Aimsun 8 API Manual

March 2015

About this Manual

The present manual describes the Aimsun Microsimulator API to be used in Aimsun 8. This component, to be used, requires a license of both Aimsun and the Aimsun Microsimulator API.

Our aim is to keep you informed of any changes so that you can continue to get the best from Aimsun. As always, please be aware that product data is subject to change without notice.

TSS-Transport Simulation Systems has made every effort to ensure that all the information contained within this manual is as accurate as possible. It should be stressed however, that this is a draft version of the latest Aimsun API Manual and as such, some of the contents may be subject to change.

As always, we welcome your feedback (support@aimsun.com) in our continued improvement and addition of new features to Aimsun

Copyright

Trademark

1 Introduction

The current trend in the development of Advanced Transport Telematic Applications, either real-time adaptive, or based on other specific approaches, is far from being standardised. It is therefore an exercise of dubious utility to try to integrate them in a fixed way in a microscopic traffic simulator. The relative gain achieved by including any of these, as an in-built function of the microsimulator is limited to simulating, on an easier way, those road networks on which the selected application is operating. However there would be no means of simulating other systems with that microsimulator. This is true whenever we address the problem of simulating adaptive traffic control systems as, for example, SCOOT, SCATS, vehicle actuated, priorizing public transport, etc., Advanced Traffic Management Systems (using VMS, traffic calming strategies, ramp metering policies, etc), Vehicle Guidance Systems, Public Transport Vehicle Scheduling and Control Systems or applications aimed at estimating the environmental impacts of pollutant emissions, and energy consumes. The main question then is: How can these Advanced Transport Telematic Applications be properly evaluated and tested by simulation?

From a conceptual point of view the operation of these modern systems can be described as follows: for certain applications the road network is suitably equipped with traffic detectors of various technologies (loop detectors, image processing detectors, etc.), with a specific layout depending on the requirements of the control approach. They supply the necessary real-time traffic data (flows, speeds, occupancies, etc) with the required degree of aggregation. These real-time traffic measurements feed the logic of the traffic control or management system which, after suitable processing, makes ad hoc control decisions: e.g. extend the green phase, change to the red phase, apply some traffic calming strategies, etc. Other applications as, for example vehicle guidance, public transport monitoring systems, or the evaluation of environmental impacts, require the access to vehicle data (position, speed, acceleration, etc.), to emulate the up-link messages in vehicle guidance applications, the vehicle tracking for the public transport monitoring or fleet management systems, or simply to provide the required data for certain fuel consumption or pollutant emissions models.

To evaluate and test any of these systems a microsimulator must be capable of incorporating in the model the corresponding traffic devices as objects: i.e. detectors, traffic lights, VMS, etc. It must also emulate their functions: provide the specific traffic measurements at the required time intervals, increase the phase timing in a given amount of time, implement a traffic calming strategy (slow down the speed on a road section, recommend an alternative route, etc). How can such evaluations be done by simulation without explicit in-built modelling of the specific Advanced Telematic Application?

1.1 Building Aimsun APIs in 64 bits

In order to build an AAPI in 64 bits, select menu *Build / Configuration Manager* from Visual Studio 2005 (x64). Then, select x64 in *Active solution platform* dropdown box.

1.2 Changes in version 8.0.4

1.2.1 Strings usage

There's a new method to delete ASCII strings

- void AKIDeleteASCIIString(const char *string);

Please refer to section 3.3.19.6 for more information about it.

1.2.2 Function relative to Network Information

There's a new method to get the total network length

- double AKIGetTotalLengthSystem();

Please refer to section 3.3.20.44 for more information about it.

1.3 Changes in version 8.0.3

1.3.1 Funtions relative to Signal Group information

Now, it is possible to get the total green, yellow and red times in the current control plan.

- int ECIGetSignalGroupGreenDuration(int idJunction, int signalPos, double timeSta)
- int ECIGetSignalGroupYellowDuration(int idJunction, int signalPos, double timeSta)
- int ECIGetSignalGroupRedDuration(int idJunction, int signalPos, double timeSta)

Please refer to sections 3.3.2.37, 3.3.2.38 and 3.3.2.39 below for more information about them.

1.3.2 Functions relative to turn information

Now, it is possible to access to turn static information by using the following API functions:

- int AKIInfNetNbTurns();
- int AKIInfNetGetTurnId(int element);
- A2KTurnInf AKIInfNetGetTurnInf(int aid);

- int AKIInfNetGetNbTurnsInNode(int idnode);
- int AKIInfNetGetOriginSectionInTurn(int idnode, int index);
- int AKIInfNetGetDestinationSectionInTurn(int idnode, int index);
- A2KTurnInf AKIInfNetGetTurnInfo(int idnode, int index);

Please refer to sections 3.3.20.33, 3.3.20.34, 3.3.20.35, 3.3.20.36, 3.3.20.37, 3.3.20.38 and 3.3.20.39 below for more information about them.

1.3.3 Function relative to Network Information

Two new methods are available to get the shortest path between two sections in the model:

- int AKIInfNetGetShortestPathNbSections(int fromSection, int toSection, void * sectionColumnCost);
- int AKIInfNetGetShortestPath(int fromSection, int toSection, void * sectionColumnCost, int * path);

Please refer to sections 3.3.20.40 and 3.3.20.41 for more information.

1.3.4 Function relative to Preemption Sets

Eight new methods have been added to access Public Transport Preemption Sets attributes from python. Function to get those attributes were already available for a C++ AAPI. The new functions available for Python APIs are:

- int ECIGetPreemptionSetNbLinesPython(int idJunction, double timeSta, int index);
- int ECIGetPreemptionSetLinesPython(int idJunction, double timeSta, int index, int nblines, int *lines);
- int ECIGetPreemptionSetNbPhasesPython(int idJunction, double timeSta);
- int ECIGetPreemptionSetPhasesPython(int idJunction, double timeSta, int index, int nbPhases , int *phases);
- int ECIGetPreemptionSetNbRequestDetectorsPython(int idJunction, double timeSta, int index);
- int ECIGetPreemptionSetRequestDetectorsPython(int idJunction, double timeSta, int index, int nbDetectors, int * detectors);

- int ECIGetPreemptionSetNbEndDetectorsPython(int idJunction, double timeSta, int index);
- int ECIGetPreemptionSetEndDetectorsPython(int idJunction, double timeSta, int index, int nbDetectors, int * detectors);

Please refer to sections 3.3.3.6, 3.3.3.7, 3.3.3.8, 3.3.3.9, 3.3.3.10, 3.3.3.11, 3.3.3.12 and 3.3.3.13 for more information.

1.3.5 Function relative Control Plans

There are two new methods to access information related to the active control plans:

- int ECIGetNbActiveControls();
- int ECIGetActiveControls(int* activeControls);

Please refer to sections 3.3.20.42 and 3.3.20.43 for more information.

1.4 Changes in version 8.0.2

1.4.1 Functions relative to node statistics

Now, all information related to the node outputs such as the number of lost vehicles, delay to calculate the level of service and number of missed turns is accessible by using the following API functions:

- int AKIEstGetGlobalStatisticsNodeLostVehicles(int aidNode, int vehTypePos);
- int AKIEstGetPartialStatisticsNodeLostVehicles(int aidNode, int vehTypePos);
- double AKIEstGetGlobalStatisticsNodeLevelOfService(int aidNode);
- double AKIEstGetPartialStatisticsNodeLevelOfService(int aidNode);
- int AKIEstGetGlobalStatisticsNodeMissedTurns(int aidNode, int vehTypePos);
- int AKIEstGetPartialStatisticsNodeMissedTurns(int aidNode, int vehTypePos);

Please refer to sections 3.3.12.43, 3.3.12.44, 3.3.12.45, 3.3.12.46, 3.3.12.47 and 3.3.12.48 below for more information about them.

1.4.2 Functions relative to pedestrians

Two new functions to generate Legion pedestrians in a micro simulation that is simulating pedestrians with Legion for Aimsun have been included.

Please refer to sections 3.3.13.7 and 3.3.13.8 for details.

1.4.3 int pointer parameters usage

Functions with *int* * representing an int array as input parameters are now supported by *intArray* class. Here's an small example:

```
nextSections = intArray( 3 )
nextSections [0] = 182
nextSections [1] = 183
nextSections [2] = 184
AKIActionAddNextSubPathODAction(182, 3, nextSections, 286, 285, 0, 110, 100, 100)
```

1.5 Changes in version 8.0.1

1.5.1 Functions relative to vehicle information

A new attribute to distinguish between the time a vehicle was generated and the time it actually was able to enter into the first section has been included.

Please refer to attributes SystemGenerationT and SystemEntranceT in the InfVeh structure. See sections 3.3.8.21, 3.3.11.3, 3.3.11.4, 3.3.11.11, 3.3.14.7 or 3.3.17.10 for details.

1.6 Changes in version 8.0

1.6.1 Functions relative to public transport preemption

Now, all information related to preemption is accessible by using the following API functions.

- int ECIGetNbPreemptionSets(int idJunction, double timeSta);
- int ECIGetPreemptionSetParameters(int idJunction, double timeSta, int index, double* delay, double *minDwell, double *reserve, double * inhibit, double *maxDwell, int *type);
- int ECIGetPreemptionSetLines(int idJunction, double timeSta, int index, int* nbLines , int *lines);
- int ECIGetPreemptionSetPhases(int idJunction, double timeSta, int index, int* nbPhases , int *phases);
- int ECIGetPreemptionSetRequestDetectors (int idJunction, double timeSta, int index, int* nbDetectors, int * detectors);
- int ECIGetPreemptionSetEndDetectors (int idJunction, double timeSta, int index, int* nbDetectors, int * detectors);

Please refer to sections 3.3.3.4, 3.3.3.5, 3.3.3.7, 3.3.3.8, 3.3.3.10 and 3.3.3.12 below for more information about them.

1.6.2 Functions relative to green meterings by lane

Two functions to get and set the green time by lane ramp meters parameters have been defined:

- int ECIGetParametersGreenMeteringByLaneById(int idmetering, double timeSta, double *amax, double *greenTime, double *amin,double *cicleTime, double *offset, double *yellowTime, double *laneOffset);
- int ECIChangeParametersGreenMeteringByLaneById(int idmetering,double timeSta,double amax, double ngreenTime, double amin, double ncicleTime, double offset, double YellowTime, double laneOffset);

Please refer to section 3.3.4.10 and 3.3.4.11 below for more information about them.

1.6.3 Functions relative to traffic demand

Now the number of slices in a traffic demand based on states are vehicle dependent so a new parameter to identify the vehicle type has been added.

- int AKIStateDemandGetNumSlices(int vehTypePos)
- int AKIStateDemandGetIniTimeSlice (int vehTypePos, int numslice)
- int AKIStateDemandGetIniTimeSlice (int vehTypePos, int numslice)

Please refer to section 3.3.23 and its subsections for more information about them.

1.6.4 Functions relative to vehicle types

Functions to modify the parameters related to the lane changing behaviour for the vehicle types have been added. The functions are:

- double AKIVehTypeGetPercentageForImprudentLaneChanging(int idVehicleType)
- int AKIVehTypeSetPercentageForImprudentLaneChanging(int idVehicleType, double newPercentage)
- double AKIVehTypeGetSensitivityForImprudentLaneChanging(int idVehicleType)
- int AKIVehTypeSetSensitivityForImprudentLaneChanging(int idVehicleType, double newSensitivity)

- double AKIVehTypeGetPercentageForStayingInFastLane(int idVehicleType)
- int AKIVehTypeSetPercentageForStayingInFastLane(int idVehicleType, double newPercentage)
- double AKIVehTypeGetPercentageForUndertaking(int idVehicleType)
- int AKIVehTypeSetPercentageForUndertaking (int idVehicleType, double newPercentage)

1.6.5 Functions relative to PT vehicles

Function to modify the parameters related to the public transport vehiccles have been added. The functions is:

- int AKIPTVehSetStaticInf(int aidVeh, StaticInfPTVeh staticinfVeh);

And, in the structure of InfPTVeh, two new attributes has been added:

```
double observedLastStopTime;
double observedLastInitialStopTime;
```

and the attribute currentLoad has been moved from the structure StaticInfPTVeh to InfPTVeh.

Please refer to sections 3.3.10.7, 3.3.10.8, 3.3.10.9, 3.3.10.10, 3.3.10.11, 3.3.10.12, 3.3.10.13 , 3.3.10.14 and 3.3.17 for more information about them.

1.7 Changes in version 7.0.4

1.7.1 Function relative to vehicle tracking

A new function to modify the next sections of a tracked vehicle has been added.

- int AKIVehTrackedModifyNextSections(int aidVeh, int sizeNextSections, const int *nextSections)

Please refer to section 3.3.14.4 for more information.

1.7.2 Functions relative to public transport preemption

Functions to check if there is any pre-emption defined in a node, to deactivate them and to activate them back have been added. The functions are:

- int ECIDisableBusPreemptionNode(int ajunction)
- int ECIEnableBusPreemptionNode(int ajunction)
- bool ECIIsBusPreemptionNodeEnabled(int idJunction)

Please refer to section 3.3.3 for more information about them.

1.8 Changes in version 7.0.1

1.8.1 Functions relative to traffic demand

Now the number of slices in a traffic demand are vehicle dependent so a new parameter to identify the vehicle type has been added.

- int AKIODDemandGetNumSlicesOD(int vehTypePos)
- int AKIODDemandGetIniTimeSlice (int vehTypePos, int numslice)
- int AKIODDemandGetIniTimeSlice (int vehTypePos, int numslice)

Trips are internally represented as integers.

- int AKIODDemandGetDemandODPair (int origin, int desti, int vehTypePos, int numslice)
- int AKIODDemandSetDemandODPair (int origen, int desti, int vehTypePos, int numslice, int anewdemand)

Please refer to section 3.3.22 and its subsections for more information about them.

1.8.2 Functions relative to public transport vehicles

Now it is possible to know the seconds a public transport vehicle has been stopped at the current public transport stop.

Please refer to section 3.3.17.10 for more information about it.

1.9 Changes in version 7

1.9.1 Functions relative to statistics

A new function to read section instant virtual queue statistics:

- double AKIEstGetInstantVirtualQueueSection(int aidarc,int vehTypePos);

New functions to read link statistics have been added:

- StructAkiEstadTurning AKIEstGetGlobalStatisticsLink(int aidsectionFrom, int aidsectionTo, int vehTypePos);
- StructAkiEstadTurning AKIEstGetParcialStatisticsLink(int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsLinkFuelCons(int aidsectionFrom, int aidsectionTo, int vehTypePos);
- double AKIEstGetParcialStatisticsLinkFuelCons(int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsLinkPollution(int indexPol, int aidsectionFrom, int aidsectionTo, int vehTypePos);
- double AKIEstGetParcialStatisticsLinkPollution(int indexPol, int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);

1.9.2 Functions relative to actuated signal control

Two new funcions to get and modify the Force Off and Permissive period parameters for an actuated control junction have been added:

- int ECIGetActuatedParamsForceOFFPermissivePeriod(int elemControl, int ajunction, int aidphase, double *forceOFF, double *permissivePeriodFrom, double *permissivePeriodTo);
- int ECISetActuatedParamsForceOFFPermissivePeriod(int elemControl, int ajunction, int aidphase, double forceOFF, double permissivePeriodFrom, double permissivePeriodTo);

Please refer to section 3.3.2.35 and 3.3.2.36 for more information about them.

1.9.3 Functions depending on vehicle type

The vehicle type reference in several functions has been changed from the vehicle type name to the vehicle type ID. The modified functions are:

- int ECIChangeParametersDelayMeteringVehTypeById(int idmetering, double timeSta, double newavg, double newdev, int idVehType)
- int ECIChangeParametersDelayMeteringVehType(int idsection, double timeSta, double newavg, double newdev, int idVehType)

- double AKIPastCostGetPastCost(int sectorig, int sectdest, double aTime, int idVehType);
- double AKIPastCostGetPastOutputCost(int sectorig, int sectdest, double aTime, int idVehType);

Please refer to sections 3.3.4.18 and 3.3.24.5 for more information about them.

Also the:

int AKIVehGetVehTypePosition(const unsigned short *aname);

function is considered as deprecated. The way to access to the internal position of a vehicle is by using its Aimsun ID.

Please refer to section 3.3.10.5 for more information about a possible workaround if the vehicle wants to be still accessed by name.

1.9.4 Functions relative to incidents

The function used to create an incident needs two additional paramers to define the incident's visibility distance and whether it would be a part of a group of incidents or not. The modified functions is:

- int AKIGenerateIncident(int asection, int alane, double position, double length, double initime, double duration, double visibilityDistance, bool updateIdGroup)

Please refer to section 3.3.15.1 for more information about it.

1.9.5 Obsolete Functions

Obsolete or redundant funcions are no longer available. These functions were:

- structA2KDetector AKIDetGetPropertiesDetectorANG(int elem)
- structA2KDetector AKIDetGetPropertiesDetectorANGById (int IdDetector)

1.10 Changes in version 6.1.4

1.10.1 Functions relative to strings in Python

New functions to read the network name and path, traffic demand name and pollutant name (if they are ascii) have been added to the Python version. These functions are:

- const char * AKIInfNetGetNetworkNameA();
- const char * AKIInfNetGetNetworkPathA();
- const char * AKIInfNetGetTrafficDemandNameA();
- const char * AKIEstGetPollutantNameA(int index);

Refer to sections 3.3.20.2, 3.3.20.3, 3.3.20.4 and 3.3.12.5 for more information about them.

1.11 Changes in version 6.1.2

1.11.1 Functions relative to Route Choice Calculation

A new high-level function to inform that a new route choice cycle is about to be calculated has been added:

- int AAPIPreRouteChoiceCalculation(double time, double timeSta);

Refer to section 3.2 for more information about it.

New functions to read and modify the section parameters that are used in the route choice calculation have been added. These functions are:

- int AKISetSectionCapacity(int idsection, double capacity);
- double AKIGetSectionCapacity(int idsection);
- int AKISetSectionUserDefinedCost(int idsection, double userdefinedcost);
- double AKIGetSectionUserDefinedCost(int idsection);
- int AKISetSectionUserDefinedCost2(int idsection, double userdefinedcost);
- double AKIGetSectionUserDefinedCost2(int idsection);
- int AKISetSectionUserDefinedCost3(int idsection, double userdefinedcost);
- double AKIGetSectionUserDefinedCost3(int idsection);

Refer to sections 3.3.20.16 to 3.3.20.23 for more information about them.

1.11.2 Functions relative to Actuated Control

New functions to read and modify the actuated parameters of a control plan have added. These functions are:

- int ECIGetActuatedParamsPassageTime(int elemControl, int ajunction, int aidphase, double *apassageTime, double *atimeBeforeReduce, double *atimeToReduce, double *aminimumGap);
- int ECISetActuatedParamsPassageTime(int elemControl, int ajunction, int aidphase, double apassageTime, double atimeBeforeReduce, double atimeToReduce, double aminimumGap);
- int ECIGetActuatedParamsMinimumGreen(int elemControl, int ajunction, int aidphase, double *aMinDurationMinGreen, double *aMaxMinGreen, double *aSecActuation);
- int ECISetActuatedParamsMinimumGreen(int elemControl, int ajunction, int aidphase, double aMinDurationMinGreen, double aMaxMinGreen, double aSecActuation);
- double ECIGetActuatedParamsMaxGreen(int elemControl, int ajunction, int aidphase);
- int ECISetActuatedParamsMaxGreen(int elemControl, int ajunction, int aidphase, double aMaxDuration);

Refer to sections 3.3.2.29 to 3.3.2.34 for more details.

1.12 Changes in version 6.1

1.12.1 Functions relative to manage Control plans

A new function to get the yellow time of a control plan phase has been added:

- int ECIGetYellowTimePhase(int idJunction,int idPhase,double timeSta,double *yellow);

Refer to section 3.3.2.12 for more information about it.

1.12.2 Functions relative to manage meterings

New functions to get and modify the parameters of the Flow-Alinea meterings has been added:

- int ECIGetParametersFlowAlineaMeteringById(int idmetering, double timeSta,double *amax, double *flow, double *amin, double *kr, double *ostar, double *intervalUpdate);
- int ECIChangeParametersFlowAlineaMeteringById(int idmetering, double timeSta, double amax, double flow, double amin, double kr, double ostar, double intervalUpdate);

Refer to section 3.3.4.14 and 3.3.4.15 for more information about them.

1.12.3 Functions relative to Traffic management

A new function to create a change speed action that affects just one lane or one segment in a section has been added:

- void * AKIActionAddDetailedSpeedAction(int sectionId, int laneId, int segmentId, double newSpeed, int vehTypePos, double acomplianceLevel);

Refer to section 3.3.7.2 for more information about them.

We have added a new function to create a close lane action specifying if the two-lanes car-following model will be considered or not.

- void * AKIActionCloseLaneDetailedAction(int sectionId, int alane, int vehTypePos, bool apply2LanesCF, double visibilityDistance);

Refer to section 3.3.7.4 for more information about them.

New functions to modify the compliance level of previously created traffic management actions have been added:

- void AKIActionModifyNextTurningODAction(void * a2kaction, double acomplianceLevel);
- void AKIActionModifyNextTurningResultAction(void * a2kaction, double acomplianceLevel);
- void AKIActionModifyChangeDestAction(void * a2kaction, double acomplianceLevel);
- void AKIActionModifyNextSubPathResultAction(void * a2kaction, double acomplianceLevel);
- void AKIActionModifyNextSubPathODAction(void * a2kaction, double acomplianceLevel);
- void AKIActionModifyCloseTurningODAction(void * a2kaction, double acomplianceLevel);

Refer to sections 3.3.7.13 to 3.3.7.17 for more information about them.

1.12.4 Functions relative to fuel consumption and pollution emission statistics

Functions to get the fuel consumption and pollution emission rates for each statistical time period as well as for the whole simulation have been added:

- double AKIEstGetGlobalStatisticsSectionFuelCons(int aidarc, int vehTypePos);
- double AKIEstGetParcialStatisticsSectionFuelCons(int aidarc, double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsTurningFuelCons(int aidsectionFrom, int aidsectionTo, int vehTypePos);
- double AKIEstGetParcialStatisticsTurningFuelCons(int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsSystemFuelCons(int vehTypePos);
- double AKIEstGetParcialStatisticsSystemFuelCons(double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsODPairFuelCons(int idOrigin, int idDestination, int vehTypePos);
- double AKIEstGetParcialStatisticsODPairFuelCons(int idOrigin, int idDestination, double time, int vehTypePos);
- double AKIEstGetParcialStatisticsStreamFuelCons(int aidstream, double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsStreamFuelCons(int aidstream, int vehTypePos);
- int AKIEstGetNbPollutants();
- const unsigned short *AKIEstGetPollutantName(int index);
- double AKIEstGetGlobalStatisticsSectionPollution(int indexPol, int aidarc, int vehTypePos);
- double AKIEstGetParcialStatisticsSectionPollution(int indexPol, int aidarc, double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsTurningPollution(int indexPol, int aidsectionFrom, int aidsectionTo, int vehTypePos);

- double AKIEstGetParcialStatisticsTurningPollution(int indexPol, int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsSystemPollution(int indexPol, int vehTypePos);
- double AKIEstGetParcialStatisticsSystemPollution(int indexPol, double timeSta, int vehTypePos);
- double AKIEstGetGlobalStatisticsODPairPollution(int indexPol, int idOrigin, int idDestination, int vehTypePos);
- double AKIEstGetParcialStatisticsODPairPollution(int indexPol, int idOrigin, int idDestination, double timeSta, int vehTypePos);
- double AKIEstGetParcialStatisticsStreamPollution(int indexPol, int aidstream, double time, int vehTypePos);
- double AKIEstGetGlobalStatisticsStreamPollution(int indexPol, int aidstream, int vehTypePos);

Refer to section 3.3.12 and subsections for more information about them.

1.12.5 Functions relative to vehicle information

The StaticInfVeh structure contains a new field to get and modify the vehicle's reaction time at traffic light. Refer to sections about obtaining and setting the information for a vehicle for more information (sections 3.3.8.20, 3.3.11.5, 3.3.11.6, 3.3.11.7, 3.3.11.8, 3.3.11.12, 3.3.11.13, 3.3.14.8, 3.3.14.9).

1.12.6 Python Version

In order to use Python to create an Aimsun API, it is required to have installed, at least, Python 2.6.2. It is possible to freely download from http://www.python.org/ website.

Please refer to the '*Before starting*' section in the Aimsun Scripting manual for information about the differences between C++ and Python syntax for functions.

1.13 Changes in version 6

1.13.1 Visual C++

In the previous version the provided project to build Aimsun API's was Visual C++ 6.0. In version 6, we provide a sample project built using Visual Studio 2005, available at **$AIMSUN_HOME/AAPI60/AAPIVisual**

1.13.2 Strings usage

In previous versions there were lots of functions using **char*** as a parameter, which could be the name of the object, the name of a column, etc. Due to the Qt change, all these function parameters have been converted into **const unsigned short***. So, there are two new functions available to transform a **char*** into **const unsigned short*** and viceversa:

- const char *AKIConvertToAsciiString(const unsigned short *string, bool deleteUshortString, bool *anyNonAsciiChar);
- const unsigned short *AKIConvertFromAsciiString(const char *ascii);

We also have added a function which deletes any unsigned short* previously created.

- void AKIDeleteUNICODEString(const unsigned short *string)

Refer to section 3.3.19 for a detailed explanation. Look at sections 6.6 and 6.7 to see some examples.

1.13.3 Basic Pointer usage

In the current version we have changed the way to create pointers to basic types such as int, double, etc.
We provide the following pointer classes:

- intp : integer pointer
- floatp: float pointer
- doublep : double pointer
- boolp: Boolean pointer

The way to assign a value to a pointer is using the function:

- void assign(type value);

The way to get the value of a pointer is using the function:

- type value();

Example:

```
aa=doublep()
aa.assign( 888.87 )
bb=aa.value()
```

An usage example is explained at Area 4.2

1.13.4 AAPIManage and AAPIPostManage functions

The four parameters that the the AAPIManage and AAPIPostManage functions receive as input: *time*, *timeSta*, *timeTrans* and *cycle* have changed from 'float' type to 'double 'type' to increase accuracy with all the possible simulation steps.

1.13.5 Functions relative to manage Meterings

In previous versions all the functions related to meterings were accesed using the section identifier where the metering was located making it impossible to have access to more than one metering per section.

New functions accessible using the metering identifier directly have been added. Old functions are kept for compatibility reasons but are considered deprecated and will be removed in future versions.

A new function to access a metering by position inside a section has also been added:

- int ECIGetMeteringIdByPos(int idsection, double position);

Refer to section 3.3.4 for the signature of all the new functions.

1.13.6 Functions relative to Actions

Functions relative to actions had the name of the vehicle type as parameters in versions prior to 6. As the name can be not unique, the functions relative to actions have been changed and the vehicle type is now identified by the position in the list of vehicle types being used. The total number of vehicles can be get using the function AKIVehGetNbVehTypes ().

Refer to section 3.3.7 for the signature of all the functions relative to actions.

Furthermore, three new functions have been added:

- void * AKIActionAddNextSubPathODAction(int ang_sectionId, int * anextSections, int aOrigin, int aDest, int vehTypePos, int asectionInPath, double acomplianceLevel);
- void * AKIActionAddNextSubPathResultAction(int ang_sectionId, int * anextSections, int vehTypePos, double acomplianceLevel);
- void * AKIActionAddCloseTurningODAction(int sectionId, int anewSection2Close, int aOrigin, int aDest, int vehTypePos, double acomplianceLevel);

Refer to sections 3.3.7.7 and 3.3.7.10 for the signature of these two functions.

1.13.7 Functions relative to Vehicles information

The function

char *AKIVehGetNameVehType (int vehTypePos)

has been considered deprecated. Three new functions have been added:

- const unsigned short *AKIVehGetVehTypeName(int vehTypePos);
- int AKIVehGetVehTypePosition(const unsigned short *aname);
- int AKIVehGetVehTypeInternalPosition(int aimsunVehicleTypeId);

Refer to section 3.3.10.2, 3.3.10.5 and 3.3.10.4 for the signature of these new functions.

New functions to access the graphical information such as the lights the vehicle has on have been added:

- int AKIVehEnableGraphicalInf()
- int AKIVehDisableGraphicalInf()
- GraphicInfVeh AKIVehGetVehicleGraphicInfSection(int asect, int indexveh);
- GraphicInfVeh AKIVehGetVehicleGraphicInfJunction(int ajunction, int indexveh);
- int AKIVehSetDrivingBackwards(int aidSec, int indexveh, bool value);

Refer to sections 3.3.11.18, 3.3.11.19, 3.3.11.20, 3.3.11.21 and 3.3.11.22 for the signature of these new functions.

1.13.8 Enrouted vehicles

The name of the vehicle attribute *guided* has been changed to attribute *enrouted* in the following structures:

- *StaticInfVeh*
- *StaticInfPTVeh*

1.14 Changes in version 5.1

1.14.1 Equipped Vehicles

The function:

bool AKIDetIsInfEquipedVehGather (int Capability)

has been changed to function

bool AKIDetIsInfEquippedVehGather (int Capability)

The name of the equipped vehicle attribute *equiped* has been changed to attribute *equipped* in the following structures:

- *StaticInfVeh*
- *StaticInfPTVeh*

The structure *EquipedInfVeh* has been changed to structure *EquippedInfVeh* .

1.14.2 Functions Relative to Manage Control Plans

The following functions have been removed.

- int ECIAddNewControl (char *name, double initime)
- int ECIRestoringInitialDurations ()

1.14.3 Retrieve current simulation time

The function AKIGetCurrentSimulationTime() described in section 3.3.16.5 has been added so that during the simulation the current simulation time can be obtained.

2 The Aimsun solution

The approach taken in Aimsun consists of considering any Advanced Telematic Application to be tested as an EXTERNAL APPLICATION that can communicate to Aimsun. The Aimsun API module gives Aimsun the ability to interface with almost any EXTERNAL APPLICATION that may need access to some internal data of Aimsun during simulation run time.

Using Aimsun functions, the detector, VMS and traffic control plan can be modelled and their attributes defined. The process of information exchange between Aimsun and the EXTERNAL APPLICATION is shown in Figure 1:

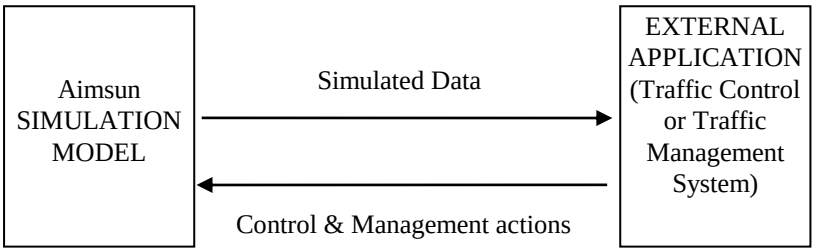

Figure 1 Process of information exchange

The Aimsun model of the road network emulates the detection process. Then, through a set of functions it provides the external application with the required "*Simulation Detection Data*" (e.g. flow, occupancy, etc.). The EXTERNAL APPLICATION (user provided) uses this data to feed some control policy and decides which control and/or management actions have to be applied on the road network. Finally, the EXTERNAL APPLICATION sends, the corresponding actuations (e.g. change the traffic signal state, the phase duration, display a message in a VMS, etc.) to the simulation model, which then emulates their operation through the corresponding model components such as traffic signals, VMS's and ramp metering signs.

Another uses of the Aimsun API are to access to detailed vehicle simulated data to feed some user's model (e.g. fuel consumption and pollution emissions), to keep track of a guided vehicle throughout the network by an external vehicle guidance system or to simulate the activities of vehicles such as floating cars.

3 Implementation

3.1 *Introduction to Aimsun API*

The Aimsun API module extends the functions of Aimsun environment including user defined applications which can exchange information and/or modify its state dynamically, with the Aimsun module.

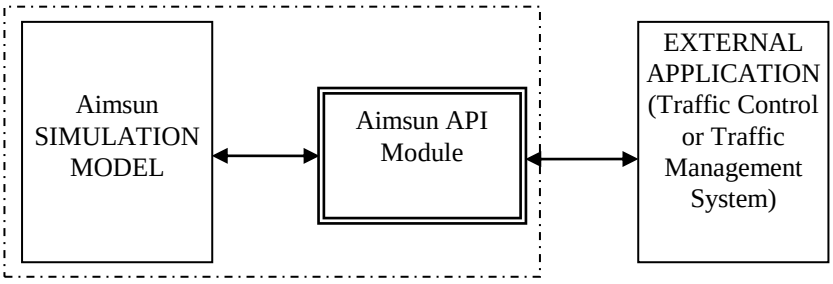

Figure 2 Schema of Aimsun API module

The Aimsun API module is placed, in the functional point of view, between the Aimsun simulation model and the external application defined by the user. Considering that, there are two types of communication processes: on one side there is a communication process between the Aimsun and the Aimsun API module and on the other side between the Aimsun API module and the external application. The communication process between Aimsun API module and the Aimsun simulation model is provided by the Aimsun environment and explained in point 3.2, but the communication between the Aimsun API module and the External Application has to be implemented by the user, depending on the requirements of the external application.

3.2 *Making an Aimsun API Module work in Aimsun: the communication process*

The Aimsun API module has six high level functions defined in order to guarantee the communication between the Aimsun API Module and the Aimsun Simulation model: AAPILoad, AAPIInit, AAPIManage, AAPIPostManage, AAPIFinish, AAPIUnLoad. It has five additional functions that are called when certain events occur. They are: AAPIEnterVehicle and AAPIExitVehicle, AAPIEnterVehicleSection, AAPIExitVehicleSection and AAPIPreRouteChoiceCalculation.

1. *AAPILoad ()*: It is called when the module is loaded by Aimsun.
2. *AAPIInit ()*: It is called when Aimsun starts the simulation and can be used to initialise whatever the module needs.
3. *AAPIManage (double time, double timeSta, double timeTrans, double cycle):* This is called in every simulation step at the beginning of the cycle, and can be used to request detector measures, vehicle information and interact with junctions, meterings and VMS in order to implement the control and management policy. This function receives four parameters in relation to time:
    - time: Absolute time of simulation in seconds. At the beginning it takes the value 0.
    - timeSta: time of simulation in stationary period, in seconds.
    - timeTrans: duration of warm-up period, in seconds
    - cycle: duration of each simulation step in seconds.

    For example:

We have started a simulation at 07:00, with 5 minutes of warm-up period and we have simulated one minute with 1 second as the simulation step. Taking these values the parameters will be:

time = 5*60 + 60 = 360.0
timeSta = 7*3600+60 = 25260.0
timeTrans = 5*60 = 300.0
cycle = 1.0

4. *AAPIPostManage (double time, double timeSta, double timeTrans, double cycle):* This is called in every simulation step at the end of the cycle, and can be used to request detector measures, vehicle information and interact with junctions, meterings and VMS in order to implement the control and management policy. This function receives four parameters in relation to time:
   - time: Absolute time of simulation in seconds. At the beginning it takes the value 0.
   - timeSta: time of simulation in stationary period, in seconds.
   - timeTrans: duration of warm-up period, in seconds
   - cycle: duration of each simulation step in seconds.

5. *AAPIFinish():* It is called when Aimsun finish the simulation and can be used to finish whatever the module needs.

6. *AAPIUnLoad()*: It is called when the module is unloaded by Aimsun.

The following figure shows graphically how Aimsun and an Aimsun API Module interact.

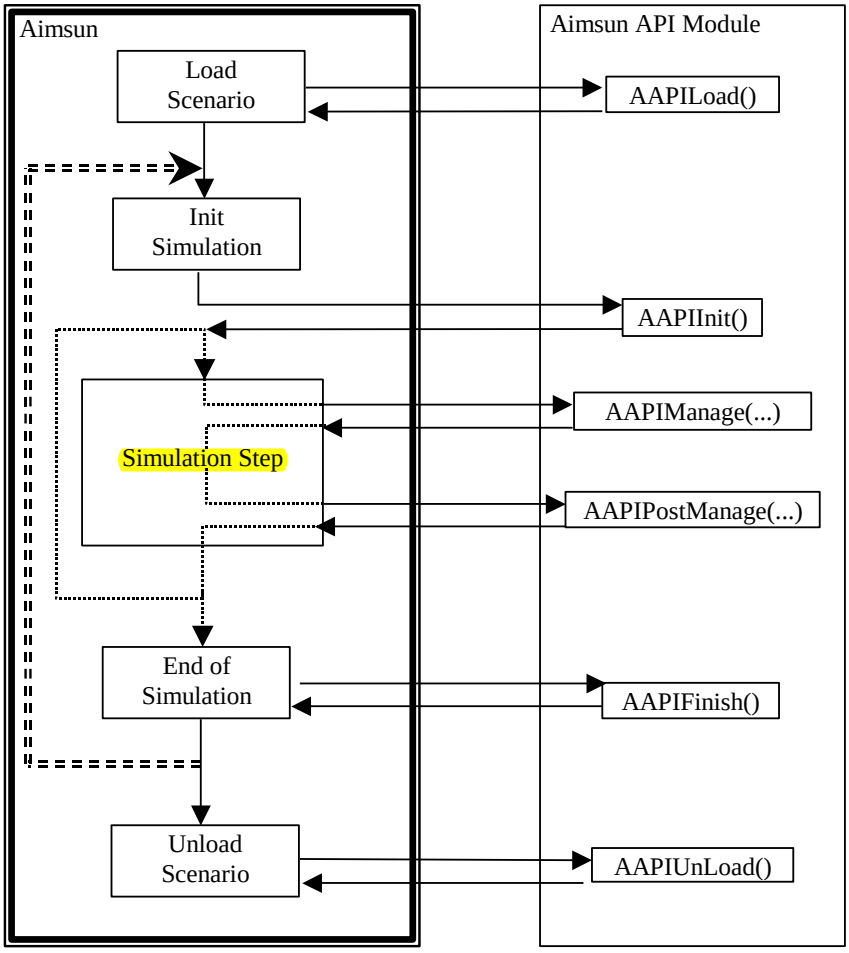

Figure 3 Scheme how Aimsun and Aimsun API Module interact

7. *AAPIEnterVehicle(int idveh, int idsectionOrTurn):* This is called when a new vehicle enters the system, that is, when the vehicle enters its first section or a turn (it is only possible when loading an initial state), not when it enters to the Virtual queue (in case it exists). This function receives two parameters:
    - idveh: Identifier of the new vehicle that enters in the system.
    - idsection: Identifier of the section where the vehicle enters in the system.

8. *AAPIExitVehicle (int idveh, int idsection):* this is called when a new vehicle exits the system. This function receives two parameters:
    - idveh: Identifier of the vehicle exits from the system.
    - idsection: Identifier of the section where the vehicle exits from the system.

9. *AAPIEnterVehicleSection(int idveh, int idsection, double atime):* This is called when a new vehicle enters a new section. This function receives three parameters:
    - idveh: Identifier of the new vehicle that enters in the system.
    - idsection: Identifier of the section the vehicle is entering in.
    - atime: absolute time of the simulation when the vehicle is entering the section.

10. *AAPIExitVehicle (int idveh, int idsection, double atime):* this is called when a new vehicle exits the system. This function receives two parameters:
    - idveh: Identifier of the vehicle exits from the system.
    - idsection: Identifier of the section the vehicle is exiting from.
    - atime: absolute time of the simulation when the vehicle is exiting the section.

11. *AAPIPreRouteChoiceCalculation(double time, double timeSta):* This function is called just before a new cycle of route choice calculation is about to begin. It can be used to modify the sections and turnings costs to affect the route choice calculation. This function receives two parameters in relation to time:
    - time: Absolute time of simulation in seconds. At the beginning it takes the value 0.
    - timeSta: time of simulation in stationary period, in seconds.

12. *char* AAPIInternalName():* returns the unique internal name of the API. It is called registering API's before simulation.

3.3 Interface provided by Aimsun to Aimsun API Module

The interaction between Aimsun and the Aimsun API Module is performed by the set of functions provided by an interface of Aimsun. This set of functions allows the Aimsun API Module to obtain information from the Aimsun simulation model and modify the simulation state.

3.3.1 Functions relative to manage control plans

3.3.1.1 Read Number of control plans loaded

In C++ and Python:

Explanation: Read the number of control plans loaded for the whole simulation.

Format:

int ECIGetNumberofControls ()

Parameters:

No parameters required.

Output:

≥ 0: Correct number of control plan

< 0: Error

3.3.1.2 Read the name of a control plan loaded

In C++:

Explanation: Read the name of a specific control plan.

Format:

const unsigned short *ECIGetNameofControl(int elemControl);

Parameters:

elemControl: the index of the control plan name to be retrieved, within the range 0 ≤ elemControl < ECIGetNumberofControls () –1

Output:

≠NULL: name of the control plan

= NULL: Error

In Python:

Not Available.

3.3.1.3 Read the Initial Time of a control plan loaded

In C++ and Python:

Explanation: Read the initial time when the control plan will be active in seconds from time 00:00:00.

Format:

double ECIGetIniTimeofControl(int elemControl);

Parameters:

elemControl: the index of the control plan name to be retrieved, within the range 0 ≤ elemControl < ECIGetNumberofControls () –1

Output:

≥ 0: initial Time

< 0: Error

3.3.1.4 Read the Offset of a control plan loaded

In C++ and Python:

Explanation: Read the offset of a control plan in seconds.

Format:

double ECIGetOffsetofControl (int elemControl);

Parameters:

elemControl: the index of the control plan for which offset is being retrieved, within the range 0 ≤ elemControl < ECIGetNumberofControls () –1

Output:

≥ 0: offset

< 0: Error

3.3.1.5 Read the name of the current control plan.

In C++:

Explanation: Read the name of the current active control plan.

Format:

const unsigned short * ECIGetNameCurrentControl ();

Parameters:

No parameters required.

Output:

≠NULL: name of the control plan

= NULL: Error

In Python:

Not Available.

3.3.1.6 Read the index of the current control plan.

In C++ and Python:

Explanation: Gives the index of the current active control plan.

Format:

int ECIGetNumberCurrentControl ();

Parameters:

No parameters required.

Output:

≥ 0: index of the control plan, 0 ≤ index < ECIGetNumberofControls () –1

< 0: Error

3.3.1.7 Remove a loaded control plan

In C++ and Python:

Explanation: Remove a previously loaded control plan using the Aimsun API Module.

Format:

int ECIRemoveControl (int elemControl);

Parameters:

elemControl: the index of the control plan to be removed, within the range

0 ≤ elemControl < ECIGetNumberofControls () – 1

Output:

= 0: No error

< 0: Error

3.3.2 Functions relative to control junctions

3.3.2.1 Read Number of junctions

In C++ and Python:

Explanation: Read the number of junctions present on the road network. A control plan must have been loaded before using this function.

