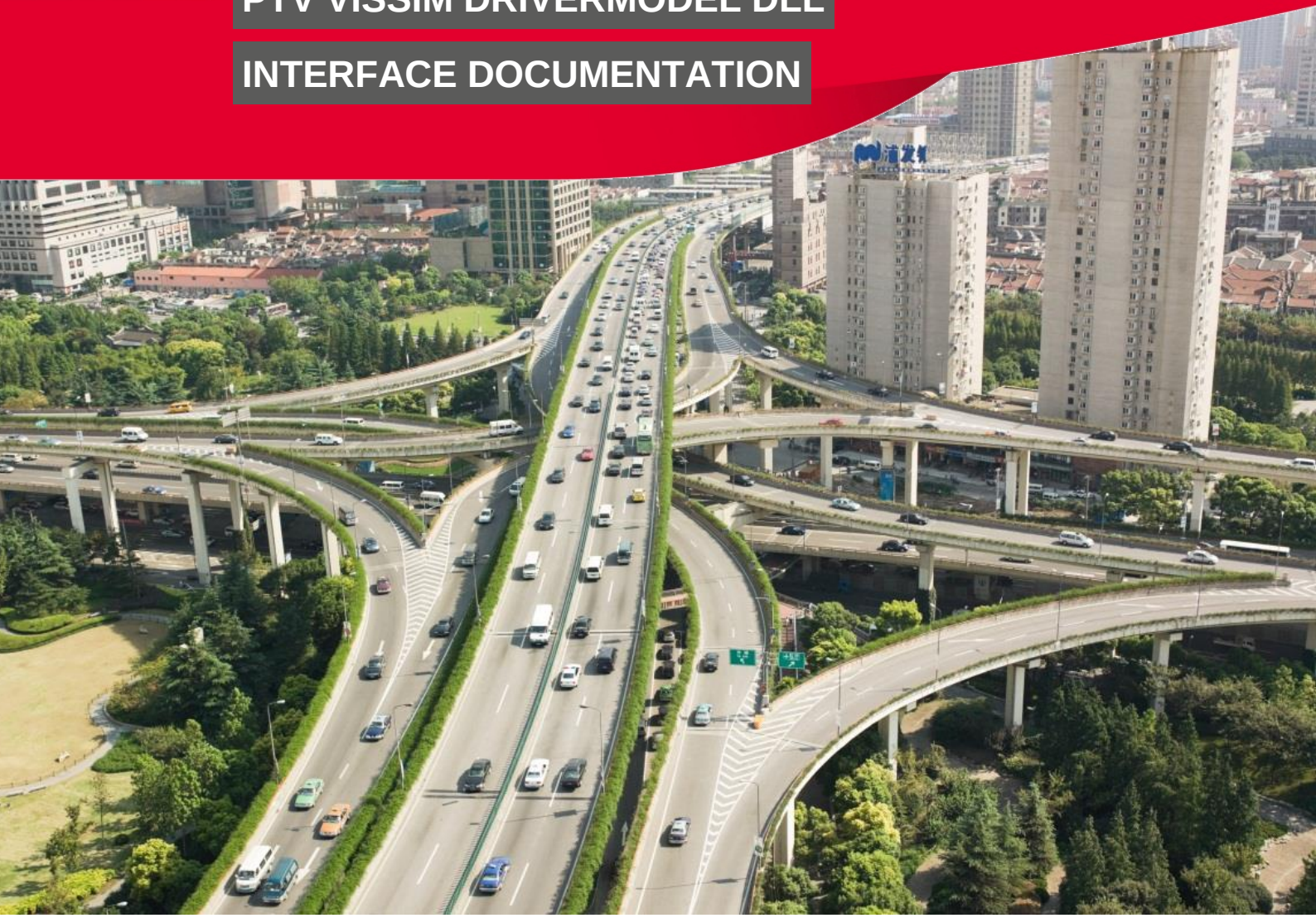
PTV GROUP

the mind of movement

PTV VISSIM DRIVERMODEL DLL INTERFACE DOCUMENTATION



Imprint

This documentation is valid for PTV Vissim version 2020.00-00 and later.
Modifications added after Vissim 11 are marked like this.
Edited by: Lukas Kautzsch

PTV Planung Transport Verkehr AG
Haid-und-Neu-Str. 15
D-76131 Karlsruhe
Germany

Phone: +49 721 9651-0
Fax: +49 721 9651-699
info@vision.ptvgroup.com
www.ptvgroup.com

Contents

1 General

The External Driver Model DLL Interface of Vissim provides the option to replace the internal driving behavior by a fully user-defined behavior for some or all vehicles in a simulation run. The user-defined algorithm must be implemented in a DLL written in C/C++ which contains specific functions (as specified below). During a simulation run, Vissim calls the DLL code for each affected vehicle in each simulation time step to determine the behavior of the vehicle. Vissim passes the current state of the vehicle and its surroundings to the DLL and the DLL computes the acceleration / deceleration of the vehicle and the lateral behavior (mainly for lane changes) and passes these values back to Vissim to be used in the current time step.

