td>
</tr>
<tr>
<td>DOM</td>
<td>The Document Object Model is a platform- and language-independent standard object model for representing HTML or XML and related formats.</td>
</tr>
<tr>
<td>DTML</td>
<td>Dynamic HTML is a combination of technologies used to create dynamic and interactive Web sites normally a combination of HTML, Style Sheets and JavaScript.</td>
</tr>
<tr>
<td>GeoRSS</td>
<td>GeoRSS is a standard for encoding location as part of an RSS feed. In GeoRSS, location content consists of geographical points, lines, and polygons of interest and related feature descriptions.</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System is a system for capturing, storing, analyzing and managing data and associated attributes which are spatially referenced to the earth.</td>
</tr>
<tr>
<td>GUI</td>
<td>A Graphical User Interface is a user interface which allows people to interact with a computer and computer-controlled devices which employ graphical icons, visual indicators or special graphical elements; along with text representing the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements.</td>
</tr>
<tr>
<td>Shortened Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language is the predominant markup language for web pages. It provides a means to describe the structure of text-based information in a document and to supplement the text with forms, images and other objects.</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation is a lightweight data-interchange format, <a href="http://www.json.org">http://www.json.org</a></td>
</tr>
<tr>
<td>KML</td>
<td>Keyhole Markup Language is an XML-based language for describing three-dimensional geospatial data and its display in application programs, such as web-based maps.</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer is a style of software architecture for distributed hypermedia systems such as the World Wide Web.</td>
</tr>
<tr>
<td>RSS</td>
<td>RDF Site Summary is a family of Web feed formats used to publish frequently updated content. RSS formats are specified using XML.</td>
</tr>
<tr>
<td>xAL</td>
<td>eXtensible Address Language is an international standard for address formatting.</td>
</tr>
<tr>
<td>XML</td>
<td>The Extensible Markup Language is a general-purpose markup language.</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language Transformations is an XML-based language used for the transformation of XML documents into other XML or &quot;human-readable&quot; documents.</td>
</tr>
</tbody>
</table>
11 References

[17] How can I get the Maps API to display in a language besides English?,
http://code.google.com/support/bin/answer.py?answer=60737&topic=10
946, visited 2007-07-06
[18] What countries is geocoding available in?,
http://code.google.com/support/bin/answer.py?answer=62668&topic=10
946, visited 2007-07-06
[19] Bret Taylor - Product Manager, Google Maps API Version 2,
http://googlemapsapi.blogspot.com/2006/04/google-maps-api-version-
2.html, visited 2007-06-26
[20] Sign Up for the Google Maps API,
[21] Multimap API V1.2 Documentation,
http://www.multimap.com/share/documentation/api/1.2/, visited 2007-11-
19
[22] Class MMRoute,
http://www.multimap.com/share/documentation/api/1.2/classes/MMRout-
e.html#lang, visited 2007-07-23
[23] Multimap Local Information v. 1.0 Product Datasheet, Multimap
visited 2007-07-24
[25] Reference Documentation,
http://dev.viamichelin.fr/wswebsite/gbr/jsp/vmdn/VMDN-Api-Maps-Drive-
Reference.jsp, visited 2007-11-19
[26] Reference Documentation,
http://dev.viamichelin.fr/wswebsite/gbr/jsp/vmdn/VMDN-Api-Maps-Drive-
Reference.jsp, visited 2007-07-16
[27] General terms and conditions for providing “ViaMichelin Maps & Drive
API”, http://dev.viamichelin.fr/wswebsite/gbr/jsp/vmdn/VMDN-Api-Maps-
Drive-Conditions.jsp, visited 2007-07-17
19
[29] John Bentley, Live search maps,
http://gadgetshow.five.tv/jsp/5gsmain.jsp?lnk=401&featureid=338&pagei-
d=792&show=s6e8&section=Features, visited 2007-06-26
[30] Andy McGovern - Virtual Earth Program Manager, Virtual Earth API V5
is released,
entry, visited 2007-07-09
[31] Virtual Earth Service API terms of use,
http://www.microsoft.com/virtualearth/control/terms.mspx, visited 2007-
07-02
[32] How to buy,
http://www.microsoft.com/virtualearth/products/webservice/howtobuy.ms-
px, visited 2007-07-06
[33] W. Gellert, S. Gottwald, M. Hellwich, H. Kästner, and H. Küstner, The
VNR Concise Encyclopedia of Mathematics, 2nd ed., ch. 12, Van
[35] Moveable Type, Calculate distance, bearing and more between two Latitude/Longitude points, http://www.movable-type.co.uk/scripts/latlong.html, visited 2007-08-10
Appendix A – load tests

