# SAFESPOT INTEGRATED PROJECT - IST-4-026963-IP

## ANNEX

### SP7 – SCORE – SAFESPOT Core Architecture

## LDM API and Usage Reference

<table>
<thead>
<tr>
<th>Deliverable No. (use the number indicated on technical annex)</th>
<th>D7.3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubProject No.</td>
<td>SP7</td>
</tr>
<tr>
<td>SubProject Title</td>
<td>SAFESPOT Core Architecture</td>
</tr>
<tr>
<td>WORKPACKAGE NO.</td>
<td>WP3</td>
</tr>
<tr>
<td>WORKPACKAGE TITLE</td>
<td>Specification</td>
</tr>
<tr>
<td>TASK NO.</td>
<td>T7.3.1</td>
</tr>
<tr>
<td>Task Title</td>
<td>Building of the Global System Reference Architecture</td>
</tr>
<tr>
<td>Authors (per company, if more than one company provide it together)</td>
<td>G. Vivo - CRF</td>
</tr>
<tr>
<td>Status (F: final; D: draft; RD: revised draft):</td>
<td>F</td>
</tr>
<tr>
<td>Version No:</td>
<td>0.7</td>
</tr>
<tr>
<td>File Name:</td>
<td>SF_D7.3.1_Annex2_LDM_API_and_Usage_Reference_v0.7.doc</td>
</tr>
<tr>
<td>Planned Date of submission according to TA:</td>
<td>30/08/07</td>
</tr>
<tr>
<td>Issue Date:</td>
<td>23/09/10</td>
</tr>
<tr>
<td>Project start date and duration</td>
<td>01 February 2006, 48 Months</td>
</tr>
</tbody>
</table>
## Revision Log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Name and Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>25-09-09</td>
<td>First draft</td>
<td>G. Vivo, CRF</td>
</tr>
<tr>
<td>0.2</td>
<td>02-10-09</td>
<td>IP chapter added</td>
<td>O. Kannenberg, Tele Atlas</td>
</tr>
<tr>
<td>0.3</td>
<td>03-01-10</td>
<td>Added content on Object Model, LDM schema and API</td>
<td>G. Vivo, CRF</td>
</tr>
<tr>
<td>0.4</td>
<td>12-01-10</td>
<td>Update chapter 3.1 PG-LDM</td>
<td>Sheung Ying Yuen-Wille, Bosch</td>
</tr>
<tr>
<td>0.5</td>
<td>14-01-10</td>
<td>Document layout update, minor style revision of chapter 3.1</td>
<td>G. Vivo, CRF</td>
</tr>
<tr>
<td>0.6</td>
<td>28-01-10</td>
<td>Reformulation of one paragraph in the IP chapter</td>
<td>G. Vivo, CRF</td>
</tr>
<tr>
<td>0.7</td>
<td>23-09-10</td>
<td>Revision implementing the remarks of the final review (Added captions to fig. 36, 40 and 42)</td>
<td>G. Vivo, CRF</td>
</tr>
</tbody>
</table>
## Abbreviation List

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAS</td>
<td>Advanced Driver Assistance Systems</td>
</tr>
<tr>
<td>ADASRP</td>
<td>NAVTEQ ADAS Research Platform</td>
</tr>
<tr>
<td>AGORA-C</td>
<td>common name of the ISO Standard 17572-3 Intelligent Transport System (ITS)—Location Referencing for Geographic Databases—Part 3: Dynamic Location References</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CVIS</td>
<td>Cooperative Vehicle-Infrastructure Systems. EC funded Integrated Project, in the context of the same EC CALL where SAFESPOT is collocated</td>
</tr>
<tr>
<td>ESRI</td>
<td>Popular vectorial format for Geographic Information System (GIS), developed by the homonymous service company</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>GML</td>
<td>Geography Markup Language</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronic Engineers</td>
</tr>
<tr>
<td>IP</td>
<td>Integrated Project</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property (*)</td>
</tr>
<tr>
<td>IPR</td>
<td>Intellectual Property Right</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Transport Systems</td>
</tr>
<tr>
<td>LDM</td>
<td>Local Dynamic Map</td>
</tr>
<tr>
<td>MMF</td>
<td>Modelling Meta File – 3D GIS modeling format</td>
</tr>
<tr>
<td>NAVTEQ-LDM</td>
<td>SQLite Local Dynamic Map – Navteq implementation of the LDM</td>
</tr>
<tr>
<td>OpenGIS</td>
<td>Former designation of the Open Geospatial Consortium (OGC), an international voluntary consensus standards organization</td>
</tr>
<tr>
<td>PG-LDM</td>
<td>PostgreSQL Local Dynamic Map – Tele Atlas implementation of the LDM</td>
</tr>
<tr>
<td>PostGIS</td>
<td>Open source Geographic Information System software program, adding support for geographic objects to the PostgreSQL object-relational database</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Object-relational database management system (ORDBMS), released under a BSD-style license (open source)</td>
</tr>
<tr>
<td>PSF</td>
<td>Physical Storage Format</td>
</tr>
<tr>
<td>SAFESPOT</td>
<td>EC funded Integrated Project where the LDM concept was originally introduced, specified and implemented</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SQLite</td>
<td>ACID-compliant relational database management system contained in a relatively small C programming library</td>
</tr>
<tr>
<td>WKT</td>
<td>Well Known Text</td>
</tr>
</tbody>
</table>

(*) Note: in the document the term “IP” is used with two different descriptions; the context where the term is used is clear enough to disambiguate its meaning.
Table of contents

REVISION LOG ......................................................................................................................... 2
ABBREVIATION LIST ............................................................................................................. 3
TABLE OF CONTENTS ........................................................................................................... 4
LIST OF FIGURES ................................................................................................................ 5
LIST OF TABLES .................................................................................................................. 7
EXECUTIVE SUMMARY ..................................................................................................... 8
1. INTRODUCTION ............................................................................................................ 9
   1.1. Document structure .............................................................................................. 9
2. OVERVIEW OF THE LDM ............................................................................................. 9
3. OBJECT MODEL AND STRUCTURE OF THE LDM ..................................................... 11
   3.1. PG-LDM .................................................................................................................. 11
   3.2. NAVTEQ-LDM ...................................................................................................... 14
   3.3. Data Model Description ....................................................................................... 15
      3.3.1. Static features .................................................................................................. 16
      3.3.2. Moving objects ............................................................................................... 23
      3.3.3. Conceptual objects ......................................................................................... 27
      3.3.4. Relationships .................................................................................................. 33
      3.3.5. Others ............................................................................................................. 35
4. APPLICATION PROGRAMMING INTERFACE (API) .................................................... 36
   4.1. Level 1 API ............................................................................................................. 36
   4.2. Level 2 API ............................................................................................................. 41
   4.3. API documentation ............................................................................................... 46
5. STATUS OF KNOWN INTELLECTUAL PROPERTY RIGHTS (IPR) .............................. 55
6. CONCLUSIONS ............................................................................................................ 56
7. REFERENCES ............................................................................................................... 56
ADDENDUM ........................................................................................................................... 56
List of Figures