The external driver model can be activated for each vehicle type separately in the dialog box “Vehicle Type” by checking the checkbox "Use external driver model” on the tab page “External Driver Model" and selecting a driver model DLL file and optionally a parameter file to be used. If this option is checked, the driving behavior of all vehicles of this vehicle type will be calculated by the selected DLL.

A subdirectory `DriverModelData\` must exist in the directory of vissim.exe in order to avoid a warning message when a simulation run is started.

If the number of cores is not set to 1 in the Simulation Parameters of the Vissim network, the DLL needs to confirm that it supports multithreading. If it doesn’t, the simulation run is canceled with an error message.

2 DLL Interface

For the implementation of the DLL, several source code files are provided:

- DriverModel.h: Header file for a driver model DLL.
  This file should not be changed. It contains the definitions of all "type" and "number" constants used by Vissim in calls of the “DriverModel*” DLL functions which are declared here, too.
- DriverModel.cpp: Main source file of a driver model DLL.
  This file is the place where calculations or calls of functions of the driving behavior model algorithm should be added. Provided is a dummy version which does "nothing" (sets DRIVER_DATA_WANTS_SUGGESTION and DRIVER_DATA_USE_INTERNAL_MODEL). (In contrast, the file DriverModelExample.cpp contains a very simple following model which actually overrides the internal model of Vissim.)
  The preprocessor #define DRIVERMODEL_EXPORTS must be set in the compiler options for compilation of DriverModel.cpp! (This is included in the provided project file – see below.)
- DriverModel.vcproj: Visual C++ 2010 project file for a driver model DLL. This file can be used if the DLL is to be created with Microsoft Visual C++.

A driver model DLL must contain and export 3 functions which are called from Vissim: `DriverModelSetValue`, `DriverModelGetValue` and `DriverModelExecuteCommand`.
Two additional functions are optional (but required for certain data): `DriverModelSetValue3` and `DriverModelGetValue3`.

The signature of these functions and their meaning is as follows:

```
int  DriverModelSetValue (long   type,
                          long   index1,
                          long   index2,
                          long   long_value,
                          double double_value,
                          char*  string_value);
int  DriverModelSetValue3 (long   type,
                           long   index1,
                           long   index2,
                           long   index3,
                           long   long_value,
                           double double_value,
                           char*  string_value);
```

Vissim passes the current value of the data item indicated by `type` and (for most types) indexed by `index1` and sometimes `index2`, in `DriverModelSetValue3` also by `index3`. The value is passed in `long_value`, `double_value` or `string_value`, depending on `type`. In some cases, values are passed in more than one of these variables.
The code in the function must make sure to save the value somewhere if it is required later for the driving behavior calculation because the next call of this function from Vissim will overwrite the local parameter.
(In the provided dummy DLL the values suggested by Vissim (via several calls to `DriverModelSetValue`) are saved in global variables in order to be able to send them back when Vissim calls `DriverModelGetValue` after the one call of `DriverModelExecuteCommand (DRIVER_COMMAND_MOVE_DRIVER)` per vehicle per time step.)
The function must return 1 for all types which are not marked as optional in the documentation below. For optional types, it can return 0 to inform Vissim that it doesn't handle this type.

```
int  DriverModelGetValue (long   type,
                          long   index1,
                          long   index2,
                          long   *long_value,
                          double *double_value,
                          char*  *string_value);
int  DriverModelGetValue3 (long   type,
                           long   index1,
                           long   index2,
                           long   index3,
                           long   *long_value,
                           double *double_value,
                           char*  *string_value);
```

Vissim retrieves the value of the data item indicated by `type` and (for most types) indexed by `index1` and sometimes `index2`, in `DriverModelSetValue3` also by `index3`. Before returning from this function, the value must be written to either `*long_value`, `*double_value` or `*string_value`, depending on the data type of the data item.
The function must return 1 for all types which are not marked as optional in the documentation below. For optional types, it can return 0 to inform Vissim that it doesn't handle this type.


```
int  DriverModelExecuteCommand (long number);
```

Vissim requests execution of the command indicated by `number`.
Currently available command constants are `DRIVER_COMMAND_INIT`, `DRIVER_COMMAND_CREATE_DRIVER`, `DRIVER_COMMAND_MOVE_DRIVER` and `DRIVER_COMMAND_KILL_DRIVER`. The function must return 1 for all these commands lest Vissim stop the simulation run.

Before Vissim requests execution of one of the available commands (`Init`, `CreateDriver`, `MoveDriver`, `KillDriver`) of the DLL there are always several calls of the DLL function `DriverModelSetValue`, one for each data item that might be used by the DLL when executing the command.

After the command `MoveDriver` has finished computation, the resulting state of the vehicle is fetched from the DLL in a similar manner again by several calls of `DriverModelGetValue`.

For a complete list of defined values for `type` and `number` see `DriverModel.h`.

3 Commands

There are 4 commands that a driver model DLL must implement:
`Init`, `CreateDriver`, `MoveDriver` and `KillDriver`.

