VIRES Simulationstechnologie GmbH

Project:	*OpenDRIVE Manager*	Document No.	Issue:
Title:	User Manual	VI2008.029	L
	Date: July 20, 2015	no. of pages:	35
	Issuing Party:	VIRES GmbH	
	Name:	Marius Dupuis	
	Distribution List:	public	

Refer to protection notice DIN ISO 16016

Revision Control:

Issue	Date	Author	Description	Affected Chapters
L	July 20, 2015	Dupuis	adaptation to OdrMgr 1.4.3 • updated names of include files (starting with prefix "Odr" now) • introduction of geographic co-ordinates • additional methods for working with surface data • laneHeight may be ignored • co-ordinate conversion between geographic and inertial positions	all 5.2.4 / 5.2.5 7.x 6.6 6.7, 6.8
K	May 30, 2014	Dupuis	• optional consistency checks for geometry of reference line	4.2
J	January 23, 2013	Dupuis	• introduced method "loadData()"	4.1 and 4.2
I	August 08, 2012	Dupuis	• updated description of distro • inertial2laneList() can be forced to search full database • getTrackLen() needs track ID as an argument • positionObject may be printed explicity • path objects may be deleted • CRG surfaces are supported; the surface scale may be set explicitly • added (brief) version history	3.1 6.4.1 6.15 5 6.32.3 7 8
H	March 20, 2009	Dupuis	adaptation to OdrMgr 1.1.24, • improved method "intersectCircle()" • query method for path direction • improved performance of inertial2laneList() in junctions • enhanced numerical tolerance for track positions at end of track • introduced validity matrix in documentation	6.23.6 7
G	November 17, 2008	Dupuis	adaptation to OdrMgr 1.1.21, • improved method "intersectCircle()" • supports signal references in add-on files • debugged signal query method • debugged "inertial2lane()" if first road hit is not driveable • debugged lane curvature calculation	
F	October 17, 2008	Dupuis	added query methods for junction and signal controllers; laneHeight evaluation added for "inertial2lane()" and "lane2inertial()"	6.3, 6.6, 6.25, 6.27
E	August 08, 2008	Dupuis	adaptation to pre-release of OdrMgr 1.1.20, added lane curvature and derivative methods	3, 6
D	July 29, 2008	Dupuis	adaptation to OdrMgr 1.1.19, In addition, a bug in the routines for the lane curvature calculation was fixed	3, 6
C	22nd May 2008	Dupuis	adaptation to OdrMgr 1.1.18	3, 6
B	21st March 2008	Dupuis	adaptation to OdrMgr 1.1.16	3, 6

A	18th March 2008	Dupuis	creation	all

TABLE OF CONTENTS

1 Introduction

1.1 Overview

This document shall provide an easy and quick introduction into the usage of the OpenDRIVE Manager library by VIRES Simulationstechnologie GmbH. It is intended to be used by programmers and will assume that the typical technical terms of programming are known to the reader.

General aspects of OpenDRIVE, like co-ordinate systems etc., are assumed to be known to the user from the respective documentation. Only where required will further details be given in this document.

Some examples will complement this manual.

1.2 References

This manual refers to:

[1] OpenDRIVE Format Specification, Rev. 1.4E-DRAFT, February 23, 2015, VIRES Simulationstechnologie GmbH

1.3 Versions

The OpenDRIVE Manager comes in two versions:

- lite version
- full version

The lite version is intended for non-commercial and evaluation purposes. It provides a subset of the functionality of the full version and allows for the high-speed evaluation of one position object only.

2 Purpose of the OpenDRIVE Manager

The OpenDRIVE manager shall provide an easy means to read and evaluate in real-time OpenDRIVE databases. It is provided by the creators of OpenDRIVE and takes into account the latest developments and user requirements of the OpenDRIVE community.

By using the OpenDRIVE Manager, the user will, to great extent, be independent of developments of the standard itself. This means that methods provided by the OpenDRIVE Manager are intended to remain independent of the underlying standard. So, upgrading the manager in correspondence with the standard should be possible with minimum adaptation effort for existing applications.

3 The Distribution

3.1 Files

The distribution of the OpenDRIVE manager contains the following files:

```
V1.4.3lite
├── data                                          examples
│   └── sample1.1.xodr
├── doc                                           manuals
│   ├── OdrMgrUserManualL.pdf
│   └── OpenDriveFormatSpecRev1.4E-DRAFT.pdf
├── inc                                           include files
│   ├── BaseNodes
│   ├── OdrCoord.hh
│   ├── OdrGeoCoord.hh
│   ├── OdrLaneCoord.hh
│   ├── OdrManager.hh
│   ├── OdrManagerLite.hh
│   ├── OpenDRIVE.hh
│   └── OdrTrackCoord.hh
├── lib                                           libraries for various platforms
│   ├── Linux
│   │   ├── i586
│   │   │   └── libODriveLite.1.4.3.a
│   │   └── x86-64
│   │       └── libODriveLite.1.4.3.a
│   └── Windows
│       ├── VS2013-32
│       │   ├── Debug
│       │   │   └── OpenDriveLibLite.lib
│       │   ├── DynamicDebug
│       │   │   └── OpenDriveLibLite.lib
│       │   ├── DynamicRelease
│       │   │   └── OpenDriveLibLite.lib
│       │   └── Release
│       │       └── OpenDriveLibLite.lib
│       └── VS2013-64
│           ├── Debug
│           │   └── OpenDriveLibLite.lib
│           ├── DynamicDebug
│           │   └── OpenDriveLibLite.lib
│           ├── DynamicRelease
│           │   └── OpenDriveLibLite.lib
│           └── Release
│               └── OpenDriveLibLite.lib
├── README.txt                                    latest notes
└── src                                           sample implementation of the manager
    └── main.cc
```

Some special files shall be mentioned:

`inc/BaseNodes/PublicNodes.hh`	inclusions for all available nodes
`inc/OdrManagerLite.hh`	base class with lite functionality
`inc/OdrManager.hh`	derived class with full functionality

NOTE: When looking for the complete instruction set of the OpenDRIVE manager, please check the headers of the **lite** version **and** the **full** version.

