



Faculty of Economic Sciences, Communication and IT
Department of Computer Science

Marie Högkvist
Rickard Johansson

Web map visualisation of road inspections for the Swedish Civil Contingencies Agency

Degree Project of 15 credit points
Bachelor of Science in Engineering,
Computer Science Engineering

Date/Term: 10-06-10
Supervisor: Donald F. Ross
Examiner: Martin Blom
Serial Number: C2010:08

**Web map visualisation of
road inspections for the
Swedish Civil Contingencies Agency**

Marie Högkvist, Rickard Johansson

This report is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not our own work has been identified and no material is included for which a degree has previously been conferred.

Marie Högvist

Rickard Johansson

Approved, 2010-06-10

Advisor: Donald F. Ross

Examiner: Martin Blom

Abstract

One of the tasks of the Swedish Civil Contingencies Agency, MSB (*Myndigheten för Samhällsskydd och Beredskap*), is to partake in relief efforts in foreign countries. Before these relief efforts, roads are inspected for accessibility to ensure that communications will work. The road inspections are currently being performed by hand with the use of pen and paper, which brings difficulties to distribution of information.

During the course of this thesis project, Sweco Position Karlstad has developed a digital road inspection tool for MSB that receives information from a road inspector and stores this information in a database. This road inspection tool may in the future replace the pen and paper inspections and ease the work of the road inspectors.

In this dissertation, a prototype of a web map that displays data from a road inspection is presented. Using a web map to display the results of a road inspection offers several advantages, as it eases distribution of information and allows end users to make selections about what data they want to view.

Acknowledgements

The authors of this dissertation would like to thank our supervisor at Sweco Position, Christian Göransson, for his support and help during the project.

We would also like to thank Bengt Djuvfeldt at Sweco Position and Stefan Andersson at MSB for including us in the project.

Finally we would like to thank our advisor Donald F. Ross for his great feedback and his enthusiasm.

Contents

1	Introduction	1
1.1	Dissertation overview	1
1.2	Project requirements	2
1.3	Chapter overview	2
2	Background	4
2.1	Introduction	4
2.2	The existing system	5
2.2.1	RAPID	5
2.2.2	Database	6
2.3	System overview	7
2.4	Adjustments of RAPID	7
2.5	Data Processing	8
2.5.1	DPAS, Data Processing Application Suite	8
2.5.2	Media	8
2.5.3	Positional data	9
2.5.4	Transmission and Publication of Data	9
2.6	The prototype	10
2.6.1	Prototype Concept	10
2.6.2	Technical Requirements	10
2.6.3	Web Map	10
2.6.4	Exporting the Map	11
2.6.5	Graphical User Interface	11
2.7	Security	12
2.7.1	Confidentiality	12
2.7.2	Integrity	12

2.7.3	Accessibility	12
2.8	Summary	12
3	Experiment	14
3.1	Introduction	14
3.2	Overview	14
3.2.1	System overview	15
3.3	From inspection to web map useable data	16
3.3.1	KML	16
3.3.2	Creating KML files	17
3.3.3	Linking KML files to the map with the use of OpenLayers	19
3.4	Web map	20
3.4.1	Overview	20
3.4.2	Map service	21
3.4.3	Map	21
3.4.4	Menu overview	24
3.4.5	Menu selections	26
3.5	CSS, Cascading Style Sheets	28
3.6	JavaScript	28
3.6.1	OpenLayers	29
3.6.2	jQuery	30
3.6.3	Thickbox	30
3.7	Summary	31
4	Result	33
4.1	Introduction	33
4.2	Result	33
4.2.1	Evaluation of the web map prototype	33

4.2.2	Known problems	33
4.3	From prototype to working project	34
4.3.1	DPAS	36
4.3.2	Publication	36
4.3.3	World map	37
4.3.4	File handling	38
4.4	Future development possibilities	38
4.5	Summary	39
5	Evaluation	40
5.1	Time planning	40
5.2	Acquired knowledge	40
5.3	What could have been done differently?	41
5.4	Other areas of application	41
5.5	Summary	42
	References	43
A	ER Diagram of the RAPID Database	45
B	Base GUI	46
C	KML file, discrete values	47
D	KML file, continuous values	49
E	Code snippet for loading KML files into web map	52

List of Figures

1.1	Overview, RAPID	1
1.2	Overview, inspection system	2
2.1	RAPID user interface [1]	6
2.2	The inspection process	7
2.3	Overview of the DPAS	8
3.1	Overview, web map prototype	14
3.2	File hierarchy	16
3.3	Prototype web page GUI	20
3.4	GUI: Viewing media popup information	22
3.5	GUI: Viewing a media file	22
3.6	GUI: An example of Google Maps street and satellite	23
3.7	Menu: The four tabs; camp, city, road and file.	24
3.8	Menu: Changing tabs	25
3.9	Menu: Subgroups	25
3.10	Menu: Discrete and continuous data	26
3.11	Menu: Icon buttons. (Selected, unselected and disabled.)	27
3.12	Menu: Group selectors and continuous selectors	28
4.1	Thickbox bug	35
A.1	ER Diagram of the RAPID Database [1]	45
B.1	The base GUI used for development	46

1 Introduction

1.1 Dissertation overview

This dissertation project was suggested by Sweco Position Karlstad [2]. The assignment was to develop a prototype of a web map for the Swedish Civil Contingencies Agency, MSB (*Myndigheten för Samhällsskydd och Beredskap*) [3], with the purpose of displaying data collected during their road inspections.

The road inspections are performed by MSB before a relief effort in a foreign country. The purpose of the road inspection is to assess road accessibility. Currently these road inspections are made with the help of pen and paper, which brings difficulty to distribution of information and consistency (as all inspectors have their own personal style of writing reports).

During the course of this project Sweco Position developed a road inspection tool for MSB for the purpose of digitalising road inspections. The tool was based on an already existing system, RAPID (*Road Assesment Programme Inspection Device*), and consists of a tablet PC, a GPS, a database and a road inspection software. The road inspector will, with the use of the software, store information about the road in the database. See figure 1.1.



Figure 1.1: Overview, RAPID

1.2 Project requirements

The goal of this project has been to create a proof of concept web map that will present the data collected during a road inspection (as described in the previous section). The web map prototype reads files which has been generated from the data in the database and displays the information on the map. See figure 1.2.

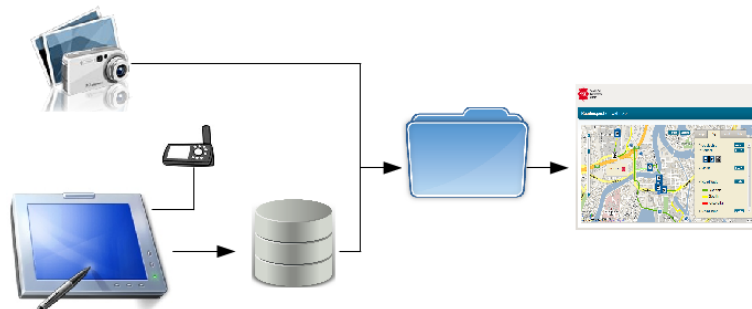


Figure 1.2: Overview, inspection system

The following parts were a requirement for the project:

- Web map displaying an area
- Display placemarks and routes on the web map
- A user should be able to hide and show different placemarks and routes

1.3 Chapter overview

The process of developing the web map prototype is described in the following chapters.

In chapter 2, the background of the project is discussed. Here, the process of a road inspection is explained and existing systems surrounding the web map are presented. In the chapter the prototype concept is also described and the security aspects of using a web

map are discussed.

The experiment itself, the implementation of the web map prototype, is presented in chapter 3. Here the details of the implementation are explained, both how already existing solutions have been implemented into the web map prototype and how new solutions have been developed. In chapter 3 the appearance of the web map prototype is also presented, as well as the different parts of the GUI and user options.

In chapter 4 the result of the experiment is evaluated, how the result matches the demands and what known problems there are in the implementation. As the web map developed in this dissertation is only a prototype, a discussion about what steps are needed to be taken in order to achieve a fully working project is also made.

In chapter 5 an evaluation of the project as a whole is presented, knowledge that have been acquired as well as other areas of application for web maps similar to the one developed during the course of this project.

2 Background

This chapter will cover the background of the project and the prototype implementation. This will be done by presenting the structure of the system as well as giving a brief overview of the underlying technology specific to our project. It will also show where the web application prototype - which is the final product of this project, together with this report - fits into the greater system to be used by the customer.

2.1 Introduction

RAPID (*Road Assessment Programme Inspection Device*) is an application developed by Sweco Position [2] and is a tool used to perform road inspections. A road inspector will use RAPID to collect data about the road, and the data collected during these inspections is later used to classify roads into different security categories. The system consists of hardware in the form of a GPS and a tablet PC together with software that stores GPS coordinates in a database and allows road inspectors to take notes about the road.

A modified version of the RAPID tool is currently being developed by Sweco Position for the Swedish Civil Contingencies Agency, MSB (*Myndigheten för Samhällsskydd och Beredskap*). MSB is a Swedish Government Agency whose task is to "*enhance and support societal capabilities for preparedness for and prevention of emergencies and crises.*" [3]

These include for example relief efforts in regions in foreign countries suffering from natural disasters. Before these missions, a report is written about the region in question. The report is used by different organisations that are involved in the effort, for example the United Nations [4] and SIDA (*Swedish International Development Cooperation Agency*) [5]. The report includes an analysis about the roads in the region since it is, on such missions, vital that communications and accessibility works well. In order to ensure this, roads are inspected for accessibility so that vehicles are able to arrive at their intended destination in time, without running into unexpected obstacles.

Currently these road inspections are produced by hand with the use of pen and paper. An inspector takes handwritten notes as they travel down a road. This way of doing inspections is time consuming and brings difficulties to distribution of information, which is why RAPID will be adapted to MSB's requirements and replace the handwritten inspections. Since RAPID was originally designed to calculate the security level of a road and not to present the data recorded by the inspectors, a way of presenting the data recorded needs to be developed. A prototype of a web application used to present data recorded by RAPID will be presented in this dissertation. The web application will consist of a web map containing the routes examined. The purpose of the map is to provide a confirmation to the inspector that the stored information is correct, and to ease the distribution of recorded data.