3.1 Init

This command is executed through a call of
`DriverModelExecuteCommand (DRIVER_COMMAND_INIT)`
at the start of a Vissim simulation run to initialize the driver model DLL.
Several basic parameters are passed to the DLL before this through
`DriverModelSetValue ()` (shortened "Set" below),
and some values are retrieved from the DLL through
`DriverModelGetValue ()` ("Get").

The sequence of calls to the DLL is as follows:

For each vehicle type which uses that DLL:

Get DRIVER_DATA_STATUS	(optional)
Get DRIVER_DATA_STATUS_DETAILS	(only if STATUS is not 0; optional)
Set DRIVER_DATA_PATH	
Set DRIVER_DATA_PARAMETERFILE	(optional)
Set DRIVER_DATA_TIMESTEP	
Set DRIVER_DATA_TIME	
Set DRIVER_DATA_VEH_TYPE	
Get DRIVER_DATA_WANTS_SUGGESTION	
Get DRIVER_DATA_SIMPLE_LANECHANGE	
Get DRIVER_DATA_WANTS_ALL_NVEHS	(optional)

Then only once:

Get DRIVER_DATA_ALLOW_MULTITHREADING (optional)
DriverModelGetValue must set *long_value to 1 and return 1 if the DLL can handle multiple threads (this is required if multiple cores are set to be used in the simulation parameters).

Get DRIVER_DATA_WANTS_ALL_SIGNALS (optional)
DriverModelGetValue can set *long_value to 0 and return 1 to avoid later transmission of the signal states of all signal groups of all controllers.

Get DRIVER_DATA_MAX_NUM_INDICES (optional)
DriverModelGetValue must set *long_value to 3 and return1 if the DLL wants the functions DriverModelGetValue3 and DriverModelSetValue3 to be used by Vissim (required for DRIVER_DATA_ROUTE_SIGNAL_SWITCH data).

For each user-defined attribute for vehicles in Vissim:

Set DRIVER_DATA_USE_UDA (optional)
index1 = key of the UDA; string_value = short name of the UDA
DriverModelSetValue must return 1 if values for this UDA are to be sent later for each vehicle and nearby vehicle, else 0.

Execute DRIVER_COMMAND_INIT
Get DRIVER_DATA_STATUS (optional)
Get DRIVER_DATA_STATUS_DETAILS (only if STATUS is not 0; optional)

3.2 CreateDriver

`DriverModelExecuteCommand (DRIVER_COMMAND_CREATE_DRIVER)` is called from Vissim during the simulation run whenever a new vehicle is set into the network in Vissim (at the start of a time step, from a vehicle input, a PT line or a parking lot). In the same time step, a command `MoveDriver` for the same vehicle will follow later.

The sequence of calls to the DLL is as follows:

Set DRIVER_DATA_TIMESTEP
Set DRIVER_DATA_TIME
Set DRIVER_DATA_VEH_TYPE = VehicleTypeNumber
Set DRIVER_DATA_VEH_ID = NumberOfNewVehicle
Set DRIVER_DATA_VEH_DESIRED_VELOCITY = InitialDesiredVelocity
Set DRIVER_DATA_VEH_X_COORDINATE
Set DRIVER_DATA_VEH_Y_COORDINATE
Set DRIVER_DATA_VEH_Z_COORDINATE (optional)
Set DRIVER_DATA_VEH_REAR_X_COORDINATE (optional)
Set DRIVER_DATA_VEH_REAR_Y_COORDINATE (optional)
Set DRIVER_DATA_VEH_REAR_Z_COORDINATE (optional)

Execute DRIVER_COMMAND_CREATE_DRIVER

3.3 MoveDriver

`DriverModelExecuteCommand (DRIVER_COMMAND_MOVE_DRIVER)` is called from Vissim during the simulation run once per stime step for each vehicle of a vehicle type which uses this DriverModel DLL. *Before* this call, there are many calls of `DriverModelSetValue ()` to pass the current state of the vehicle and its surroundings to the DLL. *After* the execution of the command, there are several calls of `DriverModelGetValue ()` to retrieve the new values for acceleration and lateral behavior and optionally user-defined attributes from the DLL. Before any vehicle specific data is exchanged, Vissim passes the states of all signal groups and priority rules to the DLL.

The sequence of calls to the DLL is as follows:

3.3.1 Global data

Set DRIVER_DATA_TIMESTEP
Set DRIVER_DATA_TIME

Only if DriverModelGetValue(DRIVER_DATA_WANTS_ALL_SIGNALS, …) has *not* set *long_value to zero in the initialization step:

For each SC (passed in index1), **for each signal head** (number passed in index2):

Set DRIVER_DATA_SIGNAL_STATE =
red = 1, amber = 2, green = 3, red/amber = 4, amber flashing = 5, off = 6, other = 0

For each priority rule section (“yield sign”) (index 1 = 0; number passed in index2):

Set DRIVER_DATA_SIGNAL_STATE =
blocked = 1, free = 3

3.3.2 Vehicle specific data

For each vehicle of a vehicle type using this driver model DLL first its own data is passed from Vissim to the DLL, then data of all nearby vehicles along the route of the vehicle and then some data about the upcoming link / lanes geometry and other network objects. Then, the command to move the vehicle is called, and finally, the data calculated by the DLL is retrieved.