Format:

int ECIGetNumberJunctions()

Parameters:

No parameters required.

Output:

≥ 0: Correct number of junctions in the road network

< 0: Error

3.3.2.2 Read the Identifier of a junction

In C++ and Python:

Explanation: Read the identifier of the elem-th junction present on the road network. A control plan must have been loaded before using this function.

Format:

int ECIGetJunctionId(int elem)

Parameters:

elem represents the index of the junction identifier to be read, within the range

0 ≤ elem < ECIGetNumberJunctions()–1

Output:

> 0: Correct identifier of the elem-th junction in the road network

< 0: Error

3.3.2.3 Read the Identifier of a junction from the External Identifier

In C++ and Python:

Explanation: Read the identifier of junction from the external identifier present on the road network. A control plan must have been loaded before using this function.

Format:

int ECIGetJunctionIdFromExternalId(const unsigned short *externalId)

Parameters:

externalId represents the external identifier for the junction

Output:

> 0: Correct identifier of the junction with external id 'externalId' in the road network

< 0: Error

3.3.2.4 Read the Name of a junction

In C++:

Explanation: Read the name of junction identified by idJunction. A control plan must have been loaded before using this function. Returns a utf16 enconded string. Has to be deleted with delete[]

Format:

const unsigned short *ECIGetJunctionName(int idJunction)

Parameters:

idJunction represents a valid junction identifier.

Output:

≠NULL: Correct name of the junction identified by idJunction

= NULL: Error

In Python:

Not Available.

3.3.2.5 Read the number of Signal Groups of a junction

In C++ and Python:

Explanation: Read the number of signal groups defined in junction identified by idJunction. A control plan must have been loaded before using this function.

Format:

int ECIGetNumberSignalGroups(int idJunction)

Parameters:

idJunction represents a valid junction identifier.

Output:

≥ 0: Total number of signal groups

< 0: Error

3.3.2.6 Read the number of Turnings assigned to a Signal Group of a junction

In C++ and Python:

Explanation: Read the number of turning movements assigned to a signal group defined in the junction identified by idJunction. A control plan must have been loaded before using this function.

Format:

int ECIGetNumberTurningsofSignalGroup (int idJunction, int asignalgroup)

Parameters:

idJunction represents a valid junction identifier.

asignalgroup represents a valid signal group identifier in the range:

1 ≤ asignalgroup ≤ ECIGetNumberSignalGroups(idJunction)

Output:

≥ 0: Total number of turnings assigned to the signal groups

< 0: Error

3.3.2.7 Read a Turning information assigned to a Signal Group of a junction

In C++ and Python:

Explanation: Read the turning information (identified by the section from and section to) assigned to the ith element of a signal group defined in the junction identified by idJunction. A control plan must have been loaded before using this function.

Format:

int ECIGetFromToofTurningofSignalGroup (int idJunction, int asignalgroup, int indexTurning, int *fromSection, int *toSection)

Parameters:

idJunction represents a valid junction identifier.

asignalgroup represents a valid signal group identifier.

1 ≤ asignalgroup ≤ ECIGetNumberSignalGroups(idJunction)

indexTurning represents a valid signal turning index, in the range:

0 ≤ indexTurning < ECIGetNumberTurningsofSignalGroup (idJunction, asignalgroup)

fromSection represents the origin section of each one of the turnings. This value is returned by the function.

toSection represents the destination section of the turning. This value is returned by the.

Output:

fromSection and toSection as the origin and destination sections of the turning

= 0: No Error

< 0: Error

Sample code in Python:

The following code reads the origin and destination sections of the first turning assigned to the first signal group in node 203:

```
fromSection = intp()
toSection = intp()
report = ECIGetFromToofTurningofSignalGroup (203, 1, 0, fromSection, toSection)
if report == 0:
   string = "From Section: " + str(fromSection.value())
   AKIPrintString(string)
   string = "To Section: " + str(toSection.value())
   AKIPrintString(string)
   AKIPrintString("OK")
else :
   AKIPrintString("Not OK")
```

3.3.2.8 Read the logical name of Signal Groups of a junction

In C++:

Explanation: Read the logical name of signal groups defined in junction identified by idJunction. A control plan must have been loaded before using this function. Returns a utf16 enconded string. Has to be deleted with delete[]

Format:

const unsigned short *ECIGetLogicalNameofSignalGroup(int ajunction, int asignalgroup)

Parameters:

idJunction represents a valid junction identifier.

asignalgroup represents a valid signal group identifier.

1 ≤ asignalgroup ≤ ECIGetNumberSignalGroups(idJunction)

Output:

≠NULL: Correct logical name of the signal group

= NULL: Error

In Python:
Not Available.

3.3.2.9 Read the Control Type of a junction

In C++ and Pyhon:
Explanation: Read the control type of a junction in the current control plan.
Format:
int ECIGetControlType (int idJunction)
Parameters:
idJunction represents a valid a junction identifier.
Output:
≥ 0: 0 Uncontrolled, 1Fixed, 2 External, 3 Actuated
< 0: Error

3.3.2.10 Read the Number of Phases of a junction

In C++ and Python:
Explanation: Read the total number of phases of a junction in the current control plan.
Format:
int ECIGetNumberPhases (int idJunction)
Parameters:
idJunction represents a valid junction identifier.
Output:
≥ 0: Total number of phases returned
< 0: Error

3.3.2.11 Read Time Duration of a phase of a junction

In C++ and Python:
Explanation: Read the maximum, minimum and current duration of a phase in junction during the current control plan. These times are returned in the parameters max, min and dur respectively.
Format:
int ECIGetDurationsPhase(int idJunction, int idPhase, double timeSta,
double *dur, double *max, double *min)
Parameters:
idJunction represents a valid junction identifier.
idPhase represents a valid phase identifier. This identifier is defined by the interval [1 ... Total ECIGetNumberPhases (idJunction)]
timeSta represent the absolute time of simulation (current time)
dur represents the current duration of the phase. This value is returned by the function.
max represents the maximum duration available for the phase. This value is returned by the function.
min represents the minimum duration available for the phase. This value is returned by the function.
Output:
= 0: No error occurred.
< 0: Error

3.3.2.12 Read Yellow Time Duration of a phase of a junction

In C++ and Python:
Explanation: Read the yellow time duration of a phase in junction during the current control plan. The time is returned in the yellow parameter.

Format:

int ECIGetYellowTimePhase(int idJunction,int idPhase,double timeSta,double *yellow)

Parameters:

idJunction represents a valid junction identifier.

idPhase represents a valid phase identifier. This identifier is defined by the interval [1 ... Total ECIGetNumberPhases (idJunction)]

timeSta represent the absolute time of simulation (current time)

yellow represents the yellow time duration of the phase. This value is returned by the function.

Output:

= 0: No error occurred.

< 0: Error

3.3.2.13 Check whether a phase of a junction is an interphase or not

In C++ and Python:

Explanation: Check whether a phase in junction during the current control plan has been defined as interphase by the user or not.

Format:

int ECIIsAnInterPhase(int idJunction,int idPhase,double timeSta)

Parameters:

idJunction represents a valid junction identifier.

idPhase represents a valid phase identifier. This identifier is defined by the interval [1 ... Total Number of phases]

timeSta represent the absolute time of simulation (current time)

Output:

≥ 0: 1 means the phase is an interphase, 0 otherwise.

< 0: Error

3.3.2.14 Read the number of Signal Groups assigned to a phase of a junction

In C++ and Python:

Explanation: Read the number of signal groups assigned to a specific phase.

Format:

int ECIGetNbSignalGroupsPhaseofJunction (int idJunction,int idPhase,double timeSta)

Parameters:

idJunction represents a valid junction identifier.

idPhase represents a valid phase identifier. This identifier is defined by the interval [1 ... Total Number of phases]

timeSta represent the absolute time of simulation (current time) in seconds

Output:

≥ 0: The number of signal groups assigned.

< 0: Error

3.3.2.15 Read the Signal Group Identifier assigned to a phase of a junction

In C++ and Python:

Explanation: Read the signal group Id assigned to a specific phase.

Format:

int ECIGetSignalGroupPhaseofJunction (int idJunction,int idPhase, int indexSG, double timeSta)

Parameters:

idJunction represents a valid junction identifier.

idPhase represents a valid phase identifier. This identifier is defined by the interval [1 ... Total Number of phases]

indexSG represents a valid index of a signal group. This index is defined by the interval [0 ... ECIGetNbSignalGroupsPhaseofJunction-1]
timeSta represent the absolute time of simulation (current time)

Output:
≥ 0: The signal groups id.
< 0: Error

3.3.2.16 Read the Current Yellow Time of a junction

In C++ and Python:

Explanation: Read the yellow time of a junction in the current control, even when the junction has disabled the fixed control plan.

Format:
int ECIGetYellowTime(int idJunction)

Parameters:
idJunction represents a valid junction identifier.

Output:
≥ 0: Current yellow time returned, expressed in seconds
< 0: Error

3.3.2.17 Read the Current Offset of a junction

In C++ and Python:

Explanation: Read the offset of a junction in the current control, even when the junction has disabled the fixed control plan.

Format:
int ECIGetOffset(int idJunction)

Parameters:
idJunction represents a valid junction identifier.

Output:
≥ 0: Current offset returned, expressed in seconds
< 0: Error

3.3.2.18 Read the Current Phase of a junction

In C++ and Python:

Explanation: Read the current phase of a junction, even when the junction has disabled the fixed control plan.

Format:
int ECIGetCurrentPhase(int idJunction)

Parameters:
idJunction represents a valid junction identifier.

Output:
> 0: Current phase returned
< 0: Error

3.3.2.19 Read the Starting Time of the Current Phase of a junction

In C++ and Python:

Explanation: Get the time when the current phase of a junction, even when the junction has been disabled in the fixed control plan, has been activated. This time is relative to Time (that is to say the beginning of the simulation)

Format:
double ECIGetStartingTimePhase (int idJunction)

Parameters:
idJunction represents a valid junction identifier.

Output:
> 0: Starting Time (relative to Time) returned

< 0: Error

3.3.2.20 Disable the control plan of a junction

In C++ and Python:

Explanation: Disables the current control plan of a junction, so the phase changing is completely controlled by the Aimsun API Module. It only works in External and Actuated control plans.

Format:

int ECIDisableEvents(int idJunction)

Parameters:

idJunction represents a valid junction identifier.

Output:

= 0: No error occurred

< 0: Error

3.3.2.21 Enable the control plan of a junction

In C++ and Python:

Explanation: Enables the current control plan of a junction, so the phase changing is completely controlled by Aimsun, taking into account the duration of each phase. It only works in External and Actuated control plans.

Format:

int ECIEnableEvents(int idJunction)

Parameters:

idJunction represents a valid junction identifier.

Output:

= 0: No error occurred

< 0: Error

3.3.2.22 Enable the control plan of a junction activating a phase

In C++ and Python:

Explanation: Enables the current control plan of a junction starting by the given phase. It only works in External and Actuated control plans.

Format:

int ECIEnableEventsActivatingPhase(int idJunction, int idPhaseToActivateNow,
double expiredTime)

Parameters:

idJunction represents a valid junction identifier.

idPhaseToActivateNow is the phase to start the control plan

expiredTime represents the seconds from the beginning of the phase. Note that it cannot be greater than the idPhaseToActivateNow duration, otherwise it will skip idPhaseToActivateNow.

Output:

= 0: No error occurred

< 0: Error

3.3.2.23 Direct Change of Phase

In C++ and Python:

Explanation: Changes to idPhase in junction identified by idJunction. If the events are enabled, Aimsun programmes the next phase change, taking into account the current duration of idPhase.

Format:

int ECIChangeDirectPhase(int idJunction, int idPhase, double timeSta, double time, double cycle, double expiredTime)

Parameters:

idJunction represents a valid junction identifier.
idPhase represents a valid phase identifier. This identifier is defined by the interval [1 ... Total Number of phases]
timeSta represents the absolute time of simulation
time represents a relative time of simulation in seconds.
cycle represents the duration of each simulation step in seconds.
expiredTime represents the seconds from the beginning of the phase. Note that it cannot be greater than the idPhase duration, otherwise it will skip idPhase.

Output:

= 0: No error occurred.
< 0: Error

3.3.2.24 Direct Change of Phase with Transition

In C++ and Python:

Explanation: Changes to idPhase in junction identified by idJunction, calculating automatically the transition from the current phase to idPhase. The transition is calculated using the following approach,:

```
if the current phase CP is not an interphase
        if CP has an interphase I defined as next phase
                Calculate the transition from CP to idPhase
                (setting green all signal groups active in both phases)
                Activate the interphase I with the duration defined
                (with the signal group states calculated)
                After the duration of interphase I, activate phase idPhase
        else
                Activate directly phase idPhase (without transition)
        endif
else
        Activate directly phase idPhase (without transition)
endif
```

If the events are enabled, Aimsun programmes the next phase change, taking into account the current duration of each phase. Otherwise the next phase change has to be done by the Aimsun API Module. IMPORTANT: this function requires a phase to be active, that means the transition will be calculated incorrectly when the state of signal group has been defined calling the function ECIChangeSignalGroupState, because this call leads to loss of the “current” phase.

Format:

int ECIChangeDirectPhaseWithInterphaseTransition (int idJunction, int idPhase, double timeSta, double time, double cycle)

Parameters:

idJunction represents a valid junction identifier.
idPhase represents a valid phase identifier. This identifier is defined by the interval [1 ... ECIGetNumberPhases (idJunction)]
timeSta represents the absolute time of simulation
time represents a relative time of simulation in seconds.
cycle represents the duration of each simulation step in seconds.

Output:

= 0: No error occurred.
< 0: Error

3.3.2.25 Change the Current Duration of Phase

In C++ and Python:

Explanation: Changes the duration of idPhase in the junction identified by idJunction in the current control plan. This change obviously only has sense when this junction has enabled the events.

Format:

int ECIChangeTimingPhase(int idJunction, int idPhase, double newTime, double timeSta)

Parameters:

idJunction represents a valid junction identifier.

idPhase represents a valid phase identifier. This identifier is defined by the interval [1 ... Total Number of phases] (that is known using the function ECIGetNumberPhases (idJunction))

newTime represents new duration of idPhase, in seconds.

timeSta represents the absolute time of simulation

Output:

= 0: No error occurred.

< 0: Error

3.3.2.26 Directly change the State of a Signal Group

In C++ and Python:

Explanation: Changes the state of signal group idSigGr in junction identified by idJunction. The events have to be disabled.

Format:

int ECIChangeSignalGroupState(int idJunction, int idSigGr, int aState, double timeSta, double time, double cycle)

int ECIChangeSignalGroupStatebyName(int idJunction, const unsigned short *nameSignalGroup, int aState, double timeSta, double time, double cycle);

Parameters:

idJunction represents a valid junction identifier.

idSigGr represents a valid Signal Group identifier. This identifier is defined by the interval [1 ... Total Number of Signal Groups in the junction]

aState represents the new state (0 means RED, 1 means GREEN, 2 means YELLOW, 3 means FLASHING GREEN, 4 means FLASHING RED and 5 means FLASHING YELLOW)

time represents the absolute time of simulation

timeSta represents a relative time of simulation in seconds.

cycle represents the duration of each simulation step in seconds.

nameSignalGroup represents the logical name of the signal group

Output:

= 0: No error occurred.

< 0: Error

3.3.2.27 Directly change the State of a Signal Group to Yellow

In C++ and Python:

Explanation: Changes the state of signal group idSigGr in junction identified by idJunction. The events have to be disabled.

Format:

int ECIChangeSignalGroupStateToYellow(int idJunction, int idSigGr, int aState, double timeSta, double time, double cycle, double greenTime)

Parameters:

idJunction represents a valid junction identifier.

idSigGr represents a valid Signal Group identifier. This identifier is defined by the interval [1 ... Total Number of Signal Groups in the junction]
aState represents the new state (0 means RED, 1 means GREEN, 2 means YELLOW, 3 means FLASHING GREEN, 4 means FLASHING RED and 5 means FLASHING YELLOW)
time represents the absolute time of simulation
timeSta represents a relative time of simulation in seconds.
cycle represents the duration of each simulation step in seconds.
nameSignalGroup represents the logical name of the signal group
greenTime represents the number of seconds from the moment the light changes to yellow that the vehicles will still cross the traffic light.

Output:
= 0: No error occurred.
< 0: Error

3.3.2.28 Get Current State of a Signal Group

In C++ and Python:

Explanation: Get the state of signal group idSigGr in junction identified by idJunction.
Format:

```
int ECIGetCurrentStateofSignalGroup(int ajunction, int asignalgroup)
int ECIGetCurrentStateofSignalGroupbyName(int idJunction, const unsigned short *nameSignalGroup);
```

Parameters:
idJunction represents a valid junction identifier.
idSigGr represents a valid Signal Group identifier. This identifier is defined by the interval [1 ... Total Number of Signal Groups in the junction].
nameSignalGroup represents the logical name of the signal group

Output:
≥ 0: The current state. (0 means RED, 1 means GREEN, 2 means YELLOW)
< 0: Error

3.3.2.29 Get the Actuated Gap Reduction and Passage Time Parameters

In C++ and Python:

Explanation: Get the actuated passage time parameters.
Format:

```
int ECIGetActuatedParamsPassageTime(int elemControl, int ajunction, int aidphase, double *apassageTime, double *atimeBeforeReduce, double *atimeToReduce, double *aminimumGap);
```

Parameters:
elemControl represents the index of the junction identifier to be read, within the range 0 ≤ elem < ECIGetNumberJunctions()–1
idJunction represents a valid junction identifier.
aidphase represents a valid Phase identifier. This identifier is defined by the interval [1 ... Total Number of phases].

Output:
≥ 0: apassagetTime, atimeBeforeReduce, atimeToReduce and aminimumGap contain the related values
< 0: Error

3.3.2.30 Set the Actuated Gap Reduction and Passage Time Parameters

In C++ and Python:

Explanation: Set the actuated passage time parameters.
Format:

int ECISetActuatedParamsPassageTime(int elemControl, int ajunction, int aidphase, double apassageTime, double atimeBeforeReduce, double atimeToReduce, double aminimumGap);

Parameters:

elemControl represents the index of the junction identifier to be read, within the range 0 ≤ elem < ECIGetNumberJunctions()-1
idJunction represents a valid junction identifier.
aidphase represents a valid Phase identifier. This identifier is defined by the interval [1 ... Total Number of phases].
apassageTime: new passage time
atimeBeforeReduce: new time before reduce
atimeToReduce: new time to reduce
aminimumGap: new minimum gap

Output:

≥ 0: Success
< 0: Error

3.3.2.31 Get the Actuated Minimum Green Time Parameters

In C++ and Python:

Explanation: Get the actuated Minimum Green time parameters.
Format:

int ECIGetActuatedParamsMinimumGreen(int elemControl, int ajunction, int aidphase, double *aMinDurationMinGreen, double *aMaxMinGreen, double *aSecActuation);

Parameters:

elemControl represents the index of the junction identifier to be read, within the range 0 ≤ elem < ECIGetNumberJunctions()-1
idJunction represents a valid junction identifier.
aidphase represents a valid Phase identifier. This identifier is defined by the interval [1 ... Total Number of phases].

Output:

≥ 0: aMinDurationMinGreen, aMaxMinGreen and aSecActuation contain the related values
< 0: Error

3.3.2.32 Set the Actuated Minimum Green Time Parameters

In C++ and Python:

Explanation: Set the actuated Minimum Green Time parameters.
Format:

int ECISetActuatedParamsMinimumGreen(int elemControl, int ajunction, int aidphase, double aMinDurationMinGreen, double aMaxMinGreen, double aSecActuation);

Parameters:

elemControl represents the index of the junction identifier to be read, within the range 0 ≤ elem < ECIGetNumberJunctions()-1
idJunction represents a valid junction identifier.
aidphase represents a valid Phase identifier. This identifier is defined by the interval [1 ... Total Number of phases].
aMinDurationMinGreen: new minimum green time
aMaxMinGreen: new maximum initial green time
aSecActuation: new seconds actuation time

Output:

≥ 0: Success
< 0: Error

3.3.2.33 Get the Actuated Maximum Green Time

In C++ and Python:

Explanation: Get the actuated Maximum Green time.

Format:

double ECIGetActuatedParamsMaxGreen(int elemControl, int ajunction, int aidphase);

Parameters:

elemControl represents the index of the junction identifier to be read, within the range 0 ≤ elem < ECIGetNumberJunctions()–1

idJunction represents a valid junction identifier.

aidphase represents a valid Phase identifier. This identifier is defined by the interval [1 ... Total Number of phases].

Output:

≥ 0: Maximum Green Time
< 0: Error

3.3.2.34 Set the Actuated Maximum Green Time

In C++ and Python:

Explanation: Set the actuated Maximum Green Time.

Format:

int ECISetActuatedParamsMaxGreen(int elemControl, int ajunction, int aidphase, double aMaxDuration);

Parameters:

elemControl represents the index of the junction identifier to be read, within the range 0 ≤ elem < ECIGetNumberJunctions()–1

idJunction represents a valid junction identifier.

aidphase represents a valid Phase identifier. This identifier is defined by the interval [1 ... Total Number of phases].

aMaxDuration: new maximum green time

Output:

≥ 0: Success
< 0: Error

3.3.2.35 Get the Actuated Force Off and Permissive Periods

In C++ and Python:

Explanation: Get the actuated Force Off and permissive periods defined for a junction.

Format:

int ECIGetActuatedParamsForceOFFPermissivePeriod(int elemControl, int ajunction, int aidphase, double *forceOFF, double *permissivePeriodFrom, double *permissivePeriodTo);

Parameters:

elemControl represents the index of the junction identifier to be read, within the range 0 ≤ elem < ECIGetNumberJunctions()–1

idJunction represents a valid junction identifier.

aidphase represents a valid Phase identifier. This identifier is defined by the interval [1 ... Total Number of phases].

forceOFF represents the force off defined for the phase. This value is returned by the function.

permissivePeriodFrom represents the permissive period from defined for the phase. This value is returned by the function.

permissivePeriodTo represents the permissive period to defined for the phase. This value is returned by the function.

Output:

≥ 0: Maximum Green Time
< 0: Error

3.3.2.36 Set the Actuated Force Off and Permissive Periods

In C++ and Python:

Explanation: Set the actuated Force Off and permissive periods for a junction.

Format:

int ECISetActuatedParamsForceOFFPermissivePeriod(int elemControl, int ajunction, int aidphase, double forceOFF, double permissivePeriodFrom, double permissivePeriodTo);

Parameters:

elemControl represents the index of the junction identifier to be read, within the range 0 ≤ elem < ECIGetNumberJunctions()–1
idJunction represents a valid junction identifier.
aidphase represents a valid Phase identifier. This identifier is defined by the interval [1 ... Total Number of phases].
forceOFF represents the force off defined for the phase.
permissivePeriodFrom represents the permissive period from defined for the phase.
permissivePeriodTo represents the permissive period to defined for the phase.

Output:

≥ 0: Maximum Green Time
< 0: Error

3.3.2.37 Get the Total Green Time of a Signal Group

In C++ only:

Explanation: Get the total time that a signal group will be green in the current control plan.

Format:

int ECIGetSignalGroupGreenDuration(int idJunction, int signalPos, double timeSta);

Parameters:

idJunction represents a valid junction identifier.
signalPos represents a valid signal group position in the junction between 1 and ECIGetNumberSignalGroups(idJunction)
timeSta is the current stationary time, used to find the control plan that is being executed

Output:

≥ 0: No Error
the total green duration
< 0: Error

3.3.2.38 Get the Total Yellow Time of a Signal Group

In C++ only:

Explanation: Get the total time that a signal group will be yellow in the current control plan.

Format:

int ECIGetSignalGroupYellowDuration(int idJunction, int signalPos, double timeSta);

Parameters:

idJunction represents a valid junction identifier.

signalPos represents a valid signal group position in the junction between 1 and ECIGetNumberSignalGroups(idJunction)

timeSta is the current stationary time, used to find the control plan that is being executed

Output:

≥ 0: No Error
the total yellow duration
< 0: Error

3.3.2.39 Get the Total Red Time of a Signal Group

In C++ only:

Explanation: Get the total time that a signal group will be red in the current control plan.

Format:

int ECIGetSignalGroupRedDuration(int idJunction, int signalPos, double timeSta);

Parameters:

idJunction represents a valid junction identifier.

signalPos represents a valid signal group position in the junction between 1 and ECIGetNumberSignalGroups(idJunction)

timeSta is the current stationary time, used to find the control plan that is being executed

Output:

≥ 0: No Error
the total red duration
< 0: Error

3.3.3 Functions relative to public transport preemption

3.3.3.1 Disable public transport pre-emption in a node

In C++ and Python:

Explanation:Sets the specified intersection to ignore the public transport pre-emptions defined.

Format:

int ECIDisableBusPreemptionNode(int ajunction)

Parameters:

ajunction is the ID of the intersection where the public transport pre-emption will be disabled.

Output:

0: If the bus pre-emption has been correctly disabled.
< 0: Error

3.3.3.2 Enable public transport pre-emption in a node

In C++ and Python:

Explanation:Sets the specified intersection to consider the public transport pre-emptions defined after they have been previously disabled.

Format:

int ECIEnableBusPreemptionNode (int ajunction)

Parameters:

ajunction is the ID of the intersection where the public transport pre-emption will be considered again.

Output:

0: If the bus pre-emption has been correctly enabled.

< 0: Error

3.3.3.3 Check public transport pre-emption state in a node

In C++ and Python:

Explanation: Returns whether the public transport pre-emptions defined are currently considered or ignored.

Format:

bool ECIIsBusPreemptionNodeEnabled (int ajunction)

Parameters:

ajunction is the ID of the intersection where the public transport pre-emption will be disabled.

Output:

true: If the bus pre-emptions are considered as defined.

false: If the bus pre-emptions defined have been disabled.

3.3.3.4 Get the Number of Preemption Sets

In C++ and Python:

Explanation: Get the Number of Preemption sets for a junction in the current control plan.

Format:

int ECIGetNbPreemptionSets(int idJunction, double timeSta);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

Output:

≥ 0: Number of preemption sets

< 0: Error

3.3.3.5 Get the Preemption Set Parameters

In C++ and Python:

Explanation: Get the Preemption set parameters for a junction in the current control plan.

Format:

int ECIGetPreemptionSetParameters(int idJunction, double timeSta, int index, double* delay, double *minDwell, double *reserve, double * inhibit, double *maxDwell, int *type);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

Output:

≥ 0: No Error

delay represents the number of seconds to delay the start of the preemption timing
minDwell represents the minimum duration in seconds of the dwell phase
reserve represents the time during which any other priority request is prevented.
inhibit represents the duration in seconds of the interval while the activation of any phase that is not the requested priority phase is prevented
maxDwell represents the maximum duration in seconds of the dwell phase
type represents the preemption type. 0 means Alternative, 1 means Serve All

< 0: Error

Sample code in Python:
The following code reads the first preemption parameters of node 203 at time 0:00:00 and if they are read successfully it prints the delay read:

```
delay = doublep()
minDwell = doublep()
reserve = doublep()
inhibit = doublep()
maxDwell = doublep()
type = intp()
report = ECIGetPreemptionSetParameters( 203, 0.0, 0, delay, minDwell, reserve, inhibit, maxDwell, type )
if report == 0:
   string = "Delay: " + str(delay.value())
   AKIPrintString(string)
   AKIPrintString("OK")
else :
   AKIPrintString("Not OK")
```

3.3.3.6 Get the Preemption Set Number of Public Transport Lines

In Python only:
Explanation: Get the Preemption set number of public transport lines that are considered for a junction in the current control plan.
Format:

int ECIGetPreemptionSetNbLinesPython(int idJunction, double timeSta, int index);

Parameters:

idJunction represents a valid junction identifier.
timeSta is the current stationary time, used to find the control plan that is being executed
index is a value between 0 and the ECIGetNbPreemptionSets -1

Output:

≥ 0: the number of lines in the preemption set
< 0: Error

3.3.3.7 Get the Preemption Set Public Transport Lines

C++ version:
Explanation: Get the Preemption set public transport lines that are considered for a junction in the current control plan.
Format:

int ECIGetPreemptionSetLines(int idJunction, double timeSta, int index, int* nbLines , int *lines);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

Output:

≥ 0: No Error

nbLines is the number of lines that will be returned

lines is a vector of line identifiers

< 0: Error

Python version:

Explanation: Get the Preemption set public transport lines that are considered for a junction in the current control plan.

Format:

int ECIGetPreemptionSetLinesPython(int idJunction, double timeSta, int index, int nbLines , int *lines);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

nbLines is the number of lines returned by ECIGetPreemptionSetNbLinesPython

lines is an integer array of size equal to nbLines.

Output:

≥ 0: No Error

lines is a vector of line identifiers

< 0: Error

Sample code in Python:

```
nblines = ECIGetPreemptionSetNbLinesPython( 203, 0.0 )
lines = intArray( nblines )
report = ECIGetPreemptionSetLinesPython( 203, 0.0, 0, nblines, lines )
if report == 0:
        string = "Number of lines: " + str(nblines)
        AKIPrintString(string)
        for i in range(nblines):
                string = "Line " + str(i) + ": " + str(lines[i])
                AKIPrintString(string)
```

3.3.3.8 Get the Preemption Set Number of Phases

In Python only:

Explanation: Get the Preemption set number of dwell phases for a junction in the current control plan.

Format:

int ECIGetPreemptionSetNbPhasesPython(int idJunction, double timeSta);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

Output:

≥ 0: number of phases

< 0: Error

3.3.3.9 Get the Preemption Set Phases

C++ version:

Explanation: Get the Preemption set dwell phases for a junction in the current control plan.

Format:

int ECIGetPreemptionSetPhases(int idJunction, double timeSta, int index, int* nbPhases , int *phases);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

Output:

≥ 0: No Error

nbPhases is the number of phases that will be returned

phases is a vector of phase identifiers, which are between 1 and N, where N is the number of phases

< 0: Error

Python version:

Explanation: Get the Preemption set dwell phases for a junction in the current control plan.

Format:

int ECIGetPreemptionSetPhasesPython(int idJunction, double timeSta, int index, int nbPhases , int *phases);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

nbPhases is the number of phases

phases is an int array of size equal to nbPhases

Output:

≥ 0: No Error

phases is a vector of phase identifiers, which are between 1 and N, where N is the number of phases

< 0: Error

Sample code in Python:

```
nbphases = ECIGetPreemptionSetNbPhasesPython( 203, 0.0 )
phases  = intArray(nbphases)
report = ECIGetPreemptionSetPhasesPython( 203, 0.0, 0, nbphases, phases )
if report == 0:
        string = "Number of phases: " + str(nbphases)
        AKIPrintString(string)
        for i in range(nbphases):
                string = "Line " + str(i) + ": " + str(phases[i])
                AKIPrintString(string)
```

3.3.3.10 Get the Preemption Set Number of Request Detectors

In Python only:

Explanation: Get the Preemption set number of request detectors for a junction in the current control plan.

Format:

int ECIGetPreemptionSetNbRequestDetectorsPython (int idJunction, double timeSta, int index);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

Output:

≥ 0: The number of request detectors

< 0: Error

3.3.3.11 Get the Preemption Set Request Detectors

C++ version:

Explanation: Get the Preemption set request detectors for a junction in the current control plan.

Format:

int ECIGetPreemptionSetRequestDetectors (int idJunction, double timeSta, int index, int* nbDetectors, int * detectors);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

Output:

≥ 0: No Error

nbDetectors is the number of request detectors that will be returned

detectors is a vector of request detectors identifiers

< 0: Error

Python version:

Explanation: Get the Preemption set request detectors for a junction in the current control plan.

Format:

int ECIGetPreemptionSetRequestDetectorsPython (int idJunction, double timeSta, int index, int nbDetectors, int * detectors);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

nbDetectors is the number of request detectors

detectors is an int array of size equal to nbDetectors.

Output:

≥ 0: No Error

detectors is a vector of request detectors identifiers

< 0: Error

Sample code in Python:

```
nbdets = ECIGetPreemptionSetNbRequestDetectorsPython( 203, 0.0 )
dets  = intArray(nbdets)
report = ECIGetPreemptionSetRequestDetectorsPython( 203, 0.0, 0, nbdets, dets
)
if report == 0:
        string = "Number of detectors: " + str(nbdets)
        AKIPrintString(string)
        for i in range(nbphases):
                string = "Detector " + str(i) + ": " + str(dets[i])
                AKIPrintString(string)
```

3.3.3.12 Get the Preemption Set Number of End Detectors

In Python only:

Explanation: Get the Preemption set number of end detectors for a junction in the current control plan.

Format:

int ECIGetPreemptionSetNbEndDetectorsPython (int idJunction, double timeSta, int index);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

Output:

≥ 0: The number of end detectors

< 0: Error

3.3.3.13 Get the Preemption Set End Detectors

C++ version:

Explanation: Get the Preemption set end detectors for a junction in the current control plan.

Format:

int ECIGetPreemptionSetEndDetectors (int idJunction, double timeSta, int index, int* nbDetectors, int * detectors);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed

index is a value between 0 and the ECIGetNbPreemptionSets -1

Output:

≥ 0: No Error

nbDetectors is the number of end detectors that will be returned

detectors is a vector of end detectors identifiers

< 0: Error

Python version:

Explanation: Get the Preemption set end detectors for a junction in the current control plan.

Format:

int ECIGetPreemptionSetEndDetectors (int idJunction, double timeSta, int index, int nbDetectors, int * detectors);

Parameters:

idJunction represents a valid junction identifier.

timeSta is the current stationary time, used to find the control plan that is being executed
index is a value between 0 and the ECIGetNbPreemptionSets -1
nbDetectors is the number of end detectors
detectors is an int array of size equal to nbDetectors

Output:
≥ 0: No Error
detectors is a vector of end detectors identifiers
< 0: Error

Sample code in Python:

```
nbdets = ECIGetPreemptionSetNbEndDetectorsPython( 203, 0.0 )
dets  = intArray(nbdets)
report = ECIGetPreemptionSetEndDetectorsPython( 203, 0.0, 0, nbdets, dets )
if report == 0:
        string = "Number of detectors: " + str(nbdets)
        AKIPrintString(string)
        for i in range(nbphases):
                string = "Detector " + str(i) + ": " + str(dets[i])
                AKIPrintString(string)
```

3.3.4 Functions relative to meterings

3.3.4.1 Read Number of meterings

In C++ and Python:

Explanation: Read the number of meterings present on the road network.
Format:
int ECIGetNumberMeterings ()
Parameters: No parameters required.
Output:
≥ 0: Correct number of meterings in the road network
< 0: Error

3.3.4.2 Read the Section Identifier of a metering

In C++ and Python:

Explanation: Read the section identifier that contains the elem-th metering present on the road network.
Format:
int ECIGetMeteringIdSection(int elem)
Parameters:
elem represents the index of the metering within the range 0 ≤ elem < Number of Meterings –1
Output:
> 0: Correct section identifier that contains the elem-th metering.
< 0: Error

3.3.4.3 Read the Metering Identifier

In C++ and Python:

Explanation: Read the metering identifier that contains the elem-th metering present on the road network.
Format:
int ECIGetMeteringIdByPosition (int elem)
Parameters:

elem represents the index of the metering within the range 0 ≤ elem < Number of Meterings –1

Output:

> 0: Correct metering identifier.

< 0: Error

3.3.4.4 Read the Metering Identifier (Deprecated function)

In C++ and Python:

Explanation: Read the metering identifier from a section. This function has been deprecated in version 6. Use the *ECIGetMeteringIdByPosition* function instead.

Format:

int ECIGetMeteringId (int idsection)

Parameters:

idsection represents the section identifier for the section where the metering is located

Output:

> 0: Correct metering identifier.

< 0: Error

3.3.4.5 Read the Metering Identifier

In C++ and Python:

Explanation: Read the metering identifier from a section and a position inside the section (from the beginning of the section).

Format:

int ECIGetMeteringIdByPos (int idsection, double position)

Parameters:

idsection represents the section identifier for the section where the metering is located

position represents the position from the beginning of the section where the metering is located, in the network units (that is, meters or feet).

Output:

> 0: Correct metering identifier.

< 0: Error

3.3.4.6 Read the Name of a metering

In C++:

Explanation: Read the name of a metering present in a section.

Format:

const unsigned short *ECIGetMeteringNameById(int idmetering);

*const unsigned short *ECIGetMeteringName(int idsection) (Deprecated function)*

Parameters:

idmetering represents the metering identifier

idsection represents the section identifier for the section where the metering is located.

Output:

≠NULL: Correct name of the metering

= NULL: Error

In Python:

Not Available.

3.3.4.7 Read the Type of metering

In C++ and Python:

Explanation: Read the type of metering present in a section. The type of a metering can be 0: None; 1: GREEN METERING; 2: FLOW METERING; 3: DELAY METERING; 4: FLOW-ALINEA METERING; 5: GREEN BY LANE METERING.
Format:

int ECIGetTypeMeteringById(int idmetering);
int ECIGetTypeMetering (int idsection) (Deprecated function)

Parameters:
idmetering represents the metering identifier
idsection represents a valid identifier of a section.

Output:
≥ 0: No error occurred.
< 0: Error

3.3.4.8 Read the Control Parameters of a Green Metering

In C++ and Python:

Explanation: Read the parameters of a green metering that are defined in the current control.
Format:

int ECIGetParametersGreenMeteringById(int idmetering, double timeSta,
double *amax, double *greenTime, double *amin,
double *cycleTime, double *offset, double *yellowTime);
int ECIGetParametersGreenMetering (int idsection, double timeSta,
*double *amax, double *greenTime, double *amin,*
*double *cycleTime, double *offset, double *yellowtime)*
(Deprecated funtion)

Parameters:
idmetering represents the metering identifier
idsection represents a valid section identifier.
timeSta represents the absolute time of simulation in stationary period, in seconds.
amax: Value returns the current maximum green time in seconds.
greenTime: Value returns the current green time, in seconds.
amin: Value returns the current minimum green time, in seconds.
cycleTime: Value returns the current cycle time, in seconds.
offset: Value returns the offset, in seconds.
yellowtime: Value returns the yellow time, in seconds.

Output:
= 0: No error occurred.
< 0: Error

3.3.4.9 Change the Control Parameters of a Green Metering

In C++ and Python:

Explanation: Changes the parameters of a green metering that are defined in the current control.
Format:

int ECIChangeParametersGreenMeteringById(int idmetering, double timeSta,
double amax, double ngreenTime, double amin,
double ncycleTime, double offset, double yellowTime);
int ECIChangeParametersGreenMetering(int idsection, double timeSta,
double amax, double ngreenTime, double amin,
double ncycleTime, double offset, double yellowtime)

Parameters:
idmetering represents the metering identifier

idsection represents a valid section identifier.
timeSta represents the absolute time of simulation in stationary period, in seconds.
amax represents the new maximum green time, in seconds
ngreenTime represents the new green time, in seconds
amin represents the new minimum green time, in seconds
ncycleTime represents the new cycle time, in seconds.
offset represents the new offset, in seconds.
yellowtime represents the new yellow time, in seconds.

Output:
= 0: No error occurred.
< 0: Error

3.3.4.10 Read the Control Parameters of a Green by Lane Metering

In C++ and Python:

Explanation: Read the parameters of a green by lane metering that are defined in the current control.

Format:

```
int ECIGetParametersGreenMeteringByLaneById(int idmetering,
                                            double timeSta, double *amax,
                                            double *greenTime, double *amin,double *cycleTime,
                                            double *offset, double *yellowTime, double
                                            *laneOffset);
```

Parameters:
idmetering represents the metering identifier
timeSta represents the absolute time of simulation in stationary period, in seconds.
amax: Value returns the current maximum green time in seconds.
greenTime: Value returns the current green time, in seconds.
amin: Value returns the current minimum green time, in seconds.
cycleTime: Value returns the current cycle time, in seconds.
offset: Value returns the offset, in seconds.
yellowtime: Value returns the yellow time, in seconds.
laneOffset: Value returns the delay activating green between lanes, in seconds.

Output:
= 0: No error occurred.
< 0: Error

3.3.4.11 Change the Control Parameters of a Green by Lane Metering

In C++ and Python:

Explanation: Changes the parameters of a green by lane metering that are defined in the current control.

Format:

```
int ECIChangeParametersGreenMeteringByLaneById(int idmetering,
                                               double timeSta,double amax,
                                               double ngreenTime, double amin,
                                               double ncycleTime, double offset,
                                               double yellowTime, double laneOffset);
```

Parameters:
idmetering represents the metering identifier
timeSta represents the absolute time of simulation in stationary period, in seconds.
amax represents the new maximum green time, in seconds
ngreenTime represents the new green time, in seconds

amin represents the new minimum green time, in seconds
ncycleTime represents the new cycle time, in seconds.
offset represents the new offset, in seconds.
yellowtime represents the new yellow time, in seconds.
laneOffset represents the delay activating green between lanes, in seconds.

Output:
= 0: No error occurred.
< 0: Error

3.3.4.12 Read the Control Parameters of a Flow Metering

In C++ and Python:

Explanation: Read the parameters of a flow metering that are defined in the current control.

Format:

int ECIGetParametersFlowMeteringById(int idmetering, double timeSta,
double *amax, double *flow, double *amin);

int ECIGetParametersFlowMetering (int idsection, double timeSta,
*double * amax,double *flow, double *amin) (Deprecated function)*

Parameters:
idmetering represents the metering identifier
idsection represents a valid section identifier.
timeSta represents the absolute time of simulation in stationary period, in seconds.
amax: Value returned for the maximum flow allowed.
flow: Value returned for the flow allowed.
amin: Value returned for the minimum flow allowed.

Output:
= 0: No error occurred.
< 0: Error

3.3.4.13 Change the Control Parameters of a Flow Metering

In C++ and Python:

Explanation: Changes the parameters of a flow metering that are defined in the current control.

Format:

int ECIChangeParametersFlowMeteringById(int idmetering, double timeSta,
double amax, double flow, double amin);

int ECIChangeParametersFlowMetering (int idsection, double timeSta, double amax, double flow, double amin) (Deprecated function)

Parameters:
idmetering represents the metering identifier
idsection represents a valid section identifier.
timeSta represents the absolute time of simulation in stationary period, in seconds.
amax represents the new maximum flow allowed, in vehicles per hour.
newflow represents the new flow allowed, in vehicles per hour.
amin represents the new minimum flow allowed, in vehicles per hour.

Output:
= 0: No error occurred.
< 0: Error

3.3.4.14 *Read the Control Parameters of a Flow-Alinea Metering*

In C++ and Python:

Explanation: Read the parameters of a flow-Alinea metering that are defined in the current control.