The dissertation will also explore options on how to link RAPID to the web application.

2.2 The existing system

2.2.1 RAPID

The RAPID software is made for tablet PCs. When a road is to be inspected the inspectors will bring a tablet PC with RAPID installed, and when they are at the start of the route to be inspected, they activate the GPS (*Global Positioning System*) [6] and start the program. The program will then wait for input from the inspectors which will in turn be written to the database.

Once started, a GUI containing several buttons appears (see figure 2.1). When the user clicks on a button, data is stored in the database along with the coordinates registered by the GPS. There are two types of buttons: One that will store data each time they are clicked and one that will store data from when they are activated by a click to when they are deactivated by another click. The first kind stores discrete data such as intersections while the latter stores continuous data, for example speed limits.

The entire route is filmed with the use of a digital video camera that has been attached to the rear-view mirror of the inspection car. The inspector may also record vocal comments about the road during an inspection. This media data is synchronised with the help of the Exif (*Exchangeable image file format*) [7] timestamps of the media files. The timestamp of the video camera has previously been synchronised with the GPS clock.

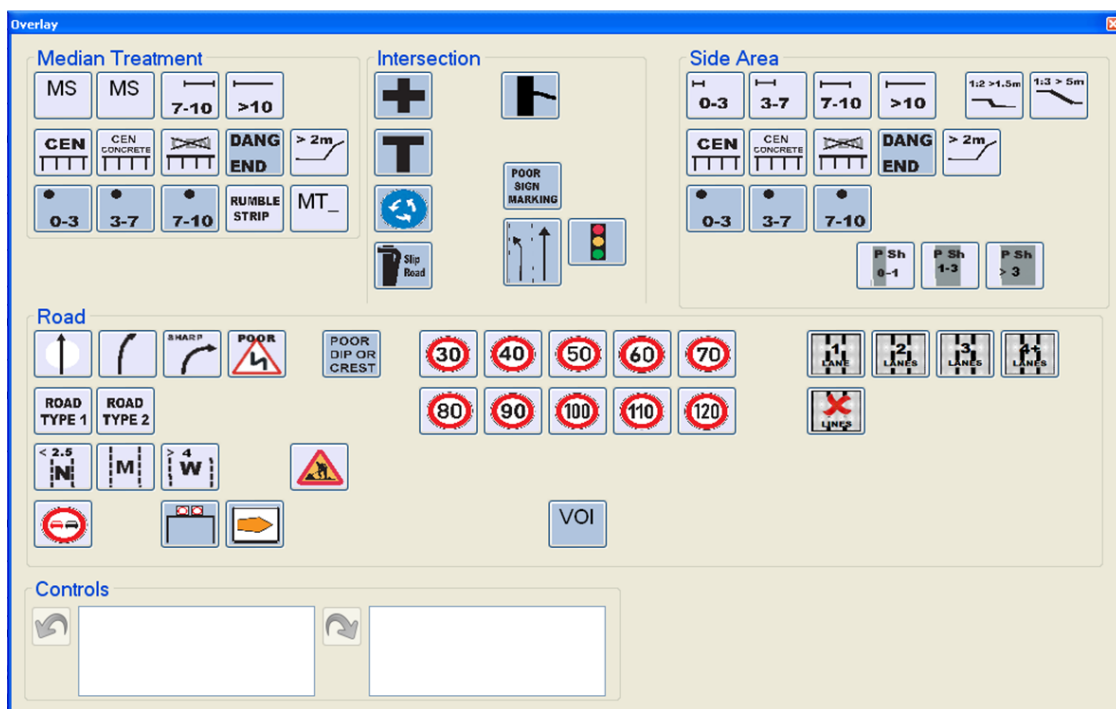


Figure 2.1: RAPID user interface [1]

2.2.2 Database

The database is an Access database [8] that is, during an inspection, stored locally on the tablet PC. Once the inspection is finished, the database is exported and sent to Sweco Position where the data is analysed and the security classification of the road is calculated according to iRap (*International Road Assessment Programme*) [9] standards. For a detailed overview of the database, see appendix A.

2.3 System overview

An inspection process consists of four major steps; inspection, processing, transmission and publication. (See figure 2.2)



Figure 2.2: The inspection process

1. The first step is the actual inspection. Data about the inspection is stored locally on the inspector's tablet PC while awaiting processing. This is where RAPID will be used.
2. The processing step will be performed after an inspection. Data will then be processed so that it can be used by the web application. A new data processing program (in this dissertation called DPAS, Data Processing Application Suite) will be developed to ensure this.
3. Since data will be processed and stored locally at a geographical location separated from the publication server, the data will need to be transmitted.
4. Finally, the data will be published so that it is accessible by all interested parties.

2.4 Adjustments of RAPID

During the course of this dissertation, the RAPID software will be modified to fit MSB's special requirements. The user interface will be modified so that the options presented to the inspector fit the kind of data that MSB wants to collect. Another major difference is that the MSB inspectors will not video film the entire road as RAPID is set up to do now. Instead, they will have the option to film certain parts of the road. They will also bring

a digital camera with them to document the route, something that was not supported in the original RAPID software.

2.5 Data Processing

2.5.1 DPAS, Data Processing Application Suite

In addition to being equipped with the modified RAPID software the inspection computers will also have a data processing application. Once the inspectors choose to run this application the DPAS will start to process the collected data into a format more suited to be handled by the end application. The positional data will be made into KML files whereas the media files can be compressed to a file size more easily transmitted.

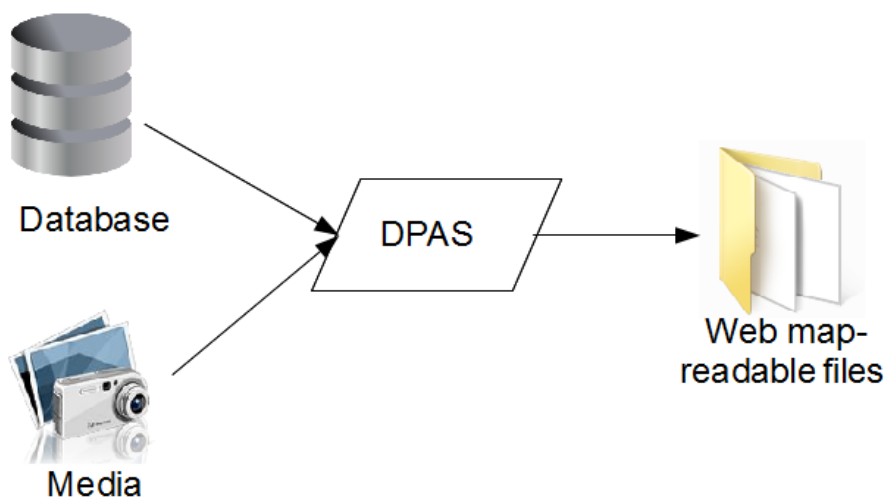


Figure 2.3: Overview of the DPAS

2.5.2 Media

Due to the various connection speeds out in the field the size of the media files can cause complications when trying to upload them to the end server where the map-service will gather its input from. To counteract this problem the media files can be compressed into a

more suitable size. There are two major ways of going about this compression. The first of these two is simply to keep the original files and apply an external compression to the file. The other way is to trade quality for smaller size. In a final version this can be handled by the DPAS where the end user could select which alternative to use or a combination of both.

2.5.3 Positional data

The DPAS also handles the positional data that can be found in the inspection computers local database. The storage of GPS positions into the database at frequent intervals helps to ensure a greater precision for the data gathered during the inspection of any given route. However, trying to represent all the gathered data in the map-service could result in a very slow system and might potentially make the service unusable on limited connections. To ensure that the system will be responsive and usable even while on a limited connection the DPAS will minimize the data points it extracts from the database. Positions of discrete information such as bridges, traffic lights or the locations where media was captured will be taken from the database along with the basic layout of the route.

Once the data has been gathered from the database it is then processed into a format that the map-service can handle. This is done by sorting the data into KML files (*Keyhole Markup Language*) [10, 11], which the map-service then can use to correctly map out the route and the points where discrete information can be found.

2.5.4 Transmission and Publication of Data

The KML files created by the DPAS as well as the media files must after creation and compression be sent from the local computer to a central host where it will be published and made accessible by all involved parties.

2.6 The prototype

This section will describe the concept of the web application prototype that will be developed and implemented during the course of this project.

2.6.1 Prototype Concept

Currently the field inspection results in a paper report which is distributed manually in order to spread the information gathered while out in the field. The purpose of the prototype is to take the information from the field inspection and represent it visually to give the users a more rapid overview of the data. As the prototype is based around an internet map service it will also be possible to quickly spread the data to the various organizations that are in need of the result of the field inspection without having to send the report to them as long as a connection to the internet is available.

2.6.2 Technical Requirements

As the system will be required to show regions in a variety of countries in great detail so as not to degrade the precision and relevance of the data it is important that the map service can provide high quality maps, even of remoter places of the world. To this end Google Maps [12] maps are used as a base for the map service as Google Maps provide high quality maps with good coverage. For the rest of the map service OpenLayers [13], an open source library for web map display, is used.

2.6.3 Web Map

The KML files generated by the DPAS are read by the OpenLayers based map service and the data is then displayed visually on the map. To make the data displayed easier to view and navigate, functions for hiding the various kinds of information are needed. This will include functionality for displaying the different types of discrete information that has been gathered as well as displaying the date when the individual inspections were made.

The data is also divided into different route ids which can also be used to sort through the data represented on the map. This is to enable the end user to focus solely on the desired route without being distracted by geographically close data.