3.2 Compiling and Running the Example

In order to get the sample implementation running, just compile the files in

```
src
```

and link them to

```
lib/<TargetPlatform>/libODrive<Lite>.1.4.3.a
```

Here's a sample command line to get this done (for full version):

```
g++ -m32 src/main.cc src/ParserSample.cc -o odrTest -Iinc \
    -Iinc/BaseNodes -Llib/Linux/i586 \
    -DLINUX -lODrive.1.4.3
```

For a quick test start the executable

```
odrTest data/sample.1.1.xodr
```

4 Loading a Database

4.1 Basic Operation

In order to load a database, create an instance of the OpenDRIVE Manager, e.g.:

```
OpenDrive::OdrManager myManager;
```

and load the file

```
myManager.loadFile( myFile );
```

This routine will return true if loading the file was successful.

Alternatively, you may directly provide the OpenDRIVE content as string argument to the following method:

```
myManager.loadData( myOdrData );
```

The string must be in valid OpenDRIVE syntax, i.e. it must be enclosed by the tags "`<OpenDRIVE>....</OpenDRIVE>`"

4.2 Options

Before executing the `loadFile()` or `loadData()` method, various options may be set which affect the way data is modified during the load process.

The following loader options are available:

```
myManager.setLoaderOption( myManager.LOADER_ADD_BORDERS, 10.0 );
```

This will instruct the loader to add border lanes with u<ser-defined width (here: 10m) to either side of each road, therefore creating a logical frame. The purpose of this option is to provide the user with valid evaluation and positioning data even if a position exceeds the original road network.

```
myManager.setLoaderOption( myManager.LOADER_VERBOSE, 1 );
```

This will instruct the loader to print information while processing the nodes during the loading of the database.

```
myManager.setLoaderOption( myManager.LOADER_SPLIT_JUNCTIONS, 1 );
```

This will instruct the loader to split junction paths with multiple lanes into multiple paths with one lane each. The junction matrix is adapted accordingly

```
myManager.setLoaderOption( myManager.LOADER_CHECK_GEOMETRY, 1 );
```

Internal consistency checks of the reference line's geometry will be performed. Upon detection of an inconsistency, a debug message will be printed to stderr.

4.3 Adding Files

Sometimes, the OpenDRIVE data of a database may be distributed over several files. The OpenDRIVE Manager is taking care of this in the special case where signal data is stored separately from the physical layout of the road network.

In order to add signal data from a file to the already loaded data, activate the loader option:

```
myManager.setLoaderOption( myManager.LOADER_XTEND_ADD_SIGNALS );
```

and add the file using the method

```
myManager.addFile( extraFile );
```

4.4 Contents of the Database

As it's being loaded into memory, the database is converted into a tree of objects derived from the base class

```
OpenDrive::Node
```

The names of the derived classes correspond to the respective names of the OpenDRIVE tags and will, therefore, not be described in further detail here.

The declarations of all nodes can be found in

```
inc/BaseNodes
```

The contents of a loaded database may be printed to standard output with the method:

```
myManager.printData();
```

4.5 Parsing the Database

If you need to parse the database (i.e. evaluate nodes that have been loaded, especially `userData` nodes), you may provide a parser callback method. This will allow you to extract information directly from the loaded nodes which may sometimes be necessary if you need additional information that is not provided directly by one of the OpenDRIVE Manager's query methods.

For parsing the database, provide a class derived from

```
OpenDrive::ParserCallback
```

and overload the method

```
readNode()
```

Then, having loaded the database, just instantiate your class and start parsing the data tree using the OpenDRIVE Manager's method, e.g.

```
ParserSample callback;
myManager.parseTree( &callback );
```

Your overloaded method will be called for each node in the database. A sample can be found in the directory

```
OdrMgr.1.1/
```

In order to determine the type of a given node, check for its opcode:

```
node->getOpcode()
```

You may then cast it yourself to the correct derived node class. The symbolic constants for all opcodes can be found in

```
OpenDRIVE.hh
```

5 Positioning

5.1 Position Objects

The real-time capability of the OpenDRIVE Manager is based on the usage of so-called „Position objects“. For successive real-time queries in the OpenDRIVE database, each of these objects makes use of the fact that it still “knows” the previous query and can start the new query from this base. In typical use cases, like driving along a road, successive queries will always be in the vicinity of previous queries, so that not the entire database must be scanned for a specific road object.

For maximum performance, one position object should be created for each contact point. Typically, one contact point is used for each tire of the ownship, hence requiring the user to create a total of four contact points for a standard car. Additionally, at least one contact point should be created for each other moving object.

The correct sequence for using and creating contact points in connection with the OpenDRIVE Manager methods is the following:

1) Create the required number of position objects and store each in a dedicated variable

```
OpenDrive::Position* myPos[nReqPos];

for( int i = 0; i < nReqPos; i++ )
   myPos[i] = myManager.createPosition();
```

2) Activate the corresponding position before modifying it or performing any calculations

```
myManager.activatePosition( myPos[i] );
```

Position objects don’t have to be created right at the beginning but may also be created during runtime (e.g. when additional moving objects are created).

The content of the currently active object can be displayed using the

```
print()
```

method of the manager.

A specific position object may also be printed explicitly using the method

```
print( posObject );
```

Position objects may be copied using the method

```
myManager.copyPos( dst, src );
```

The copying of a position object will also includes its query history, so that the copy shows the same high performance upon the next operation as the original object.

When talking about positions, the user should know which co-ordinate systems are available. This is described in the next chapter.

5.2 Co-ordinates

Three types of co-ordinates are available:

- Inertial co-ordinates:
  see `inc/OdrCoord.hh`
- Track co-ordinates:
  see `inc/OdrTrackCoord.hh`
- Lane co-ordinates:
  see `inc/OdrLaneCoord.hh`

Detailed information about the orientation and the co-ordinate systems in general can be found in [1].