Figure 1 – The four layers of the LDM ................................................................. 10
Figure 2 PG-LDM architecture in SAFESPOT .................................................... 12
Figure 3 - PG-LDM architecture in CVIS ........................................................... 12
Figure 4 - OpenJump visualisation of the Orbassano Test Track ...................... 14
Figure 5 - ADASRP server architecture ............................................................ 15
Figure 6 – colour labelling in the LDM tables .................................................... 16
Figure 7 – arealandmarks: Spatial area object modelled by a single polygon ...... 17
Figure 8 – crossing: Spatial area object modelled by a single polygon, representing a crossing of a road element with another transport mode, especially pedestrian, bicycle and railway ...... 17
Figure 9 – crossingforreferencetracks: Static relationship between a crossing and a reference track .... 17
Figure 10 – crossesignalgroup: Static relationship between a signal group and a crossing ......................... 17
Figure 11 – detectionarea: Spatial area object modelled by a single polygon, representing the coverage area of a sensor ........................................................................... 17
Figure 12 – detectionareacrossing: Static relationship between a detection area and a crossing ...... 18
Figure 13 – junction: Spatial point object modelled by a single point, representing a map database node that bounds a road element or a ferry connection ...................................................... 18
Figure 14 – linelandmarks: Spatial linear object modelled by a single polyline ........................................ 19
Figure 15 – pointlandmarks: Point landmarks are painted signs, poles or other physical spots which are used for positioning and detection purposes ............................................................. 19
Figure 16 – referencetracks: Spatial linear object modelled by a single polyline, representing a reference manoeuvre at an intersection from one road element connected to the intersection and a specific lane on that road element to another road element connected to the intersection and a specific lane on that road element ........................................................................... 19
Figure 17 – roadelement: Spatial linear object modelled by a single polyline, representing a map database edge that represents an elementary section of the road network ......................... 20
Figure 18 – roadintersection: Crossing and/or connection of two or more roads .................................. 21
Figure 19 – sensor: Spatial point object modelled by a single point representing a sensor, a device which detects or measures a physical property .......................................................................... 21
Figure 20 – sensorfordetectionarea: Static relationship between a sensor and a detection area ........ 21
Figure 21 – signalgroup: A logical object possessing a single state (see signalgroupstate) which is displayed on one or more traffic light heads (see pointlandmarks) and applying to one or more lanes of a single road element entering a junction ................................................................................................................................. 21
Figure 22 – signalgroupforreferencetrack: Static relationship between a single signalgroup and one or more referencetracks ................................................................................................................................. 22
Figure 23 – trafficsign: Spatial point object modelled by a single point representing a traffic sign or a variable message sign ................................................................................................................................. 22
Figure 24 – trafficsigninformation: Information content of a traffic sign, and relationship between this information and a traffic sign object ................................................................................................................................. 22
Figure 25 – egorsu: Information about the road side unit (RSU) that maintains the table ........................................ 23
Figure 26 – rsu: Information about other road side units (RSUs) that is maintained by a road side unit, or information about road side units that is maintained by objects that are not a road side unit ...... 23
Figure 27 – egomotorvehicle: An ego motor vehicle unit. Mutually exclusive to the ego RSU; i.e. either "egorsu" table is present in an LDM (SP2) or "egomotorvehicle" table is present (SP1), but not both. Only one entry in this table. The id of the egomotorvehicle is always 1 ........................................ 24
Figure 28 – motorvehicle: A non-ego motor vehicle unit. The value start at 2 as the 1 is reserved for the id of the egomotorvehicle................................................................................................................................. 26
Figure 29 – uo: Unidentified (or partially identified) object. Includes VRU, other "living" objects etc. Attribute list essentially consists of the motorvehicle's dynamics attributes (describing its motion) and feattyp descriptor. In SAFESPOT, these types of objects will not have a VANET, hence all attributes have to be derived remotely ................................................................................................................................. 26
Figure 30 – trailer: A trailer is connected to a tractor/truck/passenger car. It is not self-propelling. Hence its dynamics are governed by its hauling vehicle and its own characteristics ................................................................................................................................. 27
Figure 31 – accidenthotspot: Describes the results of statistical evaluation of historical accident data. 28
Figure 32 – dynamicblackspot: Describes a spot where a risk of an incident (accident) is given. This information is dynamic ................................................................................................................................. 28
Figure 33 – dynamicreferencetrackattributes: Dynamic data provided associated with the static reference track ................................................................................................................................................................. 28
Figure 34 – dynamicroadelementattribute: Dynamic attributes that can be associated to a road element -
in case the information should be provided per lane or per vehicle type several entries for a road
element can be specified. ........................................................................................................................................... 28
Figure 35 – dynamicicsensorattributes: Dynamic data provided associated with the sensor ............ 29
Figure 36 – dynamicicsensorstatus: Dynamic Sensor Status information .............................................. 29
Figure 37 – dynamictrafficsign: This feature represents a moving traffic sign (used for road works for
example) or a sub traffic sign display on a VMS .................................................................................................. 29
Figure 38 – dynamictrafficsigninformation: Identification of the dynamic traffic sign the dynamic
traffic sign information belongs to .......................................................................................................................... 29
Figure 39 – environmentalevent: This table provides measure whether information which are given in
general by weatherstations. ........................................................................................................................................ 30
Figure 40 – fcodevent: Floating Car Data events .......................................................................................... 30
Figure 41 – meteodetection: Detected weather events ................................................................................. 31
Figure 42 – roadconditionmeasurement: Measurement of the Road Condition (currently used only for
the snow thickness).................................................................................................................................................. 31
Figure 43 – roadconditionevent: Detected road conditions as defined in the roadconditionevent ........ 31
Figure 44 – signalgroupstate: Contains the dynamic attributes of a (static) signalgroup ...................... 32
Figure 45 – trafficevent: Externally generated message coming from safetycentre ................................. 32
Figure 46 – trafficobject: Traffic state information for a road element ..................................................... 33
Figure 47 – dynamicreferencepoint: recommended path when the road becomes part of a black-spot... 33
Figure 48 – alongroadelement: Map matched position of a motor vehicle or other moving object.
Explicit storage of moving object-road relationship. The field ID is the identifier of the given
relationship. This is mainly for consistency (i.e. that all tables have an "id" field). Queries concerning this
table will often/mostly/always(?) be given through the vehicle and/or road element
ids. ............................................................................................................................................................................. 33
Figure 49 – conceptualalongroadelement: m to n relationship between roadelements and conceptual
objects. The field ID is the identifier of the given relationship. This is mainly for consistency (i.e.
that all tables have an "id" field). Queries concerning this table will often/mostly/always(?) be
given through the conceptualroadelement_id and/or road_element_id ............................................................. 34
Figure 50 – trajectory: Contains the possible trajectories of moving objects. A single moving object can
have multiple trajectories. A trajectory is the predicted future path of the moving object. This may
consider the object's current dynamics (position, velocity, yaw rate etc.), the static road geometry
(possibly also reference tracks) and possibly even other indicators of driver intention (turn
indicator status etc.). The attributes here were derived from the LDM EAP file, as well as
FuturePath struct in SRresult.h from ICCS (SP1) ............................................................................................. 34
Figure 51 – gatewaycommunication: to ease communication between e.g. safetycentre-gateway and
dynamic black spot calculation allow Datareceiver to just save content of message in form of
string into this table. Using notification, interested parties can handle the data without polling.... 35
Figure 52 – messagebox: container for rsu-hmi messages. See also SP7 Data format and messages doc
Æ HMI MessageFromVenant ...................................................................................................................................... 35
Figure 53 – LDM-API classes, structs, unions and interfaces, with a brief related description .......... 46
Figure 54 – LDM-API class member functions, (1 of 22) ........................................................................ 46
Figure 55 – LDM-API class member functions, (2 of 22) ........................................................................ 47
Figure 56 – LDM-API class member functions, (3 of 22) ........................................................................ 47
Figure 57 – LDM-API class member functions, (4 of 22) ........................................................................ 47
Figure 58 – LDM-API class member functions, (5 of 22) ........................................................................ 48
Figure 59 – LDM-API class member functions, (6 of 21) ......................................................................... 48
Figure 60 – LDM-API class member functions, (7 of 22) ......................................................................... 48
Figure 61 – LDM-API class member functions, (8 of 22) ......................................................................... 50
Figure 62 – LDM-API class member functions, (9 of 22) ......................................................................... 50
Figure 63 – LDM-API class member functions, (10 of 22) ................................................................. 50
Figure 64 – LDM-API class member functions, (11 of 22) ................................................................. 50
Figure 65 – LDM-API class member functions, (12 of 22) ................................................................. 51
Figure 66 – LDM-API class member functions, (13 of 22) ................................................................. 51
Figure 67 – LDM-API class member functions, (14 of 22) ................................................................. 51
Figure 68 – LDM-API class member functions, (15 of 22) ................................................................. 52
Figure 69 – LDM-API class member functions, (16 of 22) ................................................................. 52
Figure 70 – LDM-API class member functions, (17 of 22) ................................................................. 52
Figure 71 – LDM-API class member functions, (18 of 22) ................................................................. 53
Figure 72 – LDM-API class member functions, (19 of 22) ................................................................. 53
Figure 73 – LDM-API class member functions, (20 of 22) ................................................................. 53

SF_D7.3.1_Annex2_LDM API and Usage Reference_v0.7.doc Page 6 of 56 Subproject: SP7 - SCORE
List of Tables

Table 1 - LDM Concrete Classes ........................................................................................................... 16
EXECUTIVE SUMMARY

The document describes the LDM (Structure of the database, Object Model, API) from the perspective of the final users.

Within the SAFESPOT IP, these users are the subprojects in charge to develop the vehicle based and the infrastructure based applications, together with the subprojects where the vehicle and the infrastructure platforms are in charge to fill in and maintain the consistence of the LDM database. Outside SAFESPOT, the users of the LDM are all the individuals and companies who intend to adopt this key achievement of SAFESPOT as a building block for the consistent and solid representation of the environment and the scenarios where their applications will work (typically in the domains of ITS, ADAS and Cooperative Systems).

In this respect, information related to the structure and the object model of the LDM is needed to the final users for getting a proper understanding of “what is” the LDM, where the Application Programming Interface is practically needed to “make the proper usage” of it. SAFESPOT modules access the LDM through this specific and predefined interface. The LDM API is divided into two parts: the level 1 functions, providing flexible, generic access to the LDM through low-level operations and the level 2 functions, designed to achieve a specialised access to the database.

The present annex of the D7.3.1 deliverable is released in a public form in order:

1. to describe the concept and the content of the LDM database;
2. to enable and to promote the usage of the LDM externally to the SAFESPOT IP;
3. to start the standardization activities (towards the IEEE, ETSI or other possible standardization bodies);
4. to ensure a proper level of public information is available outside of the SAFESPOT domain, in order for the original contributors (to the definition, specification and implementation of the SAFESPOT LDM) are properly acknowledged for their work on the subject.
1. Introduction

The local dynamic map, which is constructed on top of the digital map database, is considered one of the essential elements of future safety oriented cooperative systems in general, and of the SAFESPOT system in particular. In the LDM, the road geometry from a standard digital map is integrated with the information collected by vehicles and the infrastructure. It provides a real-time mapping of relevant static, temporary and dynamic infrastructure and non-infrastructure elements and objects around the system that is maintaining its consistency. The LDM is a highly dynamic data store with a relation to the road network. It enables storage and updating of objects including type, position and other characteristics, and retrieval of selected information for further processing and situation analysis, like calculation of trajectories, and detection of hazardous obstacles and potential conflicts with other road users. If the object that maintains the LDM is moving, the map window is moving as well, with the object as its center point.

1.1. Document structure

Chapter 2 gives some preliminary explanations and descriptions about the LDM, introducing the four conceptual layers composing it; chapter 3 details the object model and the structure of the LDM, with reference to the last version of the LDM implementation, as built in the SAFESPOT project (release 10.0.12 of the LDM data model). Chapter 4 contains information needed for the practical usage from the users perspective of the LDM, detailing the API adopted for the access, the reading and the updating of its content. Chapter 5 is included to make explicit the underlying intellectual property in terms of background and foreground knowledge provided by the companies who defined conceptually, designed and implemented the LDM in the SAFESPOT project. Chapter 6 contains some conclusions related to the material contained in the present document.

2. Overview of the LDM

The LDM can be considered as an innovative database where to represent, as four different conceptual layers, from bottom to top:

- layer 1 - the static (and preferably ADAS enhanced) map database
- layer 2 - additional static information not present in the standard map database
- layer 3 - temporary and dynamic information (e.g. weather and traffic conditions)
- layer 4 - dynamic and highly dynamic objects

The bottom layer is the static map as used today in navigation systems, possibly with enhanced geometry and attributes (ADAS specification). The second layer is not very different from the bottom layer: it contains static information (mainly attributes) that is not yet contained in the standard map database, but may be in the future. The third layer is for temporary and dynamic information, for instance the information related to traffic and
weather. The fourth and top layer provides a mapping of highly dynamic information, especially concerning autonomous objects that are able to move (motor vehicles, but also pedestrians, bicycles and animals), dynamic communication nodes (coinciding with certain road users, initially especially motorised ones) and other relevant non-permanent objects, in the vicinity of the vehicle, based on information from the vehicle's own sensors and on information sensed by other vehicles and the infrastructure, and transmitted via the cooperative communication channels.

Like the second and third layers, the top layer is also related to the static map in the bottom layer that forms the basis representation. Static communication nodes (infrastructure related) are represented in layer 2, and in the future may be represented in layer 1. Layers 2, 3 and 4 are currently represented as look aside tables with pointers to the static map database in layer 1. Pointers are always kept in a symbolic form (link-ID). Map database links (or edges, polylines) are linear objects that represent the network and are connected at nodes (point objects). The term static with respect to the map database in this context refers to information that is relatively permanent and thereby may be a standard content element in the map database. Other relevant types of static information may not yet have acquired the status of standard map database content element, and thereby need representation in layer 2. Most of the temporary and all dynamic information that is stored in layers 3 and 4 will by their nature never become part of the static map database. Some relatively long-term temporary information, for instance concerning road works, may in the future be incorporated temporarily in the static map database once incremental updating is commercially available [1]. The above Figure 1, depicting the four layer view of the LDM, was first presented in a presentation at the SAFESPOT project kick-off meeting.


3. Object Model and Structure of the LDM

The object model of the LDM has a hierarchical structure using associations between classes to describe their relationships. Due to the computing and bandwidth limitations in the SAFESPOT platforms, a fully object-oriented implementation was not deemed to be feasible. The overhead of transferring objects over the network, maintaining synchronicity between the objects on the various platforms as well as the relatively low number of object-specific “methods” to be provided by the LDM (with the prominent exception of spatial methods) all contributed to this conclusion. Consequently, the LDM uses a more traditional, relational database approach and only implements a sub-set of the specified classes – the concrete classes.

From the users’ point of view, concrete classes can be seen as separate tables in the database, with the attributes as columns and each new realization of an object as a row.

Two different implementations are available linked to the two map providers of static maps, Tele Atlas (PG-LDM) and NAVTEQ (NAVTEQ-LDM). The implementations are described in the following subchapters.

3.1. PG-LDM

In SAFESPOT the PG-LDM is based on a PostgreSQL database using the PostGIS extension suitable for spatial static and dynamic data handling. Level 1 and Level 2 API in C++ are provided for data access and lookup of geospatial data as well as geospatial situation analysis. For the PG-LDM implementation the remote clients can access the corresponding sources via the Ethernet network. Clients as well as servers are supported in C++ under Windows and Linux.

PostgreSQL, an open source relational database system available for multiple operating systems like Windows and Linux is used as the datastore layer of the PG-LDM. It is SQL compliant and offers a stable platform for developing databases. It has obtained a high level of acceptance and a large number of users world-wide. It claims to have high reliability, scalability and speed.

