ETFOMM Application Programming Interface Reference

Version 1.4

Prepared by:

New Global Systems for Intelligent Transportation Management
75 Cavalier Blvd Suite 221
Florence, KY 41042

May 2017

Contents

etFommInterface

loadDll

Calls LoadLibrary function to load the etfomm library module (etfomm.dll) into the address space of the calling process.

Syntax

int loadDll(std::string dllname);

Parameters

Dllname

The name of the etfomm library module to pass in LoadLibrary.

Return Value

If LoadLibrary succeeds, the return value is 0.
If the function fails, the return value is -1.

Init

Receives the addresses of exported functions from the etfomm library module.

Syntax

int Init();

Return Value

If all functions are exported successfully, the return value is 0.
If any of the function exports fails, the return value is -1.

SetInputs

Sets input parameters for etfomm simulation engine.

Syntax

int SetInputs(std::string trfFILE = "", int TSDFlag = 0, int TIDFlag = 0, int OutFlag = 0, int CSVFlag = 0);

Parameters

trfFILE

The name of the input TRF file, which is also used to compose the output file names.

TSDFlag

Writes timestep data (vehicles and signals, every time step) to the animation files. 0 = No, 1 = yes (TRAFVU format), 2 = yes (text animation files for 3d animator).

TIDFlag

Writes time interval data to the animation files. 0 = No, 1 = Yes. Only applies to the TSIS version at this time.

OutFlag

Writes to a formatted text output file. Currently not used.

CSVFlag

Writes a CSV file equivalent to the output file. Currently not used.

Return Value

If the parameters are set successfully, the return value is 0.
If the function fails, the return value is error code from etfomm.

StartUP

Starts the simulation in etfomm.dll.

Syntax

int StartUP(bool API = false);

Parameters

API

A bool variable to indicate which type of simulation to start.

Return Value

If the simulation is started successfully, the return value is 0.
If the function fails, the return value is error code from etfomm.

RunToEquilibrium

Performs simulation without data collection until the initialization phase is completed. Does not produce animation files or MOEs.

Syntax

int RunToEquilibrium(void);

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

StepSimulate

Performs one timestep of simulation with data collection. Produces animation files and MOEs if the user chose to do so.

Syntax

int StepSimulate(void);

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

ShutDown

Closes all files opened during the simulation and deallocates all dynamically allocated arrays

Syntax

int ShutDown(void);

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

WriteTRFFile

Writes TRF output file.

Syntax

void WriteTRFFile(void);

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

SetNumberOfFreewayLinks

Calls SET_NUMBER_OF_FREEWAYLINKS function in etfomm.dll to set the number of freeway links in the network.

Syntax

void SetNumberOfFreewayLinks(int n_freeway_links);

Parameters

n_freeway_links

Specifies the number of freeway links in the network. etfomm allocates freeway link arrays to the number of freeway links.

SetNumberOfStreetLinks

Calls SET_NUMBER_OF_STREETLINKS function in etfomm.dll to set the number of street links in the network.

Syntax

void SetNumberOfStreetLinks(int n_street_links);

Parameters

n_street_links

Specifies the number of street links in the network. etfomm allocates street link arrays to the number of street links.

SetNumberOfEntryNodes

Calls SET_NUMBER_OF_ENTRYNODES function in etfomm.dll to set the number of entry nodes.

Syntax

void SetNumberOfEntryNodes(int n_entrynodes);

Parameters

n_entrynodes

Specifies the number of entry nodes. etfomm allocates entry node arrays to the number of entry nodes.

SetNumberOfFTCSignals

Calls SET_NUMBER_OF_FTC_SIGNALS function in etfomm.dll to set the number of fixed-time control signals.

Syntax

void SetNumberOfFTCSignals(int n_ftcs);

Parameters

n_ftcs

Specifies the number of fixed-time control signals. etfomm allocates fixed-time control signal arrays to the number of fixed-time control signals.

SetNumberOfACSignals

Calls SET_NUMBER_OF_AC_SIGNALS function in etfomm.dll to set the number of actuated control signals.

Syntax

void SetNumberOfACSignals(int n_acs);

Parameters

n_acs

Specifies the number of actuated control signals. etfomm allocates actuated control signal arrays to the number of actuated control signals.

SetNumberOfRampMeters

Calls SET_NUMBER_OF_RAMPMETERS function in etfomm.dll to set the number of ramp meters.

Syntax

void SetNumberOfRampMeters(int n_rampmeters);

Parameters

n_rampmeters

Specifies the number of ramp meters. etfomm allocates ramp meter arrays to the number of ramp meters.