5.2.1 Inertial Co-ordinates

Inertial co-ordinates consist of the components:

`x`	x-value [m]
`y`	y-value [m]
`z`	z-value [m]
`h`	heading [rad]
`p`	pitch [rad]
`r`	roll [rad]

They may be applied to a position object in the following ways:

1) Create a co-ordinate object and apply it to the position object

   Example:

```
OpenDrive::Coord myCoord( 1667.0, 847.0, 0.0 );
myManager.setPos( myCoord );
```

2) Apply the co-ordinates directly to the position object

   Example:

```
myManager.setInertialPos( 1667.0, 847.0, 0.0 );
```

NOTE: As explained above, don't forget to activate the appropriate position object before performing these operations:

5.2.2 Track Co-ordinates

Track co-ordinates consist of the components:

`trackId`	unique ID of the road [-]
`s`	run-length along chord line [m]
`t`	lateral distance to chord line [m]
`h`	heading relative to chord line [rad]
`p`	pitch relative to chord line [rad]
`r`	roll relative to chord line [rad]

They may be applied to a position object in the following ways:

1) Create a co-ordinate object and apply it to the position object

   Example:

```
OpenDrive::TrackCoord myCoord( 17, 5.0, 3.0 );
myManager.setPos( myCoord );
```

2) Apply the co-ordinates directly to the position object

   Example:

```
myManager.setTrackPos( 17, 5.0, 3.0 );
```

In both examples, the position will be set to road no. 17, run-length 5.0m and 3.0m lateral offset from the chord line.

5.2.3 Lane Co-ordinates

Lane co-ordinates are derived from track co-ordinates and consist of the following components:

`trackId`	unique ID of the road [-]
`laneId`	unique ID of the lane within the road [-]
`offset`	lateral offset from the lane center [m]
`s`	run-length along chord line [m]
`h`	heading relative to chord line [rad]
`p`	pitch relative to chord line [rad]
`r`	roll relative to chord line [rad]

They may be applied to a position object in the following ways:

1) Create a co-ordinate object and apply it to the position object

   Example:

```
OpenDrive::LaneCoord myCoord( 5, 1, 5.0, 0.0 );
myManager.setPos( myCoord );
```

2) Apply the co-ordinates directly to the position object

   Example:

```
myManager.setLanePos( 5, 1, 5.0, 0.0 );
```

In both examples, the position will be set to road no. 5, lane no. 1, run-length 5.0m and zero offset from the lane center.

5.2.4 Geographic Co-ordinates

Geographic co-ordinates provide information about a position on the WGS84 geoid. They are defined in the header file `OdrGeoCoord.hh`:

`long`	geographic longitude [deg]
`lat`	geographic latitude [deg]
`z`	geographic altitude [m]
`h`	heading relative to chord line [rad]
`p`	pitch relative to chord line [rad]
`r`	roll relative to chord line [rad]

Geographic co-ordinates may be applied to a position object in the following ways:

1) Create a co-ordinate object and apply it to the position object

   Example:

```
OpenDrive::GeoCoord myCoord( 12.0, 48.0, 460.0 );
myManager.setPos( myCoord );
```

2) Apply the co-ordinates directly to the position object

   Example:

```
myManager.setGeoPos( 12.0, 48.0, 460.0 );
```

In both examples, the position will be set to longitude 12deg, latitude 48deg, altitude 460m.

5.2.5 Retrieving Co-ordinates

In order to retrieve the current co-ordinate settings of the active position object, the following methods are available:

```
const TrackCoord & getTrackPos() const;
const LaneCoord & getLanePos() const;
const Coord & getInertialPos() const;
const GeoCoord & getGeoPos() const;
```

6 Queries

6.1 Overview

By means of the OpenDRIVE Manager, the user queries the database and receives the corresponding results. In order to keep computation time at a minimum, only the required information is being updated during a query. Other information may be out-dated, so the user must take care of using only the updated information.

A query sequence consists of the following steps:

1) Activate the position object
2) Set the input values (e.g. known position)
3) Trigger the computation function
4) Query the result

In the following chapters, all queries available in the current version of the OpenDRIVE Manager will be described.

6.2 Convert an inertial position into a track position

This method converts an inertial position into the corresponding track position.

Example:

```
OpenDrive::Coord myCoord( 1667.0, 847.0, 0.0 );
myManager.setPos( myCoord );

bool result = myManager.inertial2track();
```

The method will return TRUE if the track position was found.

After this computation, the following methods will provide updated information:

`getTrackPos()`	track position object
`getCurvature()`	curvature of the track's chord line

In addition, the following methods may be called:

`footPoint2inertial()`	attach the inertial position to the track surface, i.e. copy the track's elevation, heading, pitch and roll values into the inertial position
`getInertialPos()`	query the modified inertial position

6.3 Convert an inertial position into a lane position

This method converts an inertial position into the corresponding lane position.

Example:

```
OpenDrive::Coord myCoord( 1667.0, 847.0, 0.0 );
myManager.setPos( myCoord );

bool result = myManager.inertial2lane();
```

The method will return TRUE if the lane position was found.

After this computation, the following methods will provide updated information:

`getLanePos()`	lane position object
`getCurvature()`	curvature of the track's chord line
`getLaneType()`	type of the lane
`getLaneWidth()`	width of the lane

In addition, the following methods may be called:

`footPoint2inertial()`	attach the inertial position to the track surface, i.e. copy the track's elevation (including lane height), heading, pitch and roll values into the inertial position
`getInertialPos()`	query the modified inertial position

If the only a specific road shall be searched for the position, the method

```
bool myManager.inertial2lane( int trackId );
```

should be used. It will return TRUE only it the inertial position can be matched to a lane position on the given road.

6.4 Convert an inertial position into a series of lane positions

6.4.1 At the Given Location

At some locations, e.g. in junctions, one inertial position may correspond to several lane positions. In this case, the user may want to have the information about all possible positions in order to select one of the positions for further processing.

The OpenDRIVE Manager first collects all lane positions and then provides the user with a query method to retrieve them one by one.

Example:

```
OpenDrive::Coord myCoord( 1667.0, 847.0, 0.0 );
myManager.setPos( myCoord );

bool result = myManager.inertial2laneList();
```

The method will return TRUE if lane positions were found.

The resulting lane positions (of type `OpenDrive::LaneCoord`) are stored in a vector. They may be queried with an iterator of type:

```
OpenDrive::LaneCoord::LaneVec::iterator it;
```

The vector will be empty if no position has been found.