Importantly, the PostGIS add-on provides support for geographic objects following the OpenGIS Simple Feature Specification. PostgreSQL-PostGIS provides an open source spatial database with support for geographic data objects as well as spatial functionality. Considering the spatial context of (almost) all objects in the LDM, this built-in spatial support is seen to be attractive.
Supporting cross-project activities to CVIS the PG-LDM has been further developed with a slightly different configuration. In CVIS, the PG-LDM bundle is an extension of the Tele Atlas Automotive Research Platform which provides further CVIS related functions for AGORA-C, map matching and routing. For all platforms Tele Atlas provides the map data files for PostgreSQL databases and a non-standard PSF.

The PG-LDM is implemented by Bosch and Tele Atlas.
The delivered SAFESPOT version of the PG-LDM-API bundle in C++ contains the following components:

- **example_code.zip** – contains a Visual Express project with a short example for using “intersects”, “buffer” and notification
- **PG-LDM-API_1_9_10_documentation.zip** – API specification in html, double click on “index.html”
- **ldm.ini** – an example for the ini-file, needed to setup a database connection, see API-documentation for “DBConn”
- **CW_R1_9_10.zip** – contains the include files and the compiled libraries for Windows
- **CL_R1_9_10.zip** – contains the include files and the compiled libraries for Linux
- **SF_SP3_Guidelines_for_PG-LDM_v1.5.doc** – an installation manual for setting up the environment for PG-LDM

For the Java PG-LDM-API, the following components are provided to CVIS as part of the collaboration with SAFESPOT:

- **code_examples** – provided on the CVIS portal Wiki page
- **html documentation** – created using javadoc
- **LDM data model** – list of the table structure
- **Java OSGI PG-LDM bundle** – as jar-file containing the PG-LDM-API and application

PostgreSQL provides administration monitoring support via pgAdminIII. pgAdminIII is an open source administration and development platform for PostgreSQL.

As visualization and analysis software, OpenJump has been selected for the PG-LDM. OpenJump is developed within an umbrella project called the JUMP Pilot Project and distributed under the GPL license. It is an open source GIS (geo-information system) software implemented in Java based on JUMP GIS by Vivid Solutions. Due to the usage of Java, OpenJump should work on any operating system that runs Java 1.5 or later. For SAFESPOT that means OpenJump can be used on both Windows and Linux platforms.

Within SAFESPOT there will be only a very small subset of OpenJump functionalities and features in use, mainly as a demo viewer and data analysing tool. The most used functionalities are:

- **Import database (LDM) content**, especially the spatial data for visualisation (see Figure 4)
- **Advanced attribute query** which can highlight the result objects
- **Spatial queries are supported**, e.g. intersection of two objects, buffer, convexhull
- **Zoom** to an user defined map scale
- **Enhanced set of selection tools**
- **Save database (LDM) content** in different standard format, e.g. GML (Geography Markup Language), WKT (Well Known Text), ESRI shapefile
3.2. NAVTEQ-LDM

As with the PG-LDM, the NAVTEQ LDM client delivery is a single static library file (ldmapi.lib for Windows XP or ldmapi.a for Linux) and associated header files that define the API classes and methods.

However, the Windows based LDM Server is implemented as a component (referred to as a plug-in) of the NAVTEQ ADAS Research Platform (ADASRP). This application platform provides the server with access to both a static map database (in the MMF physical storage format) and an SQLite dynamically updateable embedded database. The LDM Server provides a single interface for accessing data from either of these two data sources, and for writing data to the SQLite database.

The Linux based LDM Server is a standalone service that provides this same LDM functionality.

Both implementations communicate equally with either the Windows or Linux client library.
SQLite requires no installation work. Configuring an initial LDM database is done by running a simple command line script to construct the schema and/or populate some initial test site data.

Prebuilt SQLite LDM data may also be delivered as a single database file and copied into an existing installation and copied to other server installations if required. Neither the SQLite database or the MMF map database contain the full LDM data tables and attributes. Only when accessed via the LDM Server are all attributes visible. Numerous free third party tools are available for inspecting the contents of the SQLite portion of the NAVTEQ LDM database.

As well as being used by standalone applications, the Windows LDM Client library may be linked to any software component developed for use with the ADASRP in order to add LDM functionality to those applications.

3.3. Data Model Description

The data model description gives an overview of the table structure, with a detailed description of the concrete classes.

The attributes are precisely defined, including:

- a description,
- whether it is static,
- whether it has a separate timestamp associated to it,
- its specified data type (e.g. integer, floating point, character string, boolean, geometry) and
- its range of values, default values and physical unit.

From the data model, clients can identify where data is stored in the LDM, thus enabling the use of the level 1 API (see Chapter 4 for details on the subject). In the following sections most of the content is related to the release 10.0.12 of the LDM data model, which is the final release produced in the SAFESPOT project.
3.3.1. Staticfeatures

The static objects represent the information in layers 1 and 2 of the LDM. This concerns information that is today in the static map database (layer 1), and information that has a similar character, but is not (or not yet) part of the static map database (layer 2). Over time the specification of the static map database evolves, and some information that is today in layer 2 may in the future be incorporated in the static map database. In the model of the LDM the parent class of all static objects is Staticfeatures. This term is chosen as it is used in GDF, where the “feature” is defined as "the database representation of a real world object". In the LDM model the class Staticfeatures is an abstract subclass of the class WorldObject.

<table>
<thead>
<tr>
<th>Table 1 - LDM Concrete Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Staticfeatures</td>
</tr>
<tr>
<td>arealandmarks</td>
</tr>
<tr>
<td>crossing</td>
</tr>
<tr>
<td>crossingreferencetracks</td>
</tr>
<tr>
<td>crossingregion</td>
</tr>
<tr>
<td>detectionarea</td>
</tr>
<tr>
<td>detectionareaforcrossing</td>
</tr>
<tr>
<td>juction</td>
</tr>
<tr>
<td>linelandmarks</td>
</tr>
<tr>
<td>pointlandmarks</td>
</tr>
<tr>
<td>reference</td>
</tr>
<tr>
<td>roadelement</td>
</tr>
<tr>
<td>roadintersection</td>
</tr>
<tr>
<td>sensor</td>
</tr>
<tr>
<td>sensorfordetectionarea</td>
</tr>
<tr>
<td>signalgroup</td>
</tr>
<tr>
<td>signalgroupreference</td>
</tr>
<tr>
<td>traffic</td>
</tr>
<tr>
<td>trafficsign</td>
</tr>
<tr>
<td>egorp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Figure 6 – colour labeling in the LDM tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes from v9.9.1 marked in blue in worksheets</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static</th>
<th>Time stamped</th>
<th>Database</th>
<th>Data type</th>
<th>Max</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>arealandmarks</td>
<td>id, name</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1000</td>
<td>9080</td>
<td></td>
<td></td>
<td>Spatial areal object modelled by a single polygon</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Following the GDF specification all numbers above 9000 are user defined codes</td>
<td></td>
</tr>
<tr>
<td>road_element_id</td>
<td>id</td>
<td>x</td>
<td>0</td>
<td>9999</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The road element this object refers to</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>In case that the road element is not related to an arealandmark the corresponding road_element_id is set to -1 in case that there is more than one road element related, the value is null, which means not applicable</td>
<td></td>
</tr>
<tr>
<td>egorp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rsu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 7 – arealandmarks: Spatial area object modelled by a single polygon

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static</th>
<th>Timeliness</th>
<th>Data Source</th>
<th>Range</th>
<th>Default Value</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>crossing_id</td>
<td>x</td>
<td>i</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>Spatial area object modelled by a single polygon, representing a crossing of a road element with another transport mode, especially pedestrian, bicycle and railway.</td>
<td></td>
</tr>
<tr>
<td>featuretype</td>
<td>x</td>
<td>i</td>
<td>1000</td>
<td>0000</td>
<td></td>
<td></td>
<td>Feature Type</td>
<td></td>
</tr>
<tr>
<td>crossing</td>
<td>x</td>
<td>i</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>The crossing types defined for the LDM.</td>
<td></td>
</tr>
<tr>
<td>lane_mask</td>
<td>x</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Includes which lanes are crossed. Reference: situation 1 in lane_mask description. “5”=bridge, “7”=underpass.</td>
<td></td>
</tr>
<tr>
<td>road_element</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2541</td>
<td></td>
<td></td>
<td>The road being crossed by this object.</td>
<td></td>
</tr>
<tr>
<td>geom</td>
<td>x</td>
<td>g</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>Polygon geometry.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8 – crossing: Spatial area object modelled by a single polygon, representing a crossing of a road element with another transport mode, especially pedestrian, bicycle and railway

Figure 9 – crossingforreferencetracks: Static relationship between a crossing and a reference track

Figure 10 – crossingsignal group: Static relationship between a signal group and a crossing

Figure 11 – detectionarea: Spatial area object modelled by a single polygon, representing the coverage area of a sensor
Figure 12 – detectionareaforcrossing: Static relationship between a detection area and a crossing

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static</th>
<th>Timestampable</th>
<th>Range</th>
<th>Default Value</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>detectionareaforcrossing</td>
<td></td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td>Static relationship between a detection area and a crossing. Note: Can be filled either manually or via spatial queries.</td>
<td></td>
</tr>
<tr>
<td>da_id</td>
<td></td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td>Detection area ID</td>
<td></td>
</tr>
<tr>
<td>crossing_id</td>
<td></td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td>Crossing ID</td>
<td></td>
</tr>
</tbody>
</table>

Figure 13 – junction: Spatial point object modelled by a single point, representing a map database node that bounds a road element or a ferry connection

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static</th>
<th>Timestampable</th>
<th>Range</th>
<th>Default Value</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>junction</td>
<td></td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td>Spatial point object modelled by a single point, representing a map database node that bounds a road element or a ferry connection. Note: A node is defined [ISO 2004] as a zero-dimensional element that is a topological junction of two or more edges, or an end point of an edge.</td>
<td></td>
</tr>
<tr>
<td>nettype</td>
<td></td>
<td>x</td>
<td>i</td>
<td>1000</td>
<td>0</td>
<td>Feature Type</td>
<td></td>
</tr>
<tr>
<td>gsf_level2_trp</td>
<td></td>
<td>x</td>
<td>i</td>
<td>1000</td>
<td>0</td>
<td>Feature Type</td>
<td></td>
</tr>
<tr>
<td>gsf_level2_id</td>
<td></td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td>Feature identification of the gsf_level2 feature</td>
<td></td>
</tr>
<tr>
<td>geom</td>
<td></td>
<td>x</td>
<td>g</td>
<td>-</td>
<td>-</td>
<td>Point Geometry</td>
<td></td>
</tr>
</tbody>
</table>

Figure 13 – junction: Spatial point object modelled by a single point, representing a map database node that bounds a road element or a ferry connection

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static</th>
<th>Timestampable</th>
<th>Range</th>
<th>Default Value</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>linemarkings</td>
<td></td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td>Spatial linear object modelled by a single polyline. Note: Especially used to represent linear objects that are used for positioning and detection, like road markings or edges of road objects (e.g. centre line) or 3D objects (e.g. building outline).</td>
<td></td>
</tr>
<tr>
<td>nettype</td>
<td></td>
<td>x</td>
<td>i</td>
<td>1000</td>
<td>0</td>
<td>T104 Building Facade, 9962 Cut, 9952 Barriers (roadway in Saarland), T251 Barrier, 9210 Railway element single level (terminals in Saarland), T254 Road marking (painted on the road), interrupted line with long bar (default), 9693 Railway element dual level, colored above, 9692 Viaduct, 9693 Viaduct, 9694 Building Facade, 9622 Screen, 9659 Signboard, 9695 Road marking - solid line, 9696 Road marking - solid line, 9693 Road marking - combination of solid and dotted line, 9692 Road marking - interrupted line with short bars, 9693 Road marking - stop line.</td>
<td></td>
</tr>
<tr>
<td>road_element</td>
<td></td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td>The road element (this object is contained in the road_element)</td>
<td></td>
</tr>
<tr>
<td>lane_mark</td>
<td></td>
<td>x</td>
<td>i</td>
<td></td>
<td></td>
<td>Indicates the location of the lane inside the road’s lane.</td>
<td></td>
</tr>
<tr>
<td>geom</td>
<td></td>
<td>x</td>
<td>g</td>
<td>-</td>
<td>-</td>
<td>Unspecified</td>
<td></td>
</tr>
</tbody>
</table>
### Figure 14 – linelandmarks: Spatial linear object modelled by a single polyline