Google Maps

Test conducted: 06/09/2007
Latitude: 50
Longitude: 10
Zoom level: 5
Controls used: large map control, map type control

Map size:
Width: 900px
Height: 600px

<table>
<thead>
<tr>
<th>File name</th>
<th>Hosting server</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScripts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maps</td>
<td>maps.google.com</td>
<td>5</td>
</tr>
<tr>
<td>main.js</td>
<td>google.com</td>
<td>68 (190)</td>
</tr>
<tr>
<td>maps</td>
<td>maps.google.com</td>
<td>0.229</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>73.229</td>
</tr>
<tr>
<td>Images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transparent.png</td>
<td>google.com</td>
<td>0.111</td>
</tr>
<tr>
<td>poweredby.png</td>
<td>google.com</td>
<td>4</td>
</tr>
<tr>
<td>openhand.cur</td>
<td>google.com</td>
<td>0.318</td>
</tr>
<tr>
<td>lmc.png</td>
<td>google.com</td>
<td>3</td>
</tr>
<tr>
<td>lmc-bottom.png</td>
<td>google.com</td>
<td>0.4</td>
</tr>
<tr>
<td>slider.png</td>
<td>google.com</td>
<td>0.237</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>8.066</td>
</tr>
<tr>
<td>Tile images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mt</td>
<td>mt3.google.com</td>
<td>0.101</td>
</tr>
<tr>
<td>mt</td>
<td>mt0.google.com</td>
<td>0.101</td>
</tr>
<tr>
<td>mt</td>
<td>mt1.google.com</td>
<td>0.101</td>
</tr>
<tr>
<td>mt</td>
<td>mt2.google.com</td>
<td>0.265</td>
</tr>
<tr>
<td>mt</td>
<td>mt0.google.com</td>
<td>9</td>
</tr>
<tr>
<td>mt</td>
<td>mt1.google.com</td>
<td>23</td>
</tr>
<tr>
<td>mt</td>
<td>mt2.google.com</td>
<td>17</td>
</tr>
<tr>
<td>mt</td>
<td>mt3.google.com</td>
<td>21</td>
</tr>
<tr>
<td>mt</td>
<td>mt1.google.com</td>
<td>16</td>
</tr>
<tr>
<td>mt</td>
<td>mt2.google.com</td>
<td>33</td>
</tr>
<tr>
<td>mt</td>
<td>mt3.google.com</td>
<td>34</td>
</tr>
<tr>
<td>mt</td>
<td>mt0.google.com</td>
<td>14</td>
</tr>
<tr>
<td>mt</td>
<td>mt2.google.com</td>
<td>26</td>
</tr>
<tr>
<td>mt</td>
<td>mt3.google.com</td>
<td>35</td>
</tr>
<tr>
<td>mt</td>
<td>mt0.google.com</td>
<td>35</td>
</tr>
<tr>
<td>mt</td>
<td>mt1.google.com</td>
<td>12</td>
</tr>
<tr>
<td>mt</td>
<td>mt3.google.com</td>
<td>24</td>
</tr>
<tr>
<td>mt</td>
<td>mt0.google.com</td>
<td>20</td>
</tr>
<tr>
<td>mt</td>
<td>mt1.google.com</td>
<td>19</td>
</tr>
<tr>
<td>mt</td>
<td>mt2.google.com</td>
<td>18</td>
</tr>
<tr>
<td>mt</td>
<td>mt0.google.com</td>
<td>17</td>
</tr>
</tbody>
</table>