Per default, the method `inertial2laneList()` will try to minimize the searching area of potential lanes. This improves the speed considerably. In some cases, it might be necessary to force the method to look for appropriate roads within the entire road network. Then, enable the full search by calling

```
bool result = myManager.inertial2laneList( true );
```

6.4.2 At a Circle Surrounding the Given Location

Sometimes it might be necessary to know all lane positions that occur within a certain distance from a central inertial position. In this case, you may intersect the database with a circle surrounding your given inertial position.

Example:

```
OpenDrive::Coord myCoord( -60.0, 2470.0, 0.0 );
myManager.setPos( myCoord );

bool result = myManager.intersectCircle( 500.0 );
```

This intersects the database with a circle of 500m around the central position.

See the previous chapter for instructions about retrieving and interpreting the results.

6.5 Convert a track position into an inertial position

This method converts a track position into the corresponding inertial position.

Example:

```
myManager.setTrackPos( 17, 5.0, 3.0 );

bool result = myManager.track2inertial();
```

The method will return TRUE if the inertial position could be computed.

After this computation, the following methods will provide updated information:

`getInertialPos()`	inertial position object
`getCurvature()`	curvature of the track's chord line

6.6 Convert a lane position into an inertial position

This method converts a lane position into the corresponding inertial position. An existing *laneHeight* entry will be taken into account for the computation of the z co-ordinate.

Example:

```
myManager.setLanePos( 17, 1, 1.0 );

bool result = myManager.lane2inertial();
```

The method will return TRUE if the inertial position could be computed.

After this computation, the following methods will provide updated information:

`getInertialPos()`	inertial position object
`getCurvature()`	curvature of the track's chord line

Note: the consideration of the *laneHeight* may be turned off using the method

```
myManager.useLaneHeight( bool enable );
```

6.7 Convert an inertial position into a geographic position

This method converts an inertial position into the corresponding geographic position.

Example:

```
OpenDrive::Coord myCoord( 1667.0, 847.0, 0.0 );
myManager.setPos( myCoord );

bool result = myManager.inertial2geo();
```

The method will return TRUE if the position could be converted. After this computation, the following methods will provide updated information:

`getGeoPos()`	geographic position object

6.8 Convert a geographic position into an inertial position

This method converts a geographic position into the corresponding inertial position.

Example:

```
OpenDrive::GeoCoord myCoord( 12.0, 48.0, 460.0 );
myManager.setPos( myCoord );

bool result = myManager.geo2inertial();
```

The method will return TRUE if the position could be converted.

After this computation, the following methods will provide updated information:

`getInertialPos()`	inertial position object

6.9 Get the Road Material at a Location

After a successful conversion from a track or lane position into an inertial position or vice versa, you may query the road material at the given location:

Example:

```
bool gotMat = myManager.getMaterial( code, friction, roughness );
```

6.10 Get the Road Mark at a Location

After a successful query of the lane width (`myManager.getLaneWidth()`), you may query the road mark at the given location:

Example:

```
unsigned short mark = myManager.getRoadMark();
```

This method currently provides only the type of road mark. In future versions of the manager it will provide additional information stored in the RoadMark node.

6.11 Get the Junction ID at a Location

After a successful conversion from a track or lane position into an inertial position or vice versa, you may query the junction ID at the given location:

Example:

```
unsigned short jId = myManager.getJunctionId();
```

6.12 Calculate the Track Width

The method `getTrackWidth()` will calculate and return the complete width of a road at the current track position (i.e. s-position). If the position is invalid, zero width will be returned.

Example:

```
double width = myManager.getTrackWidth();
```

6.13 Calculate the Track Angles

The method `getTrackAngles()` calculates and returns the heading, pitch and roll angles of the track's chord line at the current track position (i.e. s-position). The result can be found in the H, P and R components of the returned co-ordinate container. If the position is invalid, zero angles will be returned

Example:

```
OpenDrive::Coord angles = myManager.getTrackAngles();
```

6.14 Get the Type of the Current Lane

The method `getLaneType()` returns the type of the lane that was accessed last (e.g. during conversion from inertial into lane co-ordinates). It does not perform trigger an update of the lane position and may, therefore, not be used isolated from other queries.

Example:

```
int type = myManager.getLaneType();
```

6.15 Get the Curvature within a Lane

The method `getLaneCurvature()` calculates and returns the curvature at the present lane position, taking into account, the curvature of the track's chord line, the lateral distance from the chord line, changes in the width of the own lane and any lanes between the own lane and the chord line.

Example:

```
double curvature = myManager.getLaneCurvature();
```

6.16 Get Additional Curvature Information within a Lane

After calling `getLaneCurvature()` (see above), the following additional values may be retrieved without additional computation effort:

1st derivative of lane curvature:

```
double value = myManager.getLaneCurvatureDot();
```

vertical lane curvature:

```
double value = myManager.getLaneCurvatureVert();
```

1st derivative of vertical lane curvature:

```
double value = myManager.getLaneCurvatureVertDot();
```

6.17 Get the Length of a Track

The method `getTrackLen( int trackId )` calculates and returns the total length of as specific track's chord line.

Example:

```
double length = myManager.getTrackLen( 15 );
```

6.18 Convert a Track Position into a Valid Track Position

When adding a delta s to a given track position, one may easily exceed the track´s limits on either end. If tracks connect directly to each other (i.e. no junction is involved), the OpenDRIVE Manager may proceed automatically to the successor or predecessor of the current track in order to match the given s co-ordinate.

Example:

```
myManager.setTrackPos( 5, 600.0, 3.0 );

bool retVal = myManager.track2validTrack();
```

This method will correct the track co-ordinate automatically if necessary. FALSE will only be returned if the position is not on the current track or no valid successor or predecessor was found.

6.19 Convert a Lane Position into a Valid Lane Position

Identical to 6.18 except that lane positions are queried, not track positions.
When adding a delta s to a given track position, one may easily exceed the track´s limits on either end. If tracks connect directly to each other (i.e. no junction is involved), the OpenDRIVE Manager may proceed automatically to the successor or predecessor of the current track in order to match the given s co-ordinate.

Example:

```
myManager.setLanePos( 5, 1, 600.0, 0.0 );

bool retVal = myManager.lane2validLane();
```

6.20 Calculate and Query Derivatives of Track Angles

A set of methods for calculating and querying the derivatives of the track angles is provided. The calculation has to be triggered once for a contact point. Afterwards, all query methods listed will return valid results (provided the calculation has been successful).