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static</th>
<th>Number Data Type</th>
<th>Min</th>
<th>Max</th>
<th>Default Value</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointlandmark</td>
<td>id</td>
<td>x</td>
<td>Integer</td>
<td>0</td>
<td>2</td>
<td>2^32-1</td>
<td>Point landmarks are painted signs, poles or other physical spots which are used for positioning and detection purposes.</td>
<td></td>
</tr>
<tr>
<td>height</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>road_element_id</td>
<td>-</td>
<td>i</td>
<td>Integer</td>
<td>0</td>
<td>2</td>
<td>2^32-1</td>
<td>Road element ID identification that is close to the pointlandmark.</td>
<td></td>
</tr>
<tr>
<td>geom</td>
<td>-</td>
<td>g</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Point geometry.</td>
<td></td>
</tr>
</tbody>
</table>

### Figure 15 – pointlandmarks: Point landmarks are painted signs, poles or other physical spots which are used for positioning and detection purposes

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static</th>
<th>Number Data Type</th>
<th>Min</th>
<th>Max</th>
<th>Default Value</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>referenceTrack</td>
<td>id</td>
<td>x</td>
<td>Integer</td>
<td>0</td>
<td>2</td>
<td>2^32-1</td>
<td>Spatial linear object modelled by a single polyline representing a reference manoeuvre at an intersection from one road element connected to the intersection and a specific lane on that road element to another road element connected to the intersection and a specific lane on that road element.</td>
<td></td>
</tr>
<tr>
<td>f_lane_mask</td>
<td>-</td>
<td>s</td>
<td>Integer</td>
<td>5</td>
<td></td>
<td></td>
<td>Indicates the lane at the start of the reference track.</td>
<td></td>
</tr>
<tr>
<td>t_lane_mask</td>
<td>-</td>
<td>s</td>
<td>Integer</td>
<td>5</td>
<td></td>
<td></td>
<td>Indicates the lane at the end of the reference track.</td>
<td></td>
</tr>
<tr>
<td>f_road_element</td>
<td>-</td>
<td>i</td>
<td>Integer</td>
<td>0</td>
<td>2</td>
<td>2^32-1</td>
<td>Road element at the start of the reference track.</td>
<td></td>
</tr>
<tr>
<td>t_road_element</td>
<td>-</td>
<td>i</td>
<td>Integer</td>
<td>0</td>
<td>2</td>
<td>2^32-1</td>
<td>Road element at the end of the reference track.</td>
<td></td>
</tr>
<tr>
<td>pdf_reftrack_id</td>
<td>-</td>
<td>i</td>
<td>Integer</td>
<td>1008</td>
<td>5659</td>
<td>9999</td>
<td>Reference track identifier of the reference track.</td>
<td></td>
</tr>
<tr>
<td>geom</td>
<td>-</td>
<td>g</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Reference geometry.</td>
<td></td>
</tr>
</tbody>
</table>

### Figure 16 – referencetracks: Spatial linear object modelled by a single polyline, representing a reference manoeuvre at an intersection from one road element connected to the intersection and a specific lane on that road element to another road element connected to the intersection and a specific lane on that road element

The referenceTrack table describes the reference track in more detail, including the type of reference track, the reference point, and the geometry of the reference track.
Figure 17 – roadelement: Spatial linear object modelled by a single polyline, representing a map database edge that represents an elementary section of the road network.
Figure 18 – roadintersection: Crossing and/or connection of two or more roads

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Data Type</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>roadintersection</td>
<td>id</td>
<td>x</td>
<td>0-100</td>
<td>1</td>
<td></td>
<td>Spatial road object modelled by a single point representing a sensor, a device which detects or measures a physical property</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>x</td>
<td>1</td>
<td>0-5</td>
<td></td>
<td></td>
<td>Official name: road intersection</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>z</td>
<td>9999</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>road</td>
<td>x</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td>State of line: Position of the sensor relative to a transportable element</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>x</td>
<td>0</td>
<td>9999</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 19 – sensor: Spatial point object modelled by a single point representing a sensor, a device which detects or measures a physical property

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Data Type</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensor</td>
<td>id</td>
<td>x</td>
<td>0-100</td>
<td>1</td>
<td></td>
<td>Spatial road object modelled by a single point representing a sensor, a device which detects or measures a physical property</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>x</td>
<td>1</td>
<td>0-5</td>
<td></td>
<td></td>
<td>Official name: sensor</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>z</td>
<td>9999</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>x</td>
<td>0</td>
<td>9999</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>geometry</td>
<td>x</td>
<td>0</td>
<td>9999</td>
<td></td>
<td></td>
<td>Polygon geometry</td>
<td></td>
</tr>
</tbody>
</table>

Figure 20 – sensorfordetectionarea: Static relationship between a sensor and a detection area

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Data Type</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensorfordetectionarea</td>
<td>id</td>
<td>x</td>
<td>0-100</td>
<td>1</td>
<td></td>
<td>Static relationship between a sensor and a detection area</td>
<td></td>
</tr>
<tr>
<td>sensor</td>
<td>x</td>
<td>1</td>
<td>0-100</td>
<td></td>
<td></td>
<td>Official name of the sensor</td>
<td></td>
</tr>
<tr>
<td>detectionarea</td>
<td>x</td>
<td>1</td>
<td>0-100</td>
<td></td>
<td></td>
<td>Official name of the detection area</td>
<td></td>
</tr>
</tbody>
</table>

Figure 21 – signalgroup: A logical object possessing a single state (see signalgroupstate) which is displayed on one or more traffic light heads (see signalgroupstate)
point landmarks) and applying to one or more lanes of a single road element entering a junction

Figure 22 – signalgroupforreferencetrack: Static relationship between a single signal group and one or more reference tracks

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static Distribution</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>signalgroup</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td></td>
<td>Static relationship between a single signal group and one or more reference tracks</td>
<td></td>
</tr>
<tr>
<td>referenceelements</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td></td>
<td>Static relationship between a signal group and one or more reference tracks</td>
<td></td>
</tr>
</tbody>
</table>

Figure 23 – trafficsign: Spatial point object modelled by a single point representing a traffic sign or a variable message sign

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static Distribution</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>trafficsign</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td></td>
<td>Spatial point object modelled by a single point representing a traffic sign or a variable message sign</td>
<td></td>
</tr>
<tr>
<td>note</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td></td>
<td>Notes A traffic sign may be defined ISO 20020 as a board containing symbols and (variable) some additional, expressing a traffic restriction, recommendation or information. A dynamic traffic sign is also called a variable message sign. This table only represents the traffic sign. Within the information the content of the traffic sign is static, but can be used for identifying the traffic sign by an external identifier.</td>
<td></td>
</tr>
<tr>
<td>lane_offset</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td></td>
<td>Offset relative to the starting point along the road element.</td>
<td></td>
</tr>
<tr>
<td>dynamic</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>1</td>
<td></td>
<td>indicator of a variable message sign – in the case further display units generate or values can be found in the dynamic traffic sign and dynamic_traffic-sign-preliminary table</td>
<td></td>
</tr>
<tr>
<td>sid</td>
<td>x</td>
<td>i</td>
<td>1</td>
<td>5</td>
<td></td>
<td>Side of line. Position of the Traffic Sign relative to a transportation element:</td>
<td></td>
</tr>
<tr>
<td>left</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2</td>
<td></td>
<td>1: Left.</td>
<td></td>
</tr>
<tr>
<td>right</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2</td>
<td></td>
<td>2: Left &amp; Right.</td>
<td></td>
</tr>
<tr>
<td>above</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2</td>
<td></td>
<td>3: Above.</td>
<td></td>
</tr>
<tr>
<td>lane_number</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td></td>
<td>Indicates the applicable lane(s) of the traffic sign. E.g., 0 indicates a staggered entrance or exit.</td>
<td></td>
</tr>
<tr>
<td>number</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>2^32-1</td>
<td></td>
<td>Maximum traffic sign identification of the real traffic sign. Sign number is available.</td>
<td></td>
</tr>
<tr>
<td>location</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>-</td>
<td></td>
<td>Point geometry.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 24 – trafficsigninformation: Information content of a traffic sign, and relationship between this information and a traffic sign object

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Static Distribution</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>trafficsigninformation</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>38</td>
<td></td>
<td>Information content of a traffic sign, and relationship between this information and a traffic sign object. The traffic sign and the sign information table need to be evaluated together.</td>
<td></td>
</tr>
<tr>
<td>sequencenumber</td>
<td>x</td>
<td>i</td>
<td>1</td>
<td>38</td>
<td></td>
<td>Sequence number.</td>
<td></td>
</tr>
<tr>
<td>location</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>254</td>
<td></td>
<td>Other traffic sign location.</td>
<td></td>
</tr>
<tr>
<td>d_domain</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>254</td>
<td></td>
<td>Domain domain, unit 5 g string. Refer to traffic sign. Information content of the real traffic sign.</td>
<td></td>
</tr>
<tr>
<td>syntax</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>38</td>
<td></td>
<td>Vehicle code that the traffic sign information is assigned to. Refer to vehicle type.</td>
<td></td>
</tr>
<tr>
<td>vehicle_type</td>
<td>x</td>
<td>i</td>
<td>0</td>
<td>160</td>
<td></td>
<td>Vehicle code that the traffic sign information is assigned to. Refer to vehicle type.</td>
<td></td>
</tr>
</tbody>
</table>
3.3.2. Movingobjects

The class Movingobjects is a subclass of DynamicObject, and is the basic class to represent any road user. Four subclasses are defined: EgoMotorVehicle, MotorVehicle, UnidentifiedObject and Trailer. Each moving object is characterised by one MotionState and it may have one or more Trajectory objects. Typically the moving objects represent the information in layer 4 of the LDM.
<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Range</th>
<th>Output Value</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>egomotorvehicle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 27** – egomotorvehicle: An ego motor vehicle unit. Mutually exclusive to the ego RSU; i.e. either "egorsu" table is present in an LDM (SP2) or "egomotorvehicle" table is present (SP1), but not both. Only one entry in this table. The id of the egomotorvehicle is always 1.
<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Max</th>
<th>Min</th>
<th>Default Value</th>
<th>Physical</th>
<th>Description</th>
</tr>
</thead>
</table>

**Notes**

- Deliverable N. D7.3.1 – ANNEX 2
- Dissemination Level PU
- Copyright SAFESPOT
- Contract N. IST-4-026963-IP

---

**SF_D7.3.1_Annex2_LDM_API_and_Usage_Reference_v0.7.doc**

---

Subproject: SP7 - SCORE
Figure 28 – motorvehicle: A non-ego motor vehicle unit. The value start at 2 as the 1 is reserved for the id of the egomotorvehicle

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Attributes</th>
<th>Range</th>
<th>Definition/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uo</td>
<td>id</td>
<td>x 1</td>
<td>1,255.1</td>
</tr>
<tr>
<td></td>
<td>timestamp</td>
<td>x 0</td>
<td>0 - 999</td>
</tr>
<tr>
<td></td>
<td>latлат</td>
<td>f 0</td>
<td>0 - 360°</td>
</tr>
<tr>
<td></td>
<td>lonlng</td>
<td>f 0</td>
<td>0 - 360°</td>
</tr>
<tr>
<td></td>
<td>heading</td>
<td>f 0</td>
<td>0 - 360°</td>
</tr>
<tr>
<td></td>
<td>speed</td>
<td>f 0</td>
<td>0 - 360°</td>
</tr>
<tr>
<td></td>
<td>yaw</td>
<td>f 0</td>
<td>0 - 360°</td>
</tr>
</tbody>
</table>

- **id**: Unique identifier for any vehicle, includes vehicle, pedestrian on a vehicle, pedestrian, other VRU objects etc. Subtypes in a non-ego motor vehicle contain the id of the motorvehicle's egomotorvehicle. Note: for SDFSPOT, these types of objects might have a VANET, hence all attributes have to be derived remotely.