3.3.2.1 Data of the subject vehicle (at the start of the current time step)

Set DRIVER_DATA_TIMESTEP =
time step length [simulation seconds] [0.1 .. 1.0]
Set DRIVER_DATA_TIME =
current simulation time (simulation seconds since simulation start)
Set DRIVER_DATA_VEH_ID =
ID of the vehicle to be moved
Set DRIVER_DATA_VEH_LANE =
current lane number (rightmost = 1)
Set DRIVER_DATA_VEH_ODOMETER =
total elapsed distance in the network [m]
Set DRIVER_DATA_VEH_LANE_ANGLE =
angle relative to the middle of the lane [rad]

Set DRIVER_DATA_VEH_LATERAL_POSITION =
distance of the front end from the middle of the lane [m]
Set DRIVER_DATA_VEH_VELOCITY =
current speed [m/s]
Set DRIVER_DATA_VEH_ACCELERATION =
current acceleration [m/s²]
Set DRIVER_DATA_VEH_LENGTH =
vehicle length [m]
Set DRIVER_DATA_VEH_WIDTH =
vehicle width [m]
Set DRIVER_DATA_VEH_WEIGHT =
vehicle weight [kg]
Set DRIVER_DATA_VEH_MAX_ACCELERATION =
maximum possible acceleration [m/s²]
Set DRIVER_DATA_VEH_TURNING_INDICATOR =
left = 1, right = -1, none = 0, both = 2
(non-zero only if set in the last time step from this DLL)
Set DRIVER_DATA_VEH_CATEGORY =
car = 1, truck = 2, bus = 3, tram = 4, pedestrian = 5, bike = 6
Set DRIVER_DATA_VEH_COLOR =
vehicle color (32 bit ARGB value)
Set DRIVER_DATA_VEH_PREFERRED_REL_LANE =
desired direction of a lane change because of a downstream connector
of the vehicle's route or path or because of the right-side/left-side rule,
positive = left, 0 = current lane, negative = right
Set DRIVER_DATA_VEH_USE_PREFERRED_LANE =
0 = only preferable (e.g. European highway with right-side rule),
1 = necessary (e.g. before a connector)
Set DRIVER_DATA_VEH_DESIRED_VELOCITY =
desired speed [m/s]
Set DRIVER_DATA_VEH_X_COORDINATE =
world coordinate X (vehicle front end)
Set DRIVER_DATA_VEH_Y_COORDINATE =
world coordinate Y (vehicle front end)
Set DRIVER_DATA_VEH_Z_COORDINATE =
world coordinate Z (vehicle front end), optional
(calculated correctly only if 3D visualization or a vehicle record
containing "Coordinate front" or "Coordinate rear" is active)
Set DRIVER_DATA_VEH_REAR_X_COORDINATE =
world coordinate X (vehicle rear end), optional
Set DRIVER_DATA_VEH_REAR_Y_COORDINATE =
world coordinate Y (vehicle rear end), optional
Set DRIVER_DATA_VEH_REAR_Z_COORDINATE =
world coordinate Z (vehicle rear end), optional
(calculated correctly only if 3D visualization or a vehicle record
containing "Coordinate front" or "Coordinate rear" is active)
Set DRIVER_DATA_VEH_TYPE =
vehicle type number (user defined)

Set DRIVER_DATA_VEH_CURRENT_LINK =
current link number, optional

Only if DriverModelSetValue (DRIVER_DATA_VEH_CURRENT_LINK) returned 1:
For each link in the vehicle's route/path:

Set DRIVER_DATA_NEXT_LINKS =
link number, optional

Set DRIVER_DATA_VEH_ACTIVE_LANE_CHANGE =
direction of an active lane change movement
(+1 = to the left, 0 = none, -1 = to the right), optional
Set DRIVER_DATA_VEH_REL_TARGET_LANE =
target lange (+1 = next one left, 0 = current lane, -1 = next one right), optional
Set DRIVER_DATA_VEH_INTAC_STATE =
interaction state as determined by the internal car following model
(sent only if DRIVER_DATA_WANTS_SUGGESTION has been set!):
FREE = 1, CLOSEUP = 2, FOLLOW = 3, BRAKEAX = 4, BRAKEBX = 5, BRAKEZX = 6,
BRAKESPW = 7, BRAKEKOOP = 8, PELOPS = 9,PASS = 10, SLEEP = 11, DWELL = 12,
APPROACHING_PLATOON = 13, FOLLOWING_IN_PLATOON = 14,
LEAVING_PLATOON = 15; optional
Set DRIVER_DATA_VEH_INTAC_TARGET_TYPE =
type of the relevant interaction target as determined by the internal car following
model (sent only if DRIVER_DATA_WANTS_SUGGESTION has been set!):
no target = 0, real vehicle = 1, signal head = 2, priority rule = 3, conflict area = 4,
reduced speed area = 5, stop sign = 6, parking lot = 7, PT stop = 8; optional
Set DRIVER_DATA_VEH_INTAC_TARGET_ID =
number of the relevant interaction target as determined by the internal car following
model (sent only if DRIVER_DATA_WANTS_SUGGESTION has been set!); optional