Example:

```
myManager.setLanePos( 5, 1, 600.0, 0.0 );

bool retVal = myManager.calcTrackAnglesDot();

double deriv = myManager.getDhDs();
deriv = myManager.getDpDs();        // pitch angle vs. s
deriv = myManager.getDrDs();        // roll angle vs. s
deriv = myManager.getDzDs();        // elevation vs. s
deriv = myManager.getD2zDs();       // deriv. of elevation vs. s
deriv = myManager.getDzDt();        // elevation vs. t
```

6.21 Calculate and Query Entire Cross-Section

The entire cross section of a road may be calculated by a single call to a manager method. Afterwards, the user may query a set of information containing the ID of each lane as well as its type and width.

Example:

```
myManager.setLanePos( 5, 1, 600.0, 0.0 );

bool retVal = myManager.calcCrossSection();

int noLanesInCrossSec = myManager.getCrossSectionSize();

for ( int i = 0; i < noLanesInCrossSec; i++ )
{
      int laneId;
      int laneType;
      double laneWidth;

      retVal = myManager.getCrossSectionLaneInfo( i, laneId,
                                          laneType, laneWidth );
}
```

6.22 Combined Query for Derivatives and CrossSection

For performance and user friendliness, a method has been introduced which comprises a set of single calls at a given location:

```
track2inertial()
calcTrackAnglesDot()
calcCrossSection()
```

The method is:

```
bool track2inertialAngDotCrossSec();
```

After successful completion of the method, the queries valid for all of the implicitly called methods may be used.

6.23 Get the Lane Speed at a Given Position

The method `getLaneSpeed()` calculates and returns the speed at the present lane position. It returns -1.0 if no speed record was found.

Example:

```
double speed = myManager.getLaneSpeed();
```

6.24 Get the Road Type at a Given Position

The method `getRoadType()` calculates and returns the road type at the present lane position. It returns ODR_ROAD_TYPE_NONE if no type record was found.

Example:

```
int type = myManager.getRoadType();
```

6.25 Working with Paths

6.25.1 Definition of Paths

Paths in the sense of the OpenDRIVE Manager are directed sequences of track positions. A path provides one progressive co-ordinate – s – for positioning. This co-ordinate starts at zero at the path's beginning and ends at the total of chord line length retrieved from the underlying tracks.

6.25.2 Creating a Path

A path may be created by providing a series of track positions. Between two consecutive positions there must not be more than one junction. Each position object can handle one path.

Example:

```
OpenDrive::Path* newPath = myManager.createPath( "myPath" );

// assign path to current position object for further actions
myManager.assignPath( newPath );
myManager.addPosToPath( OpenDrive::TrackCoord(  5, 600.0, 0.0 ) );
myManager.addPosToPath( OpenDrive::TrackCoord( 17, 345.0, 0.0 ) );
myManager.addPosToPath( OpenDrive::TrackCoord( 1, 10.0, 0.0 ) );
myManager.addPosToPath( OpenDrive::TrackCoord( 28, 23.0, 0.0 ) );
myManager.addPosToPath( OpenDrive::TrackCoord( 29, 10.0, 0.0 ) );
myManager.addPosToPath( OpenDrive::TrackCoord( 26, 74.0, 0.0 ) );
```

This creates a path ranging from track 5 / 600.0m to track 26 / 74.0m.

After this computation, the following methods will provide updated information:

`getPathLength()` total length of the path

6.25.3 Deleting a Path

Path objects may also be deleted from the manager.

Example:

```
OpenDrive::Path* newPath = myManager.createPath( "myPath" );
:
:
myManager.deletePath( newPath );
```

6.25.4 Convert a Lane Position into a Path Position

This method converts a lane position into the corresponding path position. The result is the path's progressive co-ordinate s.

Example:

```
myManager.activatePosition( myPos );
myManager.setLanePos( 5, 1, 600.0 );    // track 5, lane=1, s=600

// is this position on the path?
```

```
bool result = myManager.lane2path();

double pathS = getPathPos();
```

6.25.5 Convert an Inertial Position into a Path Position

This method converts an inertial position into the corresponding path position. The result is the path's progressive co-ordinate s.

Example:

```
myManager.activatePosition( myPos );
myManager.setInertialPos( 123.4, 567.8, 0.0 );

// is this position on the path?
bool result = myManager.inertial2path();

double pathS = getPathPos();
```

6.25.6 Navigate Along a Path

More frequently, paths may be used to navigate along them and have objects follow a path. For this, offsets may be applied to any path position and the resulting position may be converted into lane, track or inertial positions.

Example:

```
myManager.activatePosition( myPos );
myManager.setLanePos( 5, 1, 600.0 );
myManager.lane2path();

// proceed the indicated ds value from the start point
result = myManager.addPathOffset( ds, 0, 0.0 );
myManager.lane2inertial();
```

Now the resulting inertial position may be queried e.g. by calling:

```
myManager.getInertialPos();
```

The method `addPathOffset()` provides three arguments:

```
ds          delta in s direction
dLane       lane change
dLaneOff    change of the lane offset
```

6.25.7 Query the Relative Direction of Path and Road

After a successful mapping of an inertial or lane position into a path position, the direction of the underlying road (i.e. its design direction) relative to the path may be queried. The result is boolean, being true if the road's design direction and the path direction are the same.

Example:

```
bool goingForward = myManager.getPathFwd();
```

6.26 Querying Signals

Signals on a given road may be queried using the collect method of the OpenDRIVE Manager. With this method, a certain range of positions is queried for a given feature and all results are stored within the manager. They may afterwards be retrieved one by one.

Example:

```
myManager.activatePosition( myPos );
myManager.setLanePos( 5, 1, 600.0 );

// collect all signals in forward direction until the end of track
bool retVal = myManager.collectSignals();

for ( int i = 0;  i < myManager.getCollectionSize(); i++ )
{
      Node*  sigNode;
      double distToSignal;

      getCollectionInfo( i, distToSignal, sigNode );

      // do something with the signal node…
}
```

6.27 Querying Junctions

6.27.1 Retrieve the Junction Header Node

From any given valid track/lane position, the user may query the upcoming junction in a specified direction. Information about the distance to the junction, the possible paths through the junction and the respective deviation of the driving direction will be provided.