Format:

```
int ECIGetParametersFlowAlineaMeteringById(int idmetering, double timeSta,
                                           double *amax, double *flow, double *amin,
                                           double *kr, double *ostar, double *intervalUpdate);
```

Parameters:

idmetering represents the metering identifier
timeSta represents the absolute time of simulation in stationary period, in seconds.
amax: Value returned for the maximum flow allowed.
flow: Value returned for the flow allowed.
amin: Value returned for the minimum flow allowed.
kr: Value for the regulator parameter for this metering.
ostar: target occupancy on the main road
intervalUpdate: interval at which the flow is recalculated, in seconds

Output:

= 0: No error occurred.
< 0: Error

3.3.4.15 *Change the Control Parameters of a Flow-Alinea Metering*

In C++ and Python:

Explanation: Changes the parameters of a flow-Alinea metering that are defined in the current control.

Format:

```
int ECIChangeParametersFlowAlineaMeteringById(int idmetering, double timeSta,
                                              double amax, double flow, double amin,
                                              double kr, double ostar, double intervalUpdate);
```

Parameters:

idmetering represents the metering identifier
timeSta represents the absolute time of simulation in stationary period, in seconds.
amax represents the new maximum flow allowed, in vehicles per hour.
newflow represents the new flow allowed, in vehicles per hour.
amin represents the new minimum flow allowed, in vehicles per hour.
kr represents the regulator parameter.
ostar represents the target occupancy on the main road.
intervalUpdate: interval at which the flow will be recalculated, in seconds.

Output:

= 0: No error occurred.
< 0: Error

3.3.4.16 *Read the Control Parameters of a Delay Metering*

In C++ and Python:

Explanation: Read the parameters of a delay metering that are defined in the current control.

Format:

```
int ECIGetParametersDelayMeteringById(int idmetering, double timeSta,
                                      double *avg, double *dev);
```

*int ECIGetParametersDelayMetering (int idsection, double timeSta, double * avg, double *dev) (Deprecated function)*

Parameters:

idmetering represents the metering identifier

idsection represents a valid section identifier.

timeSta represents the absolute time of simulation in stationary period, in seconds.

avg: Value returned for the average delay time.

dev: Value returned for the deviation delay time.

Output:

= 0: No error occurred.

< 0: Error

3.3.4.17 Change the Control Parameters of a Delay Metering

In C++ and Python:

Explanation: Changes the parameters of a delay metering that are defined in the current control.

Format:

int ECIChangeParametersDelayMeteringById(int idmetering, double timeSta, double newavg, double newdev);

int ECIChangeParametersDelayMetering (int idsection, double timeSta, double newavg, double newdev) *(Deprecated function)*

Parameters:

idmetering represents the metering identifier

idsection represents a valid section identifier.

timeSta represents the absolute time of simulation in stationary period, in seconds.

newavg represents the new average delay time.

newdev: represents the new deviation delay time.

Output:

= 0: No error occurred.

< 0: Error

3.3.4.18 Change the Control Parameters of a Delay Metering for a vehicle type

In C++ and Python:

Explanation: Changes the parameters of a delay metering that are defined in the current control for a specific vehicle type.

Format:

int ECIChangeParametersDelayMeteringVehTypeById(int idmetering, double timeSta, double newavg, double newdev, int idVehType)

int ECIChangeParametersDelayMeteringVehType(int idsection, double timeSta, double newavg, double newdev, int idVehType) (Deprecated function)

Parameters:

idmetering represents the metering identifier

idsection represents a valid section identifier.

timeSta represents the absolute time of simulation in stationary period, in seconds.

newavg represents the new average delay time.

newdev: represents the new deviation delay time.

idVehType represents the vehicle type id

Output:

= 0: No error occurred.
< 0: Error

3.3.4.19 Disable the fixed control plan of a metering

In C++ and Python:

Explanation: Disables the current fixed control plan of a metering, so the state changes are completely controlled by the Aimsun API Module.

Format:

int ECIDisableEventsMeteringById(int idmetering);

int ECIDisableEventsMetering(int idsection) (Deprecated function)

Parameters:

idmetering represents the metering identifier

idsection represents a valid section identifier.

Output:

= 0: No error occurred

< 0: Error

3.3.4.20 Enable the fixed control plan of a metering

In C++ and Python:

Explanation: Enables the current fixed control plan of a metering, so the state changing is completely controlled by Aimsun, taking into account the parameters defined in the current control.

Format:

int ECIEnableEventsMeteringById(int idmetering)

int ECIEnableEventsMetering (int idsection) (Deprecated function)

Parameters:

idmetering represents the metering identifier

idsection represents a valid section identifier.

Output:

= 0: No error occurred

< 0: Error

3.3.4.21 Change the State of a metering

In C++ and Python:

Explanation: Changes the state of a metering. If the control structure is enabled, Aimsun programmes the next change, taking into account the parameters that define the control. In Green and Flow meterings it is possible to change to green and to red, but in delay meterings it is only possible to change to green and it is necessary to define in which lane.

Format:

int ECIChangeStateMeteringById(int idmetering, int aState, double time, double cycle, int identity)

int ECIChangeStateMetering(int idsection, int aState, double time, double cycle, int identity) (Deprecated function)

Parameters:

idmetering represents the metering identifier

idsection represents a valid section identifier.

aState represents a valid state(0: RED 1:GREEN 2:YELLOW).

time represents a relative time of simulation in seconds.

cycle represents the duration of each simulation step in seconds.

identity: Number of Lane 1...N. Only in Delay Meterings

Output:

= 0: No error occurred
< 0: Error

3.3.4.22 Get Current State of a metering

In C++ and Python:

Explanation: Gets the current state of a metering.

Format:

int ECIGetCurrentStateofMeteringById(int idmetering, int idlane)

int ECIGetCurrentStateofMetering(int idsection, int identity) (Deprecated function)

Parameters:

idmetering represents the metering identifier

idsection represents a valid section identifier.

identity: Number of Lane 1...N. Only in Delay Meterings

Output:

≥ 0: The Current State (0: RED 1:GREEN 2:YELLOW)

< 0: Error

3.3.5 Functions relative to traffic lights

3.3.5.1 Read Number of traffic lights/meterings in a section's lane

In C++ and Python:

Explanation: Read the number of traffic lights plus meterings present on a section's lane.

Format:

int ECIGetNumberSem(int idSection, int idLane, double timeSta)

Parameters:

idsection represents a valid section identifier.

idLane: Number of Lane 1...N.

timeSta: time of simulation in stationary period, in seconds.

Output:

≥ 0: Number of traffic lights plus meterings in the section's lane.

< 0: Error

3.3.5.2 Read the position of a traffic/light or metering in a section's lane

In C++ and Python:

Explanation: Read the position in the section of a metering or traffic light.

Format:

double ECIGetPositionSem(int idSection, int idLane, int numsem, double timeSta)

Parameters:

idsection represents a valid section identifier.

idLane: Number of Lane 1...N.

numsem: represents a valid counter from 0 to (ECIGetNumberSem(idSection, idLane, timeSta)-1)

timeSta: time of simulation in stationary period, in seconds.

Output:

-1: Traffic light located at the end of the section

>=0: distance to the beginning of the section where the metering is located.

3.3.5.3 Read the state of a traffic/light or metering in a section's lane

In C++ and Python:

Explanation: Read the state of a metering or traffic light.

Format:

double ECIGetStateSem (int idSection, int idLane, int numsem, double timeSta)

Parameters:

idsection represents a valid section identifier.

idLane: Number of Lane 1...N.

numsem: represents a valid counter from 0 to (ECIGetNumberSem(idSection, idLane, timeSta)-1)

timeSta: time of simulation in stationary period, in seconds.

Output:

0: traffic light or metering in red
1: traffic light or metering in green
2: traffic light or metering in yellow
3: traffic light or metering in flashing green

3.3.6 Functions related to VMS access

The functions related to VMS access from version v5.0 are not available because the applications related to traffic management in the current version are made using functions relative to Management Actions or functions relative to ANG Policies. The functions related to changing the message displayed in a VMS during the simulation can be done using the ANG Connections functions.

3.3.7 Functions relative to Management Actions

3.3.7.1 Activate a Change Speed Action

In C++ and Python:

Explanation: Activate a change speed action in one section per a specific vehicle type and a compliance level. In the case that the vehicle type is 0, the action is applied to all vehicle types. The returned void * has to be used to deactivate the action.

Format:

void *AKIActionAddSpeedAction (int sectionId, double newSpeed, int vehTypePos, double compliance)

Parameters:

sectionId is the section identifier

newSpeed is new speed expressed in the model units (either Km/h or Mph)

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

compliance is the compliance level (defined between 0 to 1)

Output:

!=NULL: pointer to the action, which will be necessary to disable the action or modify its compliance level later on.

= NULL: Error

3.3.7.2 Activate a Change Speed Action for one lane or segment

In C++ and Python:

Explanation: Activate a change speed action in one section for just one lane or one segment per a specific vehicle type and a compliance level. In the case that the vehicle type is 0, the action is applied to all vehicle types. The returned void * has to be used to deactivate the action.

Format:

void * AKIActionAddDetailedSpeedAction(int sectionId, int laneId, int segmentId, double newSpeed, int vehTypePos, double acomplianceLevel);

Parameters:

sectionId is the section identifier

laneId is the lane identifier (-1 for all lanes, 1 for the rightmost lane and N, where N is the number of lanes in the section, for the leftmost lane.
segmentId is the segment identifier if just one segment in the section will be affected by the action (-1 for all the segments, 1 for the first segment the vehicles face when crossing the section and N, where N is the number of segments, for the last segment the vehicles face when crossing the section.
newSpeed is new speed expressed in the model units (either Km/h or Mph)
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
compliance is the compliance level (defined between 0 to 1)

Output:

!=NULL: pointer to the action, which will be necessary to disable the action or modify its compliance level later on.
= NULL: Error

3.3.7.3 Activate a Lane Closure Action

In C++ and Python:

Explanation: Activate a lane closure action in one section per a specific vehicle type. In the case that the vehicle type is 0, the action is applied to all vehicle types. The returned void * has to be used to deactivate the action. When using this function, the lane closure created will consider the 2-lanes car-following model. If you don't want to consider it please refer to the AKIActionCloseLaneDetailedAction function.

Format:

void *AKIActionCloseLaneAction (int sectionId, int alane, int vehTypePos)

Parameters:

sectionId is the section identifier
alane is the id lane to close (1 rightmost lane, n is the left most lane)
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

!=NULL: pointer of the action, which will be necessary to disable the action or modify its compliance level later on.
= NULL: Error

3.3.7.4 Activate a Lane Closure Action specifying whether to apply the 2-lanes car-following model

In C++ and Python:

Explanation: Activate a lane closure action in one section per a specific vehicle type. In the case that the vehicle type is 0, the action is applied to all vehicle types. It can be also specified whether the 2-lanes car following model wants to be applied or not. The returned void * has to be used to deactivate the action.

Format:

void * AKIActionCloseLaneDetailedAction(int sectionId, int alane, int VehTypePos, bool apply2LanesCF, double visibilityDistance);

Parameters:

sectionId is the section identifier
alane is the id lane to close (1 rightmost lane, n is the left most lane)
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
apply2LanesCF is true if the 2-lanes car following model is going to be considered and false otherwise.

visibilityDistance is the distance at which the lane closure will start to be visible for vehicles. 200 meters is the default value when creating a lane closure action using the graphical user interface.

Output:

!=NULL: pointer of the action, which will be necessary to disable the action or modify its compliance level later on.

= NULL: Error

3.3.7.5 Activate a Lane Closure Action for a segment

In C++ and Python:

Explanation: Activate a lane closure action in one section per a specific vehicle type and for a group of segments. In the case that the vehicle type is 0, the action is applied to all vehicle types. It can be also specified whether the 2-lanes car following model wants to be applied or not. The returned void * has to be used to deactivate the action.

Format:

void *AKIActionCloseLaneActionBySegment (int sectionId, int alane, int segmentFromId, int segmentToId, int vehTypePos, bool apply2LanesCF, double visibilityDistance)

Parameters:

sectionId is the section identifier

alane is the id lane to close (1 rightmost lane, n is the left most lane)

segmentFromId is the position of the first segment (from 0 to number of segments -) where the close lane action will be applied.

segmentToId is the position of the last segment (from 0 to number of segments -) where the close lane action will be applied.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

apply2LanesCF is true if the 2-lanes car following model is going to be considered and false otherwise.

visibilityDistance is the distance at which the lane closure will start to be visible for vehicles. 200 meters is the default value when creating a lane closure action using the graphical user interface.

Output:

!=NULL: pointer of the action, which will be necessary to disable the action or modify its compliance level later on.

= NULL: Error

3.3.7.6 Activate a Force Next Turning Action

In C++ and Python:

Explanation: Activate a “force next turning” action in one section per a specific vehicle type and a compliance level. In the case that the vehicle type is 0, the action is applied to all vehicle types. The returned void * has to be used to deactivate the action.

Format:

void *AKIActionAddNextTurningODAction (int fromSectionId, int toSectionId, int originCent, int destCent, int vehTypePos, int sectionInPath, double compliance, double visibilityDistance)

void *AKIActionAddNextTurningResultAction (int fromSectionId, int oldToSectionId, int toSectionId, int vehTypePos, double compliance)

The AKIActionAddNextTurningODAction function is used when the traffic demand is based on O/D Matrices and the AKIActionAddNextTurningResultAction function is used when the traffic demand is based on traffic states.

Parameters:

fromSectionId is the section identifier origin of the turning.

toSectionId is the new next section identifier destination of the turning.

oldToSectionId is the destination section identifier of the vehicles that will be affected by these change (-1 means do not consider the current turning of the vehicle)

originCent is the origin centroid identifier to force the next turning. All vehicles with this origin can follow the action depending on the compliance level (-1 means do not consider the origin)

destCent is the destination centroid identifier to force the next turning All vehicles with this destination can follow the action depending on the compliance level (-1 means do not consider the destination).

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

sectionInPath is the section identifier that means (only using OD's) the action is applied when this section is in the vehicle path (-1 means do not consider any specific section).

compliance is the compliance level (defined between 0 to 1).

visibilityDistance is the distance at which the force turning will start to be visible for vehicles. 200 meters is the default value when creating a force turning action using the graphical user interface.

Output:

!=NULL: pointer of the action, which will be necessary to disable the action or modify its compliance level later on.

= NULL: Error

3.3.7.7 Activate a Force Next Turning Action following a sub-route

In C++ and Python:

Explanation: Activate a "force next turning" action in one section per a specific vehicle type and a compliance level. Vehicles will follow the sub-route defined. In the case that the vehicle type is 0, the action is applied to all vehicle types. The returned void * has to be used to deactivate the action.

Format:

```
void * AKIActionAddNextSubPathODAction(int sectionId, int nbNextSections , int * nextSections, int originCent, int destCent, int vehTypePos, int sectionInPath, double compliance, double visibilityDistance);
void * AKIActionAddNextSubPathResultAction(int sectionId, int nbNextSections, int * nextSections, int vehTypePos, double compliance);
```

The AKIActionAddNextSubPathODAction function is used when the traffic demand is based on O/D Matrices and the AKIActionAddNextSubPathResultAction function is used when the traffic demand is based on traffic states.

Parameters:

sectionId is the section identifier.

nextSections is the list of consecutive sections (identified using their ids) that define the route the vehicle will follow once diverted.

originCent is the origin centroid identifier to force the next turning. All vehicles with this origin can follow the action depending on the compliance level (-1 means do not consider the origin)

destCent is the destination centroid identifier to force the next turning All vehicles with this destination can follow the action depending on the compliance level (-1 means do not consider the destination).

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
sectionInPath is the section identifier that means (only using OD's) the action is applied when this section is in the vehicle path.
compliance is the compliance level (defined between 0 to 1).
visibilityDistance is the distance at which the force turning will start to be visible for vehicles. 200 meters is the default value when creating a force turning action using the graphical user interface.

Output:

!=NULL: pointer of the action, which will be necessary to disable the action or modify its compliance level later on.

= NULL: Error

3.3.7.8 Activate a Change Destination Centroid Action

In C++ and Python :

Explanation: Activate a "change the destination centroid action" in one section per a specific vehicle type and a compliance level. In the case that the vehicle type is 0, the action is applied to all vehicle types. The returned void * has to be used to deactivate the action.

Format:

void *AKIActionAddChangeDestAction (int sectionId, int anewDest, int originCent, int destCent, int vehTypePos, double compliance)

Parameters:

sectionId is the section identifier.
anewDest is the new destination centroid identifier.
originCent is the origin centroid identifier to apply the action. All vehicles with this origin can follow the action depending on the compliance level (-1 means do not consider the origin)
destCent is the destination centroid identifier to apply the action. All vehicles with this destination can follow the action depending on the compliance level (-1 means do not consider the destination)
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
compliance is the compliance level (defined between 0 to 1)

Output:

!=NULL: pointer of the action, which will be necessary to disable the action or modify its compliance level later on.

= NULL: Error

3.3.7.9 Activate a Change Turning Probability Action

In C++ only:

Explanation: Activate a change of turning probability in one section per a specific vehicle type. In the case that the vehicle type is 0, the action is applied to all vehicle types. The returned void * has to be used to deactivate the action. To use this function the demand has to be composed of traffic states.

Format:

void *AKIActionChangeTurningProbAction (int sectionId, int nbnewProb, int *anextSection, int *anewProb, int vehTypePos)

Parameters:

sectionId is the section identifier.

nbnewProb is the number of probabilities to change. That must match the number of turnings exiting the section.
anextSection is an array that contains the identifiers of the next sections (the size of the array must be equal to nbnewProb).
anewProb is an array that contains the new probabilities to the next sections (the size of the array must be equal to nbnewProb). Probabilities must be between 0 and 1.
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

!=NULL: pointer of the action, which will be necessary to disable the action later on.

= NULL: Error

3.3.7.10 Activate a Close Turning Action

In C++ and Python:

Explanation: Activate a turning closure in one turning movement per a specific vehicle type. In the case that the vehicle type is 0, the action is applied to all vehicle types. The returned void * has to be used to deactivate the action.

Format:

void * AKIActionAddCloseTurningODAction(int fromSectionId, int toSectionId, int aOrigin, int aDest, int vehTypePos, double compliance, double visibilityDistance, bool localEffect, bool isDUE)

Parameters:

fromSectionId is the turning's origin section identifier.
toSectionId is the turning's destination section identifier.
originCent is the origin centroid identifier to force the turn closing. All vehicles with this origin can follow the action depending on the compliance level (-1 means do not consider the origin)
destCent is the destination centroid identifier to force the turn closing. All vehicles with this
destination can follow the action depending on the compliance level (-1 means do not consider the destination)
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
compliance is the compliance level (defined between 0 to 1).
visibilityDistance is the distance at which the close turning will start to be visible for vehicles. 200 meters is the default value when creating a close turning action using the graphical user interface.
localEffect is true when the action will not affect the route choice calculations directly, as vehicle will notice the turn is closed when reaching it. And it is false when there is a global knowledge about the turn being closed so it is considered in the route calculation. It is true by default.
isDUE is true when a Dynamic User Equilibrium experiment is run and false when a Stochastic Route choice is run. It is false by default.

Output:

!=NULL: pointer of the action, which will be necessary to disable the action or modify its compliance level later on.

= NULL: Error

3.3.7.11 Disable an Action

In C++ and Python:

Explanation: Disable an action previously activated. The parameter required is the pointer returned when the action is enabled.
Format:
void AKIActionRemoveAction (void *anaction)
Parameters:
anaction is the pointer to an action returned when the action was activated
Output:
none

3.3.7.12 Disable all activated Actions

In C++ and Python:
Explanation: Disable all actions previously activated.
Format:
void AKIActionReset ()
Parameters:
none
Output:
none

3.3.7.13 Modify the compliance level of a Change Speed Action

In C++ and Python:
Explanation: Modifies the compliance level of a previously activated change speed. The void * returned when activating the action has to be used to modify the compliance level of the action.
Format:
void AKIActionModifyNextTurningODAction(void * action, double compliance);
Parameters:
action is the pointer to the action that was returned when activating the action
compliance is the new compliance level (defined between 0 to 1)
Output:
none

3.3.7.14 Modify the compliance level of a Force Next Turning Action

In C++ and Python:
Explanation: Modifies the compliance level of a previously activated force next turning action. The void * returned when activating the action has to be used to modify the compliance level of the action.
Format:
void AKIActionModifyNextTurningODAction(void * action, double compliance);
void AKIActionModifyNextTurningResultAction(void * action, double compliance);

The AKIActionModifyNextTurningODAction function is used when the traffic demand is based on O/D Matrices and the AKIActionModifyNextTurningResultAction function is used when the traffic demand is based on traffic states.

Parameters:
action is the pointer to the action that was returned when activating the action
compliance is the new compliance level (defined between 0 to 1)
Output:
none

3.3.7.15 Modify the compliance level of a Force Next Turning Action following a sub-route

In C++ and Python:

Explanation: Modifies the compliance level of a previously activated force next turning action following a sub-route. The void * returned when activating the action has to be used to modify the compliance level of the action.
Format:

void AKIActionModifyNextSubPathODAction (void * action, double compliance);
void AKIActionModifyNextSubPathResultAction (void * action, double compliance);

The AKIActionModifyNextSubPathODAction function is used when the traffic demand is based on O/D Matrices and the AKIActionModifyNextSubPathResultAction function is used when the traffic demand is based on traffic states.

Parameters:

action is the pointer to the action that was returned when activating the action
compliance is the new compliance level (defined between 0 to 1)

Output:

none

3.3.7.16 Modify the compliance level of a Change Destination Centroid Action

In C++ and Python:

Explanation: Modifies the compliance level of a previously activated change destination centroid action. The void * returned when activating the action has to be used to modify the compliance level of the action.
Format:

void AKIActionModifyChangeDestAction (void * action, double compliance);

Parameters:

action is the pointer to the action that was returned when activating the action
compliance is the new compliance level (defined between 0 to 1)

Output:

none

3.3.7.17 Modify the compliance level of a Close Turning Action

In C++ and Python:

Explanation: Modifies the compliance level of a previously activated close turning action. The void * returned when activating the action has to be used to modify the compliance level of the action.
Format:

void AKIActionModifyCloseTurningODAction(void * action, double compliance);

Parameters:

action is the pointer to the action that was returned when activating the action
compliance is the new compliance level (defined between 0 to 1)

Output:

none

3.3.8 Functions relative to detector measures

In the case of access to detector measures using the Detector Name, measures are given corresponding to the first Detector whose name matches with Name passed as a parameter. Therefore, when a network model has several detectors with the same name it is better to access to the Detector measures using the Detector Identifier. Instant detection refers to the attributes measured by a detector at that instant (this instant duration being defined by the detection cycle), as opposed to aggregate data collected over a detection output interval.

3.3.8.1 Read Number of detectors

In C++ and Python:

Explanation: Read the total number of detectors in the road network
Format:
int AKIDetGetNumberDetectors ()
Parameters: None
Output:
> 0: Total number of detectors
< 0: Error

3.3.8.2 Read Identifier of a detector

In C++ and Python:
Explanation: Read the identifier of the elem-th detector.
Format:
int AKIDetGetIdDetector (int elem)
Parameters:
elem: It has to be between 0 and the total number of detectors -1
Output:
> 0: The identifier of the detector
< 0: Error

3.3.8.3 Read the Information from a detector

In C++ and Python:
Explanation: Read the information of the elem-th detector.
Format:
structA2KDetector AKIDetGetPropertiesDetector(int elem)
structA2KDetector AKIDetGetPropertiesDetectorById (int IdDetector)
Parameters:
elem: It has to be between 0 and the total number of detectors -1
IdDetector is the detector identifier.
Output:

```
struct structA2KDetector {
        int report;
        int Id;
        int IdSection
        int IdFirstLane;
        int IdLastLane;
        bool DistinguishType;
        int Capabilities;
        double InitialPosition;
        double FinalPosition;
};
```

where:
int *report*: 0, OK, else error code
int *Id*: detector identifier
int *IdSection*: section identifier where the detector is located
int *IdFirstLane*: first lane that the detector covers
int *IdLastLane*: last lane that the detector covers
bool *DistinguishType*: Capability to detect distinguishing per vehicle type.
int *Capabilities*: set of bits that codifies the detection capabilities.
int *InitialPosition*: position of the beginning of the detector, with respect to the beginning of the section.
int *FinalPosition*: position of the end of the detector, with respect to the beginning of the section.

3.3.8.4 Check the detection capabilities

In C++ and Python:

Explanation: Check the detection capabilities for a detector previously read. They are used to check if a detector can detect count, presence, speed, occupancy, headway, density ot equipped vehicles.

Format:

bool AKIDetIsCountGather(int Capability);
bool AKIDetIsPresenceGather(int Capability);
bool AKIDetIsSpeedGather(int Capability);
bool AKIDetIsOccupancyGather(int Capability);
bool AKIDetIsHeadwayGather(int Capability);
bool AKIDetIsDensityGather(int Capability);
bool AKIDetIsInfEquippedVehGather (int Capability);

Parameters:

Capability: set of bits that codifies the detection capabilities. This set of bits can be obtained using the AKIDetGetPropertiesDetector and similar ones explained in section 3.3.8.3.

Output:

true: Capability allowed.
false: No Capability or Error

3.3.8.5 Read the Detection Interval

In C++ and Python:

Explanation: Read the detection interval (seconds) used to gather the aggregated measures.

Format:

double AKIDetGetIntervalDetection ()

Parameters: None

Output:

≥ 0: the detection interval returned in seconds
< 0: Error

3.3.8.6 Read the Interval of Instant Detection

Explanation: Read the interval (seconds) used to gather the instant detection measures.

Format:

double AKIDetGetCycleInstantDetection ()

Parameters: None

Output:

≥ 0: the detection interval returned in seconds
< 0: Error

3.3.8.7 Read the number of Instant Detection measures available during the last simulation step

In C++ and Python:

Explanation: Read the number of Instant detection measures available during the last simulation step.

Format:

int AKIDetGetNbMeasuresAvailableInstantDetection ()

Parameters: None

Output:

≥ 0: number of measures
< 0: Error

3.3.8.8 *Read the End Time of Instant Detection measure available during the last simulation step*

In C++ and Python:

Explanation: Read the end time of Instant detection measures available during the last simulation step.

Format:

double AKIDetGetEndTimeMeasureAvailableInstantDetection(int elem)

Parameters:

elem: it has to be ≥ 0 and < total number of measures available (see previous function)

Output:

≥ 0: the detection interval returned in seconds

< 0: Error

3.3.8.9 *Read the Instant Presence of a detector*

In C++ and Python:

Explanation: Read the instant presence during the last simulation step of a detector. This will be 0 if no vehicle has been over the detector and 1 otherwise. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function give the last measure available during the last simulation step and the second one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

int AKIDetGetPresenceCyclebyId (int IdDetector, int vehTypePos)

int AKIDetGetPresenceInstantDetectionbyId(int IdDetector, int vehTypePos, double endtime)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

endtime is the time that defines the instant measure.

Output:

≥ 0: the presence returned

< 0: Error

3.3.8.10 *Read the Instant Occupied Time of a detector*

In C++ and Python:

Explanation: Read the percentage of the time that the detector has been occupied from an instant measure. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The two function gives the last measure available during the last simulation step and the second one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

double AKIDetGetTimeOccupedCyclebyId (int IdDetector, int vehTypePos)

double AKIDetGetTimeOccupedInstantDetectionbyId(int IdDetector, int vehType, double endtime);

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

endtime is the time that defines the instant measure

Output:

≥ 0: the percentage of occupied time returned

< 0: Error

3.3.8.11 *Read the Instant Counter measure of a detector*

In C++ and Python:

Explanation: Read the number of vehicles that have crossed the detector during an instant measure. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step and the second one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

int AKIDetGetCounterCyclebyId (int IdDetector, int vehTypePos)

int AKIDetGetCounterInstantDetectionbyId(int IdDetector, int vehTypePos, double endtime)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

endtime is the time that defines the instant measure.

Output:

≥ 0: the counter returned

< 0: Error

3.3.8.12 *Read the Instant Average Speed of a detector*

In C++ and Python:

Explanation: Read the instant average of speed of vehicles (km/h or mph depending the units defined in the network) that are over the detector. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step and the last one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

double AKIDetGetSpeedCyclebyId (int IdDetector, int vehTypePos)

double AKIDetGetSpeedInstantDetectionbyId(int IdDetector, int vehTypePos, double endtime)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

endtime is the time that defines the instant measure.

Output:

≥ 0: the speed returned

< 0: Error

3.3.8.13 Read the Instant Number of Occupied Intervals of a detector

In C++ and Python:

Explanation: Read the total number of intervals that the detector has been occupied during one instant. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The two function gives the last measure available during the last simulation step and the second one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

int AKIDetGetNbintervalsOccupedCyclebyId (int IdDetector, int vehTypePos)
int AKIDetGetNbintervalsOccupedInstantDetectionbyId(int IdDetector, int vehTypePos, double endtime);

Parameters:

IdDetector is the detector identifier.
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
endtime is the time that defines the instant measure.

Output:

≥ 0: the number of intervals returned
< 0: Error

3.3.8.14 Read the Instant Initial Time of an Occupied Interval of a detector

In C++ and Python:

Explanation: Read the instant initial time of an occupied interval of a detector. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step while the last one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

double AKIDetGetIniTimeOccupedCyclebyId (int IdDetector, , int elem, int vehTypePos)
double AKIDetGetIniTimeOccupedInstantDetectionbyId(int IdDetector, int elem, int vehTypePos, double endtime);

Parameters:

IdDetector is the detector identifier.
elem: the number of interval. It has to be between 0 and the total number of intervals -1
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
endtime is the time that defines the instant measure

Output:

≥ 0: the initial time returned
< 0: Error

3.3.8.15 Read the Instant Final Time of an Occupied Interval of a detector

In C++ and Python:

Explanation: Read the Instant final time of an occupied interval of a detector. It is possible distinguish detection for different vehicle types, if the detector has this

capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step and the last one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

double AKIDetGetFinTimeOccupedCyclebyId (int IdDetector, int elem, int vehTypePos)

double AKIDetGetEndTimeOccupedInstantDetectionbyId(int IdDetector, int elem, int vehTypePos, double endtime);

Parameters:

IdDetector is the detector identifier.

elem: the number of interval. It has to be between 0 and the total number of intervals -1

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

endtime is the time that defines the instant measure

Output:

≥ 0: the final time returned

< 0: Error

3.3.8.16 Read the SCOOT Occupancy of a detector in the Last Cycle

In C++ and Python:

Explanation: Read the Instant occupancy in SCOOT format, which requires 1 second as the instant detection interval. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step while the last one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

int AKIDetGetSCOOTOccupancyCyclebyId (int IdDetector, int vehTypePos)

int AKIDetGetSCOOTOccupancyInstantDetectionbyId(int IdDetector, int vehTypePos, double endtime);

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

endtime is the time that defines the instant measure.

Output:

≥ 0: the SCOOT occupancy returned

< 0: Error

3.3.8.17 Read the Instant Density of a detector

In C++ and Python:

Explanation: Read the Instant density (expressed in veh/Km or veh/M depending the units defined in the network). It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step while the last one gives the measure for

one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection).

Format:

double AKIDetGetDensityCyclebyId (int IdDetector, int vehTypePos)
double AKIDetGetDensityInstantDetectionbyId(int IdDetector, int vehTypePos, double endtime);

Parameters:

IdDetector is the detector identifier.
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
endtime is the time that defines the instant measure.

Output:

≥ 0: the instant density returned
< 0: Error

3.3.8.18 Read the Headway of a detector

In C++ and Python:

Explanation: Read the instant average headway between vehicles (average time between from bumper to from bumper) that have been crossed the detector. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step while the last one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

double AKIDetGetHeadwayCyclebyId (int IdDetector, int vehTypePos)
double AKIDetGetHeadwayInstantDetectionbyId(int IdDetector, int vehTypePos, double endtime)

Parameters:

IdDetector is the detector identifier.
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
endtime is the time that defines the instant measure

Output:

≥ 0: the headway returned
< 0: Error

3.3.8.19 Read the Instant Number of Vehicles with its Static Information gathered by a detector

In C++ and Python:

Explanation: Read the Instant number of vehicles that have crossed the detector, when the detector has the "Equipped Vehicle" capability activated. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step while the last one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection)

Format:

int AKIDetGetNbVehsEquippedInDetectionCyclebyId (int IdDetector, int vehTypePos)

int AKIDetGetNbVehsEquippedInDetectionInstantDetectionbyId (int IdDetector, int vehTypePos, double endtime);

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

endtime is the time that defines the instant measure.

Output:

≥ 0: the number of vehicles

< 0: Error

3.3.8.20 Read the Instant Vehicle Information of a detector

In C++ and Python:

Explanation: Read the instant static vehicle information of all vehicles that have crossed the detector, when the detector has the "Equipped Vehicle" capability activated. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account. The first function gives the last measure available during the last simulation step while the last one gives the measure for one specific instant during the last simulation step (this time could be obtained using function AKIDetGetEndTimeMeasureAvailableInstantDetection). It is highly recommended that the detection cycle be equal to the simulation step. This can be done by checking the option *Same as simulation step* that is located into the *Main* tab folder of the scenario editor.

Format:

StaticInfVeh AKIDetGetInfVehInDetectionStaticInfVehCyclebyId (int iddet, int elem, int vehTypePos);

StaticInfVeh AKIDetGetInfVehInDetectionStaticInfVehInstantDetectionbyId(int iddet, int elem, int vehTypePos, double endtime);

Parameters:

iddet is the detector identifier.

elem: the number of vehicle. It has to be between 0 and the total number of vehicle identifiers -1

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

endtime is the time that defines the instant measure.

Output:

```
struct StaticInfVeh{
        int report;
        int idVeh;
        int type;
        double length;
        double width;
        double maxDesiredSpeed;
        double maxAcceleration;
        double normalDeceleration;
        double maxDeceleration;
        double speedAcceptance;
        double minDistanceVeh;
        double giveWayTime;
        double guidanceAcceptance;
        int enrouted;
```

```
                int equipped;
                int tracked;

                bool  keepfastLane;
                double headwayMin;
                double sensitivityFactor;
                double reactionTime;
                double reactionTimeAtStop;
                double reactionTimeAtTrafficLight;

                int centroidOrigin;
                int centroidDest;
                int idsectionExit;

                int idLine;
                void * internalInfo;
        };
```

where:

int *report*: 0, OK, else error code
int *idVeh*: vehicle identifier
int *type*: vehicle type (car, bus, truck, etc.)
double *length*: vehicle length (m or feet, depending on the units defined in the network).
double *width*: vehicle width (m or feet, depending on the units defined in the network).
double *maxDesiredSpeed*: Maximum desired speed of the vehicle (km/h or mph, depending on the units defined in the network).
double *maxAcceleration*: Maximum acceleration of the vehicle (m/s^2 or ft/ s^2, depending on the units defined in the network).
double *normalDeceleration*: Maximum deceleration of the vehicle that can apply under normal conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *maxDeceleration*: Maximum deceleration of the vehicle that can apply under special conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *speedAcceptance*: degree of acceptance of the speed limits.
double *minDistanceVeh*: distance that the vehicle keeps between itself and the preceding vehicle (meters or feet, depending on the units defined in the network).
double *giveWayTime*: time after which the vehicle becomes more aggressive in give-way situations (seconds).
double *guidanceAcceptance*: level of compliance of the vehicle to guidance indications.
int *enrouted*: 0 means vehicle will not change path en-route, 1 means vehicle will change path en-route depending on the percentege of enrouted vehicles defined.
int *equipped*: 0 means vehicle not equipped, 1 means vehicle equipped.
int *tracked*: 0 means vehicle not tracked, 1 means vehicle tracked.
bool *keepfastLane*: means vehicle keep fast lane during overtaking
double *headwayMin*: minimum headway to keep with its leader
double *sensitivityFactor*: estimation of the acceleration of the leader
double *reactionTime*: reaction time of the vehicle
double *reactionTimeAtStop*: reaction time at stop of the vehicle
double *reactionTimeAtTrafficLight*: reaction time of the vehicle when stopped the first one of the queue in a traffic light.

int *centroidOrigin*: Identifier of centroid origin of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *centroidDest*: : Identifier of centroid destination of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *idsectionExit*: : Identifier of exit section destination of the vehicle, when the destination centroid uses percentages as destination (otherwise is –1) and the traffic conditions are defined by an O/D matrix.
int *idLine*: Identifier of Public Transport Line, when the vehicle has been generated as a public transport vehicle.
void * *internalInfo*: only for internal use

3.3.8.21 Read the information of a Vehicle in a Detector

In C++ and Python:

Explanation: Read the information of certain vehicle in a detector. This function requires, previously, get the number of Vehicles that have crossed the detector calling either AKIDetGetNbVehsEquippedInDetectionCyclebyId or AKIDetGetNbVehsEquippedInDetectionInstantDetectionbyId.

Format:

InfVeh AKIDetGetInfVehInDetectionInfVehCyclebyId(int iddet, int elem, int VehTypePos)

InfVeh AKIDetGetInfVehInDetectionInfVehInstantDetectionbyId(int iddet, int elem, int VehTypePos, double endtime);

Parameters:

iddet: Detector Identifier
elem: vehicle index, from 0 to (Total Number of Vehicles in the detector – 1)
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
endtime is the time that defines the instant measure.

Output:

```
struct InfVeh{
        int report;
        int idVeh;
        int type;
        // Information in Vehicle when it is in a section
        int idSection;
        int segment;
        int numberLane;

        // Information in Vehicle when it is in a node
        int idJunction;
        int idSectionFrom;
        int idLaneFrom;
        int idSectionTo;
        int idLaneTo;

        double CurrentPos;
        double distance2End;
        double xCurrentPos, yCurrentPos, zCurrentPos;
        double xCurrentPosBack, yCurrentPosBack, zCurrentPosBack;
        double CurrentSpeed, PreviousSpeed;

        double TotalDistance;
```

double SystemGenerationT;
double SystemEntranceT;
double SectionEntranceT;
double CurrentStopTime;
};

where:

int *report*: 0, OK, else error code
int *idVeh*: the vehicle identifier
int *type*: the vehicle type (car, bus, truck, etc.)
int *idSection*: the section identifier
int *segment*: the segment number of the section where the vehicle is located (from 0 to n-1)
int *numberLane*: the lane number in the segment (from 1, the rightmost lanet, to N, the leftmost lane)
double *CurrentPos*: the position inside the section = distance (m or feet, depending on the units defined in the network) from the beginning of the section.
double *distance2End*: the distance to the end of the section (m or feet, depending on the units defined in the network)
double *xCurrentPos, yCurrentPos, zCurrentPos*: world coordinates of the middle point of the front bumper of the vehicle.
double *xCurrentPosBack, yCurrentPosBack, zCurrentPosBack*: world coordinates of the middle point of the rear bumper of the vehicle.
double *CurrentSpeed*: the current speed (in km/h or mph, depending on the units defined in the network
double *PreviousSpeed*: the speed in the previous simulation step (in km/h or mph, depending on the units defined in the network.
double *TotalDistance*: the total distance travelled (m or feet) by the vehicle.
double *SystemGenerationT*: the absolute generation time of the vehicle into the system. If no virtual queue found in its entrance section it will be the same as the SystemEntranceT.
double *SystemEntranceT*: the absolute entrance time of the vehicle into the system, that is into its entrance section. If no virtual queue found in its entrance section it will be the same as the SystemGenerationT.
double *SectionEntranceT*: the absolute entrance time of the vehicle into the current section.
double *CurrentStopTime*: the current stop time.

The other fields or attributes have no meaning, so their value is –1.

3.3.8.22 Read Counter Aggregated in the Last Detection Interval

In C++ and Python:

Explanation: Read the counter aggregated during the last detection interval. It is possible distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account.

Format:

int AKIDetGetCounterAggregatedbyId(int IdDetector, int VehTypePos)

Parameters:

IdDetector is the detector identifier.
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: the counter value returned
< 0: Error

3.3.8.23 Read Speed Aggregated in the Last Detection Interval

In C++ and Python:

Explanation: Read the speed aggregated during the last detection interval (km/h or mph depending the units defined in the network). It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account.

Format:

double AKIDetGetSpeedAggregatedbyId(int IdDetector, int VehTypePos)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: the speed returned

< 0: Error

3.3.8.24 Read Occupancy Aggregated in the Last Detection Interval

In C++ and Python:

Explanation: Read the percentage of the time that the detector has been occupied during the last detection interval. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account.

Format:

double AKIDetGetTimeOccupedAggregatedbyId(int IdDetector, int VehTypePos)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: the occupancy returned

< 0: Error

3.3.8.25 Read Presence Aggregated in the Last Detection Interval

In C++ and Python:

Explanation: Read the presence aggregated during the last detection interval. This will be 0 if no vehicle has been over the detector and 1 otherwise. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account.

Format:

int AKIDetGetPresenceAggregatedbyId(int IdDetector, int VehTypePos)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: the presence returned

< 0: Error

3.3.8.26 Read Density Aggregated in the Last Detection Interval

In C++ and Python:

Explanation: Read the density (expressed in veh/km or veh/mi depending the units defined in the network) during the last detection interval. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account.

Format:

double AKIDetGetDensityAggregatedbyId(int IdDetector, int VehTypePos)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: the density returned

< 0: Error

3.3.8.27 Read Headway Aggregated in the Last Detection Interval

In C++ and Python:

Explanation: Read the average headway between vehicles (average time between from bumper to bumper) that have crossed the detector during the last detection interval. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account.

Format:

double AKIDetGetHeadwayAggregatedbyId(int IdDetector, int VehTypePos)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: the headway returned, in seconds.

< 0: Error

3.3.8.28 Read Number of Equipped Vehicles with its trace in the Last Detection Interval

In C++ and Python:

Explanation: Read the number of equipped vehicles that have crossed the detector during the last detection interval, when the detector has the "Equipped Vehicle" capability activated. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account.

Format:

int AKIDetGetNbVehsInDetectionAggregatedbyId(int IdDetector, int VehTypePos)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: the number of vehicles

< 0: Error

3.3.8.29 Read the trace Equipped Vehicles in the Last Detection Interval

In C++ and Python:

Explanation: Read the information of equipped vehicles that have crossed the detector during the last detection interval, when the detector has the "Equipped Vehicle" capability activated. It is possible to distinguish detection for different vehicle types, if the detector has this capability. In the case that the vehicle type is 0, the vehicle type distinction is not taken into account.