- **timestamp**: Seconds since 1970/1/1T00:00:00

- **latлат**: Estimated covariance of latitude and longitude

- **lonlng**: Estimated covariance of latitude and longitude

- **heading**: Estimated covariance of heading and speed

- **speed**: Estimated covariance of speed and yaw

- **yaw**: Estimated covariance of speed and yaw

Figure 29 – uo: Unidentified (or partially identified) object. Includes VRU, other "living" objects etc. Attribute list essentially consists of the motorvehicle’s dynamics attributes (describing its motion) and feattyp descriptor. In SAFESPOT, these types of objects will not have a VANET, hence all attributes have to be derived remotely.
Figure 30 – trailer: A trailer is connected to a tractor/truck/passenger car. It is not self-propelling. Hence its dynamics are governed by its hauling vehicle and its own characteristics

### 3.3.3. Conceptualobjects

The class Conceptualobjects is a subclass of DynamicObject, and is the basic class to represent any traffic and environmental events. Seventeen subclasses are defined: AccidentHotSpot, DynamicBlackSpot, DynamicReferencetrackAttributes, DynamicRoadelementAttribute, DynamicSensorAttributes, DynamicSensorStatus, DynamicTrafficSign, DynamicTrafficSignInformation, EnvironmentalEvent, FcdEvent, MeteoDetection, RoadConditionEvent, RoadConditionMeasurement, SignalGroupState, TrafficEvent, TrafficObject, DynamicReferenceTrack. Typically the conceptual objects represent the information in the layer 3 of the LDM.
Figure 31 – accidenthotspot: Describes the results of statistical evaluation of historical accident data

Figure 32 – dynamicblackspot: Describes a spot where a risk of an incident (accident) is given. This information is dynamic

Figure 33 – dynamicreferencetrackattributes: Dynamic data provided associated with the static reference track

Figure 34 – dynamicroadelementattribute: Dynamic attributes that can be associated to a road element - in case the information should be provided per lane or per vehicle type several entries for a road element can be specified
Figure 35 – `dynamicsensorattributes`: Dynamic data provided associated with the sensor

Figure 36 – `dynamicsensorstatus`: Dynamic Sensor Status information

Figure 37 – `dynamictrafficsign`: This feature represents a moving traffic sign (used for road works for example) or a sub traffic sign display on a VMS

Figure 38 – `dynamictrafficsigninformation`: Identification of the dynamic traffic sign the dynamic traffic sign information belongs to
### Table 1: Table Name

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Attributes</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>environmental_event</code></td>
<td>id</td>
<td>x</td>
<td>1</td>
<td>2193-1</td>
<td>NA</td>
</tr>
<tr>
<td><code>timelapse_seconds</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>timelapse_dayseconds</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>valid_from</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>valid_to</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sql_data</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 39 – `environmental_event`: This table provides measure whether information which are given in general by weatherstations

#### Table 2: Table Name

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Attributes</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fcdevent</code></td>
<td><code>id</code></td>
<td>x</td>
<td>1</td>
<td>2193-1</td>
<td>NA</td>
</tr>
<tr>
<td><code>timelapse_seconds</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>timelapse_dayseconds</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>valid_from</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>valid_to</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sql_data</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 40 – `fcdevent`: Floating Car Data events
## Table Name: meteodetection

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp_seconds</td>
<td>≤ 3600s</td>
<td>sec</td>
<td></td>
<td>Detected weather events</td>
</tr>
<tr>
<td>timestamp_minutes</td>
<td>≤ 3600s</td>
<td>min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>visibility_range</td>
<td>≤ 255</td>
<td></td>
<td>235</td>
<td>The confidence level of the detected event. The value is in range 0-100. 0 means unknown and 100 full confidence. In addition there are two statistic type values: 994 for not updated, 255 for invalid.</td>
</tr>
<tr>
<td>weather</td>
<td>≤ 255</td>
<td></td>
<td>235</td>
<td>TIDP: rain, fog, smoke, dust, snow, sleet, snow, snow spray (strong wind), dust, hail, snow.</td>
</tr>
<tr>
<td>snow</td>
<td>≤ 255</td>
<td></td>
<td>235</td>
<td>TIDP: rain, fog, smoke, dust, snow, sleet, snow, snow spray (strong wind), dust, hail, snow.</td>
</tr>
<tr>
<td>temperature</td>
<td>≤ 4500</td>
<td></td>
<td>225</td>
<td>The confidence level of the detected event. The value is in range 0-4500. 0 means unknown and 100 full confidence. In addition there are two statistic type values: 994 for not updated, 255 for invalid.</td>
</tr>
</tbody>
</table>

## Table Name: roadconditionmeasurement

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp_seconds</td>
<td>≤ 3600s</td>
<td>sec</td>
<td></td>
<td>Second since 1970/1/1</td>
</tr>
<tr>
<td>timestamp_minutes</td>
<td>≤ 3600s</td>
<td>min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>valid</td>
<td>≤ 3600s</td>
<td></td>
<td>235</td>
<td>The confidence level of the detected event. The value is in range 0-100. 0 means unknown and 100 full confidence. In addition there are two statistic type values: 994 for not updated, 255 for invalid.</td>
</tr>
<tr>
<td>roadcondevt</td>
<td>≤ 3600s</td>
<td></td>
<td>235</td>
<td>The confidence level of the detected event. The value is in range 0-100. 0 means unknown and 100 full confidence. In addition there are two statistic type values: 994 for not updated, 255 for invalid.</td>
</tr>
</tbody>
</table>

## Table Name: roadconditionevent

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Range</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp_seconds</td>
<td>≤ 3600s</td>
<td>sec</td>
<td></td>
<td>Second since 1970/1/1</td>
</tr>
<tr>
<td>timestamp_minutes</td>
<td>≤ 3600s</td>
<td>min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>valid</td>
<td>≤ 3600s</td>
<td></td>
<td>235</td>
<td>The confidence level of the detected event. The value is in range 0-100. 0 means unknown and 100 full confidence. In addition there are two statistic type values: 994 for not updated, 255 for invalid.</td>
</tr>
<tr>
<td>event</td>
<td>≤ 3600s</td>
<td></td>
<td>235</td>
<td>Refer to the attributes prior to the roadcondition event and to the attribute valid of this document.</td>
</tr>
<tr>
<td>confidence</td>
<td>≤ 100</td>
<td></td>
<td>50</td>
<td>Probability of the event.</td>
</tr>
<tr>
<td>roadcondevt</td>
<td>≤ 3600s</td>
<td></td>
<td>235</td>
<td>The confidence level of the detected event. The value is in range 0-100. 0 means unknown and 100 full confidence. In addition there are two statistic type values: 994 for not updated, 255 for invalid.</td>
</tr>
</tbody>
</table>

### Figure 41 – meteodetection: Detected weather events

### Figure 42 – roadconditionmeasurement: Measurement of the Road Condition (currently used only for the snow thickness)

### Figure 43 – roadconditionevent: Detected road conditions as defined in the roadconditionevent
Figure 44 – signalgroupstate: Contains the dynamic attributes of a (static) signalgroup.

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Min.</th>
<th>Max.</th>
<th>Default Value</th>
<th>Physical Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signalgroupstate</td>
<td>id</td>
<td>1</td>
<td>252-1</td>
<td>NA</td>
<td>-</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>timestamp</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Fractional component of timestamp.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>signalgroup</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Actual value of Signal Group.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>color</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>red_color</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>green_color</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>yellow_color</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>amber_color</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>other_color</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>red_alternate</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>green_alternate</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>yellow_alternate</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>amber_alternate</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
<tr>
<td>signalgroupstate</td>
<td>other_alternate</td>
<td>0</td>
<td>999</td>
<td>-</td>
<td>minutes</td>
<td>Contains the dynamic attributes of a (static) signalgroup.</td>
</tr>
</tbody>
</table>

Figure 45 – trafficevent: Externally generated message coming from safetycentre.
### 3.3.4. Relationships

The class `Relationship` is used to instantiate specific relationships among other objects (dynamic or static) in the `World Object`. Three subclasses are defined: `AlongRoadElement`, `ConceptualAlongRoadElement` and `Trajectory`.

#### Table: `AlongRoadElement`

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Range</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>id</code></td>
<td></td>
<td></td>
<td></td>
<td>Identifier of the relationship. This is mainly for consistency (i.e. all tables have an &quot;id&quot;) and the logic for these tables is identical. For example, if the relationship is an &quot;along-road-element&quot; relationship, the map matched position of the road element (road object) should be in the road element table.</td>
</tr>
<tr>
<td><code>timeStamp</code></td>
<td><code>double</code></td>
<td>8</td>
<td>1,315,214.964</td>
<td>Levels of service for a road element. For example, if the relationship is an &quot;along-road-element&quot; relationship, the map matched position of the road element (road object) should be in the road element table.</td>
</tr>
<tr>
<td><code>timeStamp_milliseconds</code></td>
<td><code>long</code></td>
<td>8</td>
<td>0</td>
<td>Fractional component of time.</td>
</tr>
<tr>
<td><code>confidence</code></td>
<td><code>double</code></td>
<td>1</td>
<td>1</td>
<td>Level of confidence in the map matched position being the correct one for the specified moving object.</td>
</tr>
</tbody>
</table>

#### Table: `ConceptualAlongRoadElement`

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Range</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>id</code></td>
<td></td>
<td></td>
<td></td>
<td>Identifier of the relationship. This is mainly for consistency (i.e. all tables have an &quot;id&quot;) and the logic for these tables is identical. For example, if the relationship is an &quot;along-road-element&quot; relationship, the map matched position of the road element (road object) should be in the road element table.</td>
</tr>
<tr>
<td><code>timeStamp</code></td>
<td><code>double</code></td>
<td>8</td>
<td>1,315,214.964</td>
<td>Levels of service for a road element. For example, if the relationship is an &quot;along-road-element&quot; relationship, the map matched position of the road element (road object) should be in the road element table.</td>
</tr>
<tr>
<td><code>timeStamp_milliseconds</code></td>
<td><code>long</code></td>
<td>8</td>
<td>0</td>
<td>Fractional component of time.</td>
</tr>
<tr>
<td><code>confidence</code></td>
<td><code>double</code></td>
<td>1</td>
<td>1</td>
<td>Level of confidence in the map matched position being the correct one for the specified moving object.</td>
</tr>
</tbody>
</table>