Example:

```
myManager.setLanePos( 17, 1, 1030.0, 0.0 );

OpenDrive::JuncHeader *juncHdr = 0;
double distance;

// search in forward track direction
bool retVal = myManager.getNextJunction( true, juncHdr, distance );
```

This example will return TRUE if a junction was found in positive track direction (set the first argument to "false" in order to search the opposite direction). The pointer to the JuncHeader node will be returned as well as the distance to the entry point of the junction. The junction does not necessarily need to connect directly to the track from which the query starts. Track links will be followed automatically until a junction is found or no further link exists.

6.27.2 Get the Connection Info

The information about the possibilites to proceed through the junction will be collected during the above query and can be retrieved as shown in the following example:

```
for ( int i = 0; i < myManager.getJunctionInfoSize(); i++ )
```

```
{
        OpenDrive::RoadHeader *inRoad;
        OpenDrive::RoadHeader *connRoad;
        double turnAngle;
        int    minLaneIdIncoming;
        int    maxLaneIdIncoming;

        myManager.getJunctionInfo( i, inRoad, connRoad, turnAngle,
                                   minLaneIdIncoming, maxLaneIdIncoming );
}
```

For each possible connection from the incoming road, an info entry will be available. The query will provide the pointer to the incoming road, the connecting road (path) and the turn angle between incoming and outgoing direction. This range of this angle is [-pi;pi] with positive angles indicating left turns and negative angles indicating right turns.

6.27.3 Get the Controller Info

Once the junction header node (variable `juncHdr` in the example above) is available, one may query the signal controllers which are used within the junction. The first controller can be retrieved from the junction header itself, the other controllers may be retrieved by traversing the siblings of the first controller. The following algorithm will typically be used.

```
OpenDrive::JuncController *jCtrl = juncHdr->getFirstController();

while ( jCtrl )
{
    // do something with the controller, e.g. access ID and the referenced
    // signal controller
    // fprintf( stderr, "id = %d, signal controller: ptr=0x%x, id=%d\n",
    //                  jCtrl->mId, jCtrl->mController, jCtrl->mController->mId );
    jCtrl = dynamic_cast< OpenDrive::JuncController* >( jCtrl->getRight() );
}
```

6.27.4 Get the Signal Info from the Controller

Once the junction controller node (variable `jCtrl` in the example above) is available, one may query the individual signals of the junction per signal controller. The signal controller can be retrieved as member variable `mController` of the junction controller node (see the previous example). In the following, the example has been extended for the access to the individual signals:

```
OpenDrive::JuncController *jCtrl = juncHdr->getFirstController();

while ( jCtrl )
{
    // do something with the controller, e.g. access ID and the referenced
    // signal controller
    // fprintf( stderr, "id = %d, signal controller: ptr=0x%x, id=%d\n",
    //                  jCtrl->mId, jCtrl->mController, jCtrl->mController->mId );

    if ( jCtrl->mController )
    {
        OpenDrive::ControlEntry *entry = jCtrl->mController>getFirstEntry();

        while ( entry )
        {
            // do something with the control entry, e.g. access the referenced signal
            // fprintf( stderr, "ctrlType = %d, signal: ptr=0x%x, id=%d\n",
            //                  entry->mType, entry->mSignal, entry->mSignal->mId );

            entry = dynamic_cast< OpenDrive::ControlEntry* >( entry->getRight() );
        }
```

```
        }
        jCtrl = dynamic_cast< OpenDrive::JuncController* >( jCtrl->getRight() );
    }
```

6.28 Accessing Nodes

6.28.1 Access after a Query

If a query was successful, the corresponding nodes in the memory representation of the OpenDRIVE database may be accessed. For example, this might be used in cases where the user has a given inertial position and wants access to the corresponding road header node.

Example:

```
OpenDrive::Coord myCoord( 1667.0, 847.0, 0.0 );
myManager.setPos( myCoord );

bool result = myManager.inertial2lane();

OpenDrive::RoadHeader* hdr = myManager.getRoadHeader();
```

The following node types may be queried depending on the previously executed computation:

```
OpenDrive::RoadHeader*     getRoadHeader();
OpenDrive::LaneSection*    getLaneSection();
OpenDrive::Lane*           getLane();
OpenDrive::Elevation*      getElevation();
OpenDrive::Superelevation* getSuperelevation();
```

6.28.2 Access independent of a Query

Independent of a query, nodes may be accessed starting from the root node. This may be retrieved by:

```
OpenDrive::Node* getRootNode();
```

For the navigation through the tree of database nodes, see the file

```
inc/BaseNodes/Node.hh
```

Briefly speaking, the database may be parsed by recursively calling the methods

```
node->getRight()
```

and

```
node->getChild()
```

However, there is some limitation to calling these methods. The getRight() method will only return nodes which are of the same type (opcode) as the calling node. And getChild() will only return the first child of a node. So, in order to parse all possible child node types of a node, use the following parsing scheme:

```
unsigned int index = 0;

while ( ( childNode = node->getChildAtTypeIndex( index++ ) ) )
{
    while ( childNode )
    {
        // do something, e.g. parse the children
        childNode = childNode->getRight();
    }
}
```

6.28.3 Printing Nodes

The data of a given node or the sub-tree starting at the node may be printed to console output. For this, use the method:

```
node->print();
```

Without further arguments, this method will print all children of the node, the siblings of the node and their respective children.

You may control this behaviour by providing the corresponding arguments as can be seen from the prototype of the method:

```
void print( bool deep = true, bool siblings = true );
```

6.29 Node-specific Query Methods

Some nodes provide useful methods for querying additonal information. So, once you have the pointer to one of these nodes, you may just call the respective methods

6.29.1 JuncHeader Node

The JuncHeader node provides a method to get its influence area. By calling

```
void getCtrAndRadius( double & x, double & y, double & radius );
```

one can retrieve the x/y position of the junctions geometric center as well as the radius of the circle circumscribing the junction (i.e. the collection of all of its paths).

Alternatively, the bounding box may be queried directly from the JuncHeader node in order to get a box-shaped bounding area which is aligned with the major axes (x and y).

```
Bbox * getBoundingBox();
```

In order to retrieve all controllers used in a given junction, first call the method

```
JuncController* getFirstController();
```

and then go through all controllers with the node navigation methods (i.e. node`->getRight()`) until a null node is returned.