To give a more in depth understanding of regions deemed to be of interest by the inspectors, different media are utilised. This can come in the form of photographs taken by the inspectors or even video recordings. These points of interests are marked on the map and when selected can then be displayed or played if the user desires.

2.6.4 Exporting the Map

To improve the flow of communications and the sharing of data between the different end users, functionality for exporting data from the map service is desired. This will be done by exporting the selected view of the map either to a pdf file or into a pure bitmap. This will enable the end users to exchange files of a desired view or print the view out in order to share the information physically.

2.6.5 Graphical User Interface

With regard to the graphical user interface it is necessary to keep it simple and intuitive without losing any functionality. For the prototype this is achieved by making the GUI as familiar to the end users as possible. This is done by using established conventions for a toolbar based GUI as well as selecting a colour scheme and theme that make the map service seem like an integrated part of the overall system of the end user.

The GUI will consist of a map displaying the current region the end user is looking at and a toolbar. The toolbar will handle the displaying of desired data points as well as additional information. Along with this it will also be the part of the map service where the end user can find the additional functionality such as exporting the current view or selecting a region for closer inspection.

2.7 Security

In this section we will consider data security aspects of the new RAPID system.

2.7.1 Confidentiality

Since MSB is a Swedish agency, all data they collect will be public information according to the Swedish freedom of information legislation [14]. Thus, information confidentiality is not a major concern when it comes to the data collected with the use of RAPID.

2.7.2 Integrity

To ensure the integrity of sent data to the publisher, the senders need to authenticate themselves in some way in order to reduce the risk of spoofing, i.e someone masquerading themselves as an inspector sending data. It would seem important that false data is not spread to parties involved. If incorrect data is distributed, knowing who spread this data would of course be desirable.

2.7.3 Accessibility

The data recorded by RAPID will have to be accessed by many different organisations and by non-Swedish speakers. For this reason, the language and the symbols used in the web application must be easy to understand by as many groups as possible. It is from an accessibility perspective also vital that the host where the data is stored is reliable, and in the event of the hosting servers going offline, that a backup is available.

2.8 Summary

In this chapter the background and goal for this dissertation project have been discussed. The goal is to develop a web application where data collected during road inspections is presented on a map. This dissertation will also give design suggestions on how the

inspection system RAPID (an existing tool for performing road inspections) can be linked to the web application.

One of these modules that will be designed and that has the goal of linking RAPID with the web application is the DPAS. This module will process the data collected by the inspector and convert it into the format needed for the web application. Once the data has been processed, this data will need to be transmitted to a central host.

In this chapter, the web application concept has also been described. For the web application, it is important that a map with an high quality and good reliability is used. To ensure this, Google Maps will be used in the web application.

Since the end users of this application will be very varied, a GUI that is easy and similar to other systems already in use will be developed.

Security aspects of the new RAPID system have been considered, and two weak points that need to be examined have been identified; the reliability of the service and the integrity of data sent from the inspector.

To summarise, the following will be implemented in the project:

- A web map prototype displaying an area
- Placemarks and routes displayed on the web map
- User options for hiding and showing different placemarks and routes

3 Experiment

3.1 Introduction

This chapter will cover the implementation of the web map prototype. It will explain how the input files of the web map prototype are created and how they are read by the web map.

The chapter will also present the graphical user interface, and explain the technical details about the interface functionality.

Finally, the different systems and libraries used to build the system will be presented. Different alternatives to the various parts of the prototype will be presented along with a discussion about why, in the end, these alternatives were not used. For an overview of the different parts of the system, see figure 3.1.

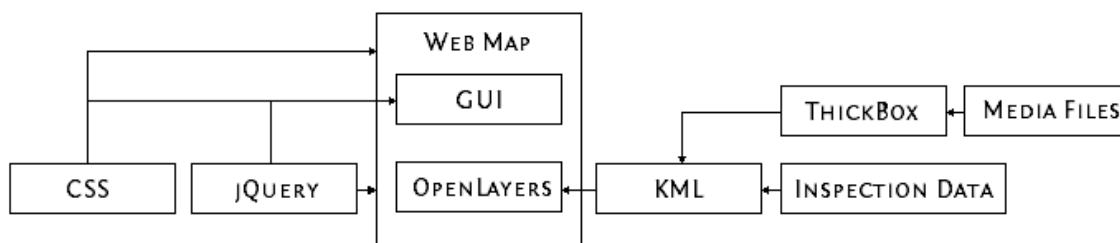


Figure 3.1: Overview, web map prototype

3.2 Overview

To view the web map prototype a browser that supports JavaScript is needed. The prototype has only been tested in Mozilla Firefox [15] version 3.6.3, which is why support in other browsers or older versions of Mozilla Firefox are not guaranteed.

Since not all content of the webmap was developed by the authors of this dissertation, special consideration was made to licenses when choosing between different external solutions. Open source licensed solutions were given priority.

The web map prototype is written in JavaScript, CSS, and HTML [16, 17, 18]. See the respective section for more information about JavaScript and CSS.

3.2.1 System overview

The web map prototype consist of several different kind of files. To make it easier for developers, files have been organised into a clear file hierarchy. (See figure 3.2) The HTML file that will be opened by the browser is located directly under the "Web map" folder. The "Web map" folder in turn consists of four subfolders, some of which in turn also contain subfolders:

- **KML.** This folder contains KML files that will be loaded into the map. The KML list file, `kmlfile.txt`, which is explained in section 3.3.2, is also located here.
- **Routedata.** This folder contains mediafiles (such as images, video, audio and documents) that can be displayed in the web map.
- **Script.** The script folder contains all JavaScript libraries.
 - **jQuery.** Contains the jQuery library.
 - **MapScript.** Contains the JavaScript developed for this web map.
 - **OpenLayers.** Contains the OpenLayers library.
 - **Thickbox.** Contains the Thickbox library.
- **Style.** This folder contains files that define the look of the web map. CSS files are located directly under this folder.
 - **Icons.** Contains icons used for the data displayed in the map.

- **Images.** Contains images used on the web map page.

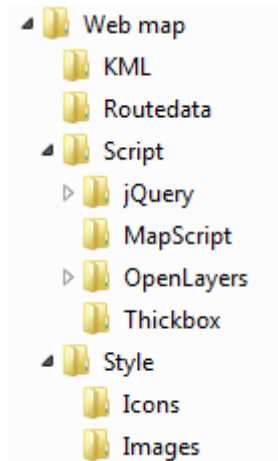


Figure 3.2: File hierarchy

3.3 From inspection to web map useable data

3.3.1 KML

KML is an XML-based[19] standard [10, 11, 20] that can be used in OpenLayers (see section 3.6.1) to read data. To use the data collected from RAPID in the web map, the data from the database has to be converted to KML files. Once it has been converted into KML, it can be used in the web map prototype. Exactly how the KML files are structured will be explained in the next section.

In the web map prototype, the same kind of data is grouped into one KML file. This means that there will be one KML file for traffic lights, one KML file for hotels, one KML file for restaurants etc. All these files are then loaded into the web map with the help of OpenLayers.

For this prototype the process of converting the inspection data into the KML files have been done manually. In a final implementation of the finished system this will be

automated to relieve the end users and administrators from additional workload.

3.3.2 Creating KML files

This section will explain the process of creating the KML files as well as how these files connect to the web map.

For the web map to be able to display the data collected during an inspection, the information gathered must be converted into KML files. There is a difference in the creation process if the data to be handled by the KML file is discrete (point based) or continuous (path based) data. The process is also slightly different depending whether the KML file is to handle with media or not.

Continuous KML files

KML files are based on an element structure where each element contains various attributes. Placemark and geographical data would be an example of elements whereas style and description would be examples of attributes. For the KML files containing continuous information, the geographical data is represented by a linestring element that consists of a list of coordinates (consisting of - in order - latitude, longitude and altitude) that form a line. Each KML file can contain one or more line elements. To form one continuous line from these elements the starting coordinates of one element would be the end coordinates for another element.

The style attribute may either be defined before the placemark elements, and will thus be a shared style attribute, or be defined specifically for each placemark element. For the continuous KML files the style attribute defines the width and colour of the lines on the web map.

See appendix D for an example of a KML file containing continuous data.

Discrete KML files without media

Discrete KML files have a structure very similar to continuous KML files. An example of a KML file with discrete values can be seen in appendix C. The difference is the point element, which takes the place of the linestring element in the continuous KML files. Where as the linestring element contains a list of coordinates, the point element only contains one coordinate.

As the discrete KML files contains points instead of lines, the style attribute defines different parameters for the discrete KML files than it does for the continuous KML files. Where the style attribute for the continuous KML files defined the width and colour of the lines, the style attribute for the discrete KML files instead defines which map marker icon to be used to indicate a point element on the web map.

Discrete KML files with media

The difference between discrete KML files that contains references to media and discrete KML files that contains no reference to media lies in how they handle the description attribute. Where the continuous and discrete KML files without media references relies more on the style attribute to convey information to the end user, the discrete KML files with media relies on the description attribute. Although the KML files containing media references still use the style attribute to place a point on the map to indicate the presence of a media file, no additional information about the media file with the exception of what kind of file - audio, video or image - can be conveyed through the style attribute. The description tag for these KML files therefore contains a thumbnail of the media file as well as a hyperlink to the media file. Both the thumbnail and the hyperlink are used by the popup that the webmap places at the point element when the map marker icon is clicked (for more information see section 3.4.3). The hyperlink to the media file utilises Thickbox - a JavaScript library - for representing the media file when clicked. More information about Thickbox can be found in section 3.6.3.

KML file list

While it would be desirable to simply iterate through an entire folder of files, JavaScript does not support this. Instead the web map prototype uses a text file (named `kmlList.txt`) containing a list of the KML files that will be loaded into the map and simply iterates through that list and reads all files that is on that list.

The text file containing the list of KML files that should be loaded into the map also determines what data icons and lines are on top of others. KML files that are added last to the map will be in the topmost layer, while KML files that are added first to the map will be in the bottom. It is therefore recommended that KML files containing lines is put first in the text file list followed by KML files containing discrete data. The discrete data should be added in rising priority, with the highest priority last. This will ensure that icons with important data will obscure icons with less important data and not the other way around.