Format:

EquippedInfVeh AKIDetGetInfVehInDetectionAggregatedbyId(int IdDetector, int vehTypePos, int elem)

Parameters:

IdDetector is the detector identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

elem: the number of the vehicle. It has to be between 0 and the total number of vehicle identifiers -1

Output:

```
struct EquippedInfVeh {
    int report;
    double timedetected;
    int idVeh;
    int vehType;
    double speed;
    double headway;
    int idptline;
}
```

where:

int *report*: 0, OK, else error code.

double *timedetected*: the instant when the equipped vehicle crossed the detector.

int *idVeh*: the vehicle identifier

int *vehType*: is the position of the vehicle type in the list of vehicles types being used. 0 corresponds to all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

double *speed*: instant vehicle speed (Km/h)

double *headway*: instant vehicle headway (sec)

int *idptline*: Public Transport line identifier (-1 when the equipped vehicle does not belongs to any PT line)

3.3.9 Functions relative to detectors clickable events

3.3.9.1 Enable the detector clickable events

In C++ and Python:

Explanation: Enables the detector clickable events in all the detectors in the model.

Format:

int AKIDetectorEventsEnable()

Parameters:

None

Output:

= 0: OK.

< 0: Error

3.3.9.2 Disable the detector clickable events

In C++ and Python:

Explanation: Disables the detector clickable events in all the detectors in the model.

Format:

int AKIDetectorEventsDisable ()

Parameters:

None

Output:

= 0: OK.

< 0: Error

3.3.9.3 Load the initial events of a detector

In C++ and Python:

Explanation: Loads the initial events of all the detectors in the model

Format:

int AKIDetectorEventsLoadInitialEvents();

Parameters:

None

Output:

= 0: OK.

< 0: Error

3.3.9.4 Save the detector events

In C++ and Python:

Explanation: Saves the events of all the detectors in the model

Format:

int AKIDetectorEventsSaveEvents();

Parameters:

None

Output:

= 0: OK.

< 0: Error

3.3.9.5 Add a detector event

In C++ and Python:

Explanation: Adds a detector event for an specified detector

Format:

void AKIDetectorEventsAddEvent(int iddet, double aIniTime, double aEndTime, int vehTypePos, double speed, double length, int idPTline);

Parameters:

Iddet: detector identifier

IniTime: absolute time of the simulation when the event will start (seconds).

Endtime: absolute time of the simulation when the event will finish (seconds).

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Speed: the detected speed

Length: the detected length

idPTLine: if the vehicle was a ptVehicle then the id of the PTLine. 0, otherwise

Output:

None

3.3.9.6 Clear the detector clickable events

In C++ and Python:

Explanation: Removes all clickable events of all detectors

Format:

int AKIDetectorEventsClear();

Parameters:

None

Output:

= 0: OK.

< 0: Error

3.3.10 Functions relative to vehicle types information

3.3.10.1 Read Number of Vehicles Types

In C++ and Python:

Explanation: Read the total number of vehicles types. If the Public Transport is loaded, it includes the public transport vehicle types.

Format:

int AKIVehGetNbVehTypes ()

Parameters:

none

Output:

> 0: Total number of vehicles Types

< 0: Error

3.3.10.2 Read the Name of a Vehicle Type

In C++:

Explanation: Read the name of a vehicle Type in utf16 and \0 terminated. The result has to be deleted with delete[]

Format:

const unsigned short *AKIVehGetVehTypeName(int vehTypePos);

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≠NULL: name of the vehicle type

= NULL: Error

In Python:

Not Available.

3.3.10.3 Read the Vehicle Type ID using the Position of a Vehicle Type

In C++ and Python:

Explanation: Read the vehicle type ID given the vehicle type position in the list of vehicle types.

Format:

int AKIVehTypeGetIdVehTypeANG (int vehTypePos);

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

>0: The Id of the vehicle type

-1: No vehicle type with the position provided found

3.3.10.4 Read the Position of a Vehicle Type using the Vehicle Type ID

In C++ and Python:

Explanation: Read the position of a vehicle Type given the vehicle type Aimsun identifier.

Format:

int AKIVehGetVehTypeInternalPosition (int aimsunVehicleTypeId);

Parameters:

aimsunVehicleTypeId: The vehicle type identifier in Aimsun.

Output:

>0: A value from 1 to the total number of vehicle types

<0: No vehicle with the name provided found

3.3.10.5 Read the Position of a Vehicle Type using the Vehicle Type Name

The name of a vehicle type, as it is user-edited, can be not unique. On the other hand, the ID is always unique. For this reason, it is suggested to use the AKIVehGetVehTypeInternalPosition explained above to get the position of a vehicle type.

In any case, if the API needs to access the vehicle type position from the vehicle type name, the following functions can be used. As an example, to get the internal veh position for vehicle type 'car' in a Python API the functions needed would be:

```
idVeh = ANGConnGetObjectId( AKIConvertFromAsciiString( "car" ), False )
vehPos = AKIVehGetVehTypeInternalPosition( idVeh )
```

Refer to section 3.3.10.4 and 3.3.29.1 for more information about these functions.

3.3.10.6 Read the Minimum and the Maximum Length of a Vehicle Type

In C++ and Python:

Explanation: Read the total number of vehicles types. If the Public Transport is loaded, it includes the public transport vehicle types.

Format:

double AKIVehGetMinLengthVehType (int vehTypePos)

double AKIVehGetMaxLengthVehType (int vehTypePos)

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

> 0: The Minimum and the Maximum Length of the vehicle type.

< 0: Error

3.3.10.7 Read the Percentage for Imprudent Lane-changing of a Vehicle Type

In C++ and Python:

Explanation: Reads the percentage specified for imprudent lane-changing of a vehicle type.

Format:

double AKIVehTypeGetPercentageForImprudentLaneChanging(int idVehicleType)

Parameters:

idVehicleType: Unique identifier of the vehicle Type.

Output:

> 0: The Percentage for Imprudent Lane-changing of the vehicle type.

< 0: Error

3.3.10.8 *Modify the Percentage for Imprudent Lane-changing of a Vehicle Type*

In C++ and Python:

Explanation: Modifies the percentage for imprudent lane-changing defined for the vehicle type.

Format:

int AKIVehTypeSetPercentageForImprudentLaneChanging(int idVehType, double newPercentage)

Parameters:

idVehType: Unique identifier of the vehicle Type.

newPercentage: value between 0 and 100 to define the new percentage that will be used from the moment the function is called.

Output:

0: OK

<0: Error

3.3.10.9 *Read the Sensitivity for Imprudent Lane-changing of a Vehicle Type*

In C++ and Python:

Explanation: Reads the sensitivity specified for imprudent lane-changing of a vehicle type.

Format:

double AKIVehTypeGetSensitivityForImprudentLaneChanging(int idVehType)

Parameters:

idVehType: Unique identifier of the vehicle Type.

Output:

> 0: The Percentage for Imprudent Lane-changing of the vehicle type.

< 0: Error

3.3.10.10 *Modify the Sensitivity for Imprudent Lane-changing of a Vehicle Type*

In C++ and Python:

Explanation: Modifies the sensitivity for imprudent lane-changing defined for the vehicle type.

Format:

int AKIVehTypeSetSensitivityForImprudentLaneChanging (int idVehType, double newSensitivity)

Parameters:

idVehType: Unique identifier of the vehicle Type.

newSensitivity: value between 0 and 1 to define the new sensitivity that will be used from the moment the function is called.

Output:

0: OK

<0: Error

3.3.10.11 *Read the Percentage for Staying in the Fast Lane of a Vehicle Type*

In C++ and Python:

Explanation: Reads the percentage specified for staying in a fast lane after a lane-changing of a vehicle type.

Format:

double AKIVehTypeGetPercentageForStayingInFastLane (int idVehType)

Parameters:

idVehType: Unique identifier of the vehicle Type.

Output:

> 0: The Percentage for Imprudent Lane-changing of the vehicle type.

< 0: Error

3.3.10.12 Modify the Percentage for Staying in the Fast Lane of a Vehicle Type

In C++ and Python:

Explanation: Modifies the percentage specified for staying in a fast lane after a lane-changing of a vehicle type.

Format:

int AKIVehTypeSetPercentageForStayingInFastLane (int idVehType, double newPercentage)

Parameters:

idVehType: Unique identifier of the vehicle Type.

newPercentage: value between 0 and 100 to define the new percentage that will be used from the moment the function is called.

Output:

0: OK

<0: Error

3.3.10.13 Read the Percentage for Undertaking of a Vehicle Type

In C++ and Python:

Explanation: Reads the percentage specified for undertaking of a vehicle type.

Format:

double AKIVehTypeGetPercentageForUndertaking (int idVehType)

Parameters:

idVehType: Unique identifier of the vehicle Type.

Output:

> 0: The Percentage for Imprudent Lane-changing of the vehicle type.

< 0: Error

3.3.10.14 Modify the Percentage for Undertaking of a Vehicle Type

In C++ and Python:

Explanation: Modifies the percentage specified for undertaking of a vehicle type.

Format:

int AKIVehTypeSetPercentageForUndertaking (int idVehicleType, double newPercentage)

Parameters:

idVehType: Unique identifier of the vehicle Type.

newPercentage: value between 0 and 100 to define the new percentage that will be used from the moment the function is called.

Output:

0: OK

<0: Error

3.3.11 Functions relative to vehicles information

3.3.11.1 Read Number of Vehicles in a Section

In C++ and Python:

Explanation: Read the total number of vehicles in a section.

Format:

int AKIVehStateGetNbVehiclesSection(int aidSec, bool considerAllSegments)

Parameters:

aidSec: Section Identifier

considerAllSegments: this parameters must always be fixed to true.

Output:

≥ 0: Total number of vehicles.

< 0: Error.

3.3.11.2 Read Number of Vehicles in a Junction

In C++ and Python:

Explanation: Read the total number of vehicles in a junction.

Format:

int AKIVehStateGetNbVehiclesJunction(int aidJunction)

Parameters:

aidJunction: Junction Identifier

Output:

≥ 0: Total number of vehicles

< 0: Error

3.3.11.3 Read the information of a Vehicle in a Section

In C++ and Python:

Explanation: Read the information of certain vehicle in a section. This function requires, previously, get the number of Vehicles in the section calling AKIVehStateGetNbVehiclesSection for the same section without any additional calls to the AKIVehStateGetNbVehiclesSection for other sections.

Format:

InfVeh AKIVehStateGetVehicleInfSection(int aidSec, int indexveh)

Parameters:

aidSec: Section Identifier

indexveh: vehicle index, from 0 to (Total Number of Vehicles in the Section – 1)

Output:

```
struct InfVeh{
        int report;
        int idVeh;
        int type;
        // Information in Vehicle when it is in a section
        int idSection;
        int segment;
        int numberLane;

        // Information in Vehicle when it is in a node
        int idJunction;
        int idSectionFrom;
        int idLaneFrom;
        int idSectionTo;
        int idLaneTo;

        double CurrentPos;
        double distance2End;
        double xCurrentPos, yCurrentPos, zCurrentPos;
        double xCurrentPosBack, yCurrentPosBack, zCurrentPosBack;
        double CurrentSpeed, PreviousSpeed;

        double TotalDistance;
```

double SystemGenerationT;
double SystemEntranceT;
double SectionEntranceT;
double CurrentStopTime;
};

where:

int *report*: 0, OK, else error code
int *idVeh*: the vehicle identifier
int *type*: is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
int *idSection*: the section identifier
int *segment*: the segment number of the section where the vehicle is located (from 0 to n-1)
int *numberLane*: the lane number in the segment (from 1, the rightmost lanet, to N, the leftmost lane)
double *CurrentPos*: the position inside the section = distance (m or feet, depending on the units defined in the network) from the beginning of the section.
double *distance2End*: the distance to the end of the section (m or feet, depending on the units defined in the network)
double *xCurrentPos, yCurrentPos, zCurrentPos*: world coordinates of the middle point of the front bumper of the vehicle.
double *xCurrentPosBack, yCurrentPosBack, zCurrentPosBack*: world coordinates of the middle point of the rear bumper of the vehicle.
double *CurrentSpeed*: the current speed (in km/h or mph, depending on the units defined in the network
double *PreviousSpeed*: the speed in the previous simulation step (in km/h or mph, depending on the units defined in the network.
double *TotalDistance*: the total distance travelled (m or feet) by the vehicle.
double *SystemGenerationT*: the absolute generation time of the vehicle into the system. If no virtual queue found in its entrance section it will be the same as the SystemEntranceT.
double *SystemEntranceT*: the absolute entrance time of the vehicle into the system, that is into its entrance section. If no virtual queue found in its entrance section it will be the same as the SystemGenerationT.
double *SectionEntranceT*: the absolute entrance time of the vehicle into the current section.
double *CurrentStopTime*: the current stop time.

The other fields or attributes have no meaning, so their value is –1.

3.3.11.4 Read the information of a Vehicle in a Junction

In C++ and Python:

Explanation: Read the information of a certain vehicle in a junction. This function requires having previously obtained the number of Vehicles in the junction by calling the function AKIVehStateGetNbVehiclesJunction for the same junction without any additional calls to the AKIVehStateGetNbVehiclesJunction for other junctions.

Format:

InfVeh AKIVehStateGetVehicleInfJunction(int aidJunction, int indexveh)

Parameters:

aidJunction: the junction identifier
indexveh: vehicle index, from 0 to (Total Number of Vehicles in the junction – 1)

Output:

```
struct InfVeh{
        int report;
        int idVeh;
        int type;
        // Information in Vehicle when it is in a section
        int idSection;
        int segment;
        int numberLane;

        // Information on Vehicle when it is in a node
        int idJunction;
        int idSectionFrom;
        int idLaneFrom;
        int idSectionTo;
        int idLaneTo;

        double CurrentPos;
        double distance2End;
        double xCurrentPos, yCurrentPos, zCurrentPos;
        double xCurrentPosBack, yCurrentPosBack, zCurrentPosBack;
        double CurrentSpeed, PreviousSpeed;

        double TotalDistance;

        double SystemGenerationT;
        double SystemEntranceT;
        double SectionEntranceT;
        double CurrentStopTime
};
```

where:

int *report*: 0, OK, else error code
int *idVeh*: the vehicle identifier
int *type*: is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
int *idSectionFrom*: the origin section identifier
int *idLaneFrom*: number of lanes of the origin section where the vehicle enters the junction
int *idSectionTo*: destination section identifier
int *idLaneTo*: number of lanes of the destination section where the vehicle exits the junction
double *CurrentPos*: the position inside the junction = distance (m or feet, depending on the units defined in the network) from the entrance to the junction.
double *distance2End*: the distance to end of the turning (m or feet, depending on the units defined in the network).
double *xCurrentPos*, *yCurrentPos*, *zCurrentPos*: world coordinates of the middle point of the front bumper of the vehicle.
double *xCurrentPosBack*, *yCurrentPosBack*, *zCurrentPosBack*: world coordinates of the middle point of the rear bumper of the vehicle.
double *CurrentSpeed*: current speed (in km/h or mph, depending on the units defined in the network
double *PreviousSpeed*: speed in the previous simulation step (in km/h or mph, depending on the units defined in the network
double *TotalDistance*: total distance travelled (m or feet)

double *SystemGenerationT*: the absolute generation time of the vehicle into the system. If no virtual queue found in its entrance section it will be the same as the SystemEntranceT.
double *SystemEntranceT*: the absolute entrance time of the vehicle into the system, that is into its entrance section. If no virtual queue found in its entrance section it will be the same as the SystemGenerationT.
double *CurrentStopTime*: the current stop time

The other fields or attributes have no meaning, so their value is -1

3.3.11.5 Read the Static Information of a Vehicle in a Section

In C++ and Python:

Explanation: Read the static information of a certain vehicle in a section. Static information means the characters of the vehicle have been set when the vehicle has entered in the system. This function requires having previously obtained the number of Vehicles in the section calling AKIVehStateGetNbVehiclesSection.

Format:

StaticInfVeh AKIVehGetVehicleStaticInfSection(int aidSec, int indexveh)

Parameters:

aidSec: section Identifier
indexveh: vehicle index, from 0 to (Total Number of Vehicles in the section – 1)

Output:

```
struct StaticInfVeh{
        int report;
        int idVeh;
        int type;
        double length;
        double width;
        double maxDesiredSpeed;
        double maxAcceleration;
        double normalDeceleration;
        double maxDeceleration;
        double speedAcceptance;
        double minDistanceVeh;
        double giveWayTime;
        double guidanceAcceptance;
        int enrouted;
        int equipped;
        int tracked;

        bool  keepfastLane;
        double headwayMin;
        double sensitivityFactor;
        double reactionTime;
        double reactionTimeAtStop;
        double reactionTimeAtTrafficLight;

        int centroidOrigin;
        int centroidDest;
        int idsectionExit;

        int idLine;
        void * internalInfo;
};
```

where:

int *report*: 0, OK, else error code
int *idVeh*: vehicle identifier
int *type*: is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
double *length*: vehicle length (m or feet, depending on the units defined in the network).
double *width*: vehicle width (m or feet, depending on the units defined in the network).
double *maxDesiredSpeed*: Maximum desired speed of the vehicle (km/h or mph, depending on the units defined in the network).
double *maxAcceleration*: Maximum acceleration of the vehicle (m/s^2 or ft/ s^2, depending on the units defined in the network).
double *normalDeceleration*: Maximum deceleration of the vehicle that can apply under normal conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *maxDeceleration*: Maximum deceleration of the vehicle that can apply under special conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *speedAcceptance*: degree of acceptance of the speed limits.
double *minDistanceVeh*: distance that the vehicle keeps between itself and the preceding vehicle (meters or feet, depending on the units defined in the network).
double *giveWayTime*: time after which the vehicle becomes more aggressive in give-way situations (seconds).
double *guidanceAcceptance*: level of compliance of the vehicle to guidance indications.
int *enrouted*: 0 means vehicle will not change path en-route, 1 means vehicle will change path en-route depending on the percentege of enrouted vehicles defined.
int *equipped*: 0 means vehicle not equipped, 1 means vehicle equipped.
int *tracked*: 0 means vehicle not tracked, 1 means vehicle tracked.
bool *keepfastLane*: means vehicle keep fast lane during overtaking
double *headwayMin*: minimum headway to keep with its leader
double *sensitivityFactor*: estimation of the acceleration of the leader
double *reactionTime*: reaction time of the vehicle
double *reactionTimeAtStop*: reaction time at stop of the vehicle
double *reactionTimeAtTrafficLight*: reaction time of the vehicle when stopped the first one of the queue in a traffic light.
int *centroidOrigin*: Identifier of centroid origin of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *centroidDest*: Identifier of centroid destination of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *idsectionExit*: Identifier of exit section destination of the vehicle, when the destination centroid uses percentages as destination (otherwise is –1) and the traffic conditions are defined by an O/D matrix.
int *idLine*: Identifier of Public Transport Line, when the vehicle has been generated as a public transport vehicle.
void * *internalInfo*: only for internal use

3.3.11.6 Read the Static Information of a Vehicle in a Junction

In C++ and Python:

Explanation: Read the static information of certain vehicle in a junction. Static information means the characteristics of the vehicle that have been set when the vehicle

has entered the system. This function requires having previously obtained the number of Vehicles in the junction by calling function AKIVehStateGetNbVehiclesJunction.

Format:

StaticInfVeh AKIVehGetVehicleStaticInfJunction(int aidJunction, int indexveh)

Parameters:

aidJunction: the junction identifier

indexveh: vehicle index, from 0 to (Total Number of Vehicles in the junction – 1)

Output:

```
struct StaticInfVeh{
        int report;
        int idVeh;
        int type;
        double length;
        double width;
        double maxDesiredSpeed;
        double maxAcceleration;
        double normalDeceleration;
        double maxDeceleration;
        double speedAcceptance;
        double minDistanceVeh;
        double giveWayTime;
        double guidanceAcceptance;
        int enrouted;
        int equipped;
        int tracked;

        bool  keepfastLane;
        double headwayMin;
        double sensitivityFactor;
        double reactionTime;
        double reactionTimeAtStop;
        double reactionTimeAtTrafficLight;

        int centroidOrigin;
        int centroidDest;
        int idsectionExit;

        int idLine;
        void * internalInfo;
};
```

where*:*

int *report*: 0, OK, else error code

int *idVeh*: vehicle identifier

int *type*: is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

double *length*: vehicle length (m or feet, depending on the units defined in the network).

double *width*: vehicle width (m or feet, depending on the units defined in the network).

double *maxDesiredSpeed*: Maximum desired speed of the vehicle (km/h or mph, depending on the units defined in the network).

double *maxAcceleration*: Maximum acceleration of the vehicle (m/s^2 or ft/ s^2, depending on the units defined in the network).

double *normalDeceleration*: Maximum deceleration of the vehicle that can apply under normal conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *maxDeceleration*: Maximum deceleration of the vehicle that can apply under special conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *speedAcceptance*: degree of acceptance of the speed limits.
double *minDistanceVeh*: distance that the vehicle keeps between itself and the preceding vehicle (meters or feet, depending on the units defined in the network).
double *giveWayTime*: time after which the vehicle becomes more aggressive in give-way situations (seconds).
double *guidanceAcceptance*: level of compliance of the vehicle to guidance indications.
int *enrouted*: 0 means vehicle will not change path en-route, 1 means vehicle will change path en-route depending on the percentege of enrouted vehicles defined.
int *equipped*: 0 means vehicle not equipped, 1 means vehicle equipped.
int *tracked*: 0 means vehicle not tracked, 1 means vehicle tracked.
bool *keepfastLane*: means vehicle keep fast lane during overtaking
double *headwayMin*: minimum headway to keep with its leader
double *sensitivityFactor*: estimation of the acceleration of the leader
double *reactionTime*: reaction time of the vehicle
double *reactionTimeAtStop*: reaction time at stop of the vehicle
double *reactionTimeAtTrafficLight*: reaction time of the vehicle when stopped the first one of the queue in a traffic light.
int *centroidOrigin*: Identifier of centroid origin of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *centroidDest*: Identifier of centroid destination of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *idsectionExit*: Identifier of exit section destination of the vehicle, when the destination centroid uses percentages as destination (otherwise is -1) and the traffic conditions are defined by an O/D matrix.
int *idLine*: Identifier of Public Transport Line, when the vehicle has been generated as a public transport vehicle.
void * *internalInfo*: only for internal use

3.3.11.7 Modify the Static Information of a Vehicle in a Section

In C++ and Python:

Explanation: Modify some static parameters of certain vehicle in a section. Static parameters mean the characteristics of the vehicle which have been set when the vehicle has entered in the system. The static parameters which may be changed are: *type, length width, maxDesiredSpeed, maxAcceleration, normalDeceleration, maxDeceleration, speedAcceptance, minDistanceVeh, giveWayTime, guidanceAcceptance, enrouted, equipped, tracked, keepfastLane, headwayMin , sensitivityFactor, reactionTime, reactionTimeAtStop, reactionTimeAtTrafficLight, centroidOrigin, centroidDest and idsectionExit* (the exit section has sense when the destination centroid uses destination percentages; if the identifier of the section is invalid or -1 then Aimsun determines the exit section according to criteria defined in the centroid). Aimsun cannot store the previous values, so it cannot recover them. This function requires having previously obtained the number of Vehicles in the section calling AKIVehStateGetNbVehiclesSection.

Format:

int AKIVehSetVehicleStaticInfSection(int aidSec, int indexveh, StaticInfVeh staticinfVeh)

Parameters:

aidSec: the section identifier

indexveh: vehicle index, from 0 to (Total Number of Vehicles in the section – 1)

staticinfVeh: new static parameters to be assigned

Output:

= 0: No Error

< 0: Error

3.3.11.8 Modify the Static Information of a Vehicle in a Junction

In C++ and Python:

Explanation: Modify some static parameters of a particular vehicle in a junction. Static parameters mean the characteristics of the vehicle which have been set when the vehicle has entered in system. The static parameters which may be changed are: *type, length width, maxDesiredSpeed, maxAcceleration, normalDeceleration, maxDeceleration, speedAcceptance, minDistanceVeh, giveWayTime, guidanceAcceptance, enrouted, equipped, tracked, keepfastLane, headwayMin , sensitivityFactor, reactionTime, reactionTimeAtStop, reactionTimeAtTrafficLight, centroidOrigin, centroidDest and idsectionExit* (the exit section has sense when the destination centroid uses destination percentages; if the identifier of the section is invalid or –1 then Aimsun determines the exit section according to criteria defined in the centroid).. Aimsun cannot store the previous values, so it cannot recover them. This function requires having previously obtained the number of Vehicles in the junction by calling AKIVehStateGetNbVehiclesJunction.

Format:

int AKIVehSetVehicleStaticInfJunction(int aidJunct,int indexveh, StaticInfVeh staticinfVeh)

Parameters:

aidJunct: junction Identifier

indexveh: the vehicle index, from 0 to (Total Number of Vehicles in the junction – 1)

staticinfVeh: new static parameters to be assigned.

Output:

= 0: No Error

< 0: Error

3.3.11.9 Remove a Vehicle from a Section

In C++ and Python:

Explanation: Removes a particular vehicle from a section. This function requires having previously obtained the number of Vehicles in the section by calling function AKIVehStateGetNbVehiclesSection.

Format:

int AKIRemoveVehicle(int aidSec, int indexveh)

Parameters:

aidSec: section Identifier

indexveh: vehicle index, from 0 to (Total Number of Vehicles in the section – 1)

Output:

= 0: No Error

< 0: Error

3.3.11.10 Remove a Vehicle from Junction

In C++ and Python:

Explanation: Removes a particular vehicle from a junction. This function requires having previously obtained the number of Vehicles in the junction by calling function AKIVehStateGetNbVehiclesJunction.

Format:

int AKIRemoveVehicleJunction(int idJunction, int indexveh)

Parameters:

idJunction: junction Identifier

indexveh: vehicle index, from 0 to (Total Number of Vehicles in the junction – 1)

Output:

= 0: No Error

< 0: Error

3.3.11.11 Read the information of a Vehicle

In C++ and Python:

Explanation: Read the information of a particular vehicle. This function can slow the simulation because every call to this function implies finding the vehicle inside the network. In order to avoid this search, it is recommended to set the vehicle as Tracked and then read its information.

Format:

InfVeh AKIVehGetInf(int aidVeh)

Parameters:

aidVeh: Vehicle Identifier

Output:

```
struct InfVeh{
        int report;
        int idVeh;
        int type;
        // Information in Vehicle when it is in a section
        int idSection;
        int segment;
        int numberLane;

        // Information in Vehicle when it is in a node
        int idJunction;
        int idSectionFrom;
        int idLaneFrom;
        int idSectionTo;
        int idLaneTo;

        double CurrentPos;
        double distance2End

        double xCurrentPos, yCurrentPos, zCurrentPos;
        double xCurrentPosBack, yCurrentPosBack, zCurrentPosBack;
        double CurrentSpeed, PreviousSpeed;

        double TotalDistance;

        double SystemGenerationT;
        double SystemEntranceT;
        double SectionEntranceT;
        double CurrentStopTime
};
```

where:

int *report*: 0, OK, otherwise value represents an error code
int *idVeh*: the vehicle identifier.
int *type*: the vehicle type (car, bus, truck, etc.) from 1 to AKIVehGetNbVehTypes ()
int *idSection*: the section identifier.
int *segment*: the segment number of the section where the vehicle is located (from 0 to n-1).
int *numberLane*: the lane number in the segment (from 1, the rightmost lane, to N, the leftmost lane).
int *idSectionFrom*: the origin section identifier.
int *idLaneFrom*: the number of the lane of the origin section where the vehicle has entered the junction.
int *idSectionTo*: the destination section identifier.
int *idLaneTo*: the number of the lane of the destination section where the vehicle exits the junction.
double *CurrentPos*: vehicle's position inside the section given by its distance (m or feet, depending on the units defined in the network) from the beginning of the section or vehicle's position inside the junction given by its distance from the entrance to the junction.
double *distance2End*: distance to the end of the section (m or feet, depending on the units defined in the network) when the vehicle is located in a section or the distance to end of the turning when the vehicle is in a junction.
double *xCurrentPos*, *yCurrentPos*, *zCurrentPos*: world coordinates of the middle point of the front bumper of the vehicle.
double *xCurrentPosBack*, *yCurrentPosBack*, *zCurrentPosBack*: world coordinates of the middle point of the rear bumper of the vehicle.
double *CurrentSpeed*: current speed (in km/h or mph, depending on the units defined in the network.
double *PreviousSpeed*: speed in the previous simulation step (in km/h or mph, depending on the units defined in the network
double *TotalDistance*: total distance travelled (m or feet)
double *SystemGenerationT*: the absolute generation time of the vehicle into the system. If no virtual queue found in its entrance section it will be the same as the SystemEntranceT.
double *SystemEntranceT*: the absolute entrance time of the vehicle into the system, that is into its entrance section. If no virtual queue found in its entrance section it will be the same as the SystemGenerationT.
double *SectionEntranceT*: the absolute entrance time of the vehicle in the current section
double *CurrentStopTime*: the current stop time

3.3.11.12 Read the Static Information of a Vehicle.

In C++ and Python:

Explanation: Read the static information of certain vehicle. Static information means the characteristics of the vehicle which have been set when the vehicle has entered in the system. This function can slow the simulation because every call of this function implies finding the vehicle inside the network. In order to avoid this search, it is recommended to set the vehicle as Tracked and then read its information.

Format:

StaticInfVeh AKIVehGetStaticInf (int aidVeh)

Parameters:

aidVeh: Vehicle Identifier

Output:

struct StaticInfVeh{

```
        int report;
        int idVeh;
        int type;
        double length;
        double width;
        double maxDesiredSpeed;
        double maxAcceleration;
        double normalDeceleration;
        double maxDeceleration;
        double speedAcceptance;
        double minDistanceVeh;
        double giveWayTime;
        double guidanceAcceptance;
        int enrouted;
        int equipped;
        int tracked;

        bool  keepfastLane;
        double headwayMin;
        double sensitivityFactor;
        double reactionTime;
        double reactionTimeAtStop;
        double reactionTimeAtTrafficLight;

        int centroidOrigin;
        int centroidDest;
        int idsectionExit;

        int idLine;
        void * internalInfo;
};
```

where:

int *report*: 0, OK, else error code
int *idVeh*: vehicle identifier
int *type*: vehicle type (car, bus, truck, etc.)
double *length*: vehicle length (m or feet, depending on the units defined in the network).
double *width*: vehicle width (m or feet, depending on the units defined in the network).
double *maxDesiredSpeed*: Maximum desired speed of the vehicle (km/h or mph, depending on the units defined in the network).
double *maxAcceleration*: Maximum acceleration of the vehicle (m/s^2 or ft/ s^2, depending on the units defined in the network).
double *normalDeceleration*: Maximum deceleration of the vehicle that can apply under normal conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *maxDeceleration*: Maximum deceleration of the vehicle that can apply under special conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *speedAcceptance*: degree of acceptance of the speed limits.
double *minDistanceVeh*: distance that the vehicle keeps between itself and the preceding vehicle (meters or feet, depending on the units defined in the network).

double *giveWayTime*: time after which the vehicle becomes more aggressive in give-way situations (seconds).
double *guidanceAcceptance*: level of compliance of the vehicle to guidance indications.
int *enrouted*: 0 means vehicle will not change path en-route, 1 means vehicle will change path en-route depending on the percentege of enrouted vehicles defined.
int *equipped*: 0 means vehicle not equipped, 1 means vehicle equipped.
int *tracked*: 0 means vehicle not tracked, 1 means vehicle tracked.
bool *keepfastLane*: means vehicle keep fast lane during overtaking
double *headwayMin*: minimum headway to keep with its leader
double *sensitivityFactor*: estimation of the acceleration of the leader
double *reactionTime*: reaction time of the vehicle
double *reactionTimeAtStop*: reaction time at stop of the vehicle
double *reactionTimeAtTrafficLight*: reaction time of the vehicle when stopped the first one of the queue in a traffic light.
int *centroidOrigin*: Identifier of centroid origin of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *centroidDest*: Identifier of centroid destination of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *idsectionExit*: Identifier of exit section destination of the vehicle, when the destination centroid uses percentages as destination (otherwise is –1) and the traffic conditions are defined by an O/D matrix.
int *idLine*: Identifier of Public Transport Line, when the vehicle has been generated as a public transport vehicle.
void * *internalInfo*: only for internal use

3.3.11.13 Modify the Static Information of a Vehicle

In C++ and Python:

Explanation: Modify some static parameters of certain vehicle. Static parameters mean the characteristics of the vehicle which have been set when the vehicle has entered in the system. The static parameters that it is possible to change are: *type, length width, maxDesiredSpeed, maxAcceleration, normalDeceleration, maxDeceleration, speedAcceptance, minDistanceVeh, giveWayTime, guidanceAcceptance, enrouted, equipped, tracked, keepfastLane, headwayMin , sensitivityFactor, reactionTime, reactionTimeAtStop, reactionTimeAtTrafficLight, equipped, tracked, centroidOrigin, centroidDest and idsectionExit* (the exit section has sense when the destination centroid uses destination percentages; if the identifier of the section is invalid or –1 then Aimsun determines the exit section according to criteria defined in the centroid). Aimsun cannot store the previous values, so it cannot recover them. This function can slow the simulation because every call to this function implies finding the vehicle inside the network. In order to avoid this search, it is recommended to set the vehicle as Tracked and then read its information.

Format:

int AKIVehSetStaticInf (int aidVeh, StaticInfVeh staticinfVeh)

Parameters:

aidVeh: Vehicle Identifier
staticinfVeh: new static parameters to be assigned.

Output:

= 0: No Error
< 0: Error

3.3.11.14 Read the positions of a Vehicle in a Section

In C++ and Python:

Explanation: Read the position of a vehicle over a section
Format:
InfVehPos AKIVehGetVehicleGetPosSection(int asect, int indexveh, int nbPos);
Parameters:
asect: Section Identifier
indexVeh: the index of the vehicle in the section
nbPos: number of positions to read
Output:
report= 0: No Error
report < 0: Error

3.3.11.15 Read the position of a Vehicle in a section reusing the InfVehPos struct

In C++ and Python:
Explanation: Reads the position of a vehicle over a section. It is faster than the previous function
Format:
int AKIVehGetVehicleGetPosSectionWithStruct(int asect, int indexveh, int nbPos, InfVehPos *pinfVehPos);
Parameters:
asect: Section Identifier
indexVeh: the index of the vehicle in the section
nbPos: number of positions to read
pinfVehPos: the pointer to the struct.
Output:
= 0: No Error
< 0: Error

3.3.11.16 Read the positions of a Vehicle over a junction

In C++ and Python:
Explanation:
Format:
InfVehPos AKIVehGetVehicleGetPosJunction(int ajunction, int indexveh, int nbPos);
Parameters:
ajunction: Junction Identifier
indexVeh: the index of the vehicle in the section
nbPos: number of positions to read
Output:
report = 0: No Error
report < 0: Error

3.3.11.17 Read the positions of a Vehicle over a junction reusing the InfVehPos struct

In C++ and Python:
Explanation: Reads the position of a vehicle over a junction. It is faster than the previous function
Format:
int AKIVehGetVehicleGetPosJunctionWithStruct(int ajunction, int indexveh, int nbPos, InfVehPos *pinfVehPos);
Parameters:
ajunction: Junction Identifier
indexVeh: the index of the vehicle in the section

nbPos: number of positions to read
pinfVehPos: the pointer to the struct.

Output:
= 0: No Error
< 0: Error

3.3.11.18 Enable the Graphical Information of a Vehicle

In C++ and Python:

Explanation: Enables the Graphical Information of a Vehicle

Format:

int AKIVehEnableGraphicalInf()

Output:
= 0: No Error

3.3.11.19 Disable the Graphical Information of a Vehicle

In C++ and Python:

Explanation: Disables the Graphical Information of a Vehicle

Format:

int AKIVehDisableGraphicalInf()

Output:
= 0: No Error

3.3.11.20 Read the Graphical Information of a Vehicle from a section

In C++ and Python:

Explanation: Reads the Graphical Information of a Vehicle from a secion

Format:

GraphicInfVeh AKIVehGetVehicleGraphicInfSection(int asect, int indexveh);

```
struct GraphicInfVeh{
        int report;
        int idVeh;
        bool leftTurnSignal;
        bool rightTurnSignal;
        bool brakeLight;
}
```

Parameters:
*a*sect = Section Identifier
indexveh = index of the vehicle in the section

Output:
report = 0: No Error
report < 0: Error

3.3.11.21 Read the Graphical Information of a Vehicle from a junction

In C++ and Python:

Explanation: Reads the Graphical Information of a Vehicle from a junction

Format:

GraphicInfVeh AKIVehGetVehicleGraphicInfJunction(int ajunction, int indexveh);

Parameters:
ajunction = Junction Identifier
indexveh = index of the vehicle in the junction

Output:

report = 0: No Error
report < 0: Error

3.3.11.22 Modify a Vehicle to drive backwards or normal

In C++ and Python:

Explanation: Modifies the drawing mode of a vehicle. If value parameter is true, the vehicle will be drawed as going backwards

Format:

int AKIVehSetDrivingBackwards(int aidSec, int indexveh, bool value);

Parameters:

aidSec = Section Identifier
indexveh = index of the vehicle in the section
value = drawing mode

Output:

report = 0: No Error
report < 0: Error

3.3.11.23 Read the number of sections to reach destination

In C++ and Python:

Explanation: Read the number of sections to reach destination of a vehicle path. The vehicle could be in a section or in a junction. Valid for OD and PT vehicles, otherwise returns 0. AKIVehStateGetNbVehiclesSection or AKIVehStateGetNbVehiclesJunction must be called before call this function. If idsection parameter is set to -1 it returns the number of sections from the beginning to destination, otherwise returns the number of sections from section with idsection Identifier to destination.

Format:

int AKIVehStateGetNbSectionsVehiclePathSection(int aidSection,
int indexveh,
int idsection)
int AKIVehStateGetNbSectionsVehiclePathJunction(int aidJunction,
int indexveh,
int idsection)

Parameters:

aidSection: Section Identifier.
aidJunction: Junction Identifier.
indexveh: Section or junction vehicle index (0<= indexveh < Number of vehicles in section or junction)
idsection: Sections are counted from idsection. If idsection is set to -1 sections are counted from the beginning.

Output:

>= 0: Number of sections to reach destination.
AKIInfNotReady: Error

3.3.11.24 Read the Identifiers of sections to reach destination

In C++ and Python:

Explanation: Read the identifier of section elem-th of the vehicle path to reach destination. The vehicle could be in a section or in a junction.Valid for OD and PT vehicles, otherwise returns 0. AKIVehStateGetNbSectionsVehiclePathSection or AKIVehStateGetNbSectionsVehiclePathJunction must be called before call this

functions. If idsection parameter is set to -1 it returns the identifiers of sections from the beginning to destination, otherwise returns the identifiers of sections from section with idsection Identifier to destination.

Format:

```
int AKIVehStateGetIdSectionVehiclePathSection(      int aidSection,
                                                    int indexveh,
                                                    int idsection,
                                                    int indexsection)
int AKIVehStateGetIdSectionVehiclePathJunction(     int aidJunction,
                                                    int indexveh,
                                                    int idsection,
                                                    int indexsection)
```

Parameters:

aidSection: Section Identifier.

aidJunction: Junction Identifier.

indexveh: Section or junction vehicle index (0<= indexveh < Number of vehicles in section or junction)

idsection: Sections are counted from idsection. If idsection is set to -1 sections are counted from the beginning.

indexsection: section index (0 <= indexsection < Number of sections)

Output:

> 0: Section Identifier.

AKIInfNotReady: Error

3.3.12 Functions relative to statistical data

The structures returned by this set of functions are one of the following:

```
struct StructAkiEstadSystem{
        int report;             /* 0, OK, else error code */
        int Flow;               /* Flow  (veh/h) */
        double TTa,TTd;         /* Travel Time  : Average & Deviation (seconds/km or
seconds/mile)*/
        double DTa, DTd;        /* Delay Time  : Average & Deviation (seconds/km or
seconds/mile) */
        double Sa, Sd;          /* Speed  : Average & Deviation (km/h or mph) */
        double SHa, SHd;                /* Harmonic Speed  : Average & Deviation (km/h
or mph)*/
        double Density;         /* Density : (Veh/km or Veh/mile) */
        double STa, STd;                /* Stop Time  : Average & Deviation (seconds/km
or seconds/mile)*/
        double NumStops;        /* Number of Stops (#/Veh/km or #/Veh/mile)*/
        double TotalTravel;     /* Total Distance  travelled */
        double TotalTravelTime;         /* Total Time  travelled */
        double virtualQueueAvg; /*VirtualQueue Avg (veh)*/
        int virtualQueueMax; /*VirtualQueue Max (veh)*/
        int count; /* volume (vehs) */
        int inputFlow;/* vehicle flow that have entered the section (veh/h)*/
        int inputCount;/* vehicle count that have entered the section  (vehs)*/
        int vehsWaiting;/*vehicles waiting to enter (vehs)*/
        int vehIn;/* vehicles in network (vehs)*/
        int vehsLostIn; /* vehicles lost inside the network (vehs)*/
        int vehsLostOut;/*vehicle lost outside the network (vehs)*/
         double missedTurns;/* number of missed turns*/
};
```

```
struct StructAkiEstadSection{
        int report;             /* 0, OK, else error code */
        int Id;                 /* Section Identifier*/
        int Flow;               /* Flow  (veh/h) */
        double TTa,TTd;                 /* Travel Time  : Average & Deviation (seconds)*/
        double DTa, DTd;        /* Delay Time  : Average & Deviation (seconds) */
        double Sa, Sd;          /* Speed  : Average & Deviation (km/h or mph) */
        double SHa, SHd;                /* Harmonic Speed  : Average & Deviation (km/h
    or mph) */
        double Density;         /* Density : (Veh/km or Veh/mile) */
        double STa, STd;                /* Stop Time  : Average & Deviation (seconds) */
        double NumStops;        /* Number of Stops (#/Veh)*/
        double LongQueueAvg; /* Average Queue Length (veh) */
        double LongQueueMax; /* Maximum Queue Length (veh) */
        double TotalTravel;     /* Total Distance  travelled */
        double TotalTravelTime;         /* Total Time  travelled */
        double virtualQueueAvg; /*VirtualQueue Avg (veh)*/
        int virtualQueueMax; /*VirtualQueue Max (veh)*/
        int count; /* volume (vehs)*/
        int inputFlow; /* vehicle flow that have entered the section (veh/h)*/
        int inputCount;/* vehicle count that have entered the section (vehs)*/
        double flowCapacity; /*flow / section capacity*/
        double laneChanges;/* nb of lane changes in section*/
};

struct StructAkiEstadTurning{
        int report;
        int IdSectionFrom;
        int IdSectionTo;
        int Flow;                       /* Flow (veh/h) */
        double TTa,TTd;                 /* Travel Time  : Average & Deviation (seconds) */
        double DTa, DTd;                /* Delay Time  : Average & Deviation (seconds) */
        double Sa, Sd;          /* Speed  : Average & Deviation (km/h or mph) */
        double SHa, SHd;                /* Harmonic Speed  : Average & Deviation (km/h
    or mph) */
        double STa, STd;                /* Stop Time  : Average & Deviation (seconds) */
        double NumStops;                /* Number of Stops */
        double LongQueueAvg; /* Average Queue Length (veh) */
        double LongQueueMax; /* Maximum Queue Length (veh) */
        double TotalTravel;     /* Total Distance  travelled (km or miles) */
        double TotalTravelTime;         /* Total Time  travelled (seconds) */
};

struct StructAkiEstadODPair{
        int report;
        int IdOrigin;
        int IdDest;
        int Flow;                       /* Flow: #vehicles */
        double TTa, TTd;                /* Travel Time  : Average & Deviation (seconds) */
        double DTa, DTd;                /* Delay Time  : Average & Deviation (seconds) */
        double Sa, Sd;          /* Speed  : Average & Deviation (km/h or mph) */
        double SHa, SHd;                /* Harmonic Speed  : Average & Deviation (km/h
    or mph) */
        double STa, STd;                /* Stop Time  : Average & Deviation (seconds) */
```

```
        double NumStops;            /* Number of Stops */
        double TotalTravel;     /* Total Distance  travelled (km or miles) */
        double TotalTravelTime;     /* Total Time  travelled */
        int vehLost;            /* Total number of vehicle lost */
};

struct StructAkiEstadStream{
        int report;
        int Id;
        int Flow;       /* Flow */
        double TTa,TTd;             /* Travel Time  : Average & Deviation */
        double DTa,DTd;             /* Delay Time  : Average & Deviation */
        double Sa,Sd;           /* Speed  : Average & Deviation */
        double SHa,SHd;             /* Harmonic Speed  : Average & Deviation */
        double Density;
        double STa,STd;             /* Stop Time  : Average & Deviation */
        double NumStops;            /* Number of Stops */
        double LongQueueAvg; /* Average Queue Length */
        double LongQueueMax; /* Maximum Queue Length */
        double TotalTravel;     /* Total Distance  travelled */
        double TotalTravelTime;     /* Total Time  travelled */
    };
```

Furthermore, information about pollutants and emission is also available.