6.29.2 JuncController Node

Each node of type JuncController provides a member variable

```
mController
```

pointing to the actual controller which is used for controlling signals in the junction.

6.29.3 Controller Node

Each node of type Controller provides a method for querying the control entries. In order to retrieve all control entries used by a given controller, first call the method

```
ControlEntry* getFirstEntry();
```

and then parse through all controllers with the node navigation methods (i.e. node`->getRight()`) until a null node is returned.

6.29.4 ControlEntry Node

Each node of type ControlEntry provides a member variable

```
mSignal
```

pointing to the actual signal which is controlled by the controller to which the control entry belongs.

6.29.5 Bounding Boxes

Some nodes carry additional information about the x/y area which they cover. This information is contained in objects of the type "Bbox" (bounding box).

Bounding boxes may be set, added to each other and queried. For detailed information, please check the header file `Bbox.hh`.

6.29.6 Signal Node

The controller of a signal may be queried directly from the signal's member variables.

`mController`	pointer to the controller
`mControlEntry`	pointer to the controller's entry handling the respective signal

7 Surface Descriptions

The OpenDRIVE Manager supports surface descriptions according to the OpenCRG standard. The correlation between the OpenDRIVE roads and the applicable OpenCRG data must be given by `<Surface>` tags within an OpenDRIVE file (see format specification)

7.1 Scaling of Surface Information

The OpenDRIVE Manager provides a global method to influence the scaling of OpenCRG data upon evaluation at a given position. The command is

```
myManager.setSurfaceScale( double factor );
```

By this, all subsequent OpenCRG information will be scaled by the given factor before being returned to the querying method.

7.2 Contact Patch

Instead of calculating the elevation at a single point within the detailed surface data, the physical averaging effect of a contact patch may be simulated. Due to performance constraints, this simulation is realized by querying and averaging only four additional locations within a given area instead of computing a "real" patch (as a tire would represent).

The patch size may be set with the method

```
myManager.setContactPatchDimension( const double & length,
const double & width );
```

`length` and `width` are given in [m].

8 Version History

The following is a very brief overview of the features within the respective versions of the OpenDRIVE Manager.

```
# version 1.4.3:  31.05.2015 - bugfixed ParamPoly3 computation for "odd" definitions
#                              (au, av, bu, bv all non-zero)
# version 1.4.2:  11.05.2015 - introduced string IDs for basic features
#                            - introduced RoadNeighbor
# version 1.4.1:  14.04.2015 - added signal attributes
#                            - added parking space and parking space markings
# version 1.4.0:  03.03.2015 - added parametric cubic polynomial
#                            - added methods for evaluating CRG data on a patch
#                            - all files now have the pre-fix "Odr", the class names have not been touched
# version 1.3.13: 05.01.2015 - on-going work...
#                            - fixed major bug in computation of first derivative of lane curvature
#                            - replaced mBigEndian with mCrgBigEndian
#                            - object type is interpreted as string variable (no longer as int)
#                            - getRoadMark() in OdrManager causes also computation of this property
#                            - added lane types for tram and rail
#                            - added queries "inTunnel" and "onBridge"
#                            - lane search may be restricted: Position::inertial2lane
#                              ( bool allowCar = true, bool allowRail = false )
#                            - added method Position::addLaneS( const double & ds )
#                            - treating spirals with very low curvature and even lower deltaCurvature as
#                              straight lines
#                            - added loader option LOADER_CHECK_GEOMETRY to enable geometry integrity tests
#                              upon loading of a database
#                            - updated to OpenCRG 1.0.6
#                            - added CornerLocal class
#                            - corrected "z" to "dz" in CornerRoad class
#                            - methods geo2inertial() and inertial2geo() introduced
#                            - fixed identification of driveable road when using registered road in
#                              inertial2lane()
#                            - fixed spiral computation, ticket #1541
#                            - introduced <repeat> bead below <object>, added "getFirstRepeat()" method to
#                              object
# version 1.3.12: 12.04.2013 - added setting of levels for nodes; new method: setLevel( level )
#                            - debugged hierarchy position of GenericNode nodes
#                            - debugged print routines for UserData and GenericNode
#                            - added RailroadSwitch as potential child of RoadHeader
#                            - fixed major bug in evaluation of friction based on CRG
#                            - releasing CRG data set upon deleting of SurfaceCRG entry
#                            - bugfixed reading of "genuine" mode in OdrManager
# version 1.3.11: 23.01.2013 - added method "loadData()", so that OpenDRIVE data can be
                               provided as string, not as file
# version 1.3.10: 28.12.2012 - debugged road mark detection
# version 1.3.9:  12.12.2012 - default friction reported as "1.0", not as "1.01"
# version 1.3.9:  04.12.2012 - OpenCRG friction no longer interpreted also as elevation
# version 1.3.7:  07.12.2012 - reading UserData child nodes as GenericNode
#                            - changed lanevalidity to validity
# version 1.3.6:  14.08.2012 - debugging OpenCRG for friction
# version 1.3.5:  12.08.2012 - duplicate reading of OpenCRG files is avoided
#                            - introduced "purpose" for OpenCRG data
#                            - bugfixed path calculation
# version 1.3.4:  28.07.2012 - bugfixed banking calculation in "inertial2lanelist"
# version 1.3.3:  13.05.2012 - VIRES internal
# version 1.3.2:  13.05.2012 - added method setSurfaceScale() for CRG data, bugfixed evaluation of CRG
# version 1.3.1:  16.04.2012 - includes "surface" tag
# version 1.1.46: 16.04.2012 - sorting lanes in lane section after they have been read
# version 1.1.45: 02.03.2012 - added computation of vertical curvature from elevation
#                            - improved GeoPoly
#                            - fixed bug in reading of unknown tags
# version 1.1.44: 09.02.2012 - corrected computation of objects' inertial heading
#                            - implemented GeoPoly
# version 1.1.43: 08.01.2012 - added red and white road mark colors as symbolic constants;
#                            - interpreting the corresponding color strings in the reader
# version 1.1.42: 04.11.2011 - copying of position objects includes deep copy of history information
# version 1.1.41: 21.11.2011 - improved performance of splitting of junctions
# version 1.1.40: 11.07.2011 - VIRES internal
# version 1.1.39: 30.06.2011 - improved check for valid road headers in inertial2lane
# version 1.1.38: 23.05.2011 - inclusion of GPS reference co-ordinate in Header
#                 03.06.2011 - added "readability" and "occlusion" to signal properties
# version 1.1.37: 25.03.2011 - performance optimization for very long databases
# version 1.1.36: 20.02.2011 - track2lane checks for lane width and does no longer return lane 0
# version 1.1.35: 10.08.2010 - debugged path calculation
#                            - addPathOffset() will automatically change lane to reach a
```

```
#                              certain point on the path if no lane change is given
#                              explicitly
# version 1.1.34: 26.04.2010 - debugged spirals (improved categorization of type2 spirals)
# version 1.1.33: 11.03.2010 - 64bit compatible
#                            - corrected computation of first derivative of horizontal curvature
# version 1.1.32: 07,03.2010 - invalidation of lanes (split junction) limited to driveable lanes
# version 1.1.31: 30.12.2009 - debugged path calculation for some odd sorts of paths
#                            - signals on cloned roads are converted into signal references
#                            - lane sequences in cloned roads are treated correctly now
# version 1.1.30: 03.12.2009 - removed memory leaks in Path calculations
#                            - debugged creation of path with initial points being on same track
# version 1.1.29: 27.11.2009 - odrMgr.getTrackLen() now returns value, not reference
#                            - odrMgr.getLaneSpeed() will determine valid lane pos if none is available
# version 1.1.28: 25.11.2009 - bounding boxes may be calculated per RoadHeader with fix width
#                            - debugged cloning of spirals
#                            - inertial2laneList of OdrMgr can explicitly be forced to
                               full search on database
# version 1.1.27: 12.10.2009 - improved path calculation for wrap-around paths
#                            - corrected computation of first derivative of lane curvature
#                            - relative heading is taken into account for operation "lane2inertial"
#                            - improved copying of path and positioning on path with minS
#                            - lane height may be given according to Odr 1.2 or 1.1 specification
#                            - debugged calculation of lane lists according to bug report
#                            - added manager method "deletePath"
# version 1.1.26: 20.07.2009 - introduced new tag (for Odr 1.3) "LaneOffset"
#                            - improved path calculation for wrap-around paths
# version 1.1.25: 17.06.2009 - corrected bug in import of roadmark weight and color
#                            - corrected import of roadmark width
#                            - debugged evaluation of lane height
#                            - signal node has new member "mState" for dynamic signals
#                            - signal node has new member "mInZ" for inertial z position
#                            - invalid signal references will no longer cause crash when collecting signals
#                            - corrected initialization of some variables
#                            - implemented upcoming options for JuncControllers in Odr 1.3
```