* Uncompressed
Multimap

Test conducted: 23/07/2007
Latitude: 50
Longitude: 10
Zoom level: 6
Controls used: pan zoom control, map type control
Map size:
Width: 900px
Height: 600px

File name  Hosting server  Size (KB)
JavaScripts

cleankey_api  clients.multimap.com  3
obfuscated-mmviewer-1.2.js  mc.multimap.com  316
copyright_info.js  clients.multimap.com  3
mapindex.cgi  clients.multimap.com  0.209

Total  322.209

Style sheets

api_styles.css  mc.multimap.com  6

Total

Images

logo.gif  mc.multimap.com  2
trial_watermark.png  mc.multimap.com  7
panzoom-bgr.png  mc.multimap.com  0.634
north.png  mc.multimap.com  0.53
west.png  mc.multimap.com  0.538
reset.png  mc.multimap.com  0.625
east.png  mc.multimap.com  0.522
south.png  mc.multimap.com  0.518
zoomin.png  mc.multimap.com  0.435
zoom-scale.png  mc.multimap.com  0.162
slider.png  mc.multimap.com  0.421
zoomout.png  mc.multimap.com  0.37
zoom-cap.png  mc.multimap.com  0.156
type_map_active.png  mc.multimap.com  0.311
type_aerial.png  mc.multimap.com  0.334
type_hybrid.png  mc.multimap.com  0.354

Total  14.91

Tile images

30201.png  mc1.multimap.com  25
30203.png  mc3.multimap.com  14
30210.png  mc0.multimap.com  16
30212.png  mc2.multimap.com  10
<table>
<thead>
<tr>
<th>File name</th>
<th>Hosting server</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>api.1.0.js</td>
<td>api.viamichelin.com</td>
<td>3</td>
</tr>
<tr>
<td>VMPrototypes.jsp</td>
<td>www2.viamichelin.com</td>
<td>15</td>
</tr>
<tr>
<td>OpeningDivObject.js</td>
<td>www2.viamichelin.com</td>
<td>3</td>
</tr>
<tr>
<td>VMCountryUtil.js</td>
<td>www2.viamichelin.com</td>
<td>10</td>
</tr>
<tr>
<td>VMGeoReferential.js</td>
<td>www2.viamichelin.com</td>
<td>0.138</td>
</tr>
<tr>
<td>VMGeoReferentialFactory.js</td>
<td>www2.viamichelin.com</td>
<td>3</td>
</tr>
<tr>
<td>Equi43Referential.js</td>
<td>www2.viamichelin.com</td>
<td>0.831</td>
</tr>
<tr>
<td>LambertReferential.js</td>
<td>www2.viamichelin.com</td>
<td>5</td>
</tr>
<tr>
<td>Mdm35Referential.js</td>
<td>www2.viamichelin.com</td>
<td>0.868</td>
</tr>
<tr>
<td>MercatorReferential.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>MillerReferential.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>TransverseMercatorReferential.js</td>
<td>www2.viamichelin.com</td>
<td>6</td>
</tr>
<tr>
<td>UTMReferential.js</td>
<td>www2.viamichelin.com</td>
<td>5</td>
</tr>
<tr>
<td>VMLonLat.js</td>
<td>www2.viamichelin.com</td>
<td>0.13</td>
</tr>
<tr>
<td>VMAAddress.js</td>
<td>www2.viamichelin.com</td>
<td>0.274</td>
</tr>
<tr>
<td>VMGeoSearch.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>VMGeocoder.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>VMReverseGeo.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>VMPoint.js</td>
<td>www2.viamichelin.com</td>
<td>0.042</td>
</tr>
<tr>
<td>VMIcon.js</td>
<td>www2.viamichelin.com</td>
<td>0.113</td>
</tr>
<tr>
<td>VMLayer.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>VMIconLayer.js</td>
<td>www2.viamichelin.com</td>
<td>0.512</td>
</tr>
</tbody>
</table>