Set DRIVER_DATA_VEH_INTAC_HEADWAY =
distance to the relevant interaction target as determined by the internal car following
model [m], front bumper to front bumper, including length of leading vehicle
(sent only if DRIVER_DATA_WANTS_SUGGESTION has been set!); optional

For each user-defined attribute for vehicles in Vissim which has been selected in the Init step (key of the UDA passed in index1):

Set DRIVER_DATA_VEH_UDA =
value of that user-defined attribute (bool passed as long (0 or 1),
all floating point types passed as double [in their currently selected unit]), optional

3.3.2.2 Data of the nearby vehicles

For each nearby vehicle (up to two each downstream and upstream, on up to 2 lanes each on both sides and on the current lane) several values are passed from Vissim:

index1 and index2 are used as follows for DRIVER_DATA_NVEH_*:
index1 = relative lane: +2 = second lane to the left, +1 = next lane to the left,
0 = current lane,
-1 = next lane to the right, -2 = second lane to the right
(exception for DRIVER_DATA_NVEH_UDA: index 1 = ID of the UDA!)

index2 = relative position: positive = downstream (+1 next, +2 second next, possibly more if DRIVER_DATA_WANTS_ALL_NVEHS is set)
negative = upstream (-1 next, -2 second next, possibly more if DRIVER_DATA_WANTS_ALL_NVEHS is set)

First, **for each index combination** with index2 in {-2, -1, +1, +2} (the DLL needs to initialize further index2 values itself) an initialization:

Set DRIVER_DATA_NVEH_ID = -1

Then, **for each *existing* nearby vehicle** the real data:

Set DRIVER_DATA_NVEH_ID =
vehicle number
Set DRIVER_DATA_NVEH_LANE_ANGLE =
angle relative to the middle of the lane [rad] (positive = turning left)
Set DRIVER_DATA_NVEH_LATERAL_POSITION =
distance of the front end from the middle of the lane [m]
(positive = left of the middle, negative = right)
Set DRIVER_DATA_NVEH_DISTANCE =
gross distance [m] (front end to front end, negative = nveh is upstream)
Set DRIVER_DATA_NVEH_REL_VELOCITY =
speed difference [m/s] (veh. speed - nveh. speed)
Set DRIVER_DATA_NVEH_ACCELERATION =
current acceleration [m/s²]
Set DRIVER_DATA_NVEH_LENGTH =
vehicle length [m]
Set DRIVER_DATA_NVEH_WIDTH =
vehicle width [m]
Set DRIVER_DATA_NVEH_WEIGHT =
vehicle weight [kg]

Set DRIVER_DATA_NVEH_TURNING_INDICATOR =
left = 1, right = -1, none = 0, both = 2
(non-zero only if set in the last time step from this DLL)
Set DRIVER_DATA_NVEH_CATEGORY =
car = 1, truck = 2, bus = 3, tram = 4, pedestrian = 5, bike = 6
Set DRIVER_DATA_NVEH_LANE_CHANGE =
direction of a current lane change (+1 = to the left, 0 = none, -1 = to the right)
Set DRIVER_DATA_NVEH_TYPE =
number of the vehicle type in Vissim, optional
Set DRIVER_DATA_NVEH_X_COORDINATE =
world coordinate X (vehicle front end), optional
Set DRIVER_DATA_NVEH_Y_COORDINATE =
world coordinate Y (vehicle front end) , optional
Set DRIVER_DATA_NVEH_Z_COORDINATE =
world coordinate Z (vehicle front end), optional
(calculated correctly only if 3D visualization or a vehicle record
containing "Coordinate front" or "Coordinate rear" is active)
Set DRIVER_DATA_NVEH_REAR_X_COORDINATE =

world coordinate X (vehicle rear end), optional

Set DRIVER_DATA_NVEH_REAR_Y_COORDINATE =
world coordinate Y (vehicle rear end), optional

Set DRIVER_DATA_NVEH_REAR_Z_COORDINATE =
world coordinate Z (vehicle rear end), optional
(calculated correctly only if 3D visualization or a vehicle record containing "Coordinate front" or "Coordinate rear" is active)

For each user-defined attribute for vehicles in Vissim which has been selected in the Init step (key of the UDA passed in index1!):

Set DRIVER_DATA_NVEH_UDA =
value of that user-defined attribute (bool passed as long (0 or 1), all floating point types passed as double [in their currently selected unit]), optional

3.3.2.3 Data of the current link

Set DRIVER_DATA_NO_OF_LANES =
number of lanes of the link the vehicle is currently on

Data of all lanes of the current link of the subject vehicle

For each lane of the current link of the vehicle, several values are passed from Vissim:

index1 = lane number (rightmost = 1), index2 is irrelevant.