SetNumberOfFreewayDetectors

Calls SET_NUMBER_OF_FREEWAY_DETECTORS function in etfomm.dll to set the number of freeway detectors.

Syntax

void SetNumberOfFreewayDetectors(int n_fdet);

Parameters

n_fdet

Specifies the number of freeway detectors. etfomm allocates freeway detector arrays to the number of freeway detectors.

SetNumberOfStreetDetectors

Calls SET_NUMBER_OF_STREET_DETECTORS function in etfomm.dll to set the number of street detectors.

Syntax

void SetNumberOfStreetDetectors(int n_sdet);

Parameters

n_sdet

Specifies the number of street detectors. etfomm allocates street detector arrays to the number of street detectors.

SetNumberOfBusRoutes

Calls SET_NUMBER_OF_BUSROUTES function in etfomm.dll to set the number of bus routes.

Syntax

void SetNumberOfBusRoutes(int n_busroutes);

Parameters

n_busroutes

Specifies the number of bus routes. etfomm allocates bus route arrays to the number of bus routes.

SetNumberOfEvents

Calls SET_NUMBER_OF_EVENTS function in etfomm.dll to set the number of blockage events.

Syntax

void SetNumberOfEvents(int n_events);

Parameters

n_events

Specifies the number of blockage events. etfomm allocates blockage event arrays to the number of blockage events.

SetNumberOfParkingZones

Calls SET_NUMBER_OF_PARKING_ZONES function in etfomm.dll to set the number of parking zones.

Syntax

void SetNumberOfParkingZones(int n_parkingzones);

Parameters

n_parkingzones

Specifies the number of parking zones. etfomm allocates parking zone arrays to the number of parking zones.

SetNumberOfIncidents

Calls SET_NUMBER_OF_INCIDENTS function in etfomm.dll to set the number of freeway incidents.

Syntax

void SetNumberOfIncidents(int n_incidents);

Parameters

n_incidents

Specifies the number of freeway incidents. etfomm allocates freeway incident arrays to the number of freeway incidents.

SetNumberOfDiversions

Calls SET_NUMBER_OF_DIVERSIONS function in etfomm.dll to set the number of freeway diversions.

Syntax

void SetNumberOfDiversions(int n_diversions);

Parameters

n_diversions

Specifies the number of freeway diversions. etfomm allocates freeway diversion arrays to the number of freeway diversions.

SetRunInputs

Defines run number for multiple runs, random number seeds and a flag to write MOEs.

Syntax

int SetRunInputs(int nrand = -1, int seed1 = -1, int seed2 = -1, int seed3 = -1);

Parameters

nrand

Indicates run numbers for multiple runs.

Seed1, seed2, seed3

Indicates random number seeds.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

SetNetworkInputs

Calls SET_NETWORK_INPUTS function in etfomm.dll to define time period durations, time interval and a flag to skip initialization.

Syntax

int SetNetworkInputs(NETWORK_INPUTS Network_Inputs);

Parameters

Network_Inputs

A NETWORK_INPUTS structure that specifies the information to define time period durations, time interval and a flag to skip initialization.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

SetFreewayNetworkInputs

Calls SET_FREEWAY_NETWORK_INPUTS function in etfomm.dll to define freeway networks.

Syntax

void SetFreewayNetworkInputs(FREEWAY_NETWORK_INPUTS Freeway_Network_Inputs);

Parameters

Freeway_Network_Inputs

A FREEWAY_NETWORK_INPUTS structure that specifies the information to define inputs for freeway networks: lag to accelerate or decelerate, friction coefficient, default HOV utilization percentage, car following factors, desired speed distribution table, percentage of cooperative drivers, lane change duration, and multiplier for discretionary lane changes.

SetStreetNetworkInputs

Calls SET_STREET_NETWORK_INPUTS function in etfomm.dll to define surface street networks.

Syntax

void SetStreetNetworkInputs(STREET_NETWORK_INPUTS Street_Network_Inputs);

Parameters

Street_Network_Inputs

A STREET_NETWORK_INPUTS structure that specifies the information to define inputs for surface street networks: acceptable gaps at signs, amber deceleration table, bus station dwell multiplier, car following factors, desired speed distribution table, percentage of cooperative drivers, lane change duration, left turn jumper and lagger percentages, maximum left and right turning speeds, probability of joining spillback, speed at which a vehicle is considered stopped, speed for vehicles going through a yield sign, driver familiarity with path, multiplier distribution for startup lost time and queue discharge headway, pedestrian distributions, multiplier distribution for short term events.

SetVehicleTypes

Calls DEFINE_VEHICLE_TYPES function in etfomm.dll to define properties of vehicle types.