3.3.3 Linking KML files to the map with the use of OpenLayers

To display KML file data on the web map, the prototype utilises the layer functionality of OpenLayers. The layer functionality allows for OpenLayers to read the KML files and represent each KML file as an individual layer on the web map. This is done with the use of the function that can be found in appendix E. The function reads the text file containing a list of KML files to be read and then creates one layer for each KML file found. For each of the layers created the function also adds a layer control to the layer (popups, see section 3.4.3) as well as activating control objects that can be found in the GUI, so that the layers can be manipulated by the end user. For more information about the control objects, see section 3.4.5. The function uses both OpenLayers built-in functions as well as features from jQuery. For more information about OpenLayers and layering see section 3.6.1 and for jQuery see section 3.6.2.

3.4 Web map

3.4.1 Overview

The web map prototype GUI is based on a GUI developed for another MSB system by Sweco Position. (See appendix B for the appearance of the already existing GUI.) Permission were given by Sweco Position to use this in the development of the new web map.

Choosing an already existing GUI gives several advantages, the most important being that the end users do not have to learn another GUI since they are familiar with the already established one. An already established GUI would also graphically fit in better with MSB's current system. Lastly, using an already established GUI would ease future updates since developers do not need to learn the structure of a completely new GUI.

The prototype GUI consists of two major parts: The map and the menu. The menu allows the user to make selections about what data they want to view, while the map presents the data to the user. (See figure 3.3 for an image of the prototype web page.)

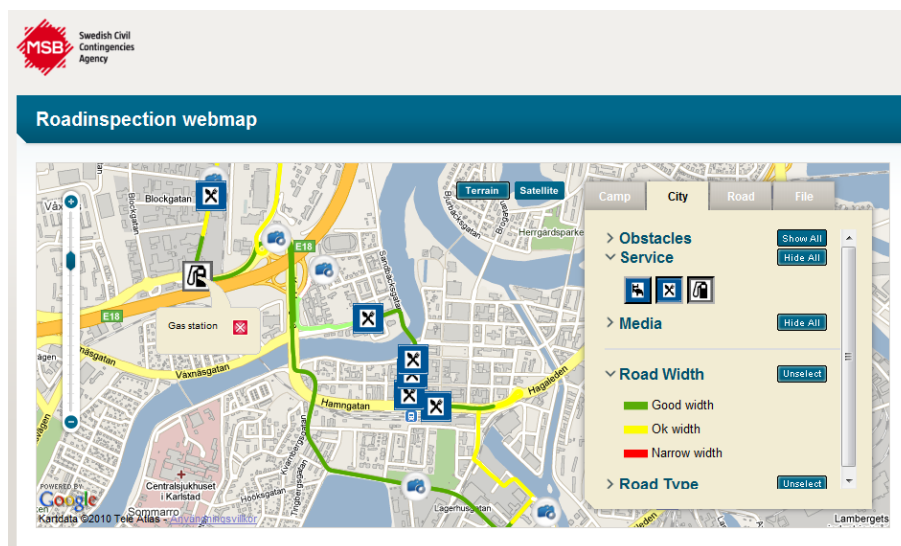


Figure 3.3: Prototype web page GUI

3.4.2 Map service

There are plenty of map services already available on the internet, the most prominent one being Google Maps. Other popular services such as Hitta.se [21] and Eniro.se [22] were considered for the prototype. However, Google Maps were chosen for this project due to its coverage and quality.

Google Maps has its own API that can be used to create specially tailored map services, locking the final product to the API provided by Google[23]. Adding the Google Map service to a domain also requires an API key, which is obtained by a free registration to the Google Maps API. Although anyone can use the API, Google Maps itself is not an open source solution - something that was requested and is striven towards by Sweco Position Karlstad. As such, while Google Maps is one of the leading Web based map service providers, the project needed something different. To this end OpenLayers were chosen. The maps provided by Google can be used in OpanLayers so the map coverage and quality did not get effected by this choise.

3.4.3 Map

The purpose of the map is to present the collected data to the user. Discrete data (data that consist of a single coordinate) will appear on the map as icons while continuous data (data that consists of several linked coordinates) will appear as coloured lines. The user can click on icons on the map to view a popup containing more information about that specific point on the map. Media icons such as images, video, audio and documents are special. Their popups will contain thumbnails of the media, and when the thumbnail is clicked the media will be displayed to the user. See figure 3.4 and 3.5 for image examples of the map functionality.

The user can choose between two types of base maps; Street or Satellite. (See figure 3.6 for an example of the two types of base maps.) Both maps come from the Google Maps [12] map service. When changing base maps, the icons and lines on the map will not be

affected.

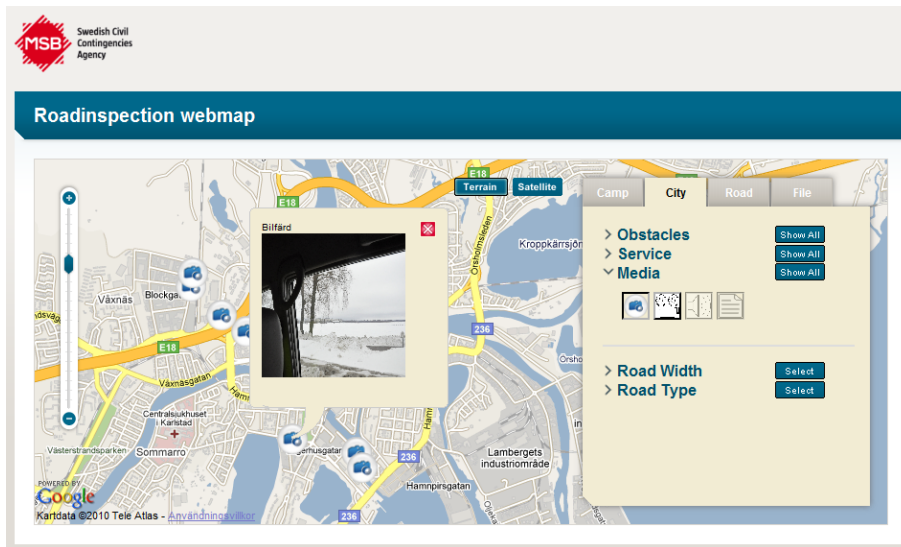
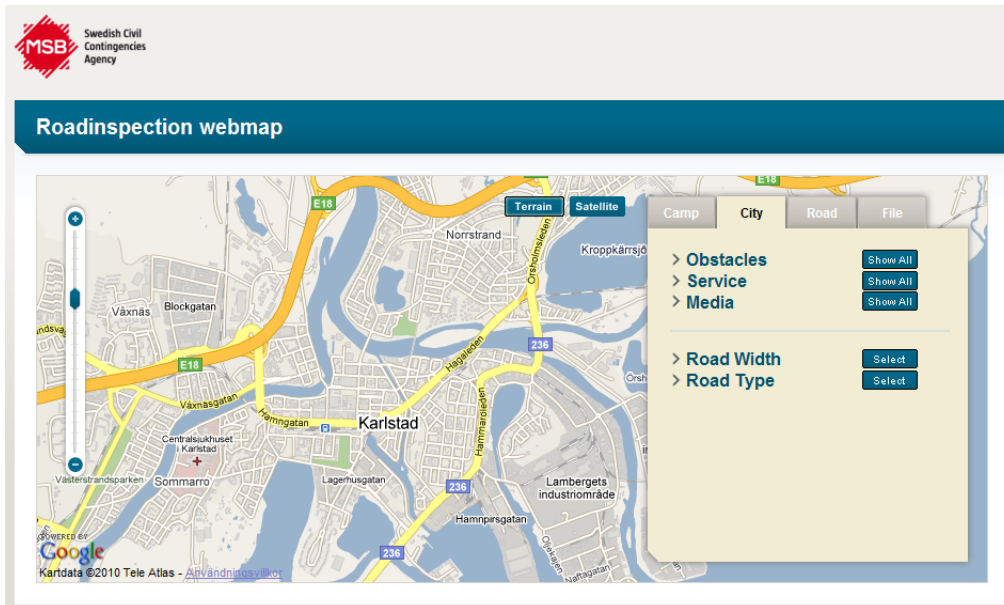


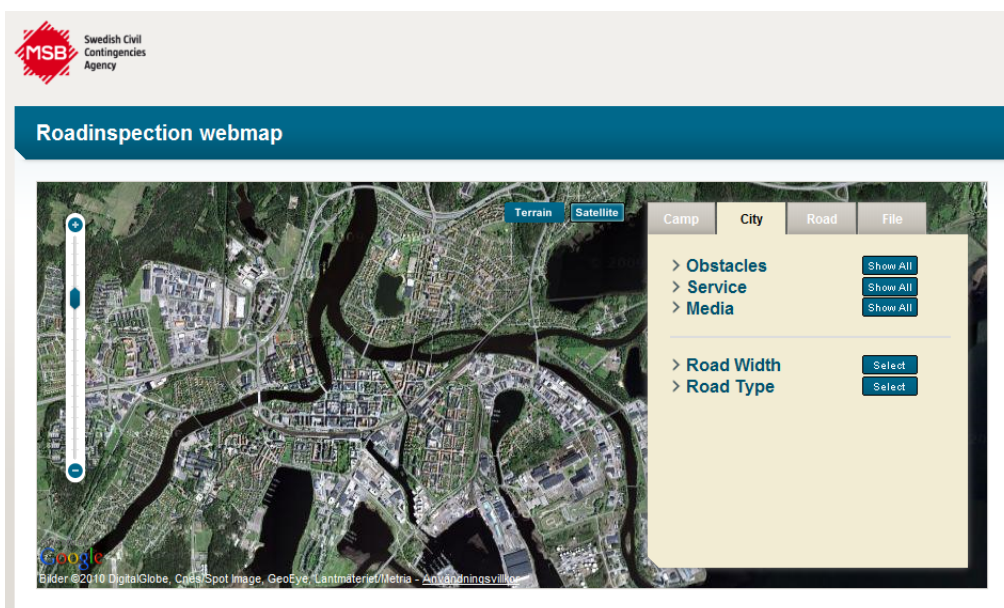
Figure 3.4: GUI: Viewing media popup information



Figure 3.5: GUI: Viewing a media file



(a) Google Maps street



(b) Google Maps satellite

Figure 3.6: GUI: An example of Google Maps street and satellite

3.4.4 Menu overview

On the right side of the screen a toolbox consisting of four different tabs can be found. These four tabs are Camp, City, Road and File. (See figure 3.7)

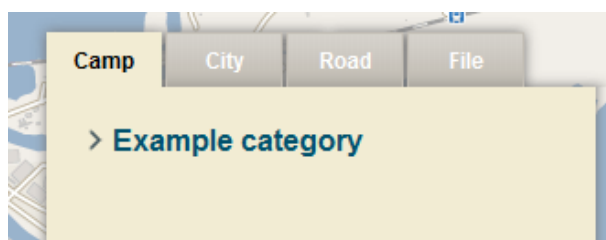


Figure 3.7: Menu: The four tabs; camp, city, road and file.