3.3.12.1 Check if the statistics are gathered

In C++ and Python:

Explanation: Check if the statistical data is being gathered during the simulation.

Format:

int AKIIsGatheringStatistics()

Parameters: none

Output:

≥ 0: 1 the statistical data is being gathered, 0 otherwise

< 0: Error

3.3.12.2 Read the statistics interval

In C++ and Python:

Explanation: Read the statistic interval (seconds).

Format:

double AKIEstGetIntervalStatistics ()

Parameters: none

Output:

≥ 0: The statistic interval. 0 means no periodic statistics are gathered.

< 0: Error

3.3.12.3 Check whether a new statistic interval is available

In C++ and Python:

Explanation: Check whether a new statistic interval is available. This function returns true just in the first postmange and mange call after the end of the new statistic interval.

Format:

bool AKIEstIsNewStatisticsAvailable()

Parameters: none

Output:

true: The statistic interval is available.

false: No is available.

3.3.12.4 Read the number of pollutants defined for the simulation

In C++ and Python:

Explanation: Read the number of pollutants defined in the network.

Format:

int AKIEstGetNbPollutants ()

Parameters:

None.

Output:

>= 0: Number of pollutants.

<0: Error

3.3.12.5 Read the Pollutant name

In C++:

Explanation: Read the name of a pollutant defined for all the vehicle types

Format:

const unsigned short * AKIEstGetPollutantName (int index)

Parameters:

index: Pollutant index, where index takes values form 0 to AKIEstGetNbPollutants( )

Output:

<>NULL: No Error

=NULL: Error

In Python:

Explanation: Read the name of a pollutant defined for all the vehicle types in ascii.

Format:

const char * AKIEstGetPollutantNameA(int index);

Parameters: none

Output:

≠ NULL: Correct traffic demand name

= NULL: Error

3.3.12.6 Read periodical statistics data for a section

In C++ and Python:

Explanation: Read the statistical data gathered during a particular time interval, for a section

Format:

StructAkiEstadSection AKIEstGetParcialStatisticsSection(int aidarc, double timeSta, int vehTypePos);

Parameters:

aidarc: section Identifier

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadSection: structure with statistical information

3.3.12.7 Read global statistics data for a section

In C++ and Python:

Explanation: Read the statistical data gathered during the whole simulation, for a section

Format:

StructAkiEstadSection AKIEstGetGlobalStatisticsSection(int aidarc, int vehTypePos)

Parameters:

aidarc: section Identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadSection: structure with statistical information

3.3.12.8 Read periodical fuel consumption data for a section

In C++ and Python:

Explanation: Read the fuel consumption data gathered during a particular time interval, for a section

Format:

double AKIEstGetParcialStatisticsSectionFuelCons(int aidarc, double timeSta, int vehTypePos);

Parameters:

aidarc: section Identifier

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for that period

< 0: Error

3.3.12.9 Read global fuel consumption data for a section

In C++ and Python:

Explanation: Read the fuel consumption data gathered during the whole simulation, for a section

Format:

double AKIEstGetGlobalStatisticsSectionFuelCons(int aidarc, int vehTypePos);

Parameters:

aidarc: section Identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for whole simulation

< 0: Error

3.3.12.10 Read periodical pollution emission data for a section

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during a particular time interval, for a section

Format:

double AKIEstGetParcialStatisticsSectionPollution(int indexPol, int aidarc, double timeSta, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

aidarc: section Identifier

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for that period

< 0: Error

3.3.12.11 Read global pollution emission data for a section

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during the whole simulation, for a section

Format:

```
double AKIEstGetGlobalStatisticsSectionPollution(int indexPol, int aidarc, int
vehTypePos);
```

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

aidarc: section Identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for whole simulation

< 0: Error

3.3.12.12 Read periodical statistics data for the system

In C++ and Python:

Explanation: Read the statistical data gathered during certain time interval, for the whole system

Format:

```
StructAkiEstadSystem AKIEstGetParcialStatisticsSystem( double timeSta, int vehTypePos);
```

Parameters:

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadSystem: structure with statistical information.

3.3.12.13 Read global statistics data for the system

In C++ and Python:

Explanation: Read the statistical data gathered during the whole simulation, for the whole system

Format:

```
StructAkiEstadSystem AKIEstGetGlobalStatisticsSystem(int vehTypePos)
```

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadSystem: structure with statistical information

3.3.12.14 Read periodical fuel consumption data for the system

In C++ and Python:

Explanation: Read the fuel consumption data gathered during a particular time interval, for the whole system

Format:

double AKIEstGetParcialStatisticsSystemFuelCons(double timeSta, int vehTypePos);

Parameters:

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for that period

< 0: Error

3.3.12.15 Read global fuel consumption data for the system

In C++ and Python:

Explanation: Read the fuel consumption data gathered during the whole simulation, for the whole system

Format:

double AKIEstGetGlobalStatisticsSystemFuelCons(int vehTypePos);

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for whole simulation

< 0: Error

3.3.12.16 Read periodical pollution emission data for the system

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during a particular time interval, for the whole system

Format:

double AKIEstGetParcialStatisticsSystemPollution(int indexPol, double timeSta, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for that period

< 0: Error

3.3.12.17 Read global pollution emission data for the system

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during the whole simulation, for the whole system

Format:

double AKIEstGetGlobalStatisticsSystemPollution(int indexPol, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for whole simulation

< 0: Error

3.3.12.18 Read periodical statistics data for an O/D pair

In C++ and Python:

Explanation: Read the statistical data gathered during certain time interval, for an O/D Pair

Format:

StructAkiEstadODPair AKIEstGetParcialStatisticsODPair(int idOrigin, int idDestination, double timeSta, int vehTypePos)

Parameters:

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

idOrigin: the origin centroid identifier.

idDestination: the destination centroid identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadODPair: structure with statistical information.

3.3.12.19 Read global statistics data for an O/D pair

In C++ and Python:

Explanation: Read the statistical data gathered during the whole simulation, for an O/D Pair.

Format:

StructAkiEstadODPair AKIEstGetGlobalStatisticsODPair(int idOrigin, int idDestination, int vehTypePos);

Parameters:

idOrigin: the origin centroid identifier.

idDestination: the destination centroid identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadODPair: structure with statistical information.

3.3.12.20 Read periodical fuel consumption data for an O/D pair

In C++ and Python:

Explanation: Read the fuel consumption data gathered during a particular time interval, for an O/D Pair

Format:

double AKIEstGetParcialStatisticsODPairFuelCons(int idOrigin, int idDestination, double timeSta, int vehTypePos);

Parameters:

idOrigin: the origin centroid identifier.

idDestination: the destination centroid identifier.

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for that period

< 0: Error

3.3.12.21 Read global fuel consumption data for an O/D pair

In C++ and Python:

Explanation: Read the fuel consumption data gathered during the whole simulation, for an O/D Pair

Format:

double AKIEstGetGlobalStatisticsODPairFuelCons(int idOrigin, int idDestination, int vehTypePos);

Parameters:

idOrigin: the origin centroid identifier.

idDestination: the destination centroid identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for whole simulation

< 0: Error

3.3.12.22 Read periodical pollution emission data for an O/D pair

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during a particular time interval, for an O/D Pair

Format:

double AKIEstGetParcialStatisticsODPairPollution(int indexPol, int idOrigin, int idDestination, double timeSta, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

idOrigin: the origin centroid identifier.

idDestination: the destination centroid identifier.

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for that period

< 0: Error

3.3.12.23 Read global pollution emission data for an O/D pair

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during the whole simulation, for an O/D Pair

Format:

double AKIEstGetGlobalStatisticsODPairPollution(int indexPol, int idOrigin, int idDestination, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants()

dOrigin: the origin centroid identifier.

idDestination: the destination centroid identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for whole simulation

3.3.12.24 Read periodical statistics data for a Turning Movement

In C++ and Python:

Explanation: Read the statistical data gathered during certain time interval, for a turning movement (it only includes the measures related to the turning itself)

Format:

StructAkiEstadTurning AKIEstGetParcialStatisticsTurning(int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);

Parameters:

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadTurning: structure with statistical information

3.3.12.25 Read global statistics data for a Turning Movement

In C++ and Python:

Explanation: Read the statistical data gathered during the whole simulation, for a turning movement (it only includes the measures related to the turning itself)

Format:

StructAkiEstadTurning AKIEstGetGlobalStatisticsTurning(int aidsectionFrom, int aidsectionTo, int vehTypePos);

Parameters:

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadTurning: structure with statistical information

3.3.12.26 Read periodical fuel consumption data for a Turning Movement

In C++ and Python:

Explanation: Read the fuel consumption data gathered during a particular time interval, for a turning movement

Format:

double AKIEstGetParcialStatisticsTurningFuelCons(int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);

Parameters:

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for that period

< 0: Error

3.3.12.27 Read global fuel consumption data for a Turning Movement

In C++ and Python:

Explanation: Read the fuel consumption data gathered during the whole simulation, for a turning movement

Format:

double AKIEstGetGlobalStatisticsTurningFuelCons(int aidsectionFrom, int aidsectionTo, int vehTypePos);

Parameters:

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for whole simulation

< 0: Error

3.3.12.28 Read periodical pollution emission data for a Turning Movement

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during a particular time interval, for a turning movement

Format:

double AKIEstGetParcialStatisticsTurningPollution(int indexPol, int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for that period

< 0: Error

3.3.12.29 Read global pollution emission data for a Turning Movement

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during the whole simulation, for a turning movement

Format:

double AKIEstGetGlobalStatisticsTurningPollution(int indexPol, int aidsectionFrom, int aidsectionTo, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants()

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for whole simulation

3.3.12.30 Read periodical statistics data for a Link

In C++ and Python:

Explanation: Read the statistical data gathered during certain time interval, for a link movement (it includes the measures related to the section origin plus the turning movement)

Format:

StructAkiEstadTurning AKIEstGetParcialStatisticsLink(int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);

Parameters:

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadTurning: structure with statistical information

3.3.12.31 Read global statistics data for a Link Movement

In C++ and Python:

Explanation: Read the statistical data gathered during the whole simulation, for a link movement (it includes the measures related to the origin section plus the turning movement)

Format:

StructAkiEstadTurning AKIEstGetGlobalStatisticsLink(int aidsectionFrom, int aidsectionTo, int vehTypePos);

Parameters:

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadTurning: structure with statistical information

3.3.12.32 Read periodical fuel consumption data for a Link Movement

In C++ and Python:

Explanation: Read the fuel consumption data gathered during a particular time interval, for a link movement (it includes the measures related to the origin section plus the turning movement)

Format:

double AKIEstGetParcialStatisticsLinkFuelCons(int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);

Parameters:

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for that period

< 0: Error

3.3.12.33 Read global fuel consumption data for a Link Movement

In C++ and Python:

Explanation: Read the fuel consumption data gathered during the whole simulation, for a link movement (it includes the measures related to the origin section plus the turning movement)

Format:

double AKIEstGetGlobalStatisticsTurningFuelCons(int aidsectionFrom, int aidsectionTo, int vehTypePos);

Parameters:

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for whole simulation

< 0: Error

3.3.12.34 Read periodical pollution emission data for a Link Movement

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during a particular time interval, for a link movement (it includes the measures related to the origin section plus the turning movement)

Format:

double AKIEstGetParcialStatisticsLinkPollution(int indexPol, int aidsectionFrom, int aidsectionTo, double timeSta, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for that period

< 0: Error

3.3.12.35 Read global pollution emission data for a Link Movement

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during the whole simulation, for a link movement (it includes the measures related to the origin section plus the turning movement)

Format:

double AKIEstGetGlobalStatisticsTurningPollution(int indexPol, int aidsectionFrom, int aidsectionTo, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

aidsectionFrom: origin section identifier of the turning movement

aidsectionTo: destination section identifier of the turning movement

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for whole simulation

3.3.12.36 Read periodical statistics data for a Statistical Stream

In C++ and Python:

Explanation: Read the statistical data gathered during certain time interval, for a statistical stream

Format:

StructAkiEstadStream AKIEstGetParcialStatisticsStream(int aidstream, double timeSta, int vehTypePos);

Parameters:

aidstream: statitistical stream identifier

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadStream: structure with statistical information

3.3.12.37 *Read global statistics data for a Statistical Stream*

In C++ and Python:

Explanation: Read the statistical data gathered during the whole simulation, for a statistical stream

Format:

StructAkiEstadStream AKIEstGetGlobalStatisticsStream(int aidstream, int vehTypePos)

Parameters:

aidstream: statitistical stream identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

StructAkiEstadStream: structure with statistical information

3.3.12.38 Read periodical fuel consumption data for a Statistical Stream

In C++ and Python:

Explanation: Read the fuel consumption data gathered during a particular time interval, for a statistical stream

Format:

double AKIEstGetParcialStatisticsStreamFuelCons(int aidstream, double timeSta, int vehTypePos);

Parameters:

aidstream: statitistical stream identifier

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for that period

< 0: Error

3.3.12.39 Read global fuel consumption data for a Statistical Stream

In C++ and Python:

Explanation: Read the fuel consumption data gathered during the whole simulation, for a statistical stream

Format:

double AKIEstGetGlobalStatisticsStreamFuelCons(int aidstream, int vehTypePos);

Parameters:

aidstream: statitistical stream identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: fuel consumption data for the section for whole simulation

< 0: Error

3.3.12.40 Read periodical pollution emission data for a Statistical Stream

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during a particular time interval, for a statistical stream

Format:

double AKIEstGetParcialStatisticsStreamPollution(int indexPol, int aidstream, double timeSta, int vehTypePos);

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

aidstream: statitistical stream identifier

timeSta: time interval (when the data is produced, which is the end of the interval), in seconds from midnight.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for that period

< 0: Error

3.3.12.41 Read global pollution emission data for a Statistical Stream

In C++ and Python:

Explanation: Read the emission for a given pollutant gathered during the whole simulation, for a statistical stream

Format:

```
double AKIEstGetGlobalStatisticsStreamPollution(int indexPol, int aidstream, int
vehTypePos);
```

Parameters:

indexPol: Pollutant index, where indexPol takes values form 0 to AKIEstGetNbPollutants( )

aidstream: statitistical stream identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: emission for the pollutant requested in the section for whole simulation

3.3.12.42 Read section instant virtual queue statistics

In C++ and Python:

Explanation: Read the instant virtual queue given the vehicle type, for a section

Format:

```
double AKIEstGetInstantVirtualQueueSection (int aidarc, int vehTypePos);
```

Parameters:

aidarc: section identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: number of vehicles in the virtual queue

3.3.12.43 Read periodical lost vehicles statistics for a node

In C++ and Python:

Explanation: Read the number of lost vehicles gathered in the current statistical interval, for a node

Format:

```
int AKIEstGetPartialStatisticsNodeLostVehicles( int aidNode, int vehTypePos)
```

Parameters:

aidNode: node Identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: number of lost vehicles in the statistical interval for the specified node

3.3.12.44 Read global lost vehicles statistics for a node

In C++ and Python:

Explanation: Read the number of lost vehicles gathered during the whole simulation, for a node

Format:

int AKIEstGetGlobalStatisticsNodeLostVehicles(int aidNode, int vehTypePos)

Parameters:

aidNode: node Identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: number of lost vehicles in the whole simulation for the specified node

3.3.12.45 Read periodical level of service statistics for a node

In C++ and Python:

Explanation: Read the delay weighted by flow gathered in the current statistical interval of all the entrance sections in a node to help calculate the level of service, for a node.

Format:

int AKIEstGetPartialStatisticsNodeLevelOfService (int aidNode)

Parameters:

aidNode: node Identifier

Output:

≥ 0: delay to calculate the level of service in the current statistical interval for the specified node

3.3.12.46 Read global level of service statistics for a node

In C++ and Python:

Explanation: Read the delay weighted by flow gathered during the whole simulation of all the entrance sections in a node to help calculate the level of service, for a node.*Format:*

int AKIEstGetGlobalStatisticsNodeLevelOfService (int aidNode)

Parameters:

aidNode: node Identifier

Output:

≥ 0: delay to calculate the level of service for the whole simulation for the specified node

3.3.12.47 Read periodical missed turn statistics for a node

In C++ and Python:

Explanation: Read the number of missed turns gathered in the current statistical interval, for all the turns in a node

Format:

int AKIEstGetGlobalStatisticsNodeMissedTurns (int aidNode, int vehTypePos)

Parameters:

aidNode: node Identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: number of number of missed turns in the current statistical interval for the specified node

3.3.12.48 Read global missed turn statistics for a node

In C++ and Python:

Explanation: Read the number of missed turns gathered during the whole simulation, for all the turns in a node

Format:

int AKIEstGetPartialStatisticsNodeMissedTurns (int aidNode, int vehTypePos)

Parameters:

aidNode: node Identifier

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

≥ 0: number of number of missed turns in the current statistical interval for the specified node

3.3.13 Functions relative to vehicle entrance

3.3.13.1 Introduce a Vehicle in flows and turning proportions traffic definition

In C++ and Python:

Explanation: Introduce a new vehicle at the entrance of the specified section only if there is enough space. This function should be used only when the traffic demand is defined by flows and turnings proportions. In case the vehicle has been entered, Aimsun assigns the vehicle lane and vehicle initial speed using the internal models. If there is not enough space, the vehicle is not generated.

Format:

int AKIEnterVehTrafficFlow(int asection, int vehTypePos, int tracking);

Parameters:

asection: section Identifier where it will enter a vehicle

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

tracking: 0 when the vehicle has not to be tracked, 1 otherwise.

Output:

≥ 0: Identifier of the vehicle entered

< 0: Error

3.3.13.2 Introduce a Vehicle in O/D matrix traffic definition

In C++ and Python:

Explanation: Introduce a vehicle at the entrance of the specified section having as origin and destination centroids the specified ones. This function should be used only when the traffic demand is O/D matrix based. The vehicle is entered in the network when the function is called only if there is enough space and Aimsun assigns the vehicle lane and vehicle initial speed using the internal models. If there is not enough space, the vehicle is not generated.

Format:

int AKIEnterVehTrafficOD(int asection, int vehTypePos, int idCentroidOrigin, int idCentroidDest, int tracking);

Parameters:

asection: section Identifier where it will enter a vehicle

vehTypePos is the position of the vehicle type in the list of vehicles types being used. The value goes from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

idCentroidOrigin: identifier of origin centroid of the vehicle.

idCentroidDest: identifier of destination centroid of the vehicle.

tracking: 0 when the vehicle will not be tracked, 1 otherwise.

Output:

≥ 0: Identifier of the vehicle entered

< 0: Error

3.3.13.3 Put a Vehicle in flows and turning proportions traffic definition

In C++ and Python:

Explanation: Generate a new vehicle in the specified section, lane and position with the specified speed. This function should be used only when the traffic demand is defined by flows and turnings proportions.

Format:

int AKIPutVehTrafficFlow (int asection, int idLane, int vehTypePos, double initPosition, double initSpeed, int nextSection, int tracking);

Parameters:

asection: section Identifier where it will enter a vehicle

idLane: idLane Identifier where it will enter a vehicle (1..N)

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

initPosition: Initial position of the vehicle (in meters or feet depending on network units)

initSpeed: Initial speed of the vehicle (in Km/h or mph depending on network units)

nextSection: next section that the vehicle will take, if it is set to –1 means the first turning feasible.

tracking: 0 when the vehicle will not be tracked, 1 otherwise.

Output:

≥ 0: Identifier of the vehicle entered

< 0: Error

3.3.13.4 Put a Vehicle in O/D matrix traffic definition

In C++ and Python:

Explanation: Generate a new vehicle in the specified section, lane and position with the specified speed. It sets also the origin and destination centroids of the new vehicle. This function should be used only when the traffic demand is O/D matrix based.

Format:

int AKIPutVehTrafficOD(int asection, int idLane, int vehTypePos, int idCentroidOr, int idCentroidDest, double initPosition, double initSpeed, int tracking);

Parameters:

asection: section Identifier where it will enter a vehicle

idLane: idLane Identifier where it will enter a vehicle (1..N)

vehTypePos is the position of the vehicle type in the list of vehicles types being used. The value goes from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

idCentroidOrigin: identifier of origin centroid of the vehicle.

idCentroidDest: identifier of destination centroid of the vehicle.

initPosition: Initial position of the vehicle.

initSpeed: Initial speed of the vehicle.

tracking: 0 when the vehicle will not be tracked, 1 otherwise.

Output:

≥ 0: Identifier of the vehicle entered

< 0: Error

3.3.13.5 Introduce a Vehicle in flows and turning proportions traffic definition using Aimsun entrance models

In C++ and Python:

Explanation: : Introduce a new vehicle at the entrance of the specified section only if there is enough space. This function should be used only when the traffic demand is defined by flows and turnings proportions. In case the vehicle has been entered, Aimsun assigns the vehicle lane and vehicle initial speed using the internal models. The user can decide , in the case that a vehicle has no space to introduced, whether to add this vehicle into the virtual entrance queue or not.

Format:

InfArrival AKIGenerateArrivalTrafficFlow(int asection, int vehTypePos, int useVirtualQueue

Parameters:

asection: section Identifier where it will enter a vehicle

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

useVirtualQueue: 1 if the vehicle has no space to enter then the vehicle is stored in the virtual entrance queue, 0 is not stored in the virtual queue.

Output:

```
struct InfArrival {
        int report;
        int idVeh;
        bool inVirtualQueue;
        int entranceSection;
};
```

where:

int *report*: 0, OK, else error code

int *idVeh*: >0 vehicle identifier entered in the system

int *entranceSection*: entrance section identifier where the vehicle has been entered or stored in the virtual entrance queue.

bool *inVirtualQueue*: true the vehicle has not enter in the system but has been stored in the virtual entrance Queue, false otherwise

3.3.13.6 Introduce a Vehicle into the O/D matrix traffic definition using Aimsun entrance models

In C++ and Python:

Explanation: Introduce a vehicle at the entrance of the specified section having as origin and destination centroids the specified ones. This function should be used only when the traffic demand is O/D matrix based. The vehicle is entered in the network when the function is called only if there is enough space and Aimsun assigns the vehicle lane and vehicle initial speed using the internal models. The user can decide , in the case that a vehicle has no space to introduced, whether to add this vehicle into the virtual entrance queue or not.

Format:

InfArrival AKIGenerateArrivalTrafficOD(int vehTypePos, int idCentroidOrigin, int idCentroidDest, int useVirtualQueue);

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used. The value goes from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

idCentroidOrigin: identifier of origin centroid of the vehicle.

idCentroidDest: identifier of destination centroid of the vehicle.

useVirtualQueue: 1 if the vehicle has no space to enter then the vehicle is stored in the virtual entrance queue, 0 is not stored in the virtual queue.

Output:

```
struct InfArrival {
        int report;
        int idVeh;
        bool inVirtualQueue;
        int entranceSection;
};
```

where:

int *report*: 0, OK, else error code

int *idVeh*: >0 vehicle identifier entered in the system

int *entranceSection*: entrance section identifier where the vehicle has been entered or stored in the virtual entrance queue.

bool *inVirtualQueue*: true the vehicle has not enter in the system but has been stored in the virtual entrance Queue, false otherwise

3.3.13.7 Introduce Legion Pedestrians into the model

In C++ and Python:

Explanation: Introduce Legion Pedestrians into the model detailing their origin, destination, route and finally the number of pedestrians. Using this method, pedestrians will enter the model as soon as they can.

Format:

int AKIGeneratePedestrians(int fromCentroid, int toCentroid, int idRoute, double nbPedestrians);

Parameters:

fromCentroid is the Legion Entrance Centroid id.

toCentroid is the Legion Exit Centroid id.

idRoute is the route id that pedestrians will follow. When providing a -1 as idRoute, pedestrians will follow the shortest path to their destination.

nbPedestrians is the number of pedestrians to generate.

Output:

≥ 0: Pedestrians have entered successfully in the model.

< 0: Error

3.3.13.8 Introduce Legion Pedestrians into the model during a time interval

In C++ and Python:

Explanation: Introduce Legion Pedestrians into the model detailing their origin, destination, route, the number of pedestrians and the interval time. Using this method, pedestrians will enter the model following a uniform distribution during the time interval.

Format:

int AKIGeneratePedestriansInTime(int fromCentroid, int toCentroid, int idRoute, double nbPedestrians, double timeInterval);

Parameters:

fromCentroid is the Legion Entrance Centroid id.

toCentroid is the Legion Exit Centroid id.

idRoute is the route id that pedestrians will follow. When providing a -1 as idRoute, pedestrians will follow the shortest path to their destination.

nbPedestrians is the number of pedestrians to generate.

timeInterval is the time interval in seconds when the pedestrians will be generated.

Output:

≥ 0: Pedestrians have entered successfully in the model.

< 0: Error

3.3.14 Functions relative to vehicle tracking

3.3.14.1 Modify the Speed of a Tracked Vehicle

In C++ and Python:

Explanation: Modify the speed of a tracked vehicle for the next simulation step.

Format:

int AKIVehTrackedModifySpeed (int aidVeh, double newSpeed);

Parameters:

aidVeh: tracked vehicle Identifier.

newSpeed: new Speed (Km/h or Mph depending the units defined in the network).

Output:

= 0: No Error

< 0: Error

3.3.14.2 Modify the Lane of a Tracked Vehicle

In C++ and Python:

Explanation: Modify the target lane of a tracked vehicle until it has performed the lane change.

Format:

int AKIVehTrackedModifyLane (int aidVeh, int nextLane);

Parameters:

aidVeh: tracked vehicle identifier.

newLane: -1 next right lane or 1 next left lane.

0 do not change lane.

-2 reset, gives back the control of the target lane to Aimsun´s model.

Output:

= 0: No Error

< 0: Error

3.3.14.3 Modify the Next Section of a Tracked Vehicle

In C++ and Python:

Explanation: Modify the next section of a tracked vehicle in order to force a turning, for the next simulation step.

Format:

int AKIVehTrackedModifyNextSection (int aidVeh, int nextSection);

Parameters:

aidVeh: tracked vehicle identifier.

nextSection: next section.

Output:

= 0: No Error

< 0: Error

3.3.14.4 Modify the Next Sections of a Tracked Vehicle

In C++ only:

Explanation: Modify the next sections of a tracked vehicle in order to force a subpath, for the next simulation step. The first section given should be a section immediately

downstream to the vehicle's current section. The list of section does not need to be up to the vehicle's destination centroid. Once off the last section of the given subpath the vehicle will calculate a route from its location to its destination.

Format:

int AKIVehTrackedModifyNextSections (int aidVeh, int sizeNextSections, const int *nextSections);

Parameters:

aidVeh: tracked vehicle identifier.

sizeNextSections: number of sections in the subpath

nextSections: array with the IDs of the next sections. As much sections as sizeNextSections must be provided

Output:

= 0: No Error

< 0: Error

Usage example:

```
int * pathArray = (int *) calloc( pathSize, sizeof(int) );
for( int i = 0; i < pathSize; i++ ){
    pathArray[i] = sectionID[i];
}
int err = AKIVehTrackedModifyNextSections( vehId, pathSize, pathArray );
free( pathArray );
```

3.3.14.5 Remove a Tracked Vehicle

In C++ and Python:

Explanation: Remove a tracked vehicle.

Format:

int AKIVehTrackedRemove (int aidVeh);

int AKIVehTrackedDelete (int aidVeh); //Old function, kept for compatibility

Parameters:

aidVeh: tracked vehicle identifier.

Output:

= 0: No Error

< 0: Error

3.3.14.6 Set a Vehicle as Tracked / Set a Vehicle as Not Tracked

In C++ and Python:

Explanation: Set a vehicle as tracked/ Set a Vehicle as not tracked

Format:

int AKIVehSetAsTracked (int aidVeh);

int AKIVehSetAsNoTracked (int aidVeh);

Parameters:

aidVeh: vehicle identifier.

Output:

= 0: No Error

< 0: Error

3.3.14.7 Read the information of a Tracked Vehicle

In C++ and Python:

Explanation: Read the information of a tracked vehicle

Format:

InfVeh AKIVehTrackedGetInf(int aidVeh)

Parameters:

aidVeh: tracked vehicle identifier.

Output:

```
struct InfVeh{
        int report;
        int idVeh;
        int type;
        // Information in Vehicle when it is in a section
        int idSection;
        int segment;
        int numberLane;

        // Information in Vehicle when it is in a node
        int idJunction;
        int idSectionFrom;
        int idLaneFrom;
        int idSectionTo;
        int idLaneTo;

        double CurrentPos;
        double distnace2End;
        double xCurrentPos, yCurrentPos, zCurrentPos;
        double xCurrentPosBack, yCurrentPosBack, zCurrentPosBack;
        double CurrentSpeed, PreviousSpeed;

        double TotalDistance;

        double SystemGenerationT;
        double SystemEntranceT;
        double SectionEntranceT;
        double CurrentStopTime

};
```

where:

int *report*: 0, OK, otherwise is an error code

int *idVeh*: the vehicle identifier

int *type*: the vehicle type (car, bus, truck, etc.).

int *idSection*: the section identifier.

int *segment*: segment number of the section where the vehicle is located (from 0 to n-1)

int *numberLane*: lane number in the segment (from 1, the rightmost lane, to N, the leftmost lane).

int *idSectionFrom*: origin section identifier.

int *idLaneFrom*: number of lane of the origin section where the vehicle enters the junction.

int *idSectionTo*: destination section identifier.

int *idLaneTo*: lane number of the destination section where the vehicle leaves the junction

double *CurrentPos*: position inside the section = distance (metres or feet, depending on the units defined in the network) from the beginning of the section or position inside the junction given as the distance from the entrance to the junction.

double *distance2End*: distance to end of the section (metres or feet, depending on the units defined in the network) when the vehicle is located in a section or the distance to the end of the turning when the vehicle is in a junction.

double *xCurrentPos*, *yCurrentPos*, *zCurrentPos*: world coordinates of the middle point of the front bumper of the vehicle.

double *xCurrentPosBack*, *yCurrentPosBack*, *zCurrentPosBack*: world coordinates of the midle point of the rear bumper of the vehicle.
double *CurrentSpeed*: current speed (in km/h or mph, depending on the units defined in the network.
double *PreviousSpeed*: speed in the previous simulation step (in km/h or mph, depending on the units defined in the network.
double *TotalDistance*: total distance travelled (metres or feet).
double *SystemGenerationT*: the absolute generation time of the vehicle into the system. If no virtual queue found in its entrance section it will be the same as the SystemEntranceT.
double *SystemEntranceT*: the absolute entrance time of the vehicle into the system, that is into its entrance section. If no virtual queue found in its entrance section it will be the same as the SystemGenerationT.
double *SectionEntranceT*: the absolute entrance time of the vehicle into the current section.
double *CurrentStopTime*: the current stop time.

3.3.14.8 Read the Static Information of a Vehicle Tracked

In C++ and Python:

Explanation: Read the static information of the tracked vehicle. Static information means the characteristics of the vehicle which have been set when the vehicle has entered the system.

Format:

StaticInfVeh AKIVehTrackedGetStaticInf (int aidVeh)

Parameters:

aidVeh: tracked vehicle identifier.

Output:

```
struct StaticInfVeh{
        int report;
        int idVeh;
        int type;
        double length;
        double width;
        double maxDesiredSpeed;
        double maxAcceleration;
        double normalDeceleration;
        double maxDeceleration;
        double speedAcceptance;
        double minDistanceVeh;
        double giveWayTime;
        double guidanceAcceptance;
        int enrouted;
        int equipped;
        int tracked;

        bool  keepfastLane;
        double headwayMin;
        double sensitivityFactor;
        double reactionTime;
        double reactionTimeAtStop;
        double reactionTimeAtTrafficLight;

        int centroidOrigin;
        int centroidDest;
```

```
        int idsectionExit;

        int idLine;
        void * internalInfo;
};
```

where:

int *report*: 0, OK, else error code
int *idVeh*: vehicle identifier
int *type*: vehicle type (car, bus, truck, etc.)
double *length*: vehicle length (m or feet, depending on the units defined in the network).
double *width*: vehicle width (m or feet, depending on the units defined in the network).
double *maxDesiredSpeed*: Maximum desired speed of the vehicle (km/h or mph, depending on the units defined in the network).
double *maxAcceleration*: Maximum acceleration of the vehicle (m/s^2 or ft/ s^2, depending on the units defined in the network).
double *normalDeceleration*: Maximum deceleration of the vehicle that can apply under normal conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *maxDeceleration*: Maximum deceleration of the vehicle that can apply under special conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *speedAcceptance*: degree of acceptance of the speed limits.
double *minDistanceVeh*: distance that the vehicle keeps between itself and the preceding vehicle (meters or feet, depending on the units defined in the network).
double *giveWayTime*: time after which the vehicle becomes more aggressive in give-way situations (seconds).
double *guidanceAcceptance*: level of compliance of the vehicle to guidance indications.
int *enrouted*: 0 means vehicle will not change path en-route, 1 means vehicle will change path en-route depending on the percentege of enrouted vehicles defined.
int *equipped*: 0 means vehicle not equipped, 1 means vehicle equipped.
int *tracked*: 0 means vehicle not tracked, 1 means vehicle tracked.
bool *keepfastLane*: means vehicle keep fast lane during overtaking
double *headwayMin*: minimum headway to keep with its leader
double *sensitivityFactor*: estimation of the acceleration of the leader
double *reactionTime*: reaction time of the vehicle
double *reactionTimeAtStop*: reaction time at stop of the vehicle
double *reactionTimeAtTrafficLight*: reaction time of the vehicle when stopped the first one of the queue in a traffic light.
int *centroidOrigin*: Identifier of centroid origin of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *centroidDest*: : Identifier of centroid destination of the vehicle, when the traffic conditions are defined by an O/D matrix.
int *idsectionExit*: : Identifier of exit section destination of the vehicle, when the destination centroid uses percentages as destination (otherwise is –1) and the traffic conditions are defined by an O/D matrix.
int *idLine*: Identifier of Public Transport Line, when the vehicle has been generated as a public transport vehicle.
void * *internalInfo*: only for internal use

3.3.14.9 Modify the Static Information of a Tracked Vehicle

In C++ and Python:

Explanation: Modify some static parameters of certain tracked vehicle. Static parameters mean the characteristics of the vehicle have been set when the vehicle has entered the system. The static parameters that it is possible to change are: *type*, *length, width, maxDesiredSpeed, maxAcceleration, normalDeceleration, maxDeceleration, speedAcceptance, minDistanceVeh, giveWayTime, guidanceAcceptance*, *enrouted, equipped, tracked, keepfastLane , headwayMin, sensitivityFactor, reactionTime, reactionTimeAtStop, reactionTimeAtTrafficLight, centroidOrigin, centroidDest and idsectionExit* (the exit section has sense when the destination centroid uses percentages as destination; if the identifier of the section is invalid or –1 then Aimsun applies determines the exit section according to criteria defined in the centroid). Aimsun cannot store the previous values, so it cannot recover them.

Format:

int AKIVehTrackedSetStaticInf (int aidVeh, StaticInfVeh staticinfVeh)

Parameters:

aidVeh: the tracked vehicle Identifier.

staticinfVeh: the new static parameters to be assigned.

Output:

= 0: No Error

< 0: Error

3.3.14.10 Read the position(s) of a Tracked Vehicle

In C++ and Python:

Explanation: Read the coordinates of the tracked vehicle during the last simulation step.

Format:

InfVehPos AKIVehTrackedGetPos(int anIdVeh, int nbPos)

Parameters:

aidVeh: tracked vehicle identifier.

nbPos: number of positions of the tracked vehicle. If this parameter is greater than 1 then the function gives the "between-simulating-step" positions for this vehicle. The last position always corresponds to the position at the end of the simulation step.

Output:

```
struct InfVehPos{
        int report;
        int idVeh;
        int Npos;
        VehPos *vehiclePos;
};

struct VehPos{
        double xPos, yPos, zPos;
        double xPosBack, yPosBack, zPosBack;
};
```

where:

int *report*: 0, OK, otherwise an error code

int *idVeh*: the vehicle identifier

int *Npos*: number of positions during last simulation step (input parameter).

VehPos **vehiclePos:* Array with the positions during last simulation step. After using this function, this array must be deallocated using “free” function.

double *xPos*, *yPos, zPos*: world coordinates of the middle point of a vehicle front bumper for the current position
double *xPosBack*, *yPosBack, zPosBack*: world coordinates of the middle point of the vehicle rear bumper for the current position

3.3.14.11 Modify the position of a Tracked Vehicle

In C++ and Python:

Explanation: Modify the position of the tracked vehicle during the last simulation step, considering the world coordinates on the middle point of the vehicle front and rear bumpers.

Format:

int AKIVehSetVehicleTrackedDynInf(int anIdVeh, DynInfVeh dynInfVeh)

Parameters:

aidVeh: tracked vehicle Identifier.
dynInfVeh: the dynamic information during the last simulation step where:

```
struct DynInfVeh{
        double xCurrentPos, yCurrentPos;
        double xCurrentPosBack, yCurrentPosBack;
        double currentSpeed;
        int turning;
};
```

double *xCurrentPos*, *yCurrentPos*: world coordinates of the middle point of the vehicle front bumper for the current position
double *xCurrentPosBack*, *yCurrentPosBack*: world coordinates of the middle point of the vehicle rear bumper for the current position
double *currentSpeed*: current speed (km/h or mph, depending the units defined in the network).
int *turning*: next turning selected by the vehicle (-1 right turning, 1 left turning, 0 other cases)

Output:

= 0: No Error
< 0: Error

3.3.14.12 Read the graphical information of a Tracked Vehicle

In C++ and Python:

Explanation: Read the graphical information of the tracked vehicle during the last simulation step.

Format:

GraphicInfVeh AKIVehTrackedGetGraphicInf (int anIdVeh)

Parameters:

aidVeh: tracked vehicle Identifier.

Output:

```
struct GraphicInfVeh{
        int report;
        int idVeh;
        bool leftTurnSignal;
        bool rightTurnSignal;
        bool brakeLight;
}
```

3.3.14.13 Read the number of sections to reach destination

In C++ and Python:

Explanation: Read the number of sections to reach destination of a tracked vehicle path. Valid for OD and PT vehicles, otherwise returns 0.

Format:

int AKIVehTrackedGetNbSectionsVehiclePath(int idveh)

Parameters:

idveh: the tracked vehicle Identifier.

Output:

>= 0: Number of sections to reach destination.

AKIVehNotTracked: Error

3.3.14.14 Read the Identifiers of sections to reach destination

In C++ and Python:

Explanation: Read the identifier of section elem-th of the tracked vehicle path to reach destination. Valid for OD and PT vehicles, otherwise returns 0. AKIVehTrackedGetNbSectionsVehiclePath must be called before call this function.

Format:

int AKIVehTrackedGetIdSectionVehiclePath(int idveh, int indexsection)

Parameters:

idveh: the tracked vehicle Identifier.

indexsection: section index (0 <= indexsection < Number of sections)

Output:

> 0: Section Identifier.

AKIVehNotTracked: Error

3.3.15 Functions relative to Incidents

3.3.15.1 Generate an Incident

In C++ and Python:

Explanation: Generate an incident.

Format:

int AKIGenerateIncident(int asection, int alane, double position, double length, double initime, double duration, double visibilityDistance, bool updateIdGroup);

Parameters:

asection: the identifier of the section where the incident will be generated.

alane: the lane where the incident will be generated (1..N).

position: the position of the incident.

length: the length of the incident.

initime: the absolute time of the simulation when the incident will start (seconds).

duration: the duration of the incident(seconds).

visibilityDistance: visibility distance of the incident to be used in the 7.0 models, in meters. 200 meters is the default value when creating an incident using the graphical user interface.

updateIdGroup: true when the incident is a new group of incidents and false if the incidents wants to be treated as a part of the last created incident (when creating incidents in adjacents lanes that need to be treated as a whole).