**ViaMichelin**

Test conducted: 17/07/2007
Latitude: 50
Longitude: 10
Zoom level: 5
Controls used: Map tools with full size
Map size: Width: 900px Height: 600px

Total 216
TOTAL 553.119
<table>
<thead>
<tr>
<th>File Name</th>
<th>URL</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMExpandLayer.js</td>
<td>www2.viamichelin.com</td>
<td>0.394</td>
</tr>
<tr>
<td>VMPolyLine.js</td>
<td>www2.viamichelin.com</td>
<td>6</td>
</tr>
<tr>
<td>VMCircleShape.js</td>
<td>www2.viamichelin.com</td>
<td>3</td>
</tr>
<tr>
<td>VMPolygonShape.js</td>
<td>www2.viamichelin.com</td>
<td>4</td>
</tr>
<tr>
<td>VMComplexLayer.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>VMMAP.js</td>
<td>www2.viamichelin.com</td>
<td>25</td>
</tr>
<tr>
<td>VMIinerary.js</td>
<td>www2.viamichelin.com</td>
<td>12</td>
</tr>
<tr>
<td>VMRoadSheet.js</td>
<td>www2.viamichelin.com</td>
<td>17</td>
</tr>
<tr>
<td>VMPOI.js</td>
<td>www2.viamichelin.com</td>
<td>8</td>
</tr>
<tr>
<td>VMPOIList.js</td>
<td>www2.viamichelin.com</td>
<td>0.564</td>
</tr>
<tr>
<td>VMPOICriteria.js</td>
<td>www2.viamichelin.com</td>
<td>0.57</td>
</tr>
<tr>
<td>VMPOIDefinition.js</td>
<td>www2.viamichelin.com</td>
<td>4</td>
</tr>
<tr>
<td>VMPOISearch.js</td>
<td>www2.viamichelin.com</td>
<td>3</td>
</tr>
<tr>
<td>VMGeoRssSearch.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>VMWeatherInformation.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>VMWeather.js</td>
<td>www2.viamichelin.com</td>
<td>0.412</td>
</tr>
<tr>
<td>VMWeatherSearch.js</td>
<td>www2.viamichelin.com</td>
<td>4</td>
</tr>
<tr>
<td>VMExportAsXVM.js</td>
<td>www2.viamichelin.com</td>
<td>0.591</td>
</tr>
<tr>
<td>VMBubble.js</td>
<td>www2.viamichelin.com</td>
<td>3</td>
</tr>
<tr>
<td>scaleTemplate.js</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>MapObject.jsp</td>
<td>www2.viamichelin.com</td>
<td>17</td>
</tr>
<tr>
<td>mapToolsTemplate2.js</td>
<td>www2.viamichelin.com</td>
<td>9</td>
</tr>
<tr>
<td>poiDemandObject.jsp</td>
<td>www2.viamichelin.com</td>
<td>0.473</td>
</tr>
<tr>
<td>PoiDemandFinderObject.jsp</td>
<td>www2.viamichelin.com</td>
<td>0.787</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total</td>
</tr>
</tbody>
</table>