Out of these four the prototype only uses the City tab, while the remaining three will be filled with functionality if the prototype is expanded for the final release. These tabs, with the exception of the File tab, will correspond to settings in the modified RAPID system being developed by Sweco Position for MSB. The idea behind the different tabs is to give the user tools suited for viewing the different types of inspections. The idea behind the file tab is to give the user alternative ways to share the data with others in need of information.

The menu is built out of images that are controlled by CSS. When the user selects a tab by clicking on it, the system will set the visibility of the selected tab and the contents within it to be displayed and set all other tabs and their respective content to be hidden. In addition, the CSS-class for the tabs will change, which will change the appearance of the tabs to clearly indicate to the user which tab is currently selected. See figure 3.8.

In addition to the tab-grouping, each tab is also divided into subgroups. (See figure 3.9) The content of each subgroup can be set to be shown or hidden by the user to fit their requirements. This is done by simply clicking on the desired subgroup. When clicked, the subgroup will toggle between expanded - showing content - or minimised - to show only the group name.

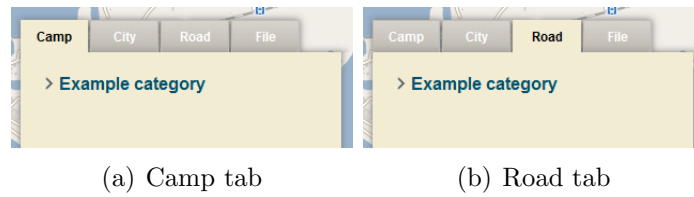


Figure 3.8: Menu: Changing tabs

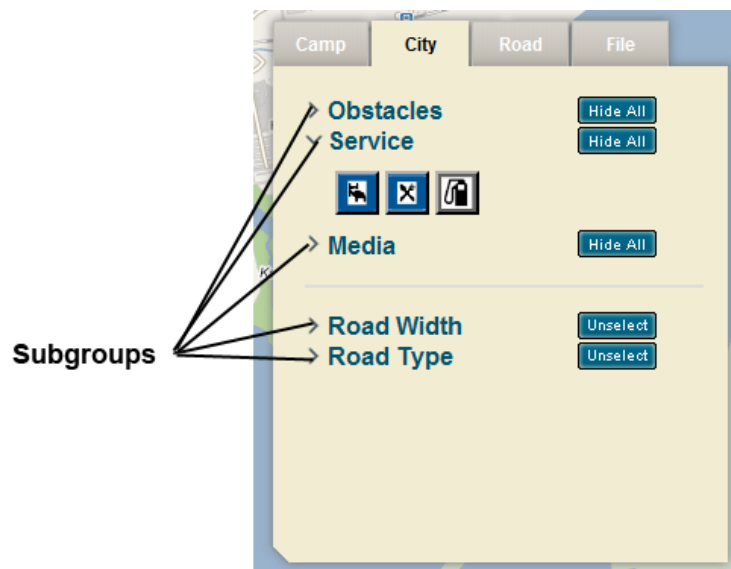


Figure 3.9: Menu: Subgroups

The two different kinds of data - discrete and continuous - are represented differently in the menu. Discrete data and continuous data are sorted into different subgroups and while the discrete data will be represented by buttons/icons, the continuous data will be represented by lines showing the color coding for that specific continuous data. (See figure 3.10.)

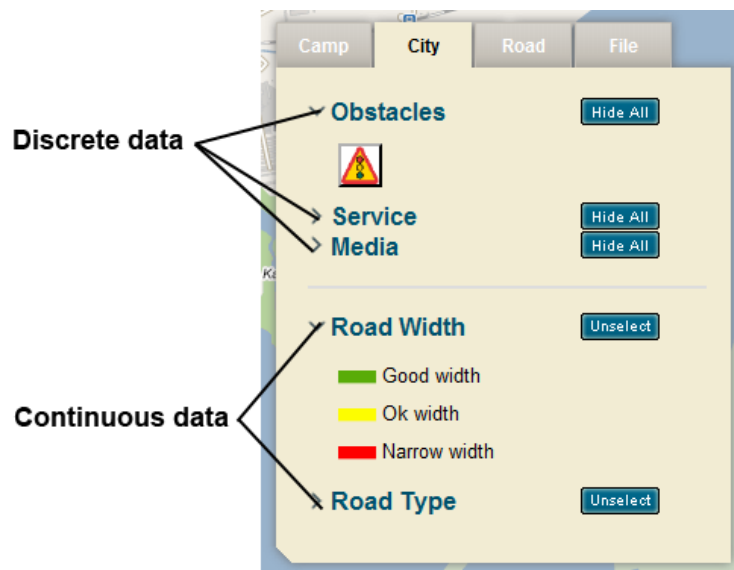


Figure 3.10: Menu: Discrete and continuous data

3.4.5 Menu selections

This section will explain the different types of selectors that can be found in the menu and how they work.

- **Icon Buttons:** Buttons corresponding to icons on the map can be found in the menu. Through the use of these buttons, the user can sort through the information and make selections so that the information displayed on the map suits that particular user's needs at that given moment. This is done so that when a button is clicked the system will toggle the visibility of the information represented by the button. In addition to this the button's CSS-class will also change which will update the graphics of the button to make it clear to the user if a button is selected or unselected. There are currently three different CSS-classes for each button; One for the selected buttons, one for the unselected buttons and one disabled class for the buttons which does not have any information linked to them for a particular inspection. (See figure 3.11.) This is done so that the GUI will not have to change between different inspections

and give the end user a consistent GUI that does not need to be relearned between inspections.

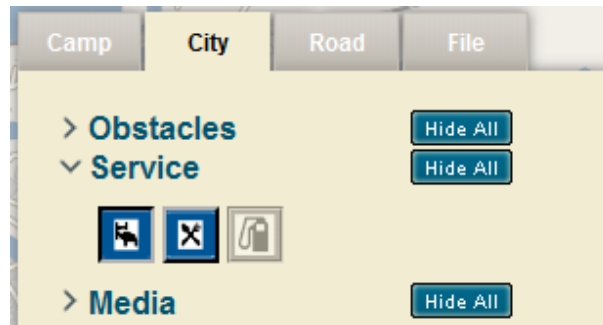


Figure 3.11: Menu: Icon buttons. (Selected, unselected and disabled.)

- Group Selectors: In addition to the individual buttons a button is assigned to each subgroup of data that contains a group of related, individual buttons. (See figure 3.12.) These buttons have the capability to select or unselect all the buttons within their respective groups. The purpose of these is to allow the user to quickly narrow down the information that they find of interest.
- Continuous Selectors: To select or deselect continuous road data, a button with similar function to the group selector is used. (See figure 3.12.) The reason for this is that only one type of continuous data can be displayed at any given time. This is to make the information readable for the end user. The main difference between the Group Selectors and the Continuous Selectors is that when the user selects a continuous road-data to be shown the visibility of the previously selected continuous data is set to hidden so that only the currently selected continuous data and all selected discrete data is visible on the map.

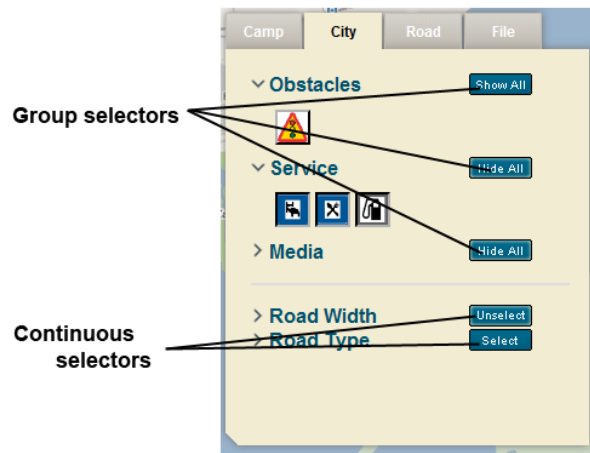


Figure 3.12: Menu: Group selectors and continuous selectors

3.5 CSS, Cascading Style Sheets

Since the web map consists of pictures and text that can be grouped together in categories, coding by hand how each picture or text should be handled would create an unnecessary workload for the developers. Instead, CSS - Cascading Style Sheets[19] - is used to determine how certain categories of elements are handled.

CSS defines rules on how a type of for example an image, text or a block-level element should be placed or appear on a web page.[24]

This approach makes for easy maintenance of the code as the developers do not have to go through the whole prototype to change the appearance of a selection of objects but can instead change the parameters in the CSS file.

3.6 JavaScript

JavaScript [25, 26] is an object oriented scripting language primarily for use in web pages. It has a wide area of use, from being able to detect and handle user events - such as mouse-over - to validating forms.