Set DRIVER_DATA_LANE_WIDTH =
lane width [m]

Set DRIVER_DATA_LANE_END_DISTANCE =
distance to end of lane [m] (can be emergency stop position before connector, negative = no end of lane in visibility range)

3.3.2.4 Data of the current lane

Set DRIVER_DATA_CURRENT_LANE_POLY_N =
number of downstream lane polygon points within visibility distance along the route/path of the vehicle, optional

For each polygon point (index1 = 0..n-1):

Set DRIVER_DATA_CURRENT_LANE_POLY_X =
X world coordinate of polygon point in the middle of the lane, optional

Set DRIVER_DATA_CURRENT_LANE_POLY_Y =
Y world coordinate of polygon point in the middle of the lane, optional

Set DRIVER_DATA_CURRENT_LANE_POLY_Z =
Z world coordinate of polygon point in the middle of the lane, optional

3.3.2.5 Data of the current and upcoming environment

Set DRIVER_DATA_RADIUS =
current curve radius [m]

Set DRIVER_DATA_MIN_RADIUS =
minimum curve radius [m] in visibility range

Set DRIVER_DATA_DIST_TO_MIN_RADIUS =
distance [m] to spot of minimum curve radius

Set DRIVER_DATA_SLOPE =
current slope (negative = drop)
Set DRIVER_DATA_SLOPE_AHEAD =
slope at end of visibility range

3.3.2.6 Data of the next signal head and priority rule stop line on the vehicle's *lane*

index1 = signal controller number / priority rule: zero,
index2 = signal head number / priority rule: number

Set DRIVER_DATA_SIGNAL_DISTANCE =
distance [m] to next signal head (negative = none visible)
Set DRIVER_DATA_SIGNAL_STATE =
red = 1, amber = 2, green = 3, red/amber = 4, amber flashing = 5, off = 6, other = 0
Set DRIVER_DATA_SIGNAL_STATE_START =
simulation time [s] when signal changed to current state

Set DRIVER_DATA_PRIO_RULE_DISTANCE =
distance [m] to next priority rule stop line (negative = none visible)
Set DRIVER_DATA_PRIO_RULE_STATE =
blocked = 1, free = 3, unknown = 0

3.3.2.7 Data of the next reduced speed area or previous desired speed decision

Set DRIVER_DATA_SPEED_LIMIT_DISTANCE =
distance [m] to "speed limit sign" (reduced speed area: real distance,
desired speed decision: 1.0 m when just passed, negative: no sign visible)
Set DRIVER_DATA_SPEED_LIMIT_VALUE =
speed limit [km/h] (0 = end of reduced speed area)

3.3.2.8 Data of the next signal head on the vehicle's *route* (including required lane changes)

index1 = signal controller number, index2 = signal head number

Set DRIVER_DATA_ROUTE_SIGNAL_DISTANCE =
distance [m] to next signal head (negative = none visible)
Set DRIVER_DATA_ROUTE_SIGNAL_STATE =
red = 1, amber = 2, green = 3, red/amber = 4, amber flashing = 5, off = 6, other = 0

Only if DriverModelSetValue3(DRIVER_DATA_MAX_NUM_INDICES, …)
has set *long_value to ≥3:
For the next 4 signal state switches of that signal head:

Set DRIVER_DATA_ROUTE_SIGNAL_SWITCH =
long = new signal state after future switch (see above)
double = time left [s] until future signal state switch
index3 = switch index (0 = next switch, 1 = second next, …)

[DriverModelSetValue3() can return 0 to this in order to skip further switch data for the current vehicle.]

Set DRIVER_DATA_ROUTE_SIGNAL_CYCLE =
cycle length [s] of the controller of the signal head (unknown = 0)

3.3.2.9 Data of the next conflict areas on the vehicle's lane (along the route, *not* including required lane changes)

Set DRIVER_DATA_CONFL_AREAS_COUNT =
number of visible downstream lane conflicts, optional
[DriverModelSetValue() can return 0 to this in order to skip further conflict area data for the current vehicle. Otherwise it needs to return 1 for all conflict area values below.]

For each of these conflict areas:

index1 = index of the lane conflict [starting at 0], index2 = conflict area number

Set DRIVER_DATA_CONFL_AREA_TYPE =
type of the lane conflict: 1 = crossing, 2 = merging, 3 = branching

Set DRIVER_DATA_CONFL_AREA_YIELD =
0 = pass (green/red), 1 = yield (red/green), 2 = undetermined (red/red)

index1 = index of the lane conflict [starting at 0]

Set DRIVER_DATA_CONFL_AREA_DISTANCE =
distance [m] to the end (!) of the lane conflict

Set DRIVER_DATA_CONFL_AREA_LENGTH =
length [m] of the lane conflict

Set DRIVER_DATA_CONFL_AREA_VEHICLES =
number of possibly conflicting vehicles/pedestrians

For each of these conflicting vehicles/pedestrians:

index1 = index of the lane conflict [starting at 0],
index2 = index of the vehicle/pedestrian [starting at 0]

Set DRIVER_DATA_CONFL_AREA_TIME_ENTER =
simulation time when the front end of the vehicle arrives in the lane conflict
(-1 for "currently not intended to enter")
(number of the vehicle is also passed in long_value, zero for pedestrians)

Set DRIVER_DATA_CONFL_AREA_TIME_IN =
simulation time when the rear end of the vehicle arrives in the lane conflict
(only used for merging areas, else -1)

Set DRIVER_DATA_CONFL_AREA_TIME_EXIT =
simulation time when the rear end of the vehicle leaves the lane conflict
(-1 for "currently not intended to enter")

3.3.2.10 Behavior data suggested for the current time step by Vissim's internal model

These values are passed only if `*long_value` has been set to 1 in the call of `DriverModelGetValue (DRIVER_DATA_WANTS_SUGGESTION)`.