#### Table: `Trajectory`

<table>
<thead>
<tr>
<th>Table name</th>
<th>Attributes</th>
<th>Range</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>id</code></td>
<td></td>
<td></td>
<td></td>
<td>Identifier of the relationship. This is mainly for consistency (i.e. all tables have an &quot;id&quot;) and the logic for these tables is identical. For example, if the relationship is an &quot;along-road-element&quot; relationship, the map matched position of the road element (road object) should be in the road element table.</td>
</tr>
<tr>
<td><code>timeStamp</code></td>
<td><code>double</code></td>
<td>8</td>
<td>1,315,214.964</td>
<td>Levels of service for a road element. For example, if the relationship is an &quot;along-road-element&quot; relationship, the map matched position of the road element (road object) should be in the road element table.</td>
</tr>
<tr>
<td><code>timeStamp_milliseconds</code></td>
<td><code>long</code></td>
<td>8</td>
<td>0</td>
<td>Fractional component of time.</td>
</tr>
<tr>
<td><code>confidence</code></td>
<td><code>double</code></td>
<td>1</td>
<td>1</td>
<td>Level of confidence in the map matched position being the correct one for the specified moving object.</td>
</tr>
</tbody>
</table>

### Figure 46 – `trafficobject`: Traffic state information for a road element

### Figure 47 – `dynamicreferencetrack`: recommended path when the road becomes part of a black-spot

### Figure 48 – `alongroadelement`: Map matched position of a motor vehicle or other moving object. Explicit storage of moving object-road relationship. The `id` field is the identifier of the given relationship. This is mainly for consistency (i.e. that all tables have an "id" field). Queries concerning this table will often/mostly/always be given through the vehicle and/or road element ids.
Figure 49 – conceptualalongroadelement: m to n relationship between roadelements and conceptual objects. The field ID is the identifier of the given relationship. This is mainly for consistency (i.e. that all tables have an "id" field). Queries concerning this table will often/mostly/always(?) be given through the conceptualobject_id and/or road_element_id.

Figure 50 – trajectory: Contains the possible trajectories of moving objects. A single moving object can have multiple trajectories. A trajectory is the predicted future path of the moving object. This may consider the object's current dynamics (position, velocity, yaw rate etc.), the static road geometry (possibly also reference tracks) and possibly even other indicators of driver intention (turn indicator status etc.). The attributes here were derived from the LDM EAP file, as well as FuturePath struct in SRresult.h from ICCS (SP1).
3.3.5. Others

The class Others is used to instantiate other specific objects (dynamic or static) in the World Object. Two subclasses are defined: GatewayCommunicaton and MessageBox.

Figure 51 – gatewaycommunication: to ease communication between e.g. safetycentre-gateway and dynamic black spot calculation allow Datareceiver to just save content of message in form of string into this table. Using notification, interested parties can handle the data without polling.

Figure 52 – messagebox: container for rsu-hmi messages. See also SP7 Data format and messages doc -> HMI MessageFromVanet.
4. Application Programming Interface (API)

The LDM software consists of:

- libraries that are used by clients to connect to and access the server
- runnable modules (including initialisation and configuration files) that implement the LDM server
- map data files containing the static object information for the relevant test sites.

The connection class (DBConn) establishes a communication channel between the client application and the LDM Server, regardless of the physical location of the server itself. The DBConn is initialized (or created) using a configuration file with a standard INI format that provides enough information for the client library to locate the LDM Server. Given a valid configuration file, the DBConn is used to open() and close() a connection to the LDM Server and determine the schema version used by the server. A connection to a server with an incompatible schema version will not be allowed. In order to get the information which schema version can be used for the release, the function getSchemaVersion() has been provided.

Once successfully open, the DBConn object may be passed to any number of instances of the Level 1 or Level 2 APIs.

SAFESPOT modules access the LDM through a defined interface. The LDM API is divided into two parts:

- **Level 1 functions** which provide flexible, generic access to the LDM through low-level operations
- **Level 2 functions** designed to provide specialised access to the database.

4.1. Level 1 API

The Level 1 API provides a very generic and flexible API that draws on SQL syntax. It essentially maps the query directly (i.e. without syntactic or semantic checking) to an SQL statement that is run on the LDM server. Due to its simplicity, and the fact that both LDM implementations utilise an SQL DBMS, it has been relatively quick and easy to implement.

A common feature of the Level 1 API is that input and output arguments are provided as strings. This emphasises the flexible nature of these methods. The same method can be used regardless of the type or number of logical arguments the query requires. Strings arguments can be used to represent single values or lists of numeric quantities, boolean flags or character strings. From a Level 1 query’s string arguments, the corresponding SQL statement can then be formed by concatenating the input string arguments with relevant SQL keywords (possibly with basic string operations to separate list elements). Clearly, without the syntactic or semantic checking of the arguments, this approach is unable to prevent/catch incorrectly formed queries from being made by clients to the LDM server. Therefore, it must be used with care by the users’ modules.

The Level 1 API consists of methods that can be grouped into the following categories:
• data modification: insert, update, delete. These operations modify the contents of the LDM – i.e. they form the T-API.
• attribute recall: query. This generic method provides attribute values stored in the LDM (it is mapped to a SELECT SQL statement).
• spatial feature generation: e.g. intersection, geomUnion, difference, envelope, convexhull, buffer, boundary, centroid. New geometries are generated by these methods based on one or more existing or explicitly defined geometries.
• spatial predicates: e.g. intersects, contains, equals. Spatial tests on the relationships between the geometries of two objects or on the properties of a single geometry.
• spatial metering: e.g. distance, length, area. Metrics of a geometry or between two geometries.
• return value processing: next, previous, first, last, size, value. These functions do not access the LDM, rather they allow the processing of the results of the operations listed above.
• transaction handling: begin, commit, rollback. These functions allow a propose transaction handling to guarantee that all corresponding LDM tables are updated in a consistent way.

As the level 1 API is low-level, in addition to the software functions provided, users should be aware of the object model used by the LDM.

Input parameters

For simplicity and to enable flexibility, the input parameters of the above functions are generally strings. Often, an input argument may consist of a list of multiple logical quantities. In these cases, the individual elements of the list are comma separated. White space is ignored. As already stated, developers should be aware that the input arguments are not parsed to ensure correctness. In most cases, the string arguments are copied verbatim into a command string that is submitted to the LDM database server.

Common types of inputs include:
• Object type list - A comma separated list of object types (tables in the LDM) as defined in the LDM object model, e.g. "motorvehicle, environmentalevent". All Level 1 queries have an object type list as their first argument
• Attribute names - Comma separated list of attribute names (columns in an LDM table), e.g. "id, speed"
• Attribute values - Comma separated list of attribute values
• Equality and inequality conditions, =,!=,<,<=,>,>=
• Conditions - An "AND" separated list of equality or inequality clauses. Each clause should have the form <attribute name> <equality/inequality> <attribute value>
• ID conditions - A specialised form of Conditions that only allows equalities based on the "id" attribute, e.g. "#1.id=24 AND #2.id=84".

The use of hash-notation above is used to reference the ordered object type list argument. In the example above for ID conditions, "#1.id=24" refers to the object whose type (table name) is given by the first object in the object type list string and with id attribute of 24.
For example,

```plaintext
tqapi.query('car, car',
    '#1.speed,#2.speed',
    '#1. vehiclesize_height>3 AND #2.vehiclemass>1.5 ')
```

translates to the following SQL statement:

```plaintext
SELECT carone.speed, cartwo.speed
FROM car AS carone, car AS cartwo
WHERE carone. vehiclesize_height>3 AND cartwo. vehiclemass>1.5
```

There are two references to a single car table which enables the comparison of the speeds of two rows of this table in the query.

**Data modification**

The data modification Level 1 API operations insert, update/modify and delete objects from the LDM (i.e. rows in object tables). Note that LDM initialisation scripts create object tables and, hence, it is not necessary to provide access to these functions to clients.

- **insert** – To insert an object into the LDM, this method is called with the following arguments:
  1. Object type - A single object type (table name) from the LDM object model.
  2. Attribute names - Any number of valid attribute names for the object can be listed in this comma separated list.
  3. Attribute values - A comma separated list of (string representations of) the object's attribute values. The order of attribute values must be given in the same order as the attribute names.

- **update** – This method modifies the attribute values of a single or multiple objects in the LDM. Input arguments have the following form:
  1. Object type - A single object type following the LDM object model. Although multiple objects may be modified in one `query.update` call, they must all be of the same type.
  2. Attribute names - A comma separated list of attribute names that are to be modified.
  3. Attribute values - The new attribute values.
  4. Condition - A logical condition that identifies the objects whose attributes are to be modified. The simplest example is an ID condition, e.g. "id=23", however, more sophisticated conditions may be defined based on other attribute values too, e.g. "speed>29.1".

- **delete** – Deleting existing LDM objects follows the same format of `update` except that there is no need for attribute names or attribute values.
Attribute recall

The *query* method provides a very powerful and flexible way to access the attributes of LDM objects. The naming and form of this method should be intuitive to those familiar with the *SELECT* statement in SQL:

1. Object type list - A comma separated list of object types, cf. *FROM* clause in SQL. A maximum of nine object types may be listed.
2. Attribute names - A list of the attributes that are requested, cf. the *SELECT* clause in SQL.
3. Condition - The condition which identifies whose (i.e. which objects') attributes are to be returned, cf. the *WHERE* clause in SQL.

Spatial properties

The spatial nature of the LDM has been mentioned in chapter 2. The queries described here perform spatial operations on the geometries of one or more objects and return a geometry, a numeric measure or a boolean. The geometrical operations follow the Simple Feature (OGC – OpenGIS Project Document 99-049) [2]. All methods require the following strings as their first two input arguments.

1. Object types - Depending on the operation, either one or two valid object types must be given. Where two object types are required, they may be identical or different.
2. ID condition - Only simple, unambiguous identification of individual objects through the *id* attribute is permitted for these methods. Although more sophisticated conditions may be allowed by implementation of the method, it is recommended that developers use other methods to identify the objects to be operated upon, and then call the methods listed here using ID conditions. This is to aid debugging and to better ensure predictable behaviour.

Only those Simple Feature Access methods requested by clients will be provided. The most commonly used methods are briefly described here.

- **intersection** – This method calculates the spatial intersection of the geometries of two objects. Two object types must be given.
- **geomUnion** – This method calculates the spatial union of the geometries of two objects. It is identical in form to the *intersection* method.
- **difference** – The spatial difference between two geometries is calculated by this method. The geometry of the second object is removed from the geometry of the first object.
- **envelope** – The bounding box of a single object is returned.
- **convexhull** – The convex hull of a single object is returned.
- **buffer** – The geometry of a single object is expanded by the amount given in the third input argument.
- **boundary** – The boundary of a single object is returned. This is different to the original geometry if holes are present.
- **centroid** – The centroid (i.e. the point at the geometric centre) of the single object is returned.
- **distance** – The minimum distance between two objects is calculated. This numeric value is represented as a string.
• **length** – Calculates the length of an object with a LINESTRING geometry (see Simple Feature specification). A string representation of the numeric value is returned.

• **area** – Calculates the area of an object with a POLYGON geometry (see Simple Feature specification). A string representation of the numeric value is returned.

• **intersects** – This function tests to see if two geometries have a non-empty intersection area (cf. *intersection*). The boundary of the geometries are/are not included.

• **contains** – This method tests if the first listed object completely surrounds/contains the second object.