This section will describe the JavaScript libraries that were utilised to add various

functionality to the system as well as the development process.

3.6.1 OpenLayers

OpenLayers is a JavaScript library that is used to build web maps. [13] OpenLayers was chosen as a base for this web map prototype because of its open source license. It has detailed documentation and many web page examples of its functionality so it is easy to get started in building a web map even for a developer that has no previous experience with web map implementations.

OpenLayers can be used in combination with many different base maps. A standard map is provided, but attaching a Google Map base map to OpenLayers is also possible. However, since maps can have different map projections [27] a conversion is necessary if the base map has a different projection than the data that is to be displayed on the map. This was not a problem in the development of the web map prototype discussed in this dissertation but it is an aspect that should be kept in mind in the future development of the prototype and for anyone using OpenLayers.

OpenLayers can read XML files with elements containing coordinates and other data in order to display this on a map. Several different types of XML-based files are supported, the most common ones being geoRSS [28] and KML. Because KML is such a widespread standard and since it is Google's own, it is used in this web map as it uses the Google Maps base map.

OpenLayers, as indicated by its name, uses layers to present data. It uses the object orientation that is supported in JavaScript so that every layer on the map is an object that can be manipulated. For example: To hide a layer, the layer property "visibility" is set to false. This has been used in the web prototype explained in this chapter. All similar types of data are all stored in one layer. This means for example that hotels are in stored one layer while restaurants are stored in another etc. When the user then clicks an button (as explained in section 3.4.4) to hide or show a type of data or a whole category of data, the

corresponding layers in OpenLayers will be hidden or shown.

The popups discussed in section 3.4.3 are also a built-in OpenLayers function (although their appearance have been altered in the web map prototype).

3.6.2 jQuery

jQuery [29] is a JavaScript library with the main purpose to speed up development. It's basic functionality lies in that it - simply put - has a lot of call functions. A developer can instead of writing regular JavaScript call pre-declared functions to minimise the amount of JavaScript that they have to write by hand to achieve the same result.

jQuery functions revolve around the call to the jQuery class. The new class object will then use the function that is called together with the jQuery class call. An example of a jQuery call would be `"$("div").css("border","9px solid red");"`

The dollar sign, "\$", is an alias for the jQuery class call, and what the example call does is to create a jQuery class object that selects all "div" elements and modify their css so that they receive a 9 pixel thick solid red border.

The reason this library was added to the prototype was in part due to a request by Sweco Position. However, using jQuery reduced the amount of code that was needed to be written to implement the web map prototype and made the utility of the library an important part of the project, that was under a tight time constraint.

3.6.3 Thickbox

The Thickbox JavaScript library [30] is used when displaying media in the prototype web map. During the course of looking for a library to handle the media display several different libraries were found. Most of the libraries found early on were only able to handle image display which made them unusable for the prototype. In terms of functionality, Shadowbox [31] seemed to be the best library to display media. However, as this library was licenced

it was put aside in favour of Thickbox. Thickbox also utilises jQuery, which meant that it would need no additional resources to enable multimedia display.

Adding Thickbox to a media file is done by adding attributes to the HTML anchor element [32] to the media file in question. The use of media files other than images becomes a bit more complex in its Thickbox compatibility. For media other than image files Thickbox utilises iframes [33] as part of the displaying process. This addition created an extra layer to the implementation as each media file - other than images - had to be imbedded into their own html documents. These documents were then displayed in the iframe by Thickbox.

It should also be noted that to show media other than images to the end user, the user must have a media player installed for their browser. If this is not the case the media will not be displayed to the user.

3.7 Summary

This chapter has discussed the various parts of the implemented prototype and has given an overview of the finished version. The chapter has discussed how the graphical user interface was built and how it has been modified from the previous system it was based on. The user aspects of the GUI and the mechanics behind them has also been discussed. Although the GUI was built on an existing system, many changes were made for the prototype and only the layout and overall design of the previous system remains in the prototype.

The chapter has also looked into map services in general and why this prototype was made using OpenLayers, as well as going into how this particular service presents the map and the inspection-data on the screen for the end user. As KML has become a standard that is widely supported this was chosen for the prototype. The addition that it is widespread also means that further additions to the system is likely to also be compatible.

What mechanics is used to store the inspection-data so that the prototype can read it and why that storage was chosen has also been a part of the discussion. The underlying

parts have been adressed. While not chosen for the benefit of the end user directly jQuery helped to free up more time for implementation of other features that in the end benefitted the end user. It also tied together with thickbox for displaying multimedia which gives the end user more easaly viewable media.

4 Result

4.1 Introduction

This chapter will aim to evaluate the finished prototype and how well the end result matched the original specifications. In addition the chapter will cover potential improvements to the prototype and the systems around the prototype as well as future uses of the system.

4.2 Result

4.2.1 Evaluation of the web map prototype

Sweco Position, the main project owner of the web map prototype, has been able to see the web map application and its functionality and to give their feedback throughout the whole project. This feedback has been predominantly positive and they have expressed great satisfaction with the end result.

As the web map developed is only a prototype, no stress testing or testing in the environment of the final product has been made. Testing of the functionality of the web map prototype has however been done on a local host. The web map will need to be integrated into MSBs system and tested within their environment to obtain a more detailed test result.

4.2.2 Known problems

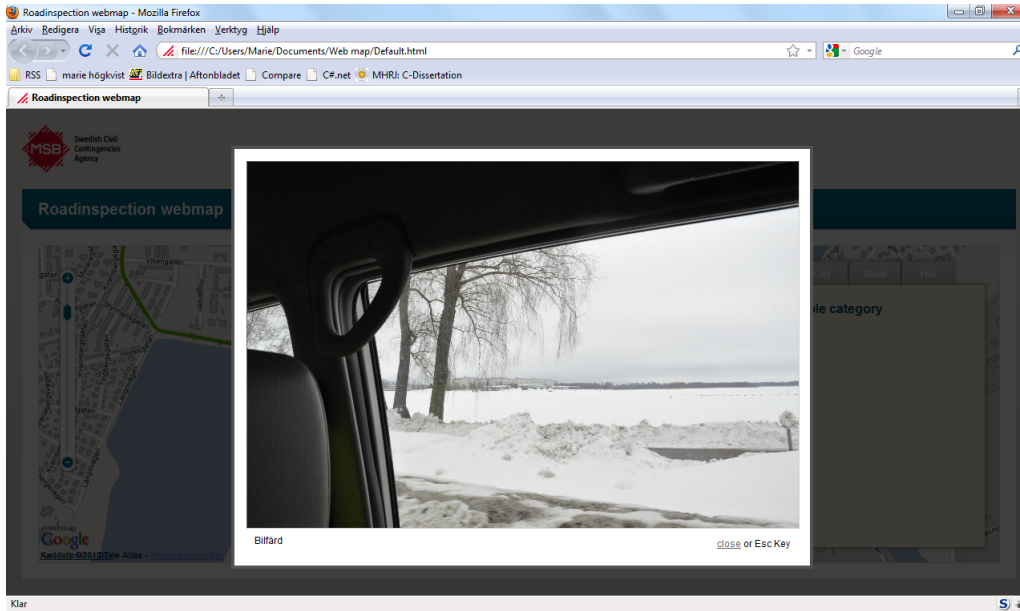
This section will discuss problems with the current implementation of the web map prototype, as well as possible solutions to these problems.

- Browser support: Currently the only supported browser for the web map prototype is Mozilla Firefox. Testing with Internet Explorer revealed a problem with loading the KML files into the browser. Other browsers have yet to be tested.

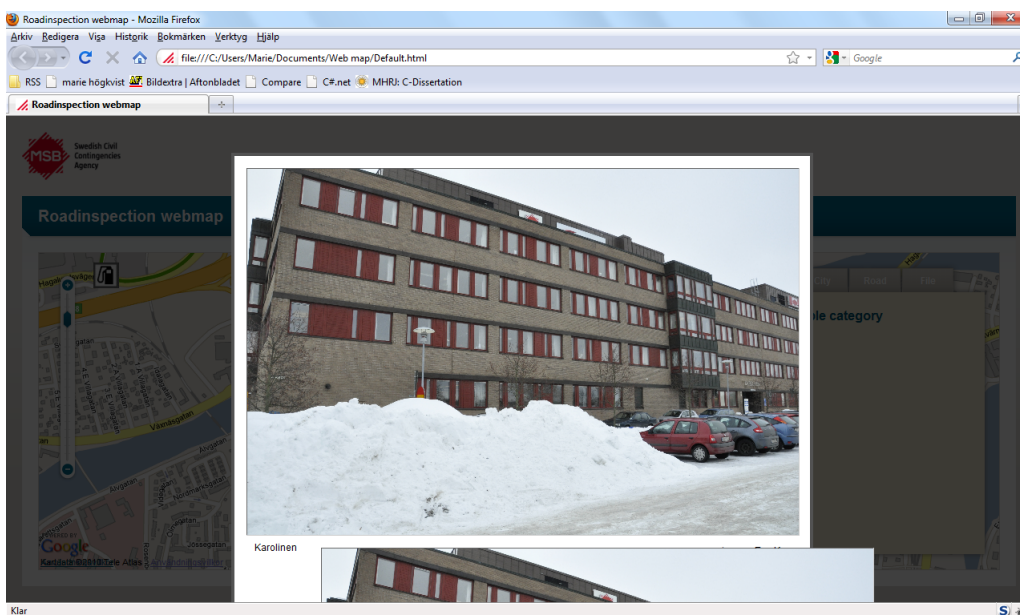
- Icon layering: When two icons that are close to each other are loaded into the web map prototype, the most recently loaded of the two will cover the other. This problem stems from how the inspection tool gathers data. To solve this the inspection tool would have to be modified, which lies outside the scope of this project, or extra functionality would need to be added to the KML generation to ensure that no two icons fully overlap each other.
- Watching videos: If the end user does not have a plugin for the video extension, watching videos in the web map prototype is not possible. This problem has no solution, as to play a media file in a browser the browser must support this functionality. An alternative to playing videos in the browser would be to allow the user to download the media instead.
- Thickbox bug: Viewing images from popups may result in a faulty image. (See figure 4.1.) The exact reason for this bug, if it is native to Thickbox or if it is an error due to the combination of Thickbox, KML and OpenLayers, is unknown. More testing would need to be done to determine what triggers this bug, if it is solvable or if an alternative to Thickbox will need to be used. Sadly due to time constraints this could not be done during the course of making this prototype but should be done before the product is taken into use.