9 Validity Matrix

The following matrix gives an overview of the validity of queries after certain actions have been performed.

action \ query	getRootNode()	getInertialPos()	getLanePos(),(id, s, laneId, offset)	getTrackPos(),(id, s, t)	getFootPoint()	getCurvature()	getLaneList()	getTrackLen()	getTrackAngles()	getTrackWidth()	getDhDs(), getDpDs() etc.	getRoadMark()	getCrossSectionSize()	getCrossSectionLaneInfo()	getCollectionSize()	getCollectionInfo()	getMaterial()	getLaneType()	getJunctionId()	getRoadHeader()	getLaneSection()	getLane()	getElevation()	getSuperelevation()	printPath()	getPathLength()	getPathPos()	getPathFwd()	getJunctionInfoSize()	getJunctionInfo()	getRoadType()	getLaneSpeed()	getLaneCurvature()	getLaneCurvatureDot()	getLaneCurvatureVert()	getLaneCurvatureVertDot()	comment
setTrackPos()	c	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
setLanePos()	c	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	-	-	-	-	-	
setInertialPos()	c	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
inertial2track()	c	c	-	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
inertial2lane()	c	c	x	x	x	x	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	-	-	-	-	-	-	-	x	x	-	-	-	-	-	
inertial2laneList()	c	c	p	p	p	p	x	x	x	-	-	-	-	-	-	-	x	x	x	x	x	x	-	-	-	-	-	-	-	x	x	-	-	-	-	-	1
lane2inertial()	c	x	c	p	-	x	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	2
lane2track()	c	-	c	x	p	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3
track2inertial()	c	x	c	p	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	x	-	-	x	x	-	-	-	-	-	-	-	-	-	-	-	-	2
track2curvature()	c	-	-	c	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
track2lane()	c	-	x	c	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
track2validTrack()	c	-	-	x	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4
calcTrackAnglesDot()	c	-	c	c	-	-	-	-	x	-	x	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
lane2validLane()	c	-	x	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-	-	x	x	-	-	-	-	-	
calcCrossSection()	c	-	-	c	-	-	-	-	-	-	-	-	x	x	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
collectSignals()	c	-	-	c	-	-	-	-	-	-	-	-	-	-	x	x	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
footPoint2inertial()	c	x	-	-	c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
track2inertialAngDotCrossSec()	c	x	-	c	-	x	-	-	-	-	x	-	x	x	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
intersectCircle()	c	c	c	c	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
assignPath()	c	c	c	c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	-	-	-	-	-	-	-	-	-	-	
lane2Path()	c	c	c	c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	-	-	-	-	-	-	-	-	
inertial2path()	c	c	c	c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	-	-	-	-	-	-	-	-	
addPathOffset()	c	c	c	c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	-	-	-	-	-	-	-	-	
addPos2Path()	c	c	c	c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	-	-	-	-	-	-	-	-	-	-	
getLaneCurvature()	c	x	-	-	c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	5
getNextJunction()	c	c	c	c	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	-	-	-	-	-	-	5
getLaneWidth()	-	-	-	-	-	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

green:
methods in OdrManagerLite

blue:
add-ons in OdrManager

Remarks:
c = constant
- = invalid
x = valid
p = partially valid (see comment)

Comments (reference numbers, see above)

1) valid for last entry in list
2) s of trackPos is automatically limited to maximum track length
3) only getFootPoint().getH() is valid
4) getTrackPos().getH() returns track heading at new position
5) mixture between action and query

Copyright note according to DIN 34