**Images**

<table>
<thead>
<tr>
<th>File Name</th>
<th>URL</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoomOutON.gif</td>
<td>www2.viamichelin.com</td>
<td>0.384</td>
</tr>
<tr>
<td>fullScaleBarShadow.png</td>
<td>www2.viamichelin.com</td>
<td>0.222</td>
</tr>
<tr>
<td>fullScaleBar.png</td>
<td>www2.viamichelin.com</td>
<td>0.348</td>
</tr>
<tr>
<td>zoomInON.gif</td>
<td>www2.viamichelin.com</td>
<td>0.398</td>
</tr>
<tr>
<td>zoomInShadow.png</td>
<td>www2.viamichelin.com</td>
<td>0.209</td>
</tr>
<tr>
<td>expand.gif</td>
<td>www2.viamichelin.com</td>
<td>0.176</td>
</tr>
<tr>
<td>closeShadow.png</td>
<td>www2.viamichelin.com</td>
<td>0.262</td>
</tr>
<tr>
<td>closeShadowDown.png</td>
<td>www2.viamichelin.com</td>
<td>0.265</td>
</tr>
<tr>
<td>marieRightShadow.png</td>
<td>www2.viamichelin.com</td>
<td>0.258</td>
</tr>
<tr>
<td>marieON.png</td>
<td>www2.viamichelin.com</td>
<td>2</td>
</tr>
<tr>
<td>marieShadow.png</td>
<td>www2.viamichelin.com</td>
<td>0.22</td>
</tr>
<tr>
<td>drag.jpg</td>
<td>www2.viamichelin.com</td>
<td>0.901</td>
</tr>
<tr>
<td>toolsRight.png</td>
<td>www2.viamichelin.com</td>
<td>0.29</td>
</tr>
<tr>
<td>toolsBottom.png</td>
<td>www2.viamichelin.com</td>
<td>0.21</td>
</tr>
<tr>
<td>pk.jpg</td>
<td>www2.viamichelin.com</td>
<td>0.755</td>
</tr>
<tr>
<td>rad.jpg</td>
<td>www2.viamichelin.com</td>
<td>0.81</td>
</tr>
<tr>
<td>sub.jpg</td>
<td>www2.viamichelin.com</td>
<td>0.801</td>
</tr>
<tr>
<td>toolsShadow.png</td>
<td>www2.viamichelin.com</td>
<td>0.24</td>
</tr>
<tr>
<td>closeH.jpg</td>
<td>www2.viamichelin.com</td>
<td>0.403</td>
</tr>
<tr>
<td>closeHShadowRight.png</td>
<td>www2.viamichelin.com</td>
<td>0.274</td>
</tr>
<tr>
<td>closeHShadowDown.png</td>
<td>www2.viamichelin.com</td>
<td>0.278</td>
</tr>
<tr>
<td>dragON.jpg</td>
<td>www2.viamichelin.com</td>
<td>0.625</td>
</tr>
<tr>
<td>MapFinderObjectFile.jsp</td>
<td>www2.viamichelin.com</td>
<td>13</td>
</tr>
<tr>
<td>bestMapFinder</td>
<td>www2.viamichelin.com</td>
<td>0.535</td>
</tr>
</tbody>
</table>
Table:

<table>
<thead>
<tr>
<th>File name</th>
<th>Hosting server</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScripts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mapcontrol.ashx</td>
<td>dev.virtualearth.net</td>
<td>140 (582)*</td>
</tr>
<tr>
<td>atlascompat.js</td>
<td>maps.live.com</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>146</strong></td>
</tr>
<tr>
<td>Stylesheets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ero.css</td>
<td>maps.live.com</td>
<td>0.742</td>
</tr>
<tr>
<td>api.css</td>
<td>maps.live.com</td>
<td>2</td>
</tr>
<tr>
<td>Traffic.css</td>
<td>maps.live.com</td>
<td>2</td>
</tr>
<tr>
<td>mapcontrol.css</td>
<td>maps.live.com</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>8.742</strong></td>
</tr>
<tr>
<td>Images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>beakLeft.gif</td>
<td>maps.live.com</td>
<td>0.146</td>
</tr>
<tr>
<td>HeaderBackGroundGraphic.gif</td>
<td>maps.live.com</td>
<td>0.206</td>
</tr>
<tr>
<td>vecssblue.gif</td>
<td>maps.live.com</td>
<td>4</td>
</tr>
<tr>
<td>vecss.gif</td>
<td>maps.live.com</td>
<td>8</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Uncompressed

Virtual Earth:

- Test conducted: 06/07/2007
- Latitude: 50
- Longitude: 10
- Zoom level: 5
- Controls used: normal dashboard
- Map size: Width: 900px, Height: 600px