4.3 From prototype to working project

This section will handle the improvements that need to be made to the prototype before it can be shipped to the end user for use in their work. This will include direct improvements to the prototype itself as well as systems that the final version will rely on, covering the entire inspection process from inspection to publication as explained in section 2.3, System overview.



(a) Correctly displayed thickbox image



(b) Thickbox bug: Note the extra image appearing at the bottom.

Figure 4.1: Thickbox bug

4.3.1 DPAS

DPAS, the Data Processing Application Suite (mentioned in section 2.5.1) is currently not implemented. As a result of this the prototype currently relies on manual labour to create the KML files needed to represent the inspection data. To create the KML files manually is time consuming as well as giving rise to human error, and as such automating this process through the DPAS would be preferable. This would mean that the DPAS needs to be implemented before the web map is taken into use.

One possible implementation of the DPAS would be to create a database view from the database in the inspector computer and through one or many applications create the KML format needed as well as compressing the media files to a desired file size.

4.3.2 Publication

The files generated by the DPAS will need to be accessible by the web map. Therefore they will need to be uploaded to a server where the web map can access and read them. There are several ways that this can be accomplished, which will be discussed below.

- DPAS upload: From a user perspective, letting the DPAS handle the upload would be the easiest solution. It is however the most time-consuming solution from the developer's perspective as more time need to be spent on error handling, time that could be used to improve other features of the web map application. This would need to be done as the DPAS would need to connect to a server and upload the files automatically, without involving the user in the upload process. The effort in creating this would however result in a more integrated system where no reliance on outside parts would be needed.
- User upload: Another solution could be to let the end user upload the generated files themselves. This could be done through, for example, a stand alone FTP application or a web interface. This is a fairly easy-to-implement solution, but for the more

inexperienced computer users this solution presents some challenge.

- Email, then admin upload: Yet another option is to let the end user send the generated files to an administrator, who in turn would upload the files to a server. This would mean that inexperienced computer users can send the files without having to worry about the final upload. The downside to this is that it would place a greater strain on the administrator as it would create a bigger workload.

Limiting the upload possibilities to merely one option would mean a slimmer and faster system, but this would also make the system more vulnerable to disruption. Enabling several different upload possibilities would create a failsafe by increasing the system redundancy as well as giving the end user a choice in how they want to upload the files to the web map application.

4.3.3 World map

The web map prototype currently displays only one inspection. When displaying several inspections, new aspects need to be taken into consideration.

Several inspections could be read into the web map at once, and the user could then go to the area that they are interested in. The problem with this is that loading the KML files for several different inspections will create a very slow system as the web map would have to load a considerable amount of data, having the browser handling numerous large files at once.

One solution could be to implement a multitude of web maps - one map per inspection. This however has the risk of creating a system structure that would be hard to obtain a good overview of. Another solution that would solve both the risk of loading too many KML files into the same map making it slow, and removes the risk of a multitude of scattered maps would be to create a world map. This map would give an overview of the world where all inspected areas are marked. An inspection could be selected from this

view, and the end user would then be redirected to a web map similar to the one in the web map prototype that would load only the KML files for the selected inspection.

4.3.4 File handling

As mentioned in section 2.6.4 - Exporting the Map, sharing a selected section of information from the map service is desired. Therefore a file handling option where the part of the map that the user is currently viewing can be exported into a, for example, pdf format should be implemented into the final web map. Additional functionality could be to let the user load a map layer into the web map. This would give the user the possibility to compare the information presented by the web map with information gathered elsewhere. Functionality for sharing a view with another end user using the map may also be implemented to give the end users more alternatives how to share the desired information.

4.4 Future development possibilities

Currently the web map prototype represents static information. To modify information about any given route the inspectors would have to make a whole new inspection. The need to modify information can arise from for example seasonal changes to the landscape as well as natural catastrophies.

To update a route with new information is currently a slow process. Future development can enable an administrator to use the web application to directly modify the data from any given route directly in their browser. This would result in a system that would not solely rely on road inspections to take place, instead new and up to date information could be called in to the administrator by radio or telephone and thus speeding up the process making the system reliable even during extreme conditions, such as information spreading during a disaster relief effort.

4.5 Summary

This chapter has given an evaluation of the result of the web map prototype. Overall the result is good and the project owner Sweco Position is satisfied with the prototype. However, as a prototype it still carries a few minor bugs and problems, such as limited browser support and bugs when viewing media.

The chapter has also discussed the steps needed to be taken to evolve the web map application into a fully functional working product. For this, the KML files needed for the web map need to be generated from the inspection database and media might need to be compressed to avoid long loading times on slower connections. Since the prototype web map only displays one inspection, a solution for displaying several inspections needs to be developed. In this chapter, a solution has been discussed where inspections are marked on a world map and the end user can then select which inspection to view.

To make the prototype fully functional, file handling should be implemented in the final product. This file handling function should be able to export maps into for example pdf format as well as import map layers into the application.

Finally, the chapter discussed future development possibilities. This consists of an admin-mode where an admin can modify and update data directly from the web map, without having to upload a completely new inspection.

5 Evaluation

In this chapter the project will be evaluated in terms of time planning, acquired knowledge and general project success.

5.1 Time planning

The project has followed a strict schedule, with several deadlines. These deadlines have been held and the time planning has been met from the authors' side. Some deadlines have however been postponed due to heavy workload on MSB.

5.2 Acquired knowledge

Knowledge acquired during the course of this project includes JavaScript and general web development with a focus on CSS, as well as knowledge in less technical areas such as how parts of international relief efforts are handled.

On the technical side a deeper understanding of the different areas of OpenLayers were acquired. While previous experience with OpenLayers existed, this project required a deeper understanding of the framework - in particular how KML is handled by OpenLayers. In addition, focus has been placed on finding different existing solutions - such as jQuery, OpenLayers and ThickBox - to achieve a fully functional application in a short period of time. Making the different parts interact with each other was also a learning experience.

On the less technical side of the project, knowledge of how relief efforts are handled influenced the development of the web map prototype. Better insight to what MSB would need from the prototype was gained, as well as in what ways the web map could evolve to give MSB an even better tool for use in the various areas of a relief effort as well as after an inspection.

5.3 What could have been done differently?

Looking back at the project, there are a few things that could have been done differently. Working more closely with the developers of RAPID would have resulted in a better knowledge about the changes that were made to RAPID while the web map prototype was being developed. This would have allowed the web map prototype to be more equipped for the changes that are to be made to the RAPID system as it would have given insight in how the finished RAPID system would work and what parameters (i.e, road width, road type) the web map application would need to work with.

How much difference this would have made for the project itself is uncertain as the demands of the prototype functionality were already defined (as described in chapter 1 and 2) and have been met. It could however have made a difference if the web map prototype is to be developed further and turned into a full application.

Closer contact with MSB would also have been preferred, to gain a deeper understanding in how the web map prototype would work in their system and what tasks the end users will use the web map for. This in order to learn more about the end users and their computer experience and expectations, to be able to implement a web map design as suitable to their needs as possible.

5.4 Other areas of application

The specific map developed during the course of this project has the purpose to display data from road inspections performed by MSB, but the principle of displaying placemarks on a web map can be used in various different areas of application.

This makes the web map explained in this dissertation highly adaptable and relevant to developers who wish to implement a similar web map displaying placemarks or routes.

5.5 Summary

To summarise the project, a proof of concept has been developed. The proof of concept is a prototype of a web map, displaying placemarks and routes from a road inspection. Menu options for hiding and showing different placemarks and routes have also been implemented.

Sweco Position Karlstad, who were the commissioners for the project, are satisfied with the result and the project can thus be considered successful.

References

- [1] “Recieved from sweco position karlstad.”
- [2] “Sweco position.” <http://www.sweco.se/position>.
- [3] “Swedish civil contingencies agency.” <http://www.msb.se/>.
- [4] “United nations.” <http://www.un.org/>.
- [5] “Sida.” <http://www.sida.se/>.
- [6] “Wikipedia article about gps.” http://en.wikipedia.org/wiki/Global_Positioning_System.
- [7] “Wikipedia article about exif.” http://en.wikipedia.org/wiki/Exchangeable_image_file_format.
- [8] “Wikipedia article about microsoft access.” http://en.wikipedia.org/wiki/Microsoft_Access.
- [9] “irap.” <http://www.irap.net/>.
- [10] “Wikipedia article about kml.” http://en.wikipedia.org/wiki/Keyhole_Markup_Language.
- [11] “Kml documentation.” <http://code.google.com/intl/en/apis/kml/documentation/>.
- [12] “Google maps.” <http://maps.google.com/>.
- [13] “Openlayers.” <http://openlayers.org/>.
- [14] “The principle of public access.” <http://www.sweden.gov.se/sb/d/2184/a/15521>.
- [15] “Mozilla firefox.” <http://www.mozilla.com/en-US/>.
- [16] “Html tutorial on w3schools.” <http://www.w3schools.com/html/default.asp>.
- [17] “Wikipedia article about html.” <http://en.wikipedia.org/wiki/HTML>.
- [18] “The html standard.” <http://www.w3.org/html/>.
- [19] “The css standard.” <http://www.w3.org/Style/CSS/>.
- [20] “Kml standard.” <http://www.opengeospatial.org/standards/kml>.