• **equals** – This spatial predicate method tests if the geometries of two objects are spatially equal. It does not test if the two geometries are defined in exactly the same manner, just that spatial relationships between the two geometries and any other geometry are equivalent.

### Output parameters

The output of a Level 1 LDM query is contained in the query object. As multiple values may be returned, they are automatically entered into an ordered list within the query object. To access these results, a number of methods are provided. Methods are available to traverse the (possibly multiple) return values, whilst another extracts a single value at the current list position:

• **next** – move to the next position in the list

• **previous** – move to the previous position in the list

• **first** – move to the first position in the list

• **last** – move to the last position in the list

• **size** – returns the size of the list

• **value** – return the value at the current list element.

### Transaction handling

In order to guarantee that the LDM content is updated, accessed and modified in a consistent way, transaction handling functions are provided. The transaction handling functions determine which functions are performed as one atomic transaction. Following actions are provided:

• **begin** – start transaction

• **rollback** – cancel function calls that have been set up after a *begin* and rollback to the status when the transaction started

• **commit** – commit (i.e. execute) all function calls which have been set up after the *begin* transaction
4.2. Level 2 API

The level 2 API consists of specialised/predefined queried. Predefined queries consider more specific application needs and terms of performance and comfort.

Whilst the Level 1 API is sufficient to allow full access to the contents of the LDM (with knowledge of the object model), it was deemed desirable to provide an additional group of functions that provided higher level access. The level 2 API methods aim to provide some of the more complicated queries used by clients. A level 2 API may be advantageous when, by exploiting its internal implementation, the LDM may be more efficient at evaluating a query than a series of level 1 queries that then need to be externally processed by the client. By implementing a series of queries internally, the LDM server may be able to reduce overall computation time and also reduce inter-component messaging (i.e. reducing LAN traffic). The cost of this is that computational load is redistributed from the clients (of which there may be many) to the LDM server. This trade-off was carefully considered when deciding which level 2 queries were to be implemented.

L2API

The Level 2 API class L2API represents an extension of the Level 1 API. It covers notification mechanism which are described more in detail in the next section and further complex functions introduced in the following:

- `getNextRoadElements` – Based on a single road element or a particular vehicle, determine the list of next road elements.
- `getPrevRoadElements` – Based on a single road element or a particular vehicle, determine the list of previous road elements.
- `getRoadElementTree` – Calculate the road element tree starting at a certain roadElement, taking a distance, the vehicle type and illegal maneuvers into account.
- `getVehiclesOnRoadElement` – Perform lookup for all vehicles on a road element, which are in front of a certain point specified by an offset. In this context, the term 'in front' depends on the direction of the road element. Attention should be paid to the last parameter 'lanesMask'.
- `getLanesForRoadElement` – Get all lanes on a certain road element with regard to the specified lane mask.
- `getRoadElement` – Determine the network road element the ego motor vehicle currently uses for travel or determine the road element for a particular motor vehicle.
- `getLanesForSignalGroup` - Get all lanes regulated by a particular traffic signal group.

The following functions are specifically implemented for SAFESPOT and will not available as part of the Java API that supports CVIS applications. These functions mainly support object refinement and data fusion components realized by sensor implementations that are not used in CVIS.
• **getLandmarks** – Return the geometry of all landmarks that can be found within a specific search radius.
• **getCurbs** – Return the geometry of all curbs that can be found within a specific search radius.
• **getOverlapProportions** – Return the proportions of overlap between a provided polygon and any occurrences of roadways, sidewalks and bicycle lanes.

**NetworkRoadElement**

In order to provide a more comfortable handling of connectivity information, the NetworkRoadElement class has been created. A NetworkRoadElement describes a road element, which is defined as an aggregate of a road element id and a direction (POSITIVE, NEGATIVE, BOTH, UNKNOWN). The NetworkRoadElement allows comfortable access to direction specific information. Following functions are defined:

• **getId** – Return the ID of this network road element.
• **getDirectionOfTravel** – Return the direction of this network road element.
• **getLanes** – Return the lanes of this network road element for the direction of travel.
• **isIllegal** – Returns whether this network road element describes an illegal move.
• **getSlope** – Get the slope of this network road element.
• **getCurvature** – Get the curvature of this network road element in 1/m.
• **getNearestPoint** – Get the nearest point for a specified point location.
• **getLongitudinalDistance** – Get the longitudinal distance for a point location.
• **getSpeedLimit** – Get speed limit for current position specified by a longitudinal distance.
• **getNextIntersection** – Get the next intersection this network road element encounters.
• **getRoad** – Return the road which this network road element belongs to.
• **getRoadElementsUntilNextIntersection** – Return the network road element leading to the next intersection.

**Road**

The Road is a complex representation of the road network comprising several road elements and junctions. This class is implemented especially to support intersection safety application where a high level representation (in GDF complex) is required.

Following functions have been defined:

• **getNetworkRoadElements** – Get all network road elements belonging to this road.
• **getId** – Return the ID of this road.
RoadElementTree

The RoadElementTree class represents all relevant road elements and connectivity information in terms of levels of a certain road network the vehicle might access on short term. The RoadElementTree supports the retrieval of further road element related information relevant especially for safety purposes.

Following functions have been defined:

- `getRootRoadElement` – Return the root element of this road element tree. For a tree, there is exactly one root element, which is on level 0.
- `getChildren` – Return the children of a certain element.
- `getRoadElementsOnLevel` – Return all road elements of a certain level of the road element tree.

Lane

In SAFESPOT Lanes are modeled accordingly to the NextMAP specification, which can be accessed using the following URL: http://www.ertico.com/download/nextmap_documents/2_d23.zip, p. 8ff

The lane class provides a more comfortable access to lane specific information.

Following functions have been defined:

- `getNetworkRoadElement` – Get the network road element this lane belongs to.
- `getType` – Get the lane type of this lane. Lane types defines lane specific information (emergency lane, overtaking etc.). The defined lane type values are defined in the constant class.
- `getWidth` – Get the lane width of this lane. The default lane width is assumed to be 3.50 meters.
- `getGeometry` – Get the geometry of this lane. This function is currently not implemented and under discussion as it might not be needed.
- `getDirectionOfTrafficFlow` – Get the direction of traffic flow of this lane, regarding to the specified vehicle types.
- `getDirectionCategory` – Get direction category of this lane. The direction category is defined in the NextMAP specification.
- `getLaneDividerTypeRight` - Get the divider type on the right side of the lane. The divider type describes whether it's a physical (like a wall) or legal barrier (like a marking on the road).
- `getLaneDividerTypeLeft` - Get the divider type on the left side of the lane. The divider type describes whether it’s a physical (like a wall) or legal barrier (like a marking on the road).
- `getLaneDividerMarkerRight` - Get the divider marker on the right side of the lane. The defined lane divider marker values are defined in the constant class.
- `getLaneDividerMarkerLeft` - Get the divider marker on the left side of the lane. The defined lane divider marker values are defined in the constant class.
Intersection

The Intersection is a complex representation of the road network comprising several road elements and junctions. This class is implemented especially to support intersection safety application where a high level representation (in GDF complex) is required. For an intersection information can as well derived using the Level 1 API in line with the defined data model on intersection.

Following functions have been defined:

- **getId** – Return the ID of this intersection.
- **getJunctions** – Get junctions belonging to this intersection.
- **getApproachingNetworkRoadElements** – Get all NetworkRoadElements approaching this intersection.
- **getDepartingNetworkRoadElements** – Get all NetworkRoadElements departing from this intersection.
- **getSignalGroupsIds** – Get all signal group ids belonging to this intersection.

Junction

The junction is a simple node of the road network where road element attributes change. In comparison with an intersection a junction can have as well less than 3 adjacent road elements.

Following functions have been defined:

- **getId** – Return the ID of this junction.
- **getApproachingLanes** – Get all Lanes approaching this junction.
- **getDepartingLanes** – Get all Lanes departing this junction.

Vehicle

The vehicle is the representation of a vehicle

Following functions have been defined:

- **getId** – Return the ID of this vehicle.
- **getVehicleType** – Return the type of the vehicle.

Notification Mechanism

A notification mechanism is available in the LDM. This allows clients to ask to be notified when a particular condition in the LDM is fulfilled. This fundamentally differs from the other API functions mentioned above, as results are “pushed” from the LDM to the clients, rather than being “pulled”.

Due to the finite resources available on the SAFESPOT platforms, care must be taken to ensure appropriate use of this feature. That is, there must be a positive overall effect on the system from using the notification mechanism. Of course this is difficult to assess since it involves trade-offs between computing resources on the clients’ and LDM server’s computers, LAN bandwidth and the timing requirements of applications. In general, the view was taken that computational load is preferred on the clients’ computers than on the LDM server’s – this is due to the fact that there is only one LDM to service the...
whole platform, whereas (in most SAFESPOT platforms) there are multiple PCs hosting all of the LDM clients (e.g. separate VANET, Laserscanner, Positioning, Application PCs).

In light of this, the notification conditions should ideally be:

- simple to assess – thus not imposing too high a load on the LDM server, and
- relatively rare – thus unsuitable for the alternative approach of regular polling (using the level 1 or 2 API).

Under these conditions, using the notification mechanism is very useful and natural when implementing applications. Consider the case of an accident hotspot warning application. Under the assumption that accident hotspot “objects” occur relatively rarely, it would be inefficient for the application to poll the LDM every cycle to see if a new hotspot object has been detected. Therefore, the application could “subscribe” or “listen” to the condition that a new hotspot object has been detected. This type of condition is relatively easy for the LDM to test and, in fact, is supported by many DBMS. Therefore, the load on the LDM server is minimal/negligible. In the meantime, the application can wait (without consuming CPU time). Only once the condition is fulfilled and the LDM server sends a notification to the application does the application act (for example, to see if the new accident hotspot object is on the predicted/potential path of the ego-vehicle).

In SAFESPOT the notification mechanisms are implemented in different ways for different programming languages. In the following implementation examples are described. Please look at the implementation specific documentation to get further details on which mechanism is supported.

The subscribe() method of L2API is used to define which object in the LDM and which operation on that object is to be monitored. An optional conditional clause further refines the definition of the triggering event by allowing a numeric range to be associated with a numerical attribute. The notification is sent only when the conditions of the subscription are met. A handler is passed to this method which implements the LDMNotify interface. Subscription also allows the client application to provide its own data pointer which will be preserved and passed to the LDMNotify::LDMCallBack function when the specified event occurs.

Any object implementing the LDMNotify interface may subscribe to event notification. Only the LDMCallBack method requires a mandatory implementation. The LDMListenerShutDown is optional.

After subscribing to an event, the LDMCallBack method will be called asynchronously (in a new thread) with information about the event that occurred.
4.3. API documentation

The API documentation is based on a set of html documents, created by means of the on-line documentation tool doxygen 1.5.6. In the present chapter, only the class names and the class member function names are reported, with the purpose of giving a preliminary understanding of the API documentation content. The full documentation, whose on-line usage requires only the availability of a browser, is included as addendum to the present annex.