**TOTAL** 336.597
<table>
<thead>
<tr>
<th>Image Name</th>
<th>URL</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>grab.cur</td>
<td>maps.live.com</td>
<td>0.18</td>
</tr>
<tr>
<td>logo_powered_by.png</td>
<td>maps.live.com</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>17.532</td>
</tr>
</tbody>
</table>

**Tile images**

<table>
<thead>
<tr>
<th>Image Name</th>
<th>URL</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0201.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>6</td>
</tr>
<tr>
<td>r0210.png</td>
<td>r0.ortho.tiles.virtualearth.net</td>
<td>7</td>
</tr>
<tr>
<td>r0211.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>5</td>
</tr>
<tr>
<td>r0300.png</td>
<td>r0.ortho.tiles.virtualearth.net</td>
<td>6</td>
</tr>
<tr>
<td>r0301.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>5</td>
</tr>
<tr>
<td>r0203.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>0.104</td>
</tr>
<tr>
<td>r0212.png</td>
<td>r2.ortho.tiles.virtualearth.net</td>
<td>9</td>
</tr>
<tr>
<td>r0213.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>10</td>
</tr>
<tr>
<td>r0302.png</td>
<td>r2.ortho.tiles.virtualearth.net</td>
<td>11</td>
</tr>
<tr>
<td>r0303.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>7</td>
</tr>
<tr>
<td>r0221.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>0.399</td>
</tr>
<tr>
<td>r0230.png</td>
<td>r0.ortho.tiles.virtualearth.net</td>
<td>5</td>
</tr>
<tr>
<td>r0231.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>12</td>
</tr>
<tr>
<td>r0320.png</td>
<td>r0.ortho.tiles.virtualearth.net</td>
<td>13</td>
</tr>
<tr>
<td>r0321.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>0.104</td>
</tr>
<tr>
<td>r0223.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>2</td>
</tr>
<tr>
<td>r0232.png</td>
<td>r2.ortho.tiles.virtualearth.net</td>
<td>0.104</td>
</tr>
<tr>
<td>r0233.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>3</td>
</tr>
<tr>
<td>r0322.png</td>
<td>r2.ortho.tiles.virtualearth.net</td>
<td>13</td>
</tr>
<tr>
<td>r0323.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>8</td>
</tr>
<tr>
<td>r03113.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>6</td>
</tr>
<tr>
<td>r12002.png</td>
<td>r2.ortho.tiles.virtualearth.net</td>
<td>16</td>
</tr>
<tr>
<td>r12003.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>24</td>
</tr>
<tr>
<td>r12012.png</td>
<td>r2.ortho.tiles.virtualearth.net</td>
<td>21</td>
</tr>
<tr>
<td>r03131.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>11</td>
</tr>
<tr>
<td>r12020.png</td>
<td>r0.ortho.tiles.virtualearth.net</td>
<td>25</td>
</tr>
<tr>
<td>r12021.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>27</td>
</tr>
<tr>
<td>r12030.png</td>
<td>r0.ortho.tiles.virtualearth.net</td>
<td>21</td>
</tr>
<tr>
<td>r03133.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>15</td>
</tr>
<tr>
<td>r12022.png</td>
<td>r2.ortho.tiles.virtualearth.net</td>
<td>29</td>
</tr>
<tr>
<td>r12023.png</td>
<td>r3.ortho.tiles.virtualearth.net</td>
<td>26</td>
</tr>
<tr>
<td>r12032.png</td>
<td>r2.ortho.tiles.virtualearth.net</td>
<td>21</td>
</tr>
<tr>
<td>r03311.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>19</td>
</tr>
<tr>
<td>r12200.png</td>
<td>r0.ortho.tiles.virtualearth.net</td>
<td>10</td>
</tr>
<tr>
<td>r12201.png</td>
<td>r1.ortho.tiles.virtualearth.net</td>
<td>13</td>
</tr>
<tr>
<td>r12210.png</td>
<td>r0.ortho.tiles.virtualearth.net</td>
<td>17</td>
</tr>
</tbody>
</table>

**Total** 433.711

**TOTAL** 605.985