Output:

> 0: No Error

< 0: Error

3.3.15.2 Remove an Incident

In C++ and Python:

Explanation: Remove an incident.
Format:
int AKIRemoveIncident(int asection, int alane, double position);
Parameters:
asection: the identifier of the section where the incident to remove is located.
alane: lane where the incident will be generated.
position: position of the incident in the section (from the beginning of the section).
Output:
= 0: No Error
< 0: Error

3.3.15.3 Remove all Incidents in a section

In C++ and Python:
Explanation: Remove all incidents active in a section.
Format:
int AKIRemoveAllIncidentsInSection (int asection);
Parameters:
asection: the identifier of the section where the incidents to remove are located.
Output:
= 0: No Error
< 0: Error

3.3.15.4 Reset All Incidents

In C++ and Python:
Explanation: Removes all incidents created using the Aimsun API Module and initializes the list of incidents with the initial incidents (incidents loaded when Aimsun loads the traffic)
Format:
int AKIResetAllIncidents();
Parameters: none
Output:
= 0: No Error
< 0: Error

3.3.16 Functions relative to Get Run Time Information

3.3.16.1 Get Initial Run Time Simulation

In C++ and Python:
Explanation: Get the initial run time simulation (seconds).
Format:
double AKIGetIniSimTime ();
Parameters: none
Output:
> 0: Initial time
< 0: Error

3.3.16.2 Get End Run Time Simulation

In C++ and Python:
Explanation: Get the end run time simulation (seconds).
Format:
double AKIGetEndSimTime ();
Parameters: none

Output:

> 0: End time

< 0: Error

3.3.16.3 Get Transitory Time

In C++ and Python:

Explanation: Get the transitory time simulation (seconds).

Format:

double AKIGetDurationTransTime ();

Parameters: none

Output:

> 0: Transitory time

< 0: Error

3.3.16.4 Get Simulation Step

In C++ and Python:

Explanation: Get the simulation step duration (seconds).

Format:

double AKIGetSimulationStepTime ();

Parameters: none

Output:

> 0: Simulation step

< 0: Error

3.3.16.5 Get Simulation Time

In C++ and Python:

Explanation: Get the current simulation time (seconds).

Format:

double AKIGetCurrentSimulationTime();

Parameters: none

Output:

> 0: The current Simulation Time, that is the number of seconds from the beginning of the simulation, taking into account the transitory period.

< 0: Error

3.3.16.6 Get Stationary Time

In C++ and Python:

Explanation: Get the current simulation time (seconds).

Format:

double AKIGetTimeSta();

Parameters: none

Output:

> 0: The current Simulation Time, that is the number of seconds from the beginning of the simulation, taking into account the transitory period.

< 0: Error

3.3.16.7 Set End Run Time Simulation

In C++ and Python:

Explanation: Set the end run time (seconds).

Format:

int AKISetEndSimTime (double atime);

Parameters:

atime: represents the new end simulation time

Output:

= 0 No error

< 0: Error

3.3.17 Functions relative to Manage Public Transport

3.3.17.1 Read number of Public Transport Lines

In C++ and Python:

Explanation: Read the number of public transport lines loaded.

Format:

int AKIPTGetNumberLines ()

Parameters: none

Output:

≥ 0: Correct number of public transport lines

< 0: Error

3.3.17.2 Read the identifier of a Public Transport Line

In C++ and Python:

Explanation: Read the identifier of a public transport line.

Format:

int AKIPTGetIdLine(int elem);

Parameters:

elem: represents the number of the public transport line and must be in the range 0 ≤ elem < Number of Lines -1.

Output:

≥ 0: Correct public transport line identifier.

< 0: Error

3.3.17.3 Read the number of sections in a Public Transport Line

In C++ and Python:

Explanation: Read the number of sections that define the route of a public transport line.

Format:

int AKIPTGetNumberSectionsInLine (int lineId)

Parameters:

lineId: represents the public transport line identifier.

Output:

≥ 0: Number of sections that define the public transport line route

< 0: Error

3.3.17.4 Read the identifier of a section in a Public Transport Line

In C++ and Python:

Explanation: Reads the section identifier of the n-th section in a public transport line route.

Format:

int AKIPTGetIdSectionInLine(int lineId, int elem);

Parameters:

lineId: represents the public transport line identifier.

elem: represents the index of the section that can be obtained. It must be in the range 0 ≤ elem < Number of Sections -1.

Output:

≥ 0: Identifier of the elem-th section in the public transport line route

< 0: Error

3.3.17.5 Read the number of stops in a Public Transport Line

In C++ and Python:

Explanation: Read the number of stops in a public transport line.

Format:

int AKIPTGetNumberStopsInLine(int lineId)

Parameters:

lineId: represents the public transport line identifier.

Output:

≥ 0: Correct number of stops in the public transport line with ID lineid

< 0: Error

3.3.17.6 Read the identifier of a stop in a Public Transport Line

In C++ and Python:

Explanation: Read the stop identifier in a public transport line.

Format:

int AKIPTGetIdStopsInLine(int lineId, int elem);

Parameters:

lineId: represents the public transport line identifier.

elem: represents the position of the stop in the list of stops of the line. A value from 0 to AKIPTGetNumberStopsInLine(for the pt line) must be used.

Output:

≥ 0: Identifier of elem-th stop in the public transport line

< 0: Error

3.3.17.7 Introduce a Public Transport Vehicle

In C++ and Python:

Explanation: Introduce a public transport vehicle into the system.

Format:

int AKIPTEnterVeh (int lineId, int vehTypePos, bool tracked);

Parameters:

lineId: the public transport line identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. A value from 1 to AKIVehGetNbVehTypes () must be used.

tracking: false when the vehicle has not to be tracked, true otherwise.

Output:

≥ 0: vehicle identifier for the vehicle entered

< 0: Error

3.3.17.8 Read a Stop Time of a Public Transport Vehicle

In C++ and Python:

Explanation: Read a stop time of a vehicle.

Format:

double AKIPTGetServiceTimeStopsInLine (int aidVeh, int nstop)

Parameters:

aidVeh: the public transport vehicle identifier.

nstop: position of the stop to modify its stop time in the list of stops of the line. A value from 0 to AKIPTGetNumberStopsInLine(for the vehicle's pt line) must be used.

Output:

≥ 0: Stop Time

< 0: Error

3.3.17.9 Modify a Stop Time of a Public Transport Vehicle

In C++ and Python:

Explanation: Modify the stop time of a vehicle. When the new stop time is set to zero the stop will be skipped by the vehicle.

Format:

int AKIPTVehModifyStopTime (int aidVeh, int nstop, double stopTime);

Parameters:

aidVeh: the public transport vehicle identifier.

nstop: position of the stop to modify its stop time in the list of stops of the line. A value from 0 to AKIPTGetNumberStopsInLine(for the vehicle's pt line) must be used.

stopTime: new stop time for the vehicle at the stop (in seconds).

Output:

= 0: No error occurred

< 0: Error

3.3.17.10 Read the information of a Public Transport Vehicle

In C++ and Python:

Explanation: Read the information of a public transport vehicle

Format:

InfPTVeh AKIPTVehGetInf(int aidVeh)

Parameters:

aidVeh: vehicle Identifier.

Output:

```
struct InfPTVeh{
        int report;
        int idVeh;
        int type;
        // Information in Vehicle when it is in a section
        int idSection;
        int segment;
        int numberLane;
        // Information in Vehicle when it is in a node
        int idJunction;
        int idSectionFrom;
        int idLaneFrom;
        int idSectionTo;
        int idLaneTo;

        double CurrentPos;
        double distance2End;
        double xCurrentPos, yCurrentPos, zCurrentPos;
        double xCurrentPosBack, yCurrentPosBack, zCurrentPosBack;
        double CurrentSpeed, PreviousSpeed;

        double TotalDistance;

        double SystemGenerationT;
        double SystemEntranceT;
        double SectionEntranceT;
        double CurrentStopTime;

        double theoricalGenerationTime;
        int nbStopsDone;
```

```
        double observedLastStopTime;
        double observedLastInitialStopTime;
        int nextStopId;
        double offsetInNextStop;
        double distanceNextStop;
        double nextServiceTime;
        double currentStoppedTmeInBusStop;
        int currentLoad;
    };
```

where:

int *report*: 0, OK, otherwise an error code
int *idVeh*: the vehicle identifier
int *type*: the vehicle type (car, bus, truck, etc.)
int *idSection*: the section identifier, when the vehicle is in a section
int *numberLane*: lane number, when the vehicle is in a section
int *idSectionFrom*: origin section identifier
int *idLaneFrom*: number of lane of the origin section from which the vehicle entered the junction
int *idSectionTo*: destination section identifier
int *idLaneTo*: number of lane of the destination section where the vehicle will exit the junction.
double *CurrentPos*: position inside the section given by distance (metres or feet, depending on the units defined in the network) from the beginning of the section, or position inside the junction given by the distance from the entrance to the junction
double *distance2End*: distance to end of the section (metres or feet, depending on the units defined in the network) when the vehicle is located in a section, or the distance to end of the turning when the vehicle is in a junction.
double *xCurrentPos*, *yCurrentPos*, *zCurrentPos*: world coordinates of the middle point of the front bumper of the vehicle.
double *xCurrentPosBack*, *yCurrentPosBack*, *zCurrentPosBack*: world coordinates of the middle point of the rear bumper of the vehicle.
double *CurrentSpeed*: the current speed (in km/h or mph, depending on the units defined in the network.
double *PreviousSpeed*: the speed in the previous simulation step (in km/h or mph, depending on the units defined in the network.
double *TotalDistance*: the total distance travelled (metres or feet).
double *SystemGenerationT*: the absolute generation time of the vehicle into the system. If no virtual queue found in its entrance section it will be the same as the SystemEntranceT.
double *SystemEntranceT*: the absolute entrance time of the vehicle into the system, that is into its entrance section. If no virtual queue found in its entrance section it will be the same as the SystemGenerationT.
double *SectionEntranceT*: the absolute entrance time of the vehicle in the current section.
double *CurrentStopTime*: the current stop time.
double *theoricalGenerationTime*: the theoreticalgeneration time according to the time table
int *nbStopsDone*: the number of stops done.
double *observedLastStopTime*: the total duration of the last stop.
double *observedLastInitialStopTime*: the initial time of the last stop.
int *nextStopId*: the identifier of the next stop if the vehicle is not currently stopping at a stop or the identifier of the current stop when it is performing the stop.
double *offsetInNextStop*: the offset assigned to the next stop

double *distanceNextStop*: the distance to the next stop (metres or feet).
double *nextServiceTime*: the next stop time in the next stop(seconds) if the vehicle is not currently stopping at a stop or the stop time at the current stop to perform the stop when it is performing the stop.
double currentStoppedTimeInBusStop: the current stop time in the current stop, if any (seconds).
int *currentLoad*: current number of people inside the pt vehicle.

3.3.17.11 Read the Static Information of a Public Transport Vehicle

In C++ and Python:

Explanation: Read the static information of a public transport vehicle. Static information means the characteristics of the vehicle have which have been set when the vehicle has entered in the system.

Format:

StaticInfPTVeh AKIPTVehGetStaticInf(int aidVeh);

Parameters:

aidVeh: vehicle Identifier.

Output:

```
struct StaticInfPTVeh{
        int report;
        int idVeh;
        int type;
        double length;
        double width;
        double maxDesiredSpeed;
        double maxAcceleration;
        double normalDeceleration;
        double maxDeceleration;
        double speedAcceptance;
        double minDistanceVeh;
        double giveWayTime;
        double guidanceAcceptance;
        int enrouted;
        int equipped;
        int tracked;
        bool  keepfastLane;
        double headwayMin;
        double sensitivityFactor;
        double reactionTime;
        double reactionTimeAtStop;
        double reactionTimeAtTrafficLight;
        int maxCapacity;
        int idLine;
};
```

where*:*

int *report*: 0, OK, otherwise an error code
int *idVeh*: the vehicle identifier
int *type*: the vehicle type (car, bus, truck, etc.).
double *length*: vehicle length (m or feet, depending on the units defined in the network).
double *width*: vehicle width (m or feet, depending on the units defined in the network).

double *maxDesiredSpeed*: Maximum desired speed of the vehicle (km/h or mph, depending on the units defined in the network).
double *maxAcceleration*: Maximum acceleration of the vehicle (m/s^2 or ft/ s^2, depending on the units defined in the network).
double *normalDeceleration*: Maximum deceleration of the vehicle that can apply under normal conditions (m/s^2 or ft/ s^2, depending on the units defined in the network).
double *maxDeceleration*: Maximum deceleration of the vehicle that can apply under special conditions (m/s^2 or ft/ s^2, depending the units defined in the network).
double *speedAcceptance*: degree of acceptance of the speed limits.
double *minDistanceVeh*: the distance that the vehicle keeps between itself and the preceding vehicle (meters or feed, depending the units defined in the network).
double *giveWayTime*: the time after which the vehicle becomes more aggressive in give-way situations (seconds).
double *guidanceAcceptance*: the level of compliance of the vehicle to guidance indications.
int *enrouted*: 0 means vehicle will not change path en-route, 1 means vehicle will change path en-route depending on the percentege of enrouted vehicles defined.
int *equipped*: 0 means vehicle not equipped, 1 means vehicle equipped.
int *tracked*: 0 means vehicle not tracked, 1 means vehicle tracked.
bool keepfastLane: means vehicle keep fast lane during overtaking
double *headwayMin*: minimum headway to keep with its leader
double *sensitivityFactor*: estimation of the acceleration of the leader
double *reactionTime*: reaction time of the vehicle
double *reactionTimeAtStop*: reaction time at stop of the vehicle
double *reactionTimeAtTrafficLight*: reaction time of the vehicle when stopped the first one of the queue in a traffic light.
int *maxCapacity*: maximum number of people inside the pt vehicle.
int *idLine*: line identifier.

3.3.17.12 Modify the Static Information of a Public Transport Vehicle

In C++ and Python:

Explanation: Modify some static parameters of certain public transport vehicle. Static parameters mean the characteristics of the vehicle which have been set when the vehicle has entered in the system. The static parameters which may be changed are: *type*, *length width, maxDesiredSpeed, maxAcceleration, normalDeceleration, maxDeceleration, speedAcceptance, minDistanceVeh, giveWayTime, guidanceAcceptance*, *enrouted, equipped, tracked, keepfastLane, headwayMin , sensitivityFactor, reactionTime, reactionTimeAtStop*, *reactionTimeAtTrafficLight*. Aimsun cannot store the previous values, so it cannot recover them.

Format:

int AKIPTVehSetStaticInf(int aidVeh, StaticInfPTVeh staticinfVeh)

Parameters:

aidVeh: vehicle Identifier.
staticinfVeh : new static parameters to be assigned

Output:

= 0: No Error
< 0: Error

3.3.17.13 Modify the current vehicle load

In C++ and Python:

Explanation: Modify the number of people inside the pt vehicle. The new value cannot be greater than the maximum capacity.

Format:

int AKIPTVehSetCurrentLoad (int aidVeh, int currentLoad);

Parameters:

aidVeh: the public transport vehicle identifier.

currentLoad: the new pt vehicle load.

Output:

= 0: No error occurred

< 0: Error

3.3.18 Functions relative to Fleet Management

All these functions are provided to allow the user to manage the behaviour of fleet vehicles during the simulation. The following methods are directly related to the Aimsun Logistics Plug-In objects, such as Clients, Depots, Fleets, Fleet Assignments, etc.

Fleet Stops represent any kind of stop that the fleet vehicles will perform. Each one has tie attributes in order to define how much time will remain stopped the vehicle at the stop, the opening time window(the earliest time when a fleet vehicle can start to serve the stop) and the closing time window(the latest time when the fleet vehicle can start to serve the stop).

Fleet Routes holds which Fleet Stops will be visited by the route (including the stop times) and the path to follow.

Fleet Assignment hold the the information about which fleet vehicle will serve every fleet route.

Fleet represents a set of fleet vehicles. Fleet vehicles can only be at one fleet. Fleets can also hold the Fleet Assignments, so fleet vehicles can have associated different routes, but they can only serve one of them for each simlation.

Fleet Vehicles follow a route starting at an origin Fleet Stop, which represents a depot, and following by Fleet Stops representing customers. At each Fleet Stop, Fleet Vehicles will remain stopped for a defined time; this time represents the time took to deliver the demanded goods. Finally, Fleet Vehicles come back to the depot. The paths between each pair of Fleet Stops are considered as the shortest paths taking into account a certain kind of cost.

3.3.18.1 Read the number of Fleets

In C++ and Python:

Explanation: Read the number of fleets loaded during the simulation

Format:

int AKIFleetGetNbFleet ( )

Parameters: none

Output:

>= 0: No Error

< 0: Error

3.3.18.2 Read the Fleet identifier

In C++ and Python:

Explanation: Read the fleet identifier
Format:
int AKIFleetGetFleetId (int index)
Parameters:
index: Fleet index, where index takes values form 0 to AKIFleetGetNbFleet ( )
Output:
>= 0: No Error
< 0: Error

3.3.18.3 Read the number of Fleet Stop

In C++ and Python:
Explanation: Read the number of fleet stops loaded during the simulation
Format:
int AKIFleetGetNbFleetStops ( )
Parameters: none
Output:
>= 0: No Error
< 0: Error

3.3.18.4 Read the Fleet Stop identifier

In C++ and Python:
Explanation: Read the fleet stop identifier
Format:
int AKIFleetGetFleetStopId (int index)
Parameters:
index: Fleet index, where index takes values form 0 to AKIFleetGetNbFleetStops ( )
Output:
>= 0: No Error
< 0: Error

3.3.18.5 Read the number of fleet assignments

In C++ and Python:
Explanation: Read the number of fleet assignment of one fleet
Format:
int AKIFleetGetNbFleetVehicles (int idFleet)
Parameters:
idFleet: Fleet Identifier
Output:
>= 0: No Error
< 0: Error

3.3.18.6 Read the Fleet Vehicle Assignment name

In C++:
Explanation: Read the fleet assignment name of one fleet
Format:
const unsigned short *AKIFleetGetFleetVehicleName(int idFleet, int index)
Parameters:
idFleet: Fleet Identifier
index: Vehicle assignmnet index, where index takes values form 0 to AKIFleetGetNbFleetVehicles ( )
Output:
<>NULL: No Error

=NULL: Error

In Python:

Not Available.

3.3.18.7 Create a new Fleet

In C++ and Python:

Explanation: Create a new Fleet

Format:

int AKIFleetAddFleet(int id)

Parameters:

id: Fleet Identifier

Output:

= 0: No Error

< 0: Error

3.3.18.8 Create a new Fleet Stop

In C++ and Python:

Explanation: Create a new Fleet Stop

Format:

int AKIFleetAddFleetStop (int id, int type, int idSect, int aidLane, double adistance, double alenght, double aMaxVeh, double openTW, double closeTW, void * realStop)

Parameters:

id: Fleet Stop Identifier

type: Fleet Stop type (Normal=1, Bay = 2, Terminal=3)

idSect: Section Identifier where the fleet stop is located

aidLane: Lane identifier (1 rightmost lane)

adistance: distance from the beginning of the section

alenght: fleet stop length

aMaxVeh: Maximum vehicles of the fleet stop (only when the type is terminus)

openTW: The opening Time Window in stationary time

closeTW: The closing Time Window in stationary time

realStop: The pointer to the related stop.

Output:

= 0: No Error

< 0: Error

3.3.18.9 Add a new Fleet Assignment to a Fleet

In C++ and Python:

*Explanation:*Add a new fleet assignment to a fleet

Format:

int AKIFleetAddFleetAssignment(int idFleet, const unsigned short *nameFleetVeh, int vehTypePos, double departureTime, double deviation);

Parameters:

idFleet: Fleet Identifier

nameFleetVeh: name of the fleet Vehicle.

vehTypePos: vehType is the position of the vehicle type in the list of vehicles types being used. A value from 1 to AKIVehGetNbVehTypes () must be used.

departureTime, *deviation*: Determines the departure time, following a Normal distribution.

Output:

= 0: No Error

< 0: Error

3.3.18.10 Add a Route to a Fleet Assignment

In C++ only:

Explanation: Add a Route to a Fleet Assignment

Format:

int AKIFleetAddFleetAssignmentRoute(int idFleet, const unsigned short *nameFleetVeh, int idRoute, int nbsect, int *idsect)

Parameters:

idFleet: Fleet Identifier

nameFleetVeh: name of the fleet Vehicle.

idRoute: Route Identifier

nbsect: Number of sections that compose the route

idsect: identifiers of the sections that compose the route

Output:

= 0: No Error

< 0: Error

3.3.18.11 Add Stops to a Fleet Assignment

In C++ only:

Explanation: Add Stops to a Fleet Assignment. The stops has to previously added in the model

Format:

int AKIFleetAddFleetAssignmentStop(int idFleet, const unsigned short *nameFleetVeh, int nbstops, int *idstops, double *avgtime, double *devtime)

Parameters:

idFleet: Fleet Identifier

nameFleetVeh: name of the fleet Vehicle.

nbstops: Number of stops

idstops: identifiers of the stop

avgtime, *devtime*: Determines the stop time of the vehicle in each stop (Norma distribution)

Output:

= 0: No Error

< 0: Error

3.3.18.12 Is Fleet Vehicle generated

In C++ and Python:

Explanation: Gets whether the fleet vehicle has been generated or not

Format:

int AKIFleetIsFleetVehicleGenerated (int idFleet, const unsigned short *nameFleetVeh)

Parameters:

idFleet: Fleet Identifier

nameFleetVeh: name of the fleet Vehicle.

Output:

= 0: No generated, =1 generated

< 0: Error

3.3.18.13 Get Current Section Identifier where is located a Fleet Vehicle

In C++ and Python:

Explanation: Gets the current section identifier where the fleet vehicle is located

Format:

int AKIFleetGetCurrentSectionIdFleetVehicle (int idFleet, const unsigned short *nameFleetVeh)

Parameters:

idFleet: Fleet Identifier
nameFleetVeh: name of the fleet Vehicle.

Output:
> 0: Section Id
< 0: Error

3.3.18.14 Modify the Route and the set of stops of a Fleet Vehicle

In C++ only:

Explanation: Modify the Route and the set of stops of a Fleet Vehicle.

Format:

int AKIFleetModifyRouteStopsFleetVehicle(int idFleet, const unsigned short *nameFleetVeh, int nbsect, int *idsect, int nbstops, int *idstops, double *avgtime, double *devtime)

Parameters:
idFleet: Fleet Identifier
nameFleetVeh: name of the fleet Vehicle.
nbsect: Number of sections that compose the route
idsect: identifiers of the sections that compose the route
nbstops: Number of stops
idstops: identifiers of the stop
avgtime, *devtime*: Determines the stop time of the vehicle in each stop (Norma distribution)

Output:
= 0: No Error
< 0: Error

3.3.18.15 Read the number of stops assigned to a Fleet Vehicle

In C++ and Python:

Explanation: Read the number of stops assigned to a fleet vehicle already generated.

Format:

int AKIFleetGetNbStopsFleetVeh (int idFleet, const unsigned short *nameFleetVeh)

Parameters:
idFleet: Fleet Identifier
nameFleetVeh: name of the fleet Vehicle.

Output:
>= 0: No Error
< 0: Error

3.3.18.16 Read the number of stops done to a Fleet Vehicle

In C++ and Python:

Explanation: Read the number of stops done to a fleet vehicle already generated.

Format:

int AKIFleetGetNbStopsDoneFleetVeh (int idFleet, const unsigned short *nameFleetVeh)

Parameters:
idFleet: Fleet Identifier
nameFleetVeh: name of the fleet Vehicle.

Output:
>= 0: No Error
< 0: Error

3.3.18.17 Read whether a fleet vehicle is doing an stop

In C++ and Python:

Explanation: Read whether a fleet vehicle is doing the stop.

Format:

int AKIFleetIsDoingStopFleetVeh (int idFleet, const unsigned short *nameFleetVeh)

Parameters:

idFleet: Fleet Identifier

nameFleetVeh: name of the fleet Vehicle.

Output:

>= 0: 0 means false, 1 means true

< 0: Error

3.3.18.18 Read the stop time of stops assigned to a Fleet Vehicle

In C++ and Python:

Explanation: Read the stop time assigned to a fleet vehicle already generated.

Format:

int AKIFleetGetStopTimeFleetVeh (int idFleet, const unsigned short *nameFleetVeh, int indexstop)

Parameters:

idFleet: Fleet Identifier

nameFleetVeh: name of the fleet Vehicle.

indexStop: index of stop where the value is between 0 and AKIFleetGetNbStopsFleetVeh() -1.

Output:

>= 0: No Error

< 0: Error

3.3.18.19 Modify the stop time of a Fleet Vehicle

In C++ and Python:

Explanation: Modify the stop time of a fleet vehicle already generated.

Format:

int AKIFleetChangeStopTimeFleetVeh (int idFleet, const unsigned short *nameFleetVeh, int indexStop, double stoptime)

Parameters:

idFleet: Fleet Identifier

nameFleetVeh: name of the fleet Vehicle.

indexStop: index of stop where the value is between 0 and AKIFleetGetNbStopsFleetVeh() -1 .

stoptime: new stop time.

Output:

= 0: No Error

< 0: Error

3.3.18.20 Modify the time window of a Fleet Stop

In C++ and Python:

Explanation: Modify the time window of a fleet stop already generated.

Format:

int AKIFleetModifyFleetStopTimeWindow(int idFleetStop, double openTW, double closeTW)

Parameters:

idFleetStop: Fleet Stop Identifier

openTW: the new opening time window in stationary time (seconds).

closeTW: the new closing time window in stationary time (seconds).

Output:

= 0: No Error
< 0: Error

3.3.19 Functions relative to Display information to Log Window

3.3.19.1 Print string to Aimsun Log Window

In C++ and Python:

Explanation: Print a string in Aimsun Log window.

Format:

int AKIPrintString(char *string);

Parameters: none

Output:

≥ 0: No Error
< 0: Error

3.3.19.2 Print string to Aimsun Log Window

In C++:

Explanation: Print a string in Aimsun Log window.

Format:

void AKIPrintAsUNICODEString(const unsigned short *string);

Parameters:

string: the string to be printed

Output:

none

In Python:

Not Available. See 3.3.17.1.

3.3.19.3 Convert string to ASCII string

In C++:

Explanation: Converts a Unicode string to char* . You must delete the returned string

Format:

const char *AKIConvertToAsciiString(const unsigned short *string, bool deleteUshortString, bool *anyNonAsciiChar);

Parameters:

string: Unicode string
deleteUshortString: true to delete the parameter *string*
anyNonAsciiChar: returns true if there was any non convertable character

Output:

The converted string

In Python:

Not Available.

3.3.19.4 ConvertASCII string to string

In C++:

Explanation: Convert an ascii string into a Unicode string.

Format:

const unsigned short *AKIConvertFromAsciiString(const char *ascii);

Parameters:

ascii: the ascii string to convert

Output:
The converted string

In Python:
Not Available.

3.3.19.5 Delete a Unicode String

In C++:
Explanation: Deletes the memory allocated for string either having previously called the AKIConvertFromAsciiString function.
Format:
void AKIDeleteUNICODEString(const unsigned short *string);
Parameters:
string: the string to delete
Output:
none

In Python:
Not Available.

3.3.19.6 Delete an ASCII String

In C++:
Explanation: Deletes the memory allocated for string either having previously called the AKIConvertToAsciiString function.
Format:
void AKIDeleteASCIIString(const char *string);
Parameters:
string: the string to delete
Output:
none

In Python:
Not Available.

3.3.20 Functions relative to Network Information

3.3.20.1 Read the name of an object

In C++ and Python:
Explanation: Read the name of an object. UNICODE version.
Format:
const unsigned short * ANGConnGetObjectName(int idObject); // Unicode version. Not available in Python.
const char * ANGConnGetObjectNameA(int idObject); // Ascii version.
Parameters:
idObject: the object identifier
Output:
≠ NULL: Correct name of the object
= NULL: Error

3.3.20.2 Read the Path of the Network

In C++:

Explanation: Read the full path of the network. The result has to be deleted with delete[].
Format:
const unsigned short * AKIInfNetGetNetworkPath();
Parameters: none
Output:
≠ NULL: Correct path of the Network
= NULL: Error

In Python:
Explanation: Read the full path of the network in ascii.
Format:
const char * AKIInfNetGetNetworkPathA();
Parameters: none
Output:
≠ NULL: Correct path of the Network
= NULL: Error

3.3.20.3 Read the Name of the Network

In C++:
Explanation: Read the name of the network file (without extension). The result has to be deleted with delete[].
Format:
const unsigned short * AKIInfNetGetNetworkName();
Parameters: none
Output:
≠ NULL: Correct path of the Network
= NULL: Error

In Python:
Explanation: Read the name of the network file (without extension) in ascii.
Format:
const char * AKIInfNetGetNetworkNameA();
Parameters: none
Output:
≠ NULL: Correct name of the Network file
= NULL: Error

3.3.20.4 Read the Traffic Demand Name

In C++:
Explanation: Read the name of the loaded traffic demand. The result has to be deleted with delete[].
Format:
const unsigned short * AKIInfNetGetTrafficDemandName ();
Parameters: none
Output:
≠ NULL: Correct traffic demand name
= NULL: Error

In Python:
Explanation: Read the name of the of the loaded traffic demand in ascii.
Format:

const char * AKIInfNetGetTrafficDemandNameA();

Parameters: none

Output:

≠ NULL: Correct traffic demand name

= NULL: Error

3.3.20.5 Read the Type of Traffic Demand

In C++ and Python:

Explanation: Read the type of the Traffic Demand

Format:

int AKIInfNetGetTrafficDemandType ();

Parameters: none

Output:

≥ 0: 1 traffic demand using O/D matrices, 2 using traffic states

< 0: Error

3.3.20.6 Read the Units of a Network

In C++ and Python:

Explanation: Read the units of a Network.

Format:

int AKIInfNetGetUnits ()

Parameters: none

Output:

≥ 0: 1 Metric Units, 0 English Units

< 0: Error

3.3.20.7 Read the World Coordinates of the Network

In C++ and Python:

Explanation: Read the world coordinates of a Network. It gives the left bottom point and the right top point of the bounding box

Format:

int AKIInfNetGetWorldCoordinates (double *min_x, double *min_y, double *max_x, double *max_y)

Parameters:

min_x, min_y, max_x, max_y are the parameters changed to return the bounding box

Output:

= 0: correct

< 0: Error

3.3.20.8 Read the Number of Sections

In C++ and Python:

Explanation: Read the number of sections present on the road network.

Format:

int AKIInfNetNbSectionsANG();

Parameters: none

Output:

> 0: Correct number of sections in the road network

< 0: Error

3.3.20.9 Read the Identifier of a Section

In C++ and Python:

Explanation: Read the elem-th section identifier present on the road network.

Format:

int AKIInfNetGetSectionANGId (int elem)

Parameters:

elem represents the number of the section and must be in the range

0 ≤ elem < Number of Sections –1

Output:

> 0: Correct identifier of the section

≤ 0: Error

3.3.20.10 Read the Section Information

In C++ and Python:

Explanation: Read the information of a section.

Format:

A2KSectionInf AKIInfNetGetSectionANGInf(int aid)

Parameters:

aid is the section identifier

Output:

```
struct A2KSectionInf{
    int report;
    int id;
    int angId;
    int nbCentralLanes;
    int nbSideLanes;
    double speedLimit;
    double visibilityDistance;
    double reservedLanesVisibilityDistance;
    double yellowBoxSpeed;
    double capacity;
    double distance_OnRamp;
    double slope_percentage;
    double length;
    double userDefinedCost;
    double maxGivewayTimeVariation;
    int nbTurnings;
};
```

where:

int *report*: 0, OK, otherwise is an error code

int *id*: section identifier

int *angId*: section identifier in Aimsun.

int *nbCentralLanes*: total number of central lanes.

int *nbSideLanes*: total number of side lanes.

double *speedLimit*: speed limit of the section (Km/h o mph).

double *visibilityDistance*: visibility distance of the section (meters or feet).

double *reservedLanesVisibilityDistance*: visibility distance of the reserved lanes of the section (meters or feet).

double *yellowBoxSpeed*: yellow box speed of the section (Km/h o mph).

double *capacity*: capacity of the section (veh/h).

double *distance_OnRamp*: distance On Ramp of the section (sec).

double *slope_percentage*: slope percentage of the section.

double *length*: length of the section (meters or feet).

double *userDefinedCost*: user-defined cost of the section.

double *maxGivewayTimeVariation*: maximum giveway time variation of the section (sec).

int *nbTurnings*: total number of turnings that has as origin this section.

3.3.20.11 *Read the Destination Sections Identifier of all turnings from one section*

In C++ and Python:

Explanation: Read the elem-th section identifier of the turning movement.

Format:

int AKIInfNetGetIdSectionANGDestinationofTurning(int aid, int elem);

Parameters:

aid is the section identifier.

elem represents the number of turning and must be in the range

0 ≤ elem < Number of Turnings–1

Output:

> 0: Corresponding section identifier

≤ 0: Error

3.3.20.12 *Read the lane numbers of one turnings from one section*

In C++ and Python:

Explanation: Read lane number of one turning from one section (lane number equal to 1 means the rightmost lane). There are four numbers to get: the first lane of the origin section of the turning (AKIInfNetGetOriginFromLaneofTurning), the last lane of the origin section of the turning (AKIInfNetGetOriginToLaneofTurning), the first lane of the destination section of the turning (AKIInfNetGetDestinationFromLaneofTurning) and the last lane of the destination section of the turning (AKIInfNetGetDestinationToLaneofTurning)

Format:

int AKIInfNetGetDestinationFromLaneofTurning(int sectId, int elem);
int AKIInfNetGetDestinationToLaneofTurning(int sectId, int elem);
int AKIInfNetGetOriginFromLaneofTurning(int sectId, int elem);
int AKIInfNetGetOriginToLaneofTurning(int sectId, int elem);

Parameters:

aid is the section identifier.

elem represents the number of turning and must be in the range

0 ≤ elem < Number of Turnings–1

Output:

> 0: Corresponding lane number (from 1 to N, where lane 1 is the rightmost lane)

≤ 0: Error

3.3.20.13 *Read the lane numbers given the origin and destination sections*

In C++ and Python:

Explanation: Read lane number of one turning (lane number equal to 1 means the rightmost lane). There are four numbers to get: the first lane of the origin section of the turning (AKIInfNetGetTurningOriginFromLane), the last lane of the origin section of the turning (AKIInfNetGetTurningOriginToLane), the first lane of the destination section of the turning (AKIInfNetGetTurningDestinationFromLane) and the last lane of the destination section of the turning (AKIInfNetGetTurningDestinationToLane)

Format:

```
int AKIInfNetGetTurningDestinationFromLane( int originSection, int destinationSection )
int AKIInfNetGetTurningDestinationToLane( int originSection, int destinationSection )
int AKIInfNetGetTurningOriginFromLane( int originSection, int destinationSection )
int AKIInfNetGetTurningOriginToLane( int originSection, int destinationSection )
```

Parameters:

originSection is the origin section identifier.

destinationSection is the origin section identifier.

Output:

> 0: Corresponding lane number (from 1 to N, where lane 1 is the rightmost lane)
≤ 0: Error

3.3.20.14 Modify the Behavioural Parameters of a Section

In C++ and Python:

Explanation: Modify the behavioural parameters of a section that will be used either when applying the models and in the route choice calculations.

Format:

int AKIInfNetSetSectionBehaviouralParam(int aid, A2KSectionBehaviourParam behaviourParam, bool allsegments);

Parameters:

aid: section identifier.
allsegments: true if the changes is expanded to all segments of the section.
behaviourParam: new parameters to be assigned with the following structure:

```
struct A2KSectionBehaviourParam{
   double speedLimit;
   double visibilityDistance;
   double reservedLanesVisibilityDistance;
   double yellowBoxSpeed;
   double capacity;
   double distance_OnRamp;
   double userDefinedCost;
   double maxGivewayTimeVariation;
   int reactionTimeVariation;
};
```

where*:*

double *speedLimit*: speed limit of the section (km/h or mph).
double *visibilityDistance*: visibility distance of the section (metres or feet).
double *reservedLanesVisibilityDistance*: visibility distance of the reserved lanes of the section (meters or feet).
double *yellowBoxSpeed*: yellow box speed of the section (km/h or mph).
double *capacity*: capacity of the section (veh/h).
double *distance_OnRamp*: distance for On Ramp of the section (sec).
double *userDefinedCost*: user-defined cost of the section.
Double *maxGivewayTimeVariation*: maximum giveway time variation of the section (sec).
int *reactionTimeVariation*: reaction time variation factor.

Output:

= 0: No Error
< 0: Error

Sample code in Python:

The following code modifies the reaction time variation of section with ID=106 to a value of 1. It reads first the current parameters to modify any of them afterwards:

```
report = intp()
behParams = AKIInfNetGetSectionBehaviouralParam(106, report)
if report.value() == 0:
        behParams.reactionTimeVariation = 1
        AKIInfNetSetSectionBehaviouralParam(106, behParams, True)
        AKIPrintString("OK")
else :
        AKIPrintString("Not OK")
```

3.3.20.15 *Read the Behavioural Parameters of a Section*

In C++ and Python:

Explanation: Reads the behavioural parameters of a section that will be used either when applying the models and in the route choice calculations.

Format:

A2KSectionBehaviourParam AKIInfNetGetSectionBehaviouralParam(int aid, int *report);

Parameters:

aid: section identifier.

Output:

report = 0: No Error. The current section behavioural parameters (see function 3.3.20.14).

report < 0: Error

Sample code in Python:

The following code reads the behavioural parameters of section with ID=106 and prints the reaction time variation read:

```
report = intp()
behParams = AKIInfNetGetSectionBehaviouralParam(106, report)
if report.value() == 0:
    string = str(behParams.reactionTimeVariation)
    AKIPrintString(string)
```

3.3.20.16 *Modify the Capacity of a Section*

In C++ and Python:

Explanation: Modify the capacity of a section that will be used either when applying the models and in the route choice calculations.

Format:

int AKISetSectionCapacity(int aid, double capacity);

Parameters:

aid: section identifier.

capacity: new capacity (veh/h) to be assigned to the section

Output:

= 0: No Error

< 0: Error

3.3.20.17 *Read the Capacity of a Section*

In C++ and Python:

Explanation: Read the capacity defined for a section.

Format:

double AKIGetSectionCapacity(int aid);

Parameters:

aid: section identifier.

Output:

> 0: Capacity of the specified section

< 0: Error

3.3.20.18 *Modify the User Defined Cost of a Section*

In C++ and Python:

Explanation: Modify the user defined cost of a section that will be used either when applying the models and in the route choice calculations.

Format:

int AKISetSectionUserDefinedCost(int aid, double userdefinedcost);

Parameters:

aid: section identifier.

userdefinedcost: new user-defined cost of the section

Output:

= 0: No Error

< 0: Error

3.3.20.19 Read the User Defined Cost of a Section

In C++ and Python:

Explanation: Read the user-defined cost defined for a section.

Format:

double AKIGetSectionUserDefinedCost(int aid);

Parameters:

aid: section identifier.

Output:

> 0: User-defined cost of the specified section

< 0: Error

3.3.20.20 Modify the Second User Defined Cost of a Section

In C++ and Python:

Explanation: Modify the second user defined cost of a section that will be used either when applying the models and in the route choice calculations.

Format:

int AKISetSectionUserDefinedCost2(int aid, double userdefinedcost);

Parameters:

aid: section identifier.

userdefinedcost: new second user-defined cost of the section

Output:

= 0: No Error

< 0: Error

3.3.20.21 Read the Second User Defined Cost of a Section

In C++ and Python:

Explanation: Read the second user-defined cost defined for a section.

Format:

double AKIGetSectionUserDefinedCost2(int aid);

Parameters:

aid: section identifier.

Output:

> 0: Second user-defined cost of the specified section

< 0: Error

3.3.20.22 Modify the Third User Defined Cost of a Section

In C++ and Python:

Explanation: Modify the third user-defined cost of a section that will be used either when applying the models and in the route choice calculations.

Format:

int AKISetSectionUserDefinedCost3(int aid, double userdefinedcost);

Parameters:

aid: section identifier.

userdefinedcost: new third user-defined cost of the section

Output:
= 0: No Error
< 0: Error

3.3.20.23 Read the Third User Defined Cost of a Section

In C++ and Python:

Explanation: Read the third user-defined cost defined for a section.
Format:
double AKIGetSectionUserDefinedCost3(int aid);
Parameters:
aid: section identifier.
Output:
> 0: Third user-defined cost of the specified section
< 0: Error

3.3.20.24 Read the Number of Junctions

In C++ and Python:

Explanation: Read the number of junctions present on the road network. This function is equivalent to ECIGetNumberJunctions(), but it does not need to have loaded the control plan.
Format:
int AKIInfNetNbJunctions ();
Parameters: none
Output:
> 0: Correct number of junctions in the road network
< 0: Error

3.3.20.25 Read the Identifier of a Junction

In C++ and Python:

Explanation: Read the elem-th junction identifier present on the road network. This function is equivalent to ECIGetJunctionId (), but it does not need to have loaded the control plan.
Format:
int AKIInfNetGetJunctionId (int elem)
Parameters: *elem* represents the number of junction and must be in the range
0 ≤ elem < Number of Junctions –1
Output:
> 0: Corresponding junction identifier
≤ 0: Error

3.3.20.26 Read the Number of Centroid

In C++ and Python:

Explanation: Read the number of centroids present on the road network.
Format:
int AKIInfNetNbCentroids ();
Parameters: none
Output:
> 0: Correct number of centroids in the road network
< 0: Error

3.3.20.27 Read the Identifier of a Centroid

In C++ and Python:

Explanation: Read the elem-th centroid identifier present on the road network.

Format:

int AKIInfNetGetCentroidId (int elem)

Parameters: *elem* represents the number of centroid and must be in the range

0 ≤ elem < Number of Centroids –1

Output:

> 0: Corresponding centroid identifier.

≤ 0: Error

3.3.20.28 Read the Centroid Information

In C++ and Python:

Explanation: Read the information of a centroid

Format:

A2KCentroidInf AKIInfNetGetCentroidInf(int aid)

Parameters:

Aid: the centroid identifier

Output:

```
struct A2KCentroidInf{
        int report;
        int id;
        bool AsDestConsider_percentage;
        bool AsOrigConsider_percentage;
        bool IsOrigin;
        bool IsDestination;
        int NumConnecTo;
        int NumConnecFrom;
};
```

where:

int *report*: 0, OK, else error code

int *id*: centroid identifier

bool *AsDestConsider_percentage*: Centroid considers destination connectors percentages when the centroid acts as destination

bool *AsOrigConsider_percentage*: Centroid considers origin connectors percentages when the centroid acts as origin

bool *IsOrigin*: true if the centroid acts as one origin

bool *IsDestination*: true if the centroid acts as one destination

int *NumConnecTo*: Number of connectors To

int *NumConnecFrom*: Number of connectors From

3.3.20.29 Read the Sections Identifier of all connectors ”to” a particular centroid (deprecated function)

In C++ and Python:

Explanation: Read the section identifier for the elem-th section connected “to” the given centroid present on the road network.

Format:

int AKIInfNetGetIdSectionofOriginCentroidConnector (int aid, int elem)

Parameters:

aid is the centroid identifier

elem represents the number of the “to” connector and must be in the range

0 ≤ elem < Number of Connectors “to”–1

Output:

> 0: Corresponding section identifier

≤ 0: Error

3.3.20.30 *Read the Sections Identifier of all connectors "to" a particular centroid*

In C++ and Python:

Explanation: Read the section identifier for the elem-th section connected "to" the given centroid present on the road network.