Set DRIVER_DATA_DESIRED_ACCELERATION =
desired acceleration [m/s²] in this time step
Set DRIVER_DATA_DESIRED_LANE_ANGLE =
desired angle relative to the middle of the lane [rad] (positive = turning left)
Set DRIVER_DATA_ACTIVE_LANE_CHANGE =
direction of an active lane change movement (+1 = to the left, 0 = none,
-1 = to the right, must be != 0 while lane change is not completed,
will be used for NVEH_LANE_CHANGE seen from other vehicles)
Set DRIVER_DATA_REL_TARGET_LANE =
target lane (+1 = next one left, 0 = current lane, -1 = next one right)

3.3.2.11 Execute Move Command

Execute DRIVER_COMMAND_MOVE_DRIVER

3.3.2.12 Pass new data calculated by the behavior model in the DLL back to Vissim

Get DRIVER_DATA_VEH_TURNING_INDICATOR =
left = 1, right = -1, none = 0, both = 2
(can be set by the DLL to make the ego vehicle in Vissim show a blinker (in the visualization) and to allow other vehicles to see that active blinker when they check `DRIVER_DATA_NVEH_TURNING_INDICATOR`)
Get DRIVER_DATA_VEH_DESIRED_VELOCITY =
desired speed [m/s]
Get DRIVER_DATA_VEH_COLOR =
vehicle color (32 bit ARGB value)
Get DRIVER_DATA_USE_INTERNAL_MODEL =
use the values passed from the driver model DLL = 0,
use Vissim's internal model for this time step = 1
(1 is only possible if `DRIVER_DATA_WANTS_SUGGESTION` has been set!), optional

Only if `*long_value` has been set to 0 in the call of `DriverModelGetValue (DRIVER_DATA_USE_INTERNAL_MODEL)`:
Get DRIVER_DATA_DESIRED_ACCELERATION =
new acceleration [m/s²], optional
[This value is limited by Vissim to the minimum of desired acceleration and maximum acceleration and to the maximum deceleration for the vehicle at the current speed. It is *not* affected by the limitation of the change of acceleration which is used in the internal car following model. If this value is set, Vissim shows the interaction state of the vehicle as "PELOPS" in the vehicle record.]

Get DRIVER_DATA_DESIRED_LANE_ANGLE =
desired angle relative to the middle of the lane [rad] (positive = turning left), optional
[If `*long_value` was set to 1 in the call of `DriverModelGetValue (DRIVER_DATA_SIMPLE_LANECHANGE)` this angle does not need to be set by the DLL and the return value of `DriverModelGetValue(DRIVER_DATA_DESIRED_LANE_ANGLE)` can be zero. If it is 1, the angle set by the DLL is used, however!]

Get DRIVER_DATA_ACTIVE_LANE_CHANGE =
direction of an active lane change movement (+1 = to the left, 0 = none, -1 = to the right, must be != 0 while lane change is not completed)
[If `*long_value` was set to 1 in the call of `DriverModelGetValue (DRIVER_DATA_SIMPLE_LANECHANGE)` setting this to +1 or -1 is sufficient to start a lane change which will be completed automatically by Vissim.]

Get DRIVER_DATA_REL_TARGET_LANE =
target lane (+1 = next one left, 0 = current lane, -1 = next one right)
[This is used by Vissim only if `*long_value` was set to 0 in the call of `DriverModelGetValue (DRIVER_DATA_SIMPLE_LANECHANGE)`.]

Regardless of `DRIVER_DATA_USE_INTERNAL_MODEL`:
For each user-defined attribute for vehicles in Vissim which has been selected in the Init step (key of the UDA passed in index1!):

Get DRIVER_DATA_VEH_UDA =
value of that user-defined attribute (bool passed as long (0 or 1), all floating point types passed as double [in their currently selected unit]), optional

3.4 KillDriver

`DriverModelExecuteCommand (DRIVER_COMMAND_KILL_DRIVER)` is called from Vissim when a vehicle reaches its destination and thus leaves the network (so memory which the DLL might have allocated for that vehicle can be freed):

Set DRIVER_DATA_VEH_ID =
ID of the vehicle to be killed

Execute DRIVER_COMMAND_KILL_DRIVER

4 Lane Change

The driver model DLL interface provides 2 different ways to control lane changes of vehicles affected by the external dll:

- Simple lane change
- Full control over the lane change

If simple lane change is selected, the driver model DLL only needs to initiate a lane change for a vehicle. Vissim assumes control of the lateral behavior of this vehicle while the lane change proceeds and informs the driver model DLL when it is done.