- [21] “Hitta.se.” <http://www.hitta.se/>.
- [22] “Eniro.se.” <http://www.eniro.se/>.
- [23] “Google api.” <http://code.google.com/apis/maps/>.
- [24] “Css tutorial on w3schools.” <http://www.w3schools.com/css/>.
- [25] “Javascript tutorial on w3schools.” <http://www.w3schools.com/JS/default.asp>.
- [26] “Wikipedia article about javascript.” <http://en.wikipedia.org/wiki/JavaScript>.
- [27] “Wikipedia article about map projection.” http://en.wikipedia.org/wiki/Map_projection.
- [28] “Wikipedia article about georss.” <http://en.wikipedia.org/wiki/GeoRSS>.
- [29] “jquery.” <http://jquery.com/>.
- [30] “Thickbox.” <http://jquery.com/demo/thickbox/>.
- [31] “Shadowbox.” <http://www.shadowbox-js.com/index.html>.
- [32] “Html anchor element.” http://www.w3schools.com/HTML/html_links.asp.
- [33] “Iframes on w3schools.” http://www.w3schools.com/TAGS/tag_iframe.asp.

A ER Diagram of the RAPID Database

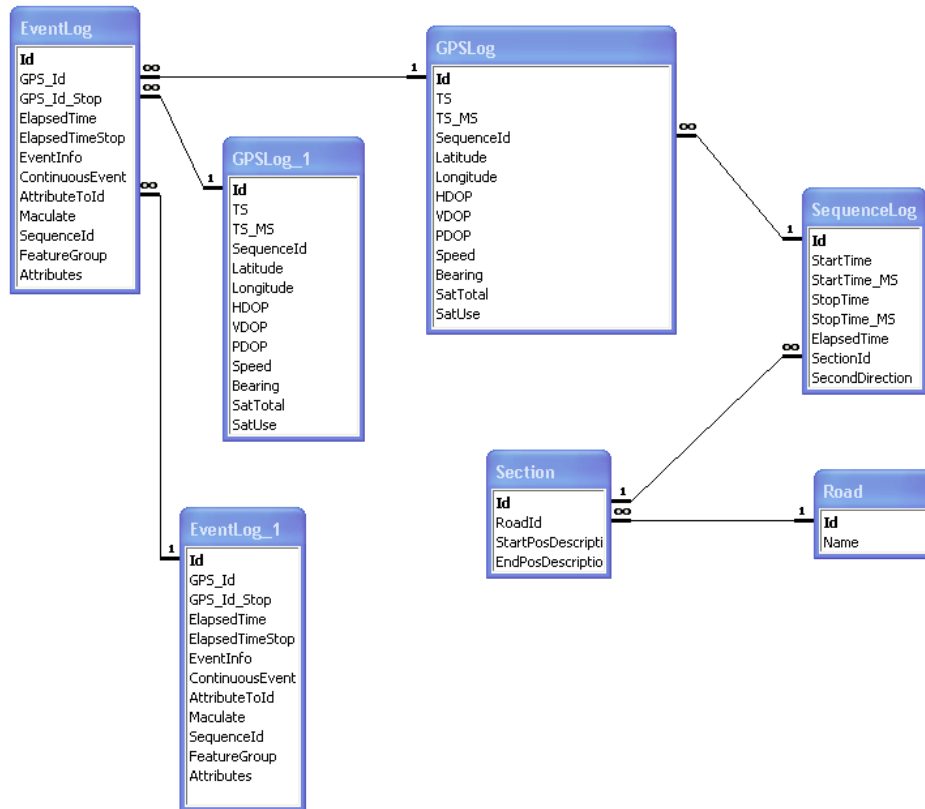


Figure A.1: ER Diagram of the RAPID Database [1]

B Base GUI

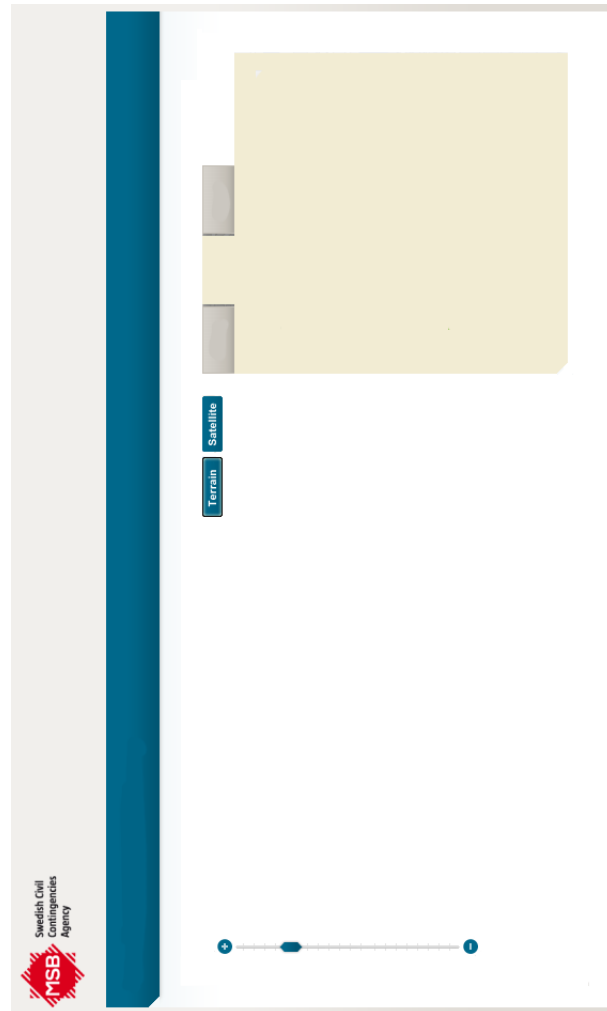


Figure B.1: The base GUI used for development

C KML file, discrete values

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <kml xmlns="http://earth.google.com/kml/2.2">
3 <Document>
4   <Style id="placemarkicon">
5     <IconStyle>
6       <Icon>
7         <href>style/icons/_hotell.png</href>
8       </Icon>
9     </IconStyle>
10  </Style>
11  <Placemark>
12    <name>Hotel</name>
13    <description>Hotel</description>
14    <styleUrl>#placemarkicon</styleUrl>
15    <Point>
16      <coordinates>13.49908666666667,59.37845166666667,0 </
          coordinates>
17    </Point>
18  </Placemark>
19  <Placemark>
20    <name>Hotel</name>
21    <description>Hotel</description>
22    <styleUrl>#placemarkicon</styleUrl>
23    <Point>
24      <coordinates>13.50157,59.37825666666667,0 </coordinates>
```

```
25     </Point>
26   </Placemark>
27 </Document>
28 </kml>
```

D KML file, continuous values

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <kml xmlns="http://www.opengis.net/kml/2.2" xmlns:atom="http://
  www.w3.org/2005/Atom" xmlns:gx="http://www.google.com/kml/ext
  /2.2">
3 <Document>
4   <Style id="style1">
5     <LabelStyle>
6       <scale>1.0</scale>
7     </LabelStyle>
8     <LineStyle>
9       <color>fff000</color>
10      <width>4</width>
11    </LineStyle>
12  </Style>
13  <Style id="style2">
14    <LabelStyle>
15      <scale>1.0</scale>
16    </LabelStyle>
17    <LineStyle>
18      <color>ff0000</color>
19      <width>4</width>
20    </LineStyle>
21  </Style>
22  <name>Continuous</name>
23  <visibility>1</visibility>
```

```

24 <Folder id="continuous1">
25   <name>Width</name>
26   <Placemark id="line1">
27     <name>line1</name>
28     <styleUrl>#style1</styleUrl>
29     <LineString>
30       <coordinates>13.5086216666667,59.3732383333333,0
                13.5087466666667,59.3733133333333,0
                13.5088683333333,59.37339,0
                13.50899,59.3734633333333,0
                13.5091116666667,59.3735366666667,0
                13.5092333333333,59.3736083333333,0
                13.5093533333333,59.3736783333333,0
                13.50947,59.3737466666667,0
                13.5095833333333,59.3738116666667,0
                13.5096883333333,59.373875,0
                13.5097783333333,59.37393,0
                13.5098533333333,59.373975,0
                13.5099166666667,59.3740083333333,0
                13.5099683333333,59.3740283333333,0 </coordinates>
31     </LineString>
32   </Placemark>
33   <Placemark id="line2">
34     <name>line2</name>
35     <styleUrl>#style2</styleUrl>
36     <LineString>
37       <coordinates>13.510005,59.3740383333333,0

```

13.51003333333333,59.3740416666667,0
13.510065,59.3740416666667,0
13.51010833333333,59.3740383333333,0
13.51015833333333,59.3740283333333,0
13.5102116666667,59.3740116666667,0
13.5102666666667,59.37399,0
13.5103216666667,59.3739633333333,0
13.51037333333333,59.373935,0
13.51042333333333,59.3739066666667,0
13.51047,59.37388,0 13.51051333333333,59.373855,0
13.5105566666667,59.37383,0
13.51058833333333,59.37381,0
13.510605,59.3737983333333,0</coordinates>

38 </LineString>

39 </Placemark>

40 </Folder>

41 </Document>

42 </kml>

E Code snippet for loading KML files into web map

```
1 function addAllKml(filename)
2 {
3     $.get(filename , function(data)
4     {
5         names=data.split("\r\n");
6         for(var i=0; i<names.length-1; i++)
7         {
8             _map.addLayer(new OpenLayers.Layer.GML("KML" , names[i]+" .
9                 kml" , {format: OpenLayers.Format.KML, CLASS_NAME: "mp" ,
10                    formatOptions: {'extractStyles': true}
11                }));
12             _map.layers[i+2].setName(names[i]);
13             $("#" + names[i]).removeClass("disabled").addClass("showL")
14             ;
15         }
16         selectControl = new OpenLayers.Control.SelectFeature(_map.
17             getLayersByClass("mp") ,
18             {onSelect: onFeatureSelect , onUnselect: onFeatureUnselect})
19             ;
20         _map.addControl(selectControl);
21         selectControl.activate();
22     });
23 }
```