Format:

int AKIInfNetGetIdSectionANGofOriginCentroidConnector (int aid, int elem)

Parameters:

aid is the centroid identifier

elem represents the number of the "to" connector and must be in the range

0 ≤ elem < Number of Connectors "to"–1

Output:

> 0: Corresponding section identifier

≤ 0: Error

3.3.20.31 *Read the Sections Identifier of all connectors "from" a particular centroid (deprecated function)*

In C++ and Python:

Explanation: Read the section identifier for the elem-th section connected "from" the given centroid present on the road network.

Format:

int AKIInfNetGetIdSectionofDestinationCentroidConnector (int aid, int elem)

Parameters:

aid is the centroid identifier

elem represents the number of the "from" connector and must be in the range

0 ≤ elem < Number of Connectors "from"–1

Output:

> 0: Corresponding section identifier

≤ 0: Error

3.3.20.32 *Read the Sections Identifier of all connectors "from" a particular centroid*

In C++ and Python:

Explanation: Read the section identifier for the elem-th section connected "from" the given centroid present on the road network.

Format:

int AKIInfNetGetIdSectionANGofDestinationCentroidConnector (int aid, int elem)

Parameters:

aid is the centroid identifier

elem represents the number of the "from" connector and must be in the range

0 ≤ elem < Number of Connectors "from"–1

Output:

> 0: Corresponding section identifier

≤ 0: Error

3.3.20.33 *Read the number of turns in the network*

In C++ and Python:

Explanation: Read the number of turns in the whole network.

Format:

int AKIInfNetNbTurns ()

Output:

the number of turns

3.3.20.34 *Read the turn identifier of a Turn*

In C++ and Python:

Explanation: Read the turn identifier for the elem-th.

Format:

int AKIInfNetGetTurnId (int elem)

Parameters:

elem represents the number of the turns in the network and must be in the range

0 ≤ elem < Number of Turns

Output:

> 0: Corresponding turn identifier

≤ 0: Error

3.3.20.35 *Read the Turn information*

In C++ and Python:

Explanation: Read the turn information for turn identifier.

Format:

A2KTurnInf AKIInfNetGetTurnInf (int turnId)

Parameters:

turnId represents a turn identifier

Output:

```
struct A2KTurnInf{
        int report;
        int id;
        double length;
        int originSectionId;
        int destinationSectionId;
        int originFromLane;
        int originToLane;
        int destinationFromLane;
        int destinationToLane;
};
```

where:

int *report*: 0, OK, else error code

int *id*: turn identifier

double *length*: turn length calculated considering Z coordinates.

int *originSectionId*: origin section identifier

int *destinationSectionId*: destination section identifier.

int *originFromLane*: origin from lane

int *originToLane*: origin to lane.

int *destinationFromLane*: destination from lane.

int *destinationToLane*: destination to lane.

3.3.20.36 *Read the Number of Turns in a Node*

In C++ and Python:

Explanation: Read the number of turns inside a node.

Format:

int AKIInfNetNbTurnsInNode (int nodeId)

Parameters:

nodeId represents a node identifier

Output:

> 0: The number of turns in the Node

≤ 0: Error

3.3.20.37 Read the Origin Section Id from a Turn

In C++ and Python:

Explanation: Read the origin section identifier from a turn.

Format:

int AKIInfNetGetOriginSectionInTurn (int nodeId, int elem)

Parameters:

nodeId represents a node identifier

elem represents a range value between 0 and "Number of Turns in node" - 1

Output:

> 0: The Origin section identifier

≤ 0: Error

3.3.20.38 Read the Destination Section Id from a Turn

In C++ and Python:

Explanation: Read the destination section identifier from a turn.

Format:

int AKIInfNetGetdestinationSectionInTurn (int nodeId, int elem)

Parameters:

nodeId represents a node identifier

elem represents a range value between 0 and "Number of Turns in node" - 1

Output:

> 0: The destination section identifier

≤ 0: Error

3.3.20.39 Read the Turn information from a Node

In C++ and Python:

Explanation: Read the turn information from the elem-th turn in a node.

Format:

A2KTurnInf AKIInfNetGetTurnInf (int nodeId, int elem)

Parameters:

nodeId represents a ndoe identifier

elem represents a range value between 0 and "Number of Turns in Node" -1

Output:

```
struct A2KTurnInf{
        int report;
        int id;
        double length;
        int originSectionId;
        int destinationSectionId;
        int originFromLane;
        int originToLane;
        int destinationFromLane;
        int destinationToLane;
};
```

where:

int *report*: 0, OK, else error code

int *id*: turn identifier
double *length*: turn length calculated considering Z coordinates.
int *originSectionId*: origin section identifier
int *destinationSectionId*: destination section identifier.
int *originFromLane*: origin from lane
int *originToLane*: origin to lane.
int *destinationFromLane*: destination from lane.
int *destinationToLane*: destination to lane.

3.3.20.40 Read the shortest path Number of sections

In C++ and Python:

Explanation: Read the shortest path number of sections between two section identifiers on the network.

Format:

int AKIInfNetGetShortestPathNbSections(int fromSection, int toSection, void * sectionColumnCost);

Parameters:

fromSection is origin section identifier
toSection is the destination section identifier
sectionColumnCost is the cost attribute that will be used to calculate the shortest path.

Output:

> 0: Number of sections
≤ 0: Error

3.3.20.41 Read the shortest path

In C++ and Python:

Explanation: Read the shortest path between two section identifiers on the network.

Format:

int AKIInfNetGetShortestPath (int fromSection, int toSection, void * sectionColumnCost, int* path);

Parameters:

fromSection is origin section identifier
toSection is the destination section identifier
sectionColumnCost is the cost attribute that will be used to calculate the shortest path.

Output:

= 0: Ok, *path* contains the shortest path defined as a section identifier vector.
< 0: Error

Example:

```
nbSections = AKIInfNetGetShortestPathNbSections( 156, 548, sectionCostColumn )
if nbSections > 0:
    path = intArray( nbSections )
    result = AKIInfNetGetShortestPath( 156, 548, sectionCostColumn, path )
    for index in range(0, nbSections):
        AKIPrintString( "%d"%path[index] )
```

3.3.20.42 Read the number of active control plans

In C++ and Python:

Explanation: Read the number of active control plans that are being simulated.

Format:

int ECIGetNbActiveControls ();

Output:

>= 0: the number of active control plans. It can be 0.

< 0: Error

3.3.20.43 Read the active control plan identifiers

In C++ and Python:

Explanation: Read the number of active control plans that are being simulated.

Format:

int ECIGetActiveControls (int * activeControls);

Parameters:

activeControls: it must be dimensioned with as many items as ECIGetNbActiveControls.

Output:

= 0: Ok, activeControls contains the active control plan ids.

< 0: Error

3.3.20.44 Read the network total length

In C++ and Python:

Explanation: Read the total network length including all section lanes and turning connections.

Format:

double AKIGetTotalLengthSystem();

Output:

> 0: Ok, the network total length.

< 0: Error

3.3.21 Functions relative to Graphical Text Objects

The functions relative to Graphical Text Objects in version v5.0 are not available and, as an alternative, can be achieved using the ANG Connections functions.

3.3.22 Functions relative to O/D Demand

3.3.22.1 Read the number of O/D Matrix Slices

In C++ and Python:

Explanation: Read the number of slices of the OD Matrix

Format:

int AKIODDemandGetNumSlicesOD(int vehTypePos)

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used.

Parameter goes from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

Output:

> 0: Number of slices

< 0: Error

3.3.22.2 Read the Initial Time of one O/D Matrix Slice

In C++ and Python:

Explanation: Read the initial time of one OD Matrix slice.

Format:

int AKIODDemandGetIniTimeSlice (int vehTypePos, int numslice)

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used. Parameter goes from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

numslice must be in the range 0 to AKIODDemandGetNumSlicesOD()-1.

Output:

≥ 0: Number seconds from 00:00:00

< 0: Error

3.3.22.3 Read the End Time of one O/D Matrix Slices

In C++ and Python:

Explanation: Read the end time of one OD Matrix slice.

Format:

int AKIODDemandGetEndTimeSlice (int vehTypePos, int numslice)

Parameters:

vehTypePos is the position of the vehicle type in the list of vehicles types being used. Parameter goes from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

numslice must be in the range 0 to AKIODDemandGetNumSlicesOD()-1.

Output:

≥ 0: Number seconds from 00:00:00 plus duration of the slice

< 0: Error

3.3.22.4 Read O/D Matrix demand.

In C++ and Python:

Explanation: Read the number of trips of one slice and vehicle type. If this name is NULL or "all" the function returns the demand aggregating all vehicle types.

Format:

double AKIODDemandGetDemandODPair (int origin, int desti, int vehTypePos, int numslice)

Parameters:

origin is the origin centroid identifier.

desti is the destination centroid identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.

numslice has to be defined from 0 to (AKIODDemandGetNumSlicesOD()-1).

Output:

≥ 0: Number of trips

< 0: Error

3.3.22.5 Modify O/D Matrix demand.

In C++ and Python:

Explanation: Modify the number of trips for one slice and vehicle type. If this name is NULL or "all" modify all vehicle types demand multiplying the number of trips of each vehicle type by a factor calculated as: anewdemand / TotalDemand (where TotalDemand is the sum of trips considering all vehicle types).

Format:

int AKIODDemandSetDemandODPair (int origen, int desti, int vehTypePos, int numslice, int anewdemand)

Parameters:

origin is the origin centroid identifier.

desti is the destination centroid identifier.
vehTypePos is the position of the vehicle type in the list of vehicles types being used. 0 must be used for all vehicle types and a value from 1 to AKIVehGetNbVehTypes (), for a specific vehicle type.
numslice has to be defined from 0 to (AKIODDemandGetNumSlicesOD()-1).
anewdemand is the new demand.

Output:
≥ 0: No Error
< 0: Error

3.3.23 Functions relative to Traffic States Demand

3.3.23.1 Read the number of Traffic States Slices

In C++ and Python:

Explanation: Read the number of slices of the traffic states

Format:
int AKIStateDemandGetNumSlices (int vehTypePos)

Parameters:
vehTypePos is the position of the vehicle type in the list of vehicles types being used. Parameter goes from 1 to AKIVehGetNbVehTypes(), for a specific vehicle type.

Output:
> 0: No Error
< 0: Error

3.3.23.2 Read the Initial Time of one Traffic State Slice

In C++ and Python:

Explanation: Read the initial time of one Traffic State slice.

Format:
double AKIStateDemandGetIniTimeSlice (int vehTypePos, int numSlice)

Parameters:
vehTypePos is the position of the vehicle type in the list of vehicles types being used.
numslice must be in the range 0 to AKIStateDemandGetNumSlices (vehTypePos)-1.

Output:
≥ 0: No Error
< 0: Error

3.3.23.3 Read the End Time of one Traffic State Slices

In C++ and Python:

Explanation: Read the end time of one Traffic State slice.

Format:
double AKIStateDemandGetEndTimeSlice (int vehTypePos, int numSlice)

Parameters:
vehTypePos is the position of the vehicle type in the list of vehicles types being used.
numslice must be in the range 0 to AKIStateDemandGetNumSlices ()-1.

Output:
≥ 0: No Error
< 0: Error

3.3.23.4 Read the Input Flow of a Traffic State demand.

In C++ and Python:

Explanation: Read the input flow of one slice and vehicle type. If this name is NULL or "all" the function returns the demand aggregating all vehicle types.

Format:

double AKIStateDemandGetDemandSection(int idSection, int vehTypePos, int numSlice)

Parameters:

idSection is the input section identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used.

numSlice has to be defined from 0 to (AKIStateDemandGetNumSlices()-1).

Output:

≥ 0: No Error

< 0: Error

3.3.23.5 Modify the Input Flow of a Traffic State demand.

In C++ and Python:

Explanation: Modify the input flow for one slice and vehicle type. If this name is NULL or "all" modify all vehicle types demand multiplying the number of trips of each vehicle type by a factor calculated as: anewdemand / TotalDemand (where TotalDemand is the sum of trips considering all vehicle types).

Format:

int AKIStateDemandSetDemandSection(int idSection, int vehTypePos, int numSlice, double anewflow)

Parameters:

idSection is the input section identifier.

vehTypePos is the position of the vehicle type in the list of vehicles types being used.

numSlice has to be defined from 0 to (AKIStateDemandGetNumSlices ()-1).

newflow is the new input flow.

Output:

≥ 0: No Error

< 0: Error

3.3.24 Functions relative to Past Costs

3.3.24.1 Read the number of intervals of Past Costs Read

In C++ and Python:

Explanation: Read the number of Past Costs previously read. If it is 0 then there is no Past Cost previously read.

Format:

int AKIPastCostGetNbIntervalsReaded ()

Parameters: none

Output:

≥ 0: No error

< 0: Error

3.3.24.2 Read whether the Past Costs are read per Vehicle Type

In C++ and Python:

Explanation: Read whether the Past Costs has been read per vehicle type

Format:

int AKIPastCostAreCostsPerVehicleType ()

Parameters: none

Output:
≥ 0: 1 means Past Cost per Vehicle Type and 0 otherwise
< 0: Error

3.3.24.3 Read the Initial Time of the PastCost Read

In C++ and Python:
Explanation: Read the initial time of the Past Costs read.
Format:
double AKIPastCostGetIniTimeReaded ()
Parameters: none
Output:
≥ 0: No Error
< 0: Error

3.3.24.4 Read the Time Interval of the PastCost Read

In C++ and Python:
Explanation: Read the time interval of the Past Costs read.
Format:
double AKIPastCostGetIntervalReaded ()
Parameters: none
Output:
≥ 0: No Error
< 0: Error

3.3.24.5 Read Past Cost.

In C++ and Python:
Explanation: Read the past costs (the past cost and the output past cost) of a link.

Format:
double AKIPastCostGetPastCost(int sectorig, int sectdest, double aTime,
int idVehType);
double AKIPastCostGetPastOutputCost(int sectorig, int sectdest, double aTime,
int idVehType);
Parameters:
sectorig is the origin section identifier of the link.
sectdest is the destination section identifier of the link.
aTime is the time of the Past Cost.
idVehType is the Vehicle Type Id.
Output:
≥ 0: No Error
< 0: Error

3.3.24.6 Modify Past Cost.

In C++ and Python:
Explanation: Modify the Past Cost. If this TypeName is NULL or “all” modify the aggregated past cost if the past cost read doesn’t distinguish per vehicle type.

Format:
int AKIPastCostSetPastCost(int sectorig, int sectdest, double aTime,
int idVehType, double acost, double aocost);
Parameters:
sectorig is the origin section identifier of the link.
sectdest is the destination section identifier of the link.
aTime is the time of the Past Cost.

idVehType is the Vehicle Type Id.
acost is the new Past Cost.
aocost is the new Past Output Cost.

Output:
= 0: No Error
< 0: Error

3.3.25 Functions relative to random number generation

3.3.25.1 Get Random Number

In C++ and Python:

Explanation: Get a random number. This function uses the internal random number generator to calculate random numbers between 0 and 1.

Format:

double AKIGetRandomNumber ();

Parameters: none

Output:

Random number between 0 and 1

3.3.26 Functions relative to ANG Replication Information

3.3.26.1 Get ANG Replication Identifier

In C++ and Python:

Explanation: Get the replication Identifier.

Format:

unsigned int ANGConnGetReplicationId ();

Parameters: none

Output:

> 0: Replication Identifier
= 0: Error

3.3.27 Functions relative to ANG Experiment Information

3.3.27.1 Get ANG Experiment Identifier

In C++ and Python:

Explanation: Get the experiment Identifier.

Format:

unsigned int ANGConnGetExperimentId ();

Parameters: none

Output:

> 0: Experiment Identifier
= 0: Error

3.3.28 Functions relative to ANG Scenario Information

3.3.28.1 Get ANG Scenario Identifier

In C++ and Python:

Explanation: Get the scenario identifier.

Format:

unsigned int ANGConnGetScenarioId ();

Parameters: none

Output:

> 0: Scenario identifier
= 0: Error

3.3.28.2 Get ANG Scenario Simulated Date and Initial Time

In C++ and Python:

Explanation: Get the scenario Simulated Date and Initial Time.

Format:

const char * ANGConnGetScenarioTime();

Parameters: none

Output:

!= NULL: Returns a string with the scenario date and time using ISO 8601 extended format (YYYY-MM-DDTHH:MM:SS),or an empty string if no information is available. Don't delete the string.

= NULL: Error

3.3.29 Functions relative to ANG Objects Connection

3.3.29.1 Get ANG Object Id

In C++ and Python:

Explanation: Get the Object Id of any object having the same name.

Format:

int ANGConnGetObjectId(const unsigned short * name, bool deleteUShortName);
// Unicode version. Not available in Python.
int ANGConnGetObjectIdA(const char * name, bool deleteUShortName); // Ascii version.

Parameters:

name: is the name of the object
deleteUShortName: true if the user wants to delete the name automatically

Output:

> 0: object id
< 0: the object doesn't exist

3.3.29.2 Get ANG Object Attribute

In C++ and Python:

Explanation: Get a pointer to an attribute of an ANG object. This pointer can be used to modify or read the value of the attribute of different objects, using functions ANGConnSetAttributeValue* or ANGConnGetAttributeValue*. In Aimsun, the internal attribute names of each object type can be displayed by selecting *Window / Type* from the main menu.

Format:

void * ANGConnGetAttribute(const unsigned short * internalAttrName);

Parameters:

internalAttrName: is the internal name of the attribute

Output:

!= NULL: pointer to attribute
= NULL: Error

3.3.29.3 Create ANG Object Attribute

In C++ and Python:

Explanation: Create an attribute of one ANG object. This pointer can be used to modify or read the value of the attribute of different objects, using functions ANGConnSetAttributeValue* or ANGConnGetAttributeValue*. In Aimsun NG, the internal

attribute names of each object type can be displayed by selecting *Window* / *Type* from the main menu.

Format:

void * ANGConnCreateAttribute(const unsigned short * typeName, const unsigned short * internalAttrName, const unsigned short * externalAttrName, int attrType, int columnType);

Parameters:

typeName: is name of the ANG object
internalAttrName: is the internal name of the attribute
externalAttrName: is the external name of the attribute (as displayed in the editors of the object)
attrType: INTEGER_TYPE as integer, DOUBLE_TYPE as double or STRING_TYPE as a string.
columnType: it defines the way to store the column as EXTERNAL_TEMPORAL, INTERNAL or EXTERNAL

Output:

!= NULL: pointer to attribute
= NULL: Error

3.3.29.4 Set value to ANG Object Attribute

In C++ and Python:

Explanation: Set the value to an attribute of an ANG object. UNICODE version.

Format:

void ANGConnSetAttributeValueString(void * attr, unsigned int objectId, const unsigned short * value); // Unicode version. Not available in Python.
void ANGConnSetAttributeValueStringA(void * attr, unsigned int objectId, const char * value); // Ascii version.
void ANGConnSetAttributeValueInt (void * attr, unsigned int objectId, int value);
void ANGConnSetAttributeValueDouble (void * attr, unsigned int objectId, double value);

Parameters:

attr: is the pointer to the attribute, obtained by calling either ANGConnGetAttribute or ANGConnCreateAttribute.
objectId: is the object identifier
value: the new value to assign

Output:

none

3.3.29.5 Get value from ANG Object Attribute

In C++ and Python:

Explanation: Get the value of an attribute of an ANG object.

Format:

const unsigned short * ANGConnGetAttributeValueString(void * attr, unsigned int objectId); // Unicode version. Not available in Python.
const char * ANGConnGetAttributeValueStringA(void * attr, unsigned int objectId); // Ascii version.
int ANGConnGetAttributeValueInt(void * attr, unsigned int objectId);
double ANGConnGetAttributeValueDouble(void * attr, unsigned int objectId);

Parameters:

attr: is the pointer of the attribute, obtained calling either ANGConnGetAttribute or ANGConnCreateAttribute
objectId: is the object identifier

Output: return the value, according to the attribute type (integer, double or string)

3.3.29.6 Get number of values in a Time Series ANG Object Attribute

In C++ and Python:

Explanation: Get number of values of a time series attribute of an ANG object.

Format:

int ANGConnGetTimeSeriesSize(void * attr, unsigned int objectId)

Parameters:

attr: is the pointer of the attribute, obtained calling either ANGConnGetAttribute or ANGConnCreateAttribute

objectId: is the object identifier

Output: return the number of values

3.3.29.7 Get value in a Time Series ANG Object Attribute

In C++ and Python:

Explanation: Get a value of a time series attribute of an ANG object.

Format:

double ANGConnGetTimeSeriesValue(void * attr, unsigned int objectId, unsigned int pos)

Parameters:

attr: is the pointer of the attribute, obtained calling either ANGConnGetAttribute or ANGConnCreateAttribute

objectId: is the object identifier

pos: position of the value in the Time Series attribute (from 0 to (ANGConnGetTimeSeriesSize(attr, objectId)-1)

Output: return the value

3.3.30 Functions relative to ANG Policies (Traffic Management)

3.3.30.1 Initialize ANG Policies

In C++ and Python:

Explanation: Initializes the policies list before a simulation

Format:

void ANGConnInitPolicies ();

Parameters: none

Output: none

3.3.30.2 Activate an ANG Policy

In C++ and Python:

Explanation: Activate a policy defined in ANG.

Format:

void ANGConnActivatePolicy(unsigned int objectId);

Parameters:

objectId: is the policy identifier

Output: none

3.3.30.3 Deactivate an ANG Policy

In C++ and Python:

Explanation: Deactivate a policy defined in ANG.

Format:
void ANGConnDeactivatePolicy (unsigned int objectId);
Parameters:
objectId: is the policy identifier
Output: none

3.3.30.4 Is Active an ANG Policy

In C++ and Python:
Explanation: Check whether a policy defined in ANG is active or not.
Format:
void ANGConnIsPolicyActive (unsigned int objectId);
Parameters:
objectId: is the policy identifier
Output:
true: Policy activated
false: Policy deactivated

3.3.30.5 Is Active an ANG Policy

In C++ and Python:
Explanation: Check whether a policy defined in ANG is active or not.
Format:
void ANGConnMarkActivatePolicy(unsigned int objectId, bool value)
Parameters:
objectId: is the policy identifier
Output:
true: Policy activated
false: Policy deactivated

3.3.31 Functions relative to ANG Text Objects

3.3.31.1 Set text in an ANG Text

In C++ and Python:
Explanation: Sets the text to be displayed in an ANG Text object.
Format:
void ANGConnSetText (unsigned int objectId, const unsigned short * text);
Parameters:
objectId: is the text object identifier
newText: is the new text to be displayed in ANG
Output: none

3.3.32 Functions relative to ANG Simulation Vehicles

When simulating in Aimsun, every vehicle is represented by two objects: one with the information related to behavioural models and the other one with the information related to the ANG Graphical Environment (that is, the GKSimVehicle that has, among others, all the information required to display the animation and this object can be managed using the ANG Objects Connections functions 3.3.29). These two objects will have different identifiers as every time a vehicle is created, the Aimsun Simulator assigns to it a consecutive identifier starting from 1 whereas the GKSimVehicle has a unique identifier that distinguishes it from all the other ANG objects. Function 3.3.32.1 can be used to obtain the GKSimVehicle identifier linked to an Aimsun vehicle identifier.
Furthermore, for efficiency reasons, the GKSimVehicles are only created and updated when the simulation is done in Interactive mode. When creating and using attributes for the GKSimVehicle type, the Aimsun Simulation Vehicles need to be created and updated independently of running

the simulation interactively or in Batch mode. To activate the creation and updating of Aimsun Simulation vehicles also in batch mode, function 3.3.32.2 must be used.

3.3.32.1 Get ANG Simulation Vehicle Identifier

In C++ and Python:

Explanation: Get the vehicle identifier defined in ANG for the simulation vehicle with Aimsun id *aidVeh*.

Format:

int ANGConnVehGetGKSimVehicleId(int aidVeh)

Parameters:

aidVeh: is the Aimsun vehicle identifier

Output:

> 0: ANG Vehicle Identifier

< 0: Error

3.3.32.2 Act ANG Vehicles Information while simulating in Batch mode

In C++ and Python:

Explanation: Activates the actualization of ANG Vehicles Information while simulating in Batch mode.

Format:

void ANGConnEnableVehiclesInBatch(bool value)

Parameters:

value: *true* when ANG vehicles want to be actualized while simulating and false otherwise

Output: None

3.3.33 Functions Relative to Vehicle Path Information

When simulating a Microscopic simulation in Aimsun the traffic demand can be either by states or O/D Matrices. When the traffic demand type is O/D Matrices vehicles move throughout the network using paths. These paths can come from three different sources:
Route Choice Paths: these are the paths that have been calculated by the route choice model.
User Defined Paths: It's a path from one origin centroid to a destination centroid. The user can edit these paths by creating Routes, or can be automatically created by a Macro simulaton.
User Defined Shortest Path Trees: This is a shorest path tree structure, where for each destination centroid there is a path from every point in the network only in those cases where the path exists. This structure can be created either by a Macro or Dynamic simulation approach.
The following functions can be used to retrieve vehicle path information as well as the path itself.

3.3.33.1 Get Vehicle Path Information

In C++ and Python:

Explanation: Read the information of the used path for a vehicle. This function requires having the vehicle identification.

Format:

PathInfVeh AKIVehInfPath(int idveh)

Parameters:

idveh: the vehicle identifier

Output:

```
struct PathInfVeh{
        int report;
        int idVeh;
        int type;
```

int entranceSectionId;
int numSectionsInPath;
double totalDistance;
};

where*:*

int *report*. 0 OK, otherwise check the error code.
int *idVeh*. Vehicle identifier.
int *type*. Path type:
1: Route choice path
2: User Defined Route
3: User Defined Shortest Path Tree
int *entranceSectionId*. Entrance section identifier. This is the section from where the vehicle enters the network.
int *numSectionsInPath*. Total number of sections that have the path.
double *totalDistance*. Total distance of the path in meters.

3.3.33.2 Get Vehicle Path Informationon in a section

In C++ and Python:

Explanation: Read the information of certain vehicle in a section. This function requires, previously, get the number of Vehicles in the section calling AKIVehStateGetNbVehiclesSection.

Format:

PathInfVeh AKIVehInfPathSection(int aidSec, int indexveh)

Parameters:

aidSec: section Identifier
indexveh: vehicle index, from 0 to (Total Number of Vehicles in the section – 1)

Output:

struct PathInfVeh*{*
int report;
int idVeh;
int type;
int entranceSectionId;
int numSectionsInPath;
double totalDistance;
};

where*:*

int *report*. 0 OK, otherwise check the error code.
int *idVeh*. Vehicle identifier.
int *type*. Path type:
1: Route choice path
2: User Defined Route
3: User Defined Shortest Path Tree
int *entranceSectionId*. Entrance section identifier. This is the section from where the vehicle enters the network.
int *numSectionsInPath*. Total number of sections that have the path.
double *totalDistance*. Total distance of the path in meters.

3.3.33.3 Get next section in vehicle path

In C++ and Python:

Explanation: Returns the next section identifier that is using a vehicle.
Format:

int AKIVehInfPathGetNextSection(int idveh, int fromsection)

Parameters:

idveh: the vehicle identifier

fromsection: This is the from section identifier.

Output:

>0: next section identifier from the fromsection

-1: fromsection is the last section of the path

otherwise: check error code

3.3.33.4 Get next section in vehicle path in section

In C++ and Python:

Explanation: Read the next section identifier of certain vehicle in a section. This function requires, previously, get the number of Vehicles in the section calling AKIVehStateGetNbVehiclesSection.

Format:

int AKIVehInfPathGetNextSectionInSection(int aidSec, int indexveh, int fromsection)

Parameters:

aidSec: section Identifier

indexveh: vehicle index, from 0 to (Total Number of Vehicles in the section – 1)

fromsection: This is the from section identifier.

Output:

>0: next section identifier from the fromsection

-1: fromsection is the last section of the path

ohterwise: check error code

3.3.34 Functions Relative to Simulation Control

3.3.34.1 Get simulation control command

In C++ and Python:

Explanation: Reads the last simulation control programmed by using the API

Format:

int ANGGetSimulationOrder(int * when)

Parameters:

when: contains the seconds from the beginning of the simulation when the simulator will stop. It is only initialised when the last command programmed is StopAt

Output:

0: No control command programmed

1: Cancel simulation

2: Rewind simulation

3: Stop simulation

4: Stop At Simulation

3.3.34.2 Set simulation control command

In C++ and Python:

Explanation: Sets a new simulation control programmed using the API

Format:

void ANGSetSimulationOrder(int order, int when)

Parameters:

order:

0: No control command programmed

1: Cancel simulation

2: Rewind simulation

3: Stop simulation

4: Stop At Simulation

when: contains the seconds from the beginning of the simulation when the simulator will stop. It is only used when the new command is StopAt

Output: None

4 Building an Aimsun API

The Aimsun API module can be implemented as a dynamic library written in C++ or as a script file written in Python (version 2.6.1) (for more information about this scripting language, visit http://www.python.org)

4.1 Building an Aimsun API using C++

A C++ Aimsun API Extension is a dynamic library that contains the desired code that will interact with the Aimsun microscopic simulator. Aimsun will load this dynamic library, if selected in the Scenario Editor, when a simulation starts.

In order to develop this library, we supplythe following files:

- Header Files:
  - AKIProxie.h
  - CIProxie.h
  - ANGConProxie.h
  - AAPI_Util.h
- Libraries:
  - Windows 32 and 64 bits versions: angext.lib
  - Linux 32 and 64 bits versions: liba2kernel.6.1.0.so (located in the Aimsun Home folder)
  - Mac version: liba2kernel.6.1.0.dylib (located in the Aimsun.app/Contents/PlugIns folder)
- Sample Files:
  - AAPI.h
  - AAPI.cxx

The user can only modify the sample file AAPI.cxx, fill in the routines AAPILoad(), AAPIInit(), AAPIManage(…), AAPIPostManage(…), AAPIFinish() and AAPIUnLoad() and add some other files if it is necessary.

By default the user receives a Visual C++ 2005 project ready to compile and build the DLL. The Visual C++ 2005 project has two configurations available: the Debugger configuration that builds the AAPI_D.dll by default and the Release configurations that builds the AAPI_R.dll by default.

A very simple example of the Aimsun API Module could be defined by the following contents of AAPI.cxx:

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>

// Procedures could be modified by the user

int AAPILoad()
{
    AKIPrintString("LOAD");
    return 0;
}

int AAPIInit()
{
```

```
    AKIPrintString("\tInit");
    return 0;
}

int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{
    return 0;
}

int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
    return 0;
}

int AAPIFinish()
{
    AKIPrintString("\tFinish");
    return 0;
}

int AAPIUnLoad()
{
    AKIPrintString("UNLOAD");
    return 0;
}
```

4.2 Building an Aimsun API using a Python Script

The Aimsun API Extension can be implemented using a Python script. This Python script is an ASCII file with extension *.py* and may be created in any text editor. To load an Aimsun API Module written in Python, the file should be placed in a user defined folder with a file named *AAPI.py* (supplied with Aimsun).

The user can write a Python script defining the six high level functions: AAPILoad(), AAPIIni(), AAPIManage(…), AAPIPostManage(…), AAPIFinish() and AAPIUnLoad().

A very simple example of Aimsun API Module with the following contents is provided in the *sample.py* file:

```
from AAPI import *

def AAPILoad():
   AKIPrintString("Load....")
   return 0

def AAPIInit():
   AKIPrintString("Init....")
   return 0

def AAPIManage(time, timeSta, timeTrans, acycle):
    AKIPrintString("Manage....")
    return 0

def AAPIPostManage(time, timeSta, timeTrans, acycle):
   AKIPrintString("PostManage....")
   return 0
```

```
def AAPIFinish():
    AKIPrintString("Finish....")
    return 0

def AAPIUnLoad():
    AKIPrintString("Unload....")
    return 0
def AAPIEnterVehicle( idveh,  idsectionOrTurn):
    return 0

def AAPIExitVehicle( idveh,  idsection):
    return 0
```

There are some functions that need to pass parameters by reference. In Python it is not possible to pass parameters by reference, but there are some extra functions that let us use this kind of parameters.

In the following code there is an example of how functions with reference parameters can be used in a Python script. The function calculates the cycle duration of a junction.

```
def getCycleTime(junctionId , numPhases, t):
    cycle = 0
    pdur = doublep()
    pcmax = doublep()
    pcmin = doublep()
    for i in range(1, numPhases):
        ECIGetDurationsPhase(junctionId, i, t, pdur, pcmax, pcmin)
        cycle = cycle + pdur.value()
    del pdur
    del pcmax
    del pcmin
    return cycle
```

5 How to enable and load an Aimsun API Entension from Aimsun

To enable the user-defined Aimsun API Extension, it is necessary to add it in the Scenario definition. Clicking on the Scenario and by selecting the *Aimsun API* tab folder the following dialog appears:

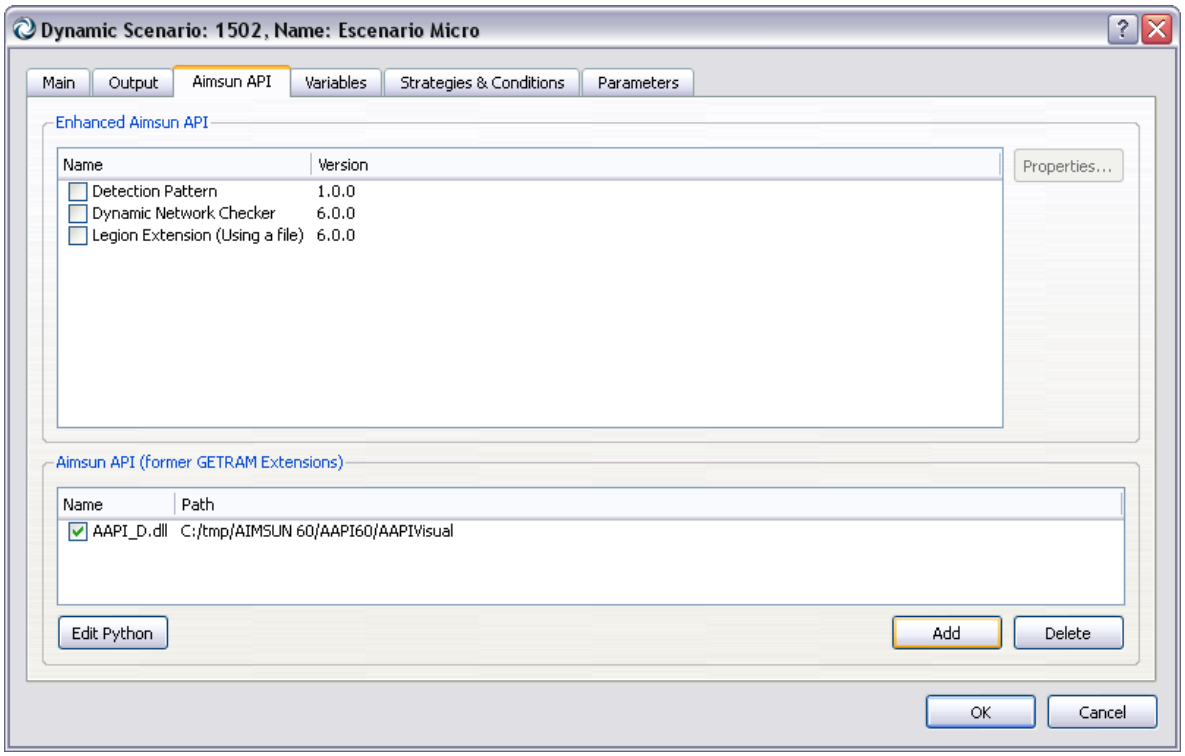

Figure 4 Extensions Dialog

In this dialog, using the Add button or Remove button the user can load or unload a set of Aimsun Extensions (Dynamic libraries or Python scripts).

6 Examples

6.1 Example obtaining information about Vehicles

This example allows the information of every vehicle inside the system to be obtained for each simulation step.

6.1.1 C++ Version

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>

// Procedures could be modified by the user

char astring[128];

int AAPILoad()
{
	return 0;
}

int AAPIInit()
{
	return 0;
}

int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{

	InfVeh infVeh;
	int nba = AKIInfNetNbSectionsANG();
	for(int i=0; i<nba;i++){
		int id = AKIInfNetGetSectionANGId(i);
		int nb = AKIVehStateGetNbVehiclesSection(id,true);
		for (int j=0; j<nb;j++){
			infVeh = AKIVehStateGetVehicleInfSection(id,j);
			sprintf(astring,"Vehicle %d , Section %d , Lane %d, CurrentPos %f,
CurrentSpeed %f\n",infVeh.idVeh, infVeh.idSection, infVeh.numberLane, infVeh.CurrentPos,
infVeh.CurrentSpeed);
			AKIPrintString(astring);
		}
	}
	int nbj = AKIInfNetNbJunctions();
	for( int i=0; i<nbj;i++){
		int id = AKIInfNetGetJunctionId(i);
		int nb = AKIVehStateGetNbVehiclesJunction(id);
		for ( int j=0; j<nb;j++){
			infVeh = AKIVehStateGetVehicleInfJunction(id,j);
			sprintf(astring,"Vehicle %d , Node %d , From %d, To %d, CurrentPos
%f,CurrentSpeed %f\n", infVeh.idVeh, infVeh.idJunction, infVeh.idSectionFrom,
infVeh.idSectionTo, infVeh.CurrentPos, infVeh.CurrentSpeed);
			AKIPrintString(astring);
```

```
                }
        }
        return 0;
}

int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
        return 0;
}

int AAPIFinish()
{
        return 0;
}

int AAPIUnLoad()
{
        return 0;
}
```

6.1.2 Python Version

```
from AAPI import *

def AAPILoad():
        return 0

def AAPIInit():

        return 0

def AAPIManage(time, timeSta, timTrans, SimStep):

        nba = AKIInfNetNbSectionsANG()
        for i in range(nba):
                id = AKIInfNetGetSectionANGId(i)
                nb = AKIVehStateGetNbVehiclesSection(id,True)
                for j in range(nb):
                        infVeh = AKIVehStateGetVehicleInfSection(id,j)
                        astring = "Vehicle " + str(infVeh.idVeh) + ", Section " +
str(infVeh.idSection) + " , Lane " + str(infVeh.numberLane) + ", CurrentPos " +
str(infVeh.CurrentPos) + ", CurrentSpeed " + str(infVeh.CurrentSpeed)
                        AKIPrintString(astring)

        nbj = AKIInfNetNbJunctions()
        for i in range(nbj):
                id = AKIInfNetGetJunctionId(i)
                nb = AKIVehStateGetNbVehiclesJunction(id)
                for j in range(nb):
                        infVeh = AKIVehStateGetVehicleInfJunction(id,j)
                        astring = "Vehicle " + str(infVeh.idVeh) + ", Node " + str(infVeh.idJunction)
+ " , From " +  str(infVeh.idSectionFrom) + ", To " + str(infVeh.idSectionTo) + ", CurrentPos " +
str(infVeh.CurrentPos) + ", CurrentSpeed " + str(infVeh.CurrentSpeed)
                        AKIPrintString(astring)

        return 0
```

```
def AAPIPostManage(time, timeSta, timTrans, SimStep):
        return 0

def AAPIFinish():
        return 0

def AAPIUnLoad():
        return 0

def AAPIEnterVehicle(idveh,idsectionOrTurn):
        return 0

def AAPIExitVehicle(idveh,idsection):
        return 0
```

6.2 Example obtaining Statistical information

This example shows how to access to the different statistical information during and at the end of the simulation.