Without simple lane change the driver model DLL has full control of the vehicle and manages the lane change on its own – the driver model DLL must inform Vissim about the current state of the vehicle.

How lane changes are actually handled is determined by the data types `DRIVER_DATA_SIMPLE_LANECHANGE` and `DRIVER_DATA_WANTS_SUGGESTION`.

The lane change itself involves the data types `DRIVER_DATA_VEH_ACTIVE_LANE_CHANGE`, `DRIVER_DATA_VEH_REL_TARGET_LANE`, `DRIVER_DATA_ACTIVE_LANE_CHANGE`, `DRIVER_DATA_REL_TARGET_LANE` and `DRIVER_DATA_DESIRED_LANE_ANGLE`.

4.1 Simple lane change - handled by Vissim

This mode is chosen by setting `*long_value` to 1 as result of Vissim's initial request for `DRIVER_DATA_SIMPLE_LANECHANGE`.

If the driver model sets `*long_value` to 1 as result for `DRIVER_DATA_WANTS_SUGGESTION` as well, Vissim will send a suggestion whenever it detects that a lane change is necessary.
As long as the driver model sets `*long_value` to 1 as result of Vissim's requests for `DRIVER_DATA_USE_INTERNAL_MODEL`, Vissim has complete control of lane changes.

A simple lane change can be initiated from the driver model DLL the following way:

1. The DLL sets `DRIVER_DATA_ACTIVE_LANE_CHANGE` to 1 (to the left) or -1 (to the right). (It does not need to set `DRIVER_DATA_REL_TARGET_LANE` to the same value.)
2. Vissim starts the lane change for the current vehicle, initially with a lateral movement sufficient to complete the lane change within 3 simulation seconds. (A lane angle set by the DLL is ignored in this first time step.)
3. During the simple lane change, Vissim ignores any values sent by the DLL for `DRIVER_DATA_ACTIVE_LANE_CHANGE` and `DRIVER_DATA_REL_TARGET_LANE`, so a simple lane change cannot be interrupted by the DLL, but it uses the value sent by the DLL for `DRIVER_DATA_DESIRED_LANE_ANGLE`, provided `DriverModelGetValue` returns 1 (it may return 0). That value must have the same sign as `DRIVER_DATA_ACTIVE_LANE_CHANGE` and `DRIVER_DATA_REL_TARGET_LANE` lest Vissim reports an error about inconsistent data.
4. Vissim sets `DRIVER_DATA_VEH_REL_TARGET_LANE`, `DRIVER_DATA_VEH_LANE` and (a suggested) `DRIVER_DATA_VEH_LANE_ANGLE` during a simple lane change: `DRIVER_DATA_VEH_REL_TARGET_LANE` is set to zero when the middle of the front end

of the vehicle has reached the target lane, and DRIVER_DATA_VEH_LANE is set to the new lane at this time, too.

5. When Vissim has finished the lane change (i.e. when the whole width of the vehicle is on the new lane), it sends 0 as value of DRIVER_DATA_VEH_ACTIVE_LANE_CHANGE. (This does not necessarily mean that the vehicle has reached its desired lateral position within the destination lane.)

4.2 Lane change - handled by the driver model DLL

This mode is chosen by setting *long_value to 0 as result of Vissim's initial request for DRIVER_DATA_SIMPLE_LANECHANGE.

A lane change can be executed by the driver model DLL the following way:

1. The driver model DLL sets DRIVER_DATA_ACTIVE_LANE_CHANGE, DRIVER_DATA_REL_TARGET_LANE and DRIVER_DATA_DESIRED_LANE_ANGLE to the desired values.
2. Vissim moves the vehicle according to these values and sets DRIVER_DATA_VEH_REL_TARGET_LANE to zero when the middle of the front end of the vehicle has reached the target lane and sets DRIVER_DATA_VEH_LANE to the new lane number at this time, too.
   (If there is no new lane on this side, Vissim sets the vehicle back to the middle of the old lane and stops the lane change itself, setting DRIVER_DATA_ACTIVE_LANE_CHANGE and DRIVER_DATA_DESIRED_LANE_ANGLE to zero.)
3. The driver model DLL has to determine itself when the lane change is over – it has to reset the values for DRIVER_DATA_ACTIVE_LANE_CHANGE and DRIVER_DATA_DESIRED_LANE_ANGLE to zero.

Hints:

- The driver model has full control in this mode – it can even interrupt a current lane change.
- Vissim does not finish a lane change on its own in this mode. This means that the vehicle will continue moving laterally even beyond the next lane if the driver model does not stop the lane change.

5 Multithreading

If multiple cores are used for a simulation run, vehicles on different Vissim links can be handled by different threads. The assignment of links to these threads can be different in each time step. The DLL functions are called during one time step in a non-deterministic sequence from multiple threads, i.e. possibly alternating and overlapping for different subject vehicles. So the data can only be assigned correctly if thread-local storage is used (instead of global variables which are fine for singlethreaded use) or if there is only one subject vehicle (using that DLL) or if all subject vehicles are on the same link.