Figure 53 – LDM-API classes, structs, unions and interfaces, with a brief related description

- a -

- add() : LDM::TQAPI
- addRoadElementsToTree() : LDM::L2API
- addSibling() : LDM::RoadElementTree
- area() : LDM::TQAPI

Figure 54 – LDM-API class member functions, (1 of 22)
- b -

- begin(): `LDM::TQAPI`
- boundary(): `LDM::TQAPI`
- buffer(): `LDM::TQAPI`

Figure 55 – LDM-API class member functions, (2 of 22)

- c -

- calculateVt(): `LDM::L2API`
- centroid(): `LDM::TQAPI`
- checkObjectName(): `LDM::TQAPI`
- close(): `LDM::DBConn`
- commit(): `LDM::TQAPI`
- contains(): `LDM::TQAPI`
- convexhull(): `LDM::TQAPI`

Figure 56 – LDM-API class member functions, (3 of 22)

- d -

- DBConn(): `LDM::DBConn`
- difference(): `LDM::TQAPI`
- distance(): `LDM::TQAPI`
- double2String(): `LDM::L2API`

Figure 57 – LDM-API class member functions, (4 of 22)
- e -
  
  - envelope() : LDM::TQAPI
  - equals() : LDM::TQAPI
  - execQuery() : LDM::TQAPI
  - execTransaction() : LDM::TQAPI

  Figure 58 – LDM-API class member functions, (5 of 22)

- f -
  
  - first() : LDM::TQAPI

  Figure 59 – LDM-API class member functions, (6 of 21)

- g -
  
  - geomUnion() : LDM::TQAPI
  - getApproachingLanes() : LDM::Junction
  - getApproachingNetworkRoadElements() : LDM::Intersection
  - getBoolean() : LDM::TQAPI
  - getChains() : LDM::RoadElementTree
  - getConnection() : LDM::DBConn
  - getCurbs() : LDM::L2API
  - getCurvature() : LDM::NetworkRoadElement
  - getDepartingLanes() : LDM::Junction
  - getDepartingNetworkRoadElements() : LDM::Intersection
  - getDirectionCategory() : LDM::Lane
  - getDirectionOfTrafficFlow() : LDM::Lane
  - getDirectionOfTravel() : LDM::NetworkRoadElement
  - getDistanceFromRoot() : LDM::RoadElementTree
  - getDouble() : LDM::TQAPI
  - getFD() : LDM::RoadElementTransition
  - getFID() : LDM::TQAPI
  - getFMeters() : LDM::RoadElementTransition
  - getRoadElementId() : LDM::RoadElementTransition
  - getGeometry() : LDM::Lane
  - getID() : LDM::Intersection
  - getId() : LDM::Junction, LDM::NetworkRoadElement, LDM::NetworkVehicle, LDM::Road
  - getLn16() : LDM::TQAPI
  - getLn32() : LDM::TQAPI
  - getLn64() : LDM::TQAPI

  Figure 60 – LDM-API class member functions, (7 of 22)
- g -

- getIntersection() : LDM::L2API, LDM::NetworkRoadElement
- getFunctions() : LDM::Intersection
- getLandmarks() : LDM::L2API
- getLandmarkType() : LDM::Landmark, LDM::LinestripLandmark, LDM::PointLandmark, LDM::PolygonLandmark
- getLaneDividerMarkerLeft() : LDM::Lane
- getLaneDividerMarkerRight() : LDM::Lane
- getLaneDividerTypeLeft() : LDM::Lane
- getLaneDividerTypeRight() : LDM::Lane
- getLaneMask() : LDM::Lane
- getLanes() : LDM::NetworkRoadElement
- getLanesForRoadElement() : LDM::L2API
- getLastError() : LDM::TQAPI
- getLongitudinalDistance() : LDM::NetworkRoadElement
- getLongitudinalOffset() : LDM::NetworkVehicle
- getNearestPoint() : LDM::NetworkRoadElement
- getNextRoadElement() : LDM::Lane
- getNextRoadElements() : LDM::Road
- getNextRoadElement() : LDM::NetworkRoadElement
- getNextRoadElement() : LDM::NetworkRoadElement
- getNextRoadElements() : LDM::L2API
- getOverlapProportion() : LDM::L2API
- getPrevRoadElements() : LDM::L2API
- getRoad() : LDM::NetworkRoadElement
- getRoadElement() : LDM::L2API
- getRoadElementFrom() : LDM::RoadElementTransition
- getRoadElementID() : LDM::NetworkVehicle
- getRoadElementMap() : LDM::RoadElementTree
- getRoadElementsOnLevel() : LDM::RoadElementTree
- getRoadElementsUntilNextIntersection() : LDM::NetworkRoadElement
- getRoadElementTo() : LDM::RoadElementTransition
- getRoadElementTransitions() : LDM::L2API
- getRoadElementTree() : LDM::L2API
- getRootRoadElement() : LDM::RoadElementTree
- getRuleId() : LDM::L2API
- getSchemaVersion() : LDM::DBCconn
- getSignalGroupIds() : LDM::Intersection
- getSlope() : LDM::NetworkRoadElement
- getSpeedLimit() : LDM::NetworkRoadElement
- getString() : LDM::TQAPI
- getSubscriptionId() : LDM::L2API
- getTDL() : LDM::RoadElementTransition
- getTMeets() : LDM::RoadElementTransition
- getTMeet() : LDM::RoadElementTransition
- getType() : LDM::TQAPI, LDM::Lane
- getUTMZone() : LDM::TQAPI
- getVehicleOnRoadElement() : LDM::L2API
- getVehicleType() : LDM::NetworkVehicle
- getWidth() : LDM::Lane
- getx() : LDM::Point2D
- gety() : LDM::Point2D
Figure 61 – LDM-API class member functions, (8 of 22)

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Namespaces</th>
<th>Classes</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class List</td>
<td>Class Hierarchy</td>
<td>Class Members</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>Functions</td>
<td>Variables</td>
<td>Typedefs</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

- i -

- Intersection() : LDM::Intersection
- intersection() : LDM::TQAPI
- intersects() : LDM::TQAPI
- isillegal() : LDM::NetworkRoadElement
- isLeftLane() : LDM::Lane
- isRightLane() : LDM::Lane
- isValid() : LDM::Intersection, LDM::Lane, LDM::NetworkVehicle, LDM::NetworkRoadElement, LDM::Road, LDM::Junction, LDM::DBConn

Figure 62 – LDM-API class member functions, (9 of 22)

| a | b | c | d | e | f | g | i | j | l | n | o | p | q | r | s | t | u | v | w | ~ |

- j -

- Junction() : LDM::Junction

Figure 63 – LDM-API class member functions, (10 of 22)

| a | b | c | d | e | f | g | i | j | l | n | o | p | q | r | s | t | u | v | w | ~ |

- l -

- L2AP() : LDM::L2API
- Lane() : LDM::Lane
- last() : LDM::TQAPI
- length() : LDM::TQAPI
- Linestrip2D() : LDM::Linestrip2D
- LinestripLandmark() : LDM::LinestripLandmark
- listen() : LDM::L2API
- logError() : LDM::Logging, LDM::TQAPI

Figure 64 – LDM-API class member functions, (11 of 22)
Figure 65 – LDM-API class member functions, (12 of 22)

- n -
  - NetworkRoadElement() : LDM::NetworkRoadElement
  - NetworkVehicle() : LDM::NetworkVehicle
  - next() : LDM::TQAPI

Figure 66 – LDM-API class member functions, (13 of 22)

- o -
  - open() : LDM::DBCConn
  - operator1() : LDM::NetworkRoadElement , LDM::NetworkVehicle
  - operator*() : LDM::Point2D
  - operator+=() : LDM::Point2D
  - operator-=() : LDM::Point2D
  - operator/=() : LDM::Point2D
  - operator<() : LDM::NetworkRoadElement , LDM::NetworkVehicle
  - operator<=(()) : LDM::NetworkVehicle , LDM::NetworkRoadElement
  - operator>() : LDM::NetworkVehicle , LDM::NetworkRoadElement

Figure 67 – LDM-API class member functions, (14 of 22)

- p -
  - Point2D() : LDM::Point2D
  - Point2WKT() : LDM::L2API
  - PointLandmark() : LDM::PointLandmark
  - Polygon2D() : LDM::Polygon2D
  - PolygonLandmark() : LDM::PolygonLandmark
  - previous() : LDM::TQAPI
- q -

- query() : LDM::TQAPI

Figure 68 – LDM-API class member functions, (15 of 22)

- r -

- remove() : LDM::TQAPI
- resetConn() : LDM::DBConn
- resetCurrentResultTuple() : LDM::TQAPI
- resolveTableName() : LDM::L2API
- Road() : LDM::Road
- RoadElementTransition() : LDM::RoadElementTransition
- RoadElementTree() : LDM::RoadElementTree
- rollback() : LDM::TQAPI

Figure 69 – LDM-API class member functions, (16 of 22)

- s -

- set() : LDM::Point2D
- setDB() : LDM::TQAPI
- setDistanceFromRoot() : LDM::RoadElementTree
- setInit() : LDM::DBConn
- setValid() : LDM::NetworkRoadElement, LDM::Road, LDM::NetworkVehicle, LDM::Intersection, LDM::Lane, LDM::Junction
- setX() : LDM::Point2D
- setY() : LDM::Point2D
- size() : LDM::TQAPI
- subscribe() : LDM::L2API

Figure 70 – LDM-API class member functions, (17 of 22)
- t -
  - toggle(): \texttt{LDM::L2API}
  - TQAPI(): \texttt{LDM::TQAPI}

  \textbf{Figure 71 – LDM-API class member functions, (18 of 22)}

- u -
  - unsubscribe(): \texttt{LDM::L2API}
  - update(): \texttt{LDM::TQAPI}

  \textbf{Figure 72 – LDM-API class member functions, (19 of 22)}

- v -
  - value(): \texttt{LDM::TQAPI}

  \textbf{Figure 73 – LDM-API class member functions, (20 of 22)}

- w -
  - WKT2Point(): \texttt{LDM::L2API}

  \textbf{Figure 74 – LDM-API class member functions, (21 of 22)}
Figure 75 – LDM-API class member functions, (22 of 22)
5. Status of known Intellectual Property Rights (IPR)

The project partners undertook an effort to check for relevant IPR concerning the technology described in this Deliverable.

The content of the present document, including its two addendum (SAFESPOT LDM World Object Model and SAFESPOT LDM API) is released in public form in order to provide an official and public description of the concept and the content of the SAFESPOT LDM; to enable and to promote the usage of the LDM externally to the SAFESPOT IP; to start the standardization activities (towards the IEEE, ETSI or other possible standardization bodies); to ensure a proper level of public information is available outside of the SAFESPOT domain, in order for the original contributors (to the definition, specification and implementation of the SAFESPOT LDM) are properly acknowledged for their work on the subject.

After the specific request to all SAFESPOT participants, no claims have been raised concerning IPR declared by the partners on the whole content of the present document or in its parts.

Similarly, at the time of writing the release 0.6 of this Annex2 of the deliverable D7.3.1 – Global System Reference Architecture Specifications, no specific information is known to exist including patents and patent applications on the arguments dealt with or related to the SAFESPOT LDM API and Usage.
6. Conclusions

The present document fulfills the need to describe in a public form the SAFESPOT LDM (Structure of the database, Object Model, API) from the perspective of the final users.

These users need a proper level of information related to the structure and the object model of the LDM, together with the Application Programming Interface, which is practically needed to “make the proper usage” of it.

With the release of the present document, the SAFESPOT consortium would like to enable and to promote the usage of the LDM externally to the SAFESPOT IP and to make feasible the standardization activities towards the IEEE, ETSI or other possible standardization bodies).

7. References


Addendum

- The world object model, described in chapter 3.3, is based on a MS Excel document, which is included as an additional document to the present annex (LDM data model_v10.0.12.xls), for the ease of consultation.

- The documentation of the LDM-API, described in chapter 4.3, is composed by a set of html files, generated by means of the documentation tool doxygen 1.5.6. Also these reference documents are included to the present annex as a zipped archive (PG-LDM-API-Documentation_1_7_10.zip).