6.2.1 C++ Version

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>

// Procedures could be modified by the user

char astring[128];
int idVeh, carPosition;

int AAPILoad()
{
        return 0;
}

int AAPIInit()
{
     idVeh = ANGConnGetObjectId( AKIConvertFromAsciiString( "car" ), false )
     carPosition = AKIVehGetVehTypeInternalPosition( idVeh )
        return 0;
}

int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{
        return 0;
}

int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
        StructAkiEstadSystem estad = AKIEstGetParcialStatisticsSystem(timeSta, 0);
        if (estad.report==0){
                AKIPrintString("\t\t SYSTEM");
```

```
        sprintf(astring,"\t\t Report :%d",estad.report);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Flow :%d ",estad.Flow);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Travel Time :%f",estad.TTa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Delay Time :%f",estad.DTa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Speed :%f",estad.Sa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Stop Time :%f \n",estad.STa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t NumStops :%f",estad.NumStops);
        AKIPrintString(astring);
}

StructAkiEstadSection estad2 = AKIEstGetParcialStatisticsSection(1, timeSta, 0);
if (estad2.report==0){
        sprintf(astring,"\t\t SECTION");
        AKIPrintString(astring);
        sprintf(astring,"\t\t Report :%d",estad2.report);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Flow :%d ",estad2.Flow);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Travel Time :%f",estad2.TTa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Delay Time :%f",estad2.DTa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Speed :%f",estad2.Sa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Stop Time :%f \n",estad2.STa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t NumStops :%f",estad2.NumStops);
        AKIPrintString(astring);
        sprintf(astring,"\t\t LongQueueAvg :%f ",estad2.LongQueueAvg);
        AKIPrintString(astring);
        sprintf(astring,"\t\t LongQueueMax :%f ",estad2.LongQueueMax);
        AKIPrintString(astring);
}

estad = AKIEstGetParcialStatisticsSystem(timeSta, carPosition);
if (estad.report==0){
        sprintf(astring,"\t\t SYSTEM CAR\n");
        AKIPrintString(astring);
        sprintf(astring,"\t\t Report :%d",estad.report);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Flow :%d ",estad.Flow);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Travel Time :%f",estad.TTa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Delay Time :%f",estad.DTa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Speed :%f",estad.Sa);
        AKIPrintString(astring);
        sprintf(astring,"\t\t Stop Time :%f \n",estad.STa);
```

```
		AKIPrintString(astring);
		sprintf(astring,"\t\t NumStops :%f",estad.NumStops);
		AKIPrintString(astring);
	}

	estad2 = AKIEstGetParcialStatisticsSection(1, timeSta, carPosition);
	if (estad2.report==0){
		sprintf(astring,"\t\t SECTION CAR\n");
		AKIPrintString(astring);
		sprintf(astring,"\t\t Report :%d",estad2.report);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Flow :%d ",estad2.Flow);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Travel Time :%f",estad2.TTa);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Delay Time :%f",estad2.DTa);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Speed :%f",estad2.Sa);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Stop Time :%f \n",estad2.STa);
		AKIPrintString(astring);
		sprintf(astring,"\t\t NumStops :%f",estad2.NumStops);
		AKIPrintString(astring);
		sprintf(astring,"\t\t LongQueueAvg :%f ",estad2.LongQueueAvg);
		AKIPrintString(astring);
		sprintf(astring,"\t\t LongQueueMax :%f ",estad2.LongQueueMax);
		AKIPrintString(astring);
	}
	return 0;
}

int AAPIFinish()
{
	StructAkiEstadSystem estad = AKIEstGetGlobalStatisticsSystem(0);
	if (estad.report==0){
		sprintf(astring,"\t\t SYSTEM\n");
		AKIPrintString(astring);
		sprintf(astring,"\t\t Report :%d",estad.report);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Flow :%d ",estad.Flow);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Travel Time :%f",estad.TTa);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Delay Time :%f",estad.DTa);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Speed :%f",estad.Sa);
		AKIPrintString(astring);
		sprintf(astring,"\t\t Stop Time :%f \n",estad.STa);
		AKIPrintString(astring);
		sprintf(astring,"\t\t NumStops :%f",estad.NumStops);
		AKIPrintString(astring);
	}

	StructAkiEstadSection estad2 = AKIEstGetGlobalStatisticsSection(1, 0);
	if (estad2.report==0){
```

```
                sprintf(astring,"\t\t SECTION\n");
                AKIPrintString(astring);
                sprintf(astring,"\t\t Report :%d",estad2.report);
                AKIPrintString(astring);
                sprintf(astring,"\t\t Flow :%d ",estad2.Flow);
                AKIPrintString(astring);
                sprintf(astring,"\t\t Travel Time :%f",estad2.TTa);
                AKIPrintString(astring);
                sprintf(astring,"\t\t Delay Time :%f",estad2.DTa);
                AKIPrintString(astring);
                sprintf(astring,"\t\t Speed :%f",estad2.Sa);
                AKIPrintString(astring);
                sprintf(astring,"\t\t Stop Time :%f \n",estad2.STa);
                AKIPrintString(astring);
                sprintf(astring,"\t\t NumStops :%f",estad2.NumStops);
                AKIPrintString(astring);
                sprintf(astring,"\t\t LongQueueAvg :%f ",estad2.LongQueueAvg);
                AKIPrintString(astring);
                sprintf(astring,"\t\t LongQueueMax :%f ",estad2.LongQueueMax);
                AKIPrintString(astring);
        }
        return 0;
}


int AAPIUnLoad()
{
        return 0;
}
```

6.2.2 Python Version

```
from AAPI import *

carPosition = 0

def AAPILoad():
        return 0

def AAPIInit():
    idVeh = ANGConnGetObjectId( AKIConvertFromAsciiString( "car" ), False )
    carPosition = AKIVehGetVehTypeInternalPosition( idVeh )
        return 0

def AAPIManage(time, timeSta, timTrans, SimStep):
        return 0

def AAPIPostManage(time, timeSta, timTrans, SimStep):
        estad = AKIEstGetParcialStatisticsSystem(timeSta, 0)
        if (estad.report==0):
                AKIPrintString("\t\t SYSTEM")
                astring = "\t\t Report : " + str(estad.report)
                AKIPrintString(astring)
                astring = "\t\t Flow : " + str(estad.Flow)
                AKIPrintString(astring)
                astring = "\t\t Travel Time : " + str(estad.TTa)
```

```
        AKIPrintString(astring)
        astring = "\t\t Delay Time : " + str(estad.DTa)
        AKIPrintString(astring)
        astring = "\t\t Speed : " + str(estad.Sa)
        AKIPrintString(astring)
        astring = "\t\t Stop Time : " + str(estad.STa)
        AKIPrintString(astring)
        astring = "\t\t NumStops : " + str(estad.NumStops)
        AKIPrintString(astring)

estad2 = AKIEstGetParcialStatisticsSection(1, timeSta, 9)
if (estad2.report==0):
        AKIPrintString("\t\t SECTION")
        astring = "\t\t Report : " + str(estad2.report)
        AKIPrintString(astring)
        astring = "\t\t Flow : " + str(estad2.Flow)
        AKIPrintString(astring)
        astring = "\t\t Travel Time : " + str(estad2.TTa)
        AKIPrintString(astring)
        astring = "\t\t Delay Time : " + str(estad2.DTa)
        AKIPrintString(astring)
        astring = "\t\t Speed : " + str(estad2.Sa)
        AKIPrintString(astring)
        astring = "\t\t Stop Time : " + str(estad2.STa)
        AKIPrintString(astring)
        astring = "\t\t NumStops : " + str(estad2.NumStops)
        AKIPrintString(astring)
        astring = "\t\t LongQueueAvg : " + str(estad2.LongQueueAvg)
        AKIPrintString(astring)
        astring = "\t\t LongQueueMax : " + str(estad2.LongQueueMax)
        AKIPrintString(astring)

estad = AKIEstGetParcialStatisticsSystem(timeSta, carPosition)
if (estad.report==0):
        AKIPrintString("\t\t SYSTEM CAR\n")
        astring = "\t\t Report : " + str(estad.report)
        AKIPrintString(astring)
        astring = "\t\t Flow : " + str(estad.Flow)
        AKIPrintString(astring)
        astring = "\t\t Travel Time : " + str(estad.TTa)
        AKIPrintString(astring)
        astring = "\t\t Delay Time : " + str(estad.DTa)
        AKIPrintString(astring)
        astring = "\t\t Speed : " + str(estad.Sa)
        AKIPrintString(astring)
        astring = "\t\t Stop Time : " + str(estad.STa)
        AKIPrintString(astring)
        astring = "\t\t NumStops : " + str(estad.NumStops)
        AKIPrintString(astring)

estad2 = AKIEstGetParcialStatisticsSection(1, timeSta, carPosition)
if (estad2.report==0):
        AKIPrintString("\t\t SECTION CAR\n")
        astring = "\t\t Report : " + str(estad2.report)
        AKIPrintString(astring)
```

```
                astring = "\t\t Flow : " + str(estad2.Flow)
                AKIPrintString(astring)
                astring = "\t\t Travel Time : " + str(estad2.TTa)
                AKIPrintString(astring)
                astring = "\t\t Delay Time : " + str(estad2.DTa)
                AKIPrintString(astring)
                astring = "\t\t Speed : " + str(estad2.Sa)
                AKIPrintString(astring)
                astring = "\t\t Stop Time : " + str(estad2.STa)
                AKIPrintString(astring)
                astring = "\t\t NumStops : " + str(estad2.NumStops)
                AKIPrintString(astring)
                astring = "\t\t LongQueueAvg : " + str(estad2.LongQueueAvg)
                AKIPrintString(astring)
                astring = "\t\t LongQueueMax : " + str(estad2.LongQueueMax)
                AKIPrintString(astring)

        return 0

def AAPIFinish():
        estad = AKIEstGetGlobalStatisticsSystem(0)
        if (estad.report==0):
                AKIPrintString("\t\t SYSTEM\n")
                astring = "\t\t Report : " + str(estad.report)
                AKIPrintString(astring)
                astring = "\t\t Flow : " + str(estad.Flow)
                AKIPrintString(astring)
                astring = "\t\t Travel Time : " + str(estad.TTa)
                AKIPrintString(astring)
                astring = "\t\t Delay Time : " + str(estad.DTa)
                AKIPrintString(astring)
                astring = "\t\t Speed : " + str(estad.Sa)
                AKIPrintString(astring);
                astring = "\t\t Stop Time : " + str(estad.STa)
                AKIPrintString(astring);
                astring = "\t\t NumStops : " + str(estad.NumStops)
                AKIPrintString(astring)

        estad2 = AKIEstGetGlobalStatisticsSection(1, 0)
        if (estad2.report==0):
                AKIPrintString("\t\t SECTION\n")
                astring = "\t\t Report : " + str(estad2.report)
                AKIPrintString(astring)
                astring = "\t\t Flow : " + str(estad2.Flow)
                AKIPrintString(astring)
                astring = "\t\t Travel Time : " + str(estad2.TTa)
                AKIPrintString(astring)
                astring = "\t\t Delay Time : " + str(estad2.DTa)
                AKIPrintString(astring)
                astring = "\t\t Speed : " + str(estad2.Sa)
                AKIPrintString(astring)
                astring = "\t\t Stop Time : " + str(estad2.STa)
                AKIPrintString(astring)
                astring = "\t\t NumStops : " + str(estad2.NumStops)
                AKIPrintString(astring)
```

```
                astring = "\t\t LongQueueAvg : " + str(estad2.LongQueueAvg)
                AKIPrintString(astring)
                astring = "\t\t LongQueueMax : " + str(estad2.LongQueueMax)
                AKIPrintString(astring)

        return 0

def AAPIUnLoad():
        return 0

def AAPIEnterVehicle(idveh,idsectionOrTurn):
        return 0

def AAPIExitVehicle(idveh,idsection):
        return 0
```

6.3 Example of vehicle entrance and vehicles tracking

This example shows how to implement vehicle tracking and how introduce vehicles into the system using an external entrance distribution.

6.3.1 C++ Version

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>

int idVehFirst = 0;
int idVehSecond = 0;
int idVeh = 0;
int carPosition = 0;

int AAPILoad()
{
        return 0;
}

int AAPIInit()
{
     idVeh = ANGConnGetObjectId( AKIConvertFromAsciiString( "car" ), false )
     carPosition = AKIVehGetVehTypeInternalPosition( idVeh )
     idVehFirst = AKIEnterVehTrafficOD(108, carPosition, 285, 288, True);
     idVehSecond = AKIEnterVehTrafficOD(108, carPosition, 285, 288, True);
     return 0;
}

int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{

        AKIVehTrackedModifySpeed(idVehFirst, 10);
        AKIVehTrackedModifySpeed(idVehSecond, 10);
        AKIVehTrackedModifyNextSection(idVehSecond, 546);
        return 0;
}
```

```
int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
	return 0;
}

int AAPIFinish()
{
	return 0;
}

int AAPIUnLoad()
{
	return 0;
}
```

6.3.2 Python Version

```
from AAPI import *

idVehFirst = 0
idVehSecond = 0
carPosition = 0

def AAPILoad():
	return 0

def AAPIInit():
	global idVehFirst
	global idVehSecond
	global carPosition

	idVeh = 8
	carPosition = AKIVehGetVehTypeInternalPosition( idVeh )

	idVehFirst = AKIEnterVehTrafficOD(108, carPosition, 285, 288, True)
	idVehSecond = AKIEnterVehTrafficOD(108, carPosition, 285, 288, True)
	return 0

def AAPIManage(time, timeSta, timTrans, SimStep):
	global idVehFirst
	global idVehSecond

	AKIVehTrackedModifySpeed(idVehFirst, 10)
	AKIVehTrackedModifySpeed(idVehSecond, 10)
	AKIVehTrackedModifyNextSection(idVehSecond, 546)
	return 0

def AAPIPostManage(time, timeSta, timTrans, SimStep):
	return 0

def AAPIFinish():
	return 0

def AAPIUnLoad():
	return 0
```

```
def AAPIEnterVehicle(idveh,idsection):
        return 0

def AAPIExitVehicle(idveh,idsection):
        return 0
```

6.4 Example of incident generation

This example shows how to implement incident generation. It generates two incidents and then it removes the first one.

6.4.1 C++ Version

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>

// Procedures could be modified by the user
int AAPILoad()
{
        return 0;
}

int AAPIInit()
{
        AKIGenerateIncident(329, 1, 150.0f, 1.0f, AKIGetIniSimTime()+5.0f, 300.0f, 200.0f, true);
        AKIGenerateIncident(329, 2, 150.0f, 1.0f, AKIGetIniSimTime()+5.0f, 300.0f, 200.0f, true);
        return 0;
}

int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{

        if (time >= 400){
                AKIRemoveIncident(329, 1, 150);
                AKIRemoveIncident(329, 2, 150);
        }

        return 0;
}

int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
        return 0;
}

int AAPIFinish()
{
        AKIResetAllIncidents();
        return 0;
}

int AAPIUnLoad()
```

```
{
        return 0;
}
```

6.4.2 Python Version

```
from AAPI import *

def AAPILoad():
        return 0

def AAPIInit():
        AKIGenerateIncident(329, 1, 150, 1, AKIGetIniSimTime() + 5, 300, 200, True)
        AKIGenerateIncident(329, 2, 150, 1, AKIGetIniSimTime() + 5, 300, 200, True)
        return 0

def AAPIManage(time, timeSta, timTrans, SimStep):
        if (time >= 400):
                AKIRemoveIncident(329, 1, 150)
                AKIRemoveIncident(329, 2, 150)
        return 0
def AAPIPostManage(time, timeSta, timTrans, SimStep):
        return 0

def AAPIFinish():
        AKIResetAllIncidents()
        return 0

def AAPIUnLoad():
        return 0

def AAPIEnterVehicle(idveh,idsection):
        return 0

def AAPIExitVehicle(idveh,idsection):
        return 0
```

6.5 Example of Public Transport Management

This example shows how to manage public transport. It generates a new bus at 18h01m and disables its first bus stop. At the same time the Aimsun API Module stores its distance travelled.

6.5.1 C++ Version

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <fstream>

char                    astring[128];
std::ofstream   fileFloatCar;
int                     idveh = 0;
int                     busVehicleTypeId = 79;
bool                    timeChanged = false;
// Procedures could be modified by the user
int AAPILoad()
```

```
{
	return 0;
}

int AAPIInit()
{
	AKIPrintString("Init....");
	fileFloatCar.open("C:/tmp/BusyPy.txt", std::ios::out);
	idveh = 0;
	return 0;
}

int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{
	double tempsEntrada = AKIGetIniSimTime() + 60;
	if (timeSta == tempsEntrada) {
		AKIPrintString("Entering a Vehicle");
		//getting the bus vehicle type position
		int busPosition = AKIVehGetVehTypeInternalPosition(busVehicleTypeId);
		idveh = AKIPTEnterVeh(1418, busPosition, true );
		sprintf(astring, " idveh = %d", idveh);
		AKIPrintString(astring);
	}
	if (timeSta >= tempsEntrada + 1 && timeChanged == false) {
		AKIPrintString("Modifying Stop Time of first bus stop");
		double retorn = 0;
		retorn = AKIPTVehModifyStopTime( idveh, 0, 0 );
		sprintf(astring, " retorn = %d", retorn);
		AKIPrintString(astring);
		timeChanged = true;
	}
	return 0;
}

int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
	if (idveh > 0) {
		InfVeh infveh = AKIVehTrackedGetInf(idveh);
		if (infveh.report == 0) {
			sprintf(astring, "%f %f\n", timeSta, infveh.TotalDistance);
			fileFloatCar << astring;
		} else {
			idveh = 0;
			fileFloatCar.close();
		}
	}
	return 0;
}

int AAPIFinish()
{
	AKIPrintString("...Finish");
	return 0;
}
```

```
int AAPIUnLoad()
{
	return 0;
}
```

6.5.2 Python Version

```
from AAPI import *

#variables globals
fileFloatCar = 0
idveh=0
busVehicleTypeId = 79
timeChanged = False

def AAPILoad():
	return 0
def AAPIInit():
	AKIPrintString("Init....")
	global idveh, fileFloatCar
	fileFloatCar=open('C:/tmp/BusPy.txt', 'w');
	idveh = 0
	return 0

def AAPIManage(time, timeSta, timeTrans, acycle):
	global idveh
	global busVehicleTypeId
	global timeChanged

	tempsEntrada = AKIGetIniSimTime()+60
	if timeSta == tempsEntrada:
		AKIPrintString("Entering a Vehicle")
		#getting the bus vehicle type position
		busPosition = AKIVehGetVehTypeInternalPosition(busVehicleTypeId)
		idveh = AKIPTEnterVeh( 1418, busPosition, True )
		astring = ' idveh = %d' %(idveh)
		AKIPrintString(astring)


	if timeSta >= tempsEntrada + 1 and timeChanged == False:
		AKIPrintString("Modifying Stop Time of first bus stop")
		retorn = AKIPTVehModifyStopTime( idveh, 0, 0 )
		astring = ' retorn = %d' %(retorn)
		AKIPrintString(astring)
		timeChanged = True
	return 0

def AAPIPostManage(time, timeSta, timeTrans, acycle):
	global idveh, fileFloatCar
	if idveh > 0 :
		infveh = AKIVehTrackedGetInf(idveh)
		if infveh.report == 0:
			astring = "%f %f\n" %(timeSta, infveh.TotalDistance)
			fileFloatCar.write(astring)
		else:
```

```
                idveh = 0
                fileFloatCar.close()
        return 0

def AAPIFinish():
        AKIPrintString("...Finish")
        return 0

def AAPIUnLoad():
        return 0
```

6.6 Example of ANG Connection (Creating a new attribute)

This example shows how to create a new attribute “Occupied” (as integer) in the section object and then set a value depending on the value of an attribute already defined (in that case the road type).

6.6.1 C++ Version

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>
#include <string.h>
// Procedures could be modified by the user

int AAPILoad()
{
        return 0;
}

int AAPIInit()
{
        //getting the attributes by name. A UNICODE conversion is required by using
AKIConvertFromAsciiString method

        void * roadTypeAtt = ANGConnGetAttribute(
                AKIConvertFromAsciiString("GKSection::roadTypeAtt") );
        void * Occupied = ANGConnGetAttribute(
AKIConvertFromAsciiString("GKSection::Occupied" ));
        if (Occupied == NULL){
                Occupied = ANGConnCreateAttribute( AKIConvertFromAsciiString("GKSection"),
                                        AKIConvertFromAsciiString("GKSection::Occupied"),
                                        AKIConvertFromAsciiString("Occupied"), INTEGER_TYPE,
EXTERNAL);
        }
        int NbSections=AKIInfNetNbSectionsANG();
        for(int i=0; i<NbSections;i++)
        {
                bool anyNonAsciiChar;
                int SecId = AKIInfNetGetSectionANGId(i);
                //getting the value
                int roadTypeId = ANGConnGetAttributeValueInt( roadTypeAtt, SecId );
                bool nonChar;
```

```
                char* roadTypeName = AKIConvertToAsciiString(
ANGConnGetObjectName(roadTypeId), false, &nonChar );
                if (roadTypeName == "Parking" ){
                        ANGConnSetAttributeValueInt( Occupied, SecId, 1 );
                }else{
                        ANGConnSetAttributeValueInt( Occupied, SecId, 0 );
                }
        }

        return 0;
}
int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{
        return 0;
}

int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
        return 0;
}

int AAPIFinish()
{
        return 0;
}

int AAPIUnLoad()
{
        return 0;
}
```

6.6.2 Python Version

```
from AAPI import *
from PyANGKernel import *

def AAPILoad():
        return 0

def AAPIInit():
        model = GKSystem.getSystem().getActiveModel()

        roadTypeAtt = ANGConnGetAttribute(
AKIConvertFromAsciiString("GKSection::roadTypeAtt" ))
        if roadTypeAtt!= None:
                Occupied = ANGConnGetAttribute(
AKIConvertFromAsciiString("GKSection::Occupied" ))

                if Occupied == None:
                        Occupied = ANGConnCreateAttribute(
AKIConvertFromAsciiString("GKSection"), AKIConvertFromAsciiString("GKSection::Occupied"),
AKIConvertFromAsciiString("Occupied"), INTEGER_TYPE, EXTERNAL)

                NbSections = AKIInfNetNbSectionsANG()
```

```
            for i in range(0, NbSections):
                SecId = AKIInfNetGetSectionANGId(i)
        value = ANGConnGetAttributeValueInt( roadTypeAtt, SecId )
        nonChar = boolp()
        roadTypeName = AKIConvertToAsciiString( ANGConnGetObjectName(roadTypeId),
 False, nonChar )
                if roadTypeName == "Parking" :
                    ANGConnSetAttributeValueInt( Occupied, SecId, 1 )
                else:
                    ANGConnSetAttributeValueInt( Occupied, SecId, 0 )

    return 0

def AAPIManage(time,timeSta,timTrans,acicle):
    return 0

def AAPIPostManage(time,timeSta,timTrans,acicle):
    return 0

def AAPIFinish():
    return 0

def AAPIUnLoad():
    return 0
```

6.7 Example of ANG Connection (Updating a ANG Simulation Vehicle attribute)

This example shows how to create a new attribute "VehStatus" (as an integer) in the ANG Simulation Vehicle object and then set a value depending on the section where the vehicle is located (in section 241 the attribute is set to 1 meanwhile in section 250 is set to 2).

6.7.1 C++ Version

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>

void * vehStatusAtt;

// Procedures could be modified by the user
int AAPILoad()
{
    return 0;
}

int AAPIInit()
{
    ANGConnEnableVehiclesInBatch(true);
  //getting the attributes by name. A UNICODE conversion is required by using
AKIConvertFromAsciiString method
    vehStatusAtt = ANGConnCreateAttribute( AKIConvertFromAsciiString("GKSimVehicle"),
                                AKIConvertFromAsciiString("GKSimVehicle::VehStatusInt"),
```

```
                                        AKIConvertFromAsciiString("Vehicle Status"),
INTEGER_TYPE, EXTERNAL);
        return 0;
}
int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{
        return 0;
}

int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
        int                     nbvehs, i, idVehANG;
        InfVeh          infveh;

        nbvehs = AKIVehStateGetNbVehiclesSection(241, true);
        for (i=0; i< nbvehs; i++) {
                infveh = AKIVehStateGetVehicleInfSection(241,i);
                if (infveh.report>=0) {
                        idVehANG = ANGConnVehGetGKSimVehicleId(infveh.idVeh);
                        if ( idVehANG > 0 ) {
                                ANGConnSetAttributeValueInt(vehStatusAtt, idVehANG, 1);
                        }
                }
        }
        nbvehs = AKIVehStateGetNbVehiclesSection(250, true);
        for (i=0; i< nbvehs; i++) {
                infveh = AKIVehStateGetVehicleInfSection(250,i);
                if (infveh.report>=0) {
                        idVehANG = ANGConnVehGetGKSimVehicleId(infveh.idVeh);
                        if ( idVehANG > 0 ) {
                                ANGConnSetAttributeValueInt(vehStatusAtt, idVehANG, 2);
                        }
                }
        }
        return 0;
}

int AAPIFinish()
{
        return 0;
}

int AAPIUnLoad()
{
        return 0;
}
```

6.7.2 Python Version

```
from AAPI import *

def AAPILoad():
        return 0

def AAPIInit():
        ANGConnEnableVehiclesInBatch(True);
```

```
        # getting the attributes by name. A UNICODE conversion is required by using
AKIConvertFromAsciiString method
        global vehStatusAtt
        vehStatusAtt = ANGConnCreateAttribute( AKIConvertFromAsciiString("GKSimVehicle"),
                                        AKIConvertFromAsciiString("GKSimVehicle::VehStatusInt"),
                                        AKIConvertFromAsciiString("Vehicle Status"), 1, 2)
        return 0

def AAPIManage(time, timeSta, timeTrans, acycle):
        return 0

def AAPIPostManage(time, timeSta, timeTrans, acycle):
        nbvehs = AKIVehStateGetNbVehiclesSection(241, True)
        for i in range(0, nbvehs):
                infveh = AKIVehStateGetVehicleInfSection(241,i)
                if infveh.report >= 0 :
                        idVehANG = ANGConnVehGetGKSimVehicleId(infveh.idVeh)
                        if idVehANG > 0:
                                ANGConnSetAttributeValueInt(vehStatusAtt, idVehANG, 1)
        nbvehs = AKIVehStateGetNbVehiclesSection(250, True)
    for i in range(0, nbvehs):
                infveh = AKIVehStateGetVehicleInfSection(250,i)
                if infveh.report >= 0:
                        idVehANG = ANGConnVehGetGKSimVehicleId(infveh.idVeh)
                        if idVehANG > 0:
                                ANGConnSetAttributeValueInt(vehStatusAtt, idVehANG, 2)
        return 0

def AAPIFinish():
        return 0

def AAPIUnLoad():
        return 0
```

6.8 Example of Control (Bus Preemption on a junction)

This example shows how to use Aimsun API function to manage the control plan, so that buses approaching a junction have green light when they reach the junction. Once the bus leaves the junction, the control plan is restored to when the bus call was detected.

6.8.1 C++ Version

```
#include "AKIProxie.h"
#include "CIProxie.h"
#include "ANGConProxie.h"
#include "AAPI.h"
#include <stdio.h>

double previousPhaseIndex, previousPhaseTime;

// Procedures could be modified by the user
int AAPILoad()
{
        return 0;
}
```

```
int AAPIInit()
{
	ANGConnEnableVehiclesInBatch(true);
	previousPhaseIndex = -1;
	previousPhaseTime = -1;
	return 0;
}

int AAPIManage(double time, double timeSta, double timeTrans, double acycle)
{
	return 0;
}

int AAPIPostManage(double time, double timeSta, double timeTrans, double acycle)
{
	int busPhase = 3;
	int intersection = 203;
	int busCallDetector = 383;
	int busExitDetector = 378;
	//get bus internal position
	int busVehiclePosition = AKIVehGetVehTypeInternalPosition( 9 );

	int currentPhase = ECIGetCurrentPhase( intersection );
	//check bus presence over busCallDetector
	if( AKIDetGetCounterCyclebyId( busCallDetector, busVehiclePosition ) > 0 &&
currentPhase != busPhase && previousPhaseIndex == -1){
		AKIPrintString( "bus detected");
		#change the control to bus phase
		previousPhaseIndex = currentPhase;
		previousPhaseTime  = time - ECIGetStartingTimePhase( intersection );
		ECIChangeDirectPhase( intersection, busPhase, timeSta, time, acycle, 0 );
	}
	//check bus presence over busExitDetector
	if( AKIDetGetCounterCyclebyId( busExitDetector, busVehiclePosition ) > 0 ){
		//go back to previous phase
		if( previousPhaseIndex > 0 ){
			AKIPrintString( "go back to previous state”);
			ECIChangeDirectPhase( intersection, previousPhaseIndex, timeSta, time,
acycle,  previousPhaseTime);
			previousPhaseIndex = -1;
			previousPhaseTime = -1;
		}
	}
	return 0;
}

int AAPIFinish()
{
	return 0;
}

int AAPIUnLoad()
{
	return 0;
}
```

6.8.2 Python Version

```
from AAPI import *

def AAPILoad():
        return 0

def AAPIInit():
        return 0

def AAPIManage(time, timeSta, timeTrans, acycle):
        return 0

def AAPIPostManage(time, timeSta, timeTrans, acycle):
        global previousPhaseIndex
        global previousPhaseTime

        busPhase = 3
        intersection = 203
        busCallDetector = 383
        busExitDetector = 378
        #get bus internal position
        busVehiclePosition = AKIVehGetVehTypeInternalPosition( 9 )

        currentPhase = ECIGetCurrentPhase( intersection )
        #check bus presence over busCallDetector
        if AKIDetGetCounterCyclebyId( busCallDetector, busVehiclePosition ) > 0 and
currentPhase != busPhase and previousPhaseIndex == -1:
                print "bus detected"
                #change the control to bus phase
                previousPhaseIndex = currentPhase
                previousPhaseTime  = time - ECIGetStartingTimePhase( intersection )
                ECIChangeDirectPhase( intersection, busPhase, timeSta, time, acycle, 0 )

        #check bus presence over busExitDetector
        if AKIDetGetCounterCyclebyId( busExitDetector, busVehiclePosition ) > 0:
                #go back to previous phase
                if previousPhaseIndex > 0:
                        print "go back to previous state"
                        ECIChangeDirectPhase( intersection, previousPhaseIndex, timeSta, time,
acycle,  previousPhaseTime)
                        previousPhaseIndex = -1
                        previousPhaseTime = -1


        return 0

def AAPIFinish():
        return 0

def AAPIUnLoad():
        return 0
```

previousPhaseIndex = -1
previousPhaseTime = -1

7 Error Codes

CtrlFatalError = -1
- Fatal Error

CtrlControlNotLoaded –1000
-Error. Control Not Loaded
CtrlControlNotInitialazed –1013
-Error. Control Not Initialized
CtrlNotCorrectControl = -1014
-Warning: The number of control is incorrect, it does not accomplish 0 ≤ elem < Number of Controls
CtrlUnknownSignal = -2002
-Warning: The signal group is unknown
CtrlUnknownState = -2003
-Warning: The state must be 0 : RED or 1: GREEN or 2:YELLOW or 3:FLASH_GREEN
CtrlEventsEnabled = -2004
-Warning: The junction has the events enabled
CtrlUnknownPhase = -2005
-Warning : Phase unknown
CtrlUnknownJunction = -2007
-Warning : Junction unknown
CtrlUnknownMet =-2008
-Warning : Unknown metering
CtrlIncorrectTypeMet =-2009
-Warning : Incorrect type of metering
CtrlJunctionNotControled = -2010
-Warning : Junction is fixed or uncontrolled
CtrlMetNotControled =-2011
-Warning : Metering is fixed or uncontrolled
CtrlNotExternalControl = -2012
-Warning : The control has not been loaded using Aimsun API Module
CtrlSameIniTimeControl = -2013
-Warning : The control is not loaded, because has the same initial time as another
CtrlRemovingCurrentControl =-2014
-Warning : The control has not been removed because is the current control.
CtrlUnknownPhaseName =-2015
-Warning : The name of the phase is incorrect.
CtrlUnknownVehTypeName =-2021
-Warning : The vehicle type name is incorrect.
CtrlInvalidIndex =-2022
-Warning : The index is incorrect, either it is negative or greater or equal to the total number of elements.
CtrlGreenTimeLesserThanZero=-2023
-Error : greenTime parameter must be greater than or equal to 0.
CtrlPhaseNotActuated = -2024
- Warning : The phase does not belong to an actuated control junction
CtrlJunctionNotCoordinated = -2025
- Warning : The junction is not defined as coordinated
CtrlPhaseWithCoordinationRecall = -2026
- Warning : The phase is defined as coordinated, do it does not have the force off and permissive periods defined

AKIVmsUnknownPanel = -3001
- Error : Unknown Panel. The VMS name or Identifier are incorrect.

AKIVmsUnknownMessage = -3002
- Error : Unknown Message. The Message is incorrect because it is not defined in the VMS.

AKIVmsIndexNotValid = -3003
- Error : Unknown Message. The Message is incorrect because it is not defined in the VMS.

AKIDETUnknownDetector = -3010
- Warning : Unknown Detector

AKIDETIncorrectInterval = -3011
- Warning : Unknown Interval

AKIDETMeasureNotGathered = -3012
- Warning : The measure is not available in the detector.

AKIDETNoAggregatedDetection = -3013
- Warning : The aggregated detection is not available.

AKIDETIncorrectTypeName = -3014
- Warning : The name of vehicle type is incorrect.

AKIDETNoInstantDetection = -3015
- Warning : The instant detection is not available.

AKIInfVehGetMem = -4001
- Warning : Not enough memory for information vehicle.

AKIInfVehUnknownSection = -4002
- Warning : The identifier of the section is incorrect.

AKIInfIndexNotValid = -4003
- Warning : The index is incorrect.

AKIInfNotReady = -4004
- Warning : The information of vehicle type is not ready.

AKIInfVehUnknownJunction = -4005
- Warning : The identifier of the junction is incorrect.

AKIInfVehNotFound = -4006
- Warning : The identifier of the vehicle is incorrect.

AKIInfVehInvalidParam = -4007
- Warning : The parameter is incorrect.

AKIInfVehNotAvalaible = -1
- Warning : The vehicle is not available.

AKIVehInvalidVehicleTypeId = -4008
- Warning : The provided vehicle type identifier is invalid

AKIInfNetGetMem =-5001
- Warning : No enough memory for information network.

AKIInfUnknownId =-5002
- Warning : Invalid Section Identifier.

AKIInfUnknownTurning =-5003
- Warning : Invalid Turning Identifier.

AKIESTUnknownSection = -6001;
- Warning : The identifier of the section is incorrect.

AKIESTNotAvalaible = -6002;
- Warning : The statistical information is not available.

AKIESTUnknownCentroid = -6004;
- Warning : The Centroid identifier is incorrect.

AKIESTUnknownStream = -6005;
- Warning : The Statistical stream identifier is incorrect.

AKIESTFuelConsumptionNotAvailable = -6006
- Warning : The Fuel consumption information is not available

AKIESTPollutionEmissionNotAvailable = -6007

- Warning : The Pollution emission information is not available

AKIESTUnknownPollutant = -6008

- Warning : The index of the pollutant is not correct.

AKIESTWrongLane = -6009

- Warning : The index of the lane is not correct.

AKIESTUnknownNode = -6010

- Warning : The node identifier is incorrect.

AKIEnterVehUnknownSection = -7001;

- Warning : The section identifier is incorrect.

AKIEnterVehUnFeasibleLane = -7003;

- Warning : The lane is not feasible.

AKIEnterVehNotSpace = -7004;

- Warning : There is no space.

AKIEnterVehUnknownCentroid = -7005;

- Warning : The centroid identifier is incorrect.

AKIEnterVehUnFeasiblePath = -7006;

- Warning : There is not path feasible to reach the destination.

AKIEnterVehNoTrafficFlow = -7008;

- Warning : The traffic conditions are not defined by flow and turning proportions.

AKIEnterVehNoTrafficOD = -7009;

- Warning : The traffic conditions are not defined by the O/D matrix.

AKIEnterVehUnknownLane = -7010;

- Warning : The lane identifier is incorrect.

AKIEnterVehUnknownNextSection = -7011;

- Warning : The next section identifier is incorrect.

AKIVehNotTracked = -7012;

- Warning : The vehicle is not tracked.

AKIVehInvalidParameter = -7013;

- Warning : Invalid parameter (new speed or new lane).

AKIVehNextSectionUnreachable = -7014;

- Warning : The next section is unreachable.

AKIEnterVehUnknownVehType = -7016;

- Warning : The vehicle type index specified does not exist.

AKILegionNotLoaded = -7017

- Error: Legion Simulator Extension is not loaded

AKIZeroPedestriansToGenerate -7018

- Error: The number of pedestrians to generate must be greater than 0

AKIZeroTimeInterval -7019

- Error: The time interval to generate pedestrians must be greater than 0

AKIEntranceCentroidNotFound -7020

- Error: Origin Legion centroid does not exist or has not been loaded in the simulation

AKIExitCentroidNotFound -7021

- Error: Destination Centroid does not exist or has not been loaded in the simulation

AKIRouteNotFound -7022

- Error: Legion OD Route does not exist

AKIEntranceCentroidNotInRoute -7023

- Error: The entrance centroid id does not belong to the route

AKIExitCentroidNotInRoute -7024

- Error: The exit centroid id does not belong to the route

AKIIncidentWrongIniTime = -8001

- Warning : The initial time of the incident is invalid.

AKIIncidentWrongPosition = -8002

- Warning : The position of the incident is invalid.

AKIIncidentUnknownLane = -8003
- Warning : The lane of the incident is invalid.

AKIIncidentUnknownSection = -8004
- Warning : The section of the incident is invalid.

AKIIncidentNotPresent = -8005
- Warning : The incident is not correct because is not present.

AKIIncidentWrongLength = -8006
- Warning : The length of the incident is invalid.

AKIIncidentWrongDuration = -8007
- Warning : The duration of the incident is invalid.

AKIPTNotLoaded = -9001
- Warning : The public transport is not loaded.

AKIPTVehUnknown = -9002
- Warning : The vehicle is not a public transport vehicle.

AKIPTStopUnknown = -9003
- Warning : The stop is not valid.

AKIPTVehTypeUnknown = -9004
- Warning : The vehicle type is incorrect.

AKIPTLineUnknown = -9005
- Warning : The public transport line is incorrect.

AKIPTVehNotSpace = -9006
- Warning : There is no space.

AKIPTIndexNotValid = -9007
- Warning : The index is not valid.

AKIPTVehUnFeasibleLane = -9008
- Warning : the lane is not feasible.

AKIPTVehCapacityOverflow = -9009
- Warning : the lane is not feasible.

AKIPTNotAvalaible = -1
- Warning : information not available.

AKIDetectorEventsNoTraffic = -10000
-Warning: The traffic demand is not loaded and it is required

AKIODDemandNoTrafficOD = -11000
-Warning: The traffic conditions are not defined by O/D matrix.

AKIODDemandIncorrectNumSlice = -11002
-Warning: The Number of Slice is incorrect.

AKIODDemandUnknownCentroid = -11003
-Warning: The origin or destination identifier is unknown.

AKIODDemandUnknownODPair = -11004
-Warning: The OD Pair doesn't have demand defined or is not connected.

AKIPastCostUnknownLink = -12001
-Warning: The link defined by section origin and destination is not correct.

AKIPastCostNoPerVehType = -12002
-Warning: Past Cost is not defined per vehicle type.

AKIPastCostPerVehType = -12003
-Warning: Past Cost is defined per vehicle type.

AKIPastCostPerVehType = -12004
-Warning: Past Cost is not read, so is not available.

AKIPastCostIncorrectTypeName = -12005
-Warning: The vehicle type is incorrect.

AKIInfVehPathNotAvailable = -13001
- Warning The vehicle path does not exist.

AKIInfVehDestinationUnreachable = -13002
- Warning The vehicle cannot reach the destination centroid.

AKIInfVehInvalidDestinationCentroid = -13003
- Warning The new destination centroid does not exist.

AKIFleetVehUnknown = -14002
- Warning. The fleet doesn't have any fleet assignment

AKIFleetStopUnknown = -14003
- Warning. The fleet stop doesn't exist

AKIFleetVehNotSpace = -14005
- Warning. There's not enough space to enter a new fleet vehicle

AKIFleetVehUnFeasibleLane = -14006
- Warning. There's no lane type in the section for the fleet vehicle

AKIFleetSectUnknown = -14007
- Warning. The section doesn't exist

AKIFleetStopTypeUnknown = -14008
- Warning. Fleet stop type not valid

AKIFleetInvalidFleetId = -14009
- Warning. The fleet doesn't exist

AKIFleetRouteAlreadyAssigned = -14010
- Warning. The fleet vehicle already has a route

AKIFleetSectNotConnected = -14011
- Warning. There is any unconnected section in the fleet route

AKIFleetRouteUnknown = -14012
- Warning. The fleet vehicle assignment doesn't have any route

AKIFleetFleetVehNotGenerated = -14013
- Warning. The fleet vehicle assignment doesn't have any vehicle

AKIFleetInvalidFirstSection = -14014
- Warning. The first section of the new route is unreachable

AKIFleetInvalidDistance = -14015
- Warning. The position of the fleet stop at the section is greater than the section length

AKIFleetInvalidLane = -14016
- Warning. The lane of the fleet stop is invalid

ANGConUnknownObject = -15001
- Warning. There isn't any object having the provided name

AKIStateDemandNoTrafficState = -15000
- Warning. The demand is not defined using traffic states

AKIStateDemandIncorrectNumSlice = -15002
- Warning. The slice provided name does not exist.

AKIStateDemandUnknownVehType = -15004
- Warning: The vehicle type index does not exist.

8 Frequently asked questions

- **Getting Started**
  The Aimsun API is a set of methods that allow to read and modify information used during an Aimsun Microscopic simulation (concerning vehicles, demand, control, detection data, …).

  The Aimsun API is provided in C++ and in Python. It is up to the user to decide which one to use.

  The files to build an Aimsun API in C++ are located in the Aimsun installation folder (*programming\Aimsun API\AAPIVisual*). In there you will find a Visual Studio project that contains all the files needed to build a C++ AAPI. The only files that you may modify are AAPI.cxx and AAPI.h

  The files to build an Aimsun API in Python are located in the Aimsun installation folder (*programming\Aimsun API\AAPIPython*). The only file that you may modify is sample.py

- **Does Aimsun API require a special licence?**
  Yes. You need the Aimsun API licence that is available for Aimsun Standard, Aimsun Professional for Micro, Aimsun Advanced and Aimsun Expert editions.

8.1 FAQ for C++ Aimsun APIs

- **Which Visual Studio version I should use?**
  The provided Visual Studio project is made in MVS 2005, so we recommend to use the same version. If it is not possible, we have succesfully tested AAPIs built in MVS 2008.

- **How to debug my AAPI**
  Start the debugger and attach it to Aimsun (*Tools/Attach...* menu). Place the breakpoints where you want and start the simulation. Note that, it might not know that the dll is there until you start the simulation for the first time. If you try to debug AAPIInit or AAPILoad it may not reach breakpoints placed in these methods unless you restart the debugger as follows (Ctrl+Shift+F5), then you should be able to debug the AAPILoad and AAPIInit.

- **How to use strings**
  There are two methods in AAPI that allow ascii to unicode conversions:

  - *const unsigned short *AKIConvertFromAsciiString(const char *ascii)*: This method convert into unicode any ascii string.

  - *const char *AKIConvertToAsciiString(const unsigned short *string, bool deleteUshortString, bool *anyNonAsciiChar)*: This method converts into ascii any unicode string. Note that the unicode string can contain non-convertable characters, in that case, *anyNonAsciiChar* will be set to *true*.

```
00 const unsigned short* name =
          AKIConvertFromAsciiString("GKSection::nblanesAtt");
01 bool anyNonAsciiChar;
02 const char* ascii = AKIConvertToAsciiString( name, false,
                                &anyNonAsciiChar );
03 AKIPrintString( ascii );
04 AKIPrintAsUNICODEString( name );
```

Line 00: Ascii to Unicode conversion
Line 02: Unicode to Ascii conversion
Line 03, 04: print in both modes. In this case, it prints exactly the same string

8.2 FAQ for Python Aimsun APIs

- **File myScript.py not found or Wrong Syntaxis myScript.py is not a valid extension**
  Be sure that the AAPI.py provided with your current Aimsun version is located at the same folder ar the myScript.py file is.

- **Python Error (): No module named new**
  This error arises when there isn't any python installed in the computer. At www.python.org website, it is able to download for free version 2.6.2.

- **How to use in/out parameters**
  We provide basic types pointers in order to use them in some methods. The available types are: boolp, intp, floatp and doublep.

```
00 min_x = doublep()
01 min_y = doublep()
02 max_x = doublep()
03 max_y = doublep()
04 AKIInfNetGetWorldCoordinates( min_x, min_y, max_x, max_y )
05 print "min(%f, %f) max(%f, %f)"%(min_x.value(), min_y.value(),
max_x.value(), max_y.value() )
```

Line 00, 03: double pointers declaration.
Line 04: Method call that initialises the minimum and maximum world coordinates.
Line 05: Priting pointer variables content.

Functions with *int** representing an int array as input parameters are supported by *intArray* class. Here's an small example:

```
nextSections = intArray( 3 )
nextSections [0] = 182
nextSections [1] = 183
nextSections [2] = 184
AKIActionAddNextSubPathODAction(182, 3, nextSections, 286, 285, 0, 110, 100, 100)
```