Syntax

void SetVehicleTypes(VEHICLE_TYPE_DATA *Vehicle_Type_Inputs);

Parameter

Vehicle_Type_Inputs

Points to an array of VEHICLE_TYPE_DATA variables that specify the information to define properties of vehicle types: length, headway factor, average occupancy, emergency deceleration, distribution of types within fleets for freeway and street networks.

SetFreewayLinks

Calls DEFINE_FREEWAYLINKS function in etfomm.dll to define properties of freeway links.

Syntax

int SetFreewayLinks(FREEWAY_LINK *freeway_link_data);

Parameter

freeway_link_data

Points to an array of FREEWAY_LINK variables that specify the information to define properties of freeway links: upstream node, downstream node, length, number of full lanes, lane alignments, type of link, auxiliary lanes, added or dropped lanes, mean freeflow speed, car following multiplier, exit percentage, detectors, grade, tilt, curvature, pavement type, lane widths, vehicle-type exclusions, anticipatory lane change inputs, vehicle-type exit multipliers, mean startup time from a rampmeter, HOV lane inputs, through receiving link, offramp receiving link, shoulder width, barriers and datastation settings.

SetStreetLinks

Calls DEFINE_STREETLINKS function in etfomm.dll to define properties of street links.

Syntax

int SetStreetLinks(STREET_LINK *street_link_data);

Parameter

street_link_data

Points to an array of STREET_LINK variables that specify the information to define properties of street links: upstream node, downstream node, length, number of full lanes, receiving links, number of left and right pocket lanes, length of left and right pocket lanes, mean freeflow speed, lane alignments, car following multiplier, sight distance, right turn on red code, pedestrian code, grade, vehicle-type exclusions, turn percentages, vehicle-type specific turn multipliers, and the width of the intersection at the upstream end of the link.

SetEntryNodes

Calls DEFINE_ENTRYNODES function in etfomm.dll to define properties of entry nodes.

Syntax

void SetEntryNodes(int typedist, int erlanga, float minsep, ENTRYNODES_DATA *entrynode_inputs);

Parameters

typedist

Specifies the type of distribution.

erlanga

Specifies erlang parameter.

minsep

Specifies minimum separation between vehicles.

entrynode_inputs

Points to an array of ENTRYNODES_DATA variables that specify the information to define properties of entry nodes: flowrate, truck percentage, carpool percentage, percentage of HOV lane violators, lane distribution.

SetFTCSignals

Calls DEFINE_FTC_SIGNALS function in etfomm.dll to define operating parameters for fixed-time control signals.

Syntax

void SetFTCSignals(FTC_DATA *ftc_data_inputs);

Parameters

ftc_data_inputs

Points to an array of FTC_DATA variables that specify the information to define operating parameters for fixed-time control signals: approach links, number of intervals, duration of intervals, offset, and signal codes for each interval.

SetACSignals

Calls DEFINE_AC_SIGNALS function in etfomm.dll to define operating parameters for actuated control signals.

Syntax

void SetACSignals(AC_DATA *ac_data_inputs);

Parameters

ac_data_inputs

Points to an array of AC_DATA variables that specify the information to define operating parameters for actuated control signals: approach links, phase inputs, detectors.

SetRampMeters

Calls DEFINE_RAMPMETERS function in etfomm.dll to define operating parameters for ramp meters.

Syntax

void SetRampMeters(RM_DATA *rampmeter_inputs);

Parameters

rampmeter_inputs

Points to an array of RM_DATA variables that specify the information to define operating parameters for ramp meters: type of metering control, onset time, headway for clock-time metering, capacity for demand/capacity metering, speed thresholds for speed control metering, detectors used for metering, detector update interval and two-per-green setting.

SetFDetectors

Calls DEFINE_FREEWAY_DETECTORS function in etfomm.dll to define operating parameters for freeway detectors.

Syntax

void SetFDetectors(DETECTOR_DATA *fdet_inputs);

Parameters

fdet_inputs

Points to an array of DETECTOR_DATA variables that specify the information to define operating parameters for freeway detectors: ID, link, lanes covered, location on the link, length of the sensing area, carryover time, delay time and operation code.

SetSDetectors

Calls DEFINE_STREET_DETECTORS function in etfomm.dll to define operating parameters for street detectors.

Syntax

void SetSDetectors(DETECTOR_DATA *sdet_inputs);

Parameters

fdet_inputs

Points to an array of DETECTOR_DATA variables that specify the information to define operating parameters for street detectors: ID, link, lanes covered, location on the link, length of the sensing area, carryover time, delay time and operation code.

SetBusRoutes

Calls DEFINE_BUSROUTES function in etfomm.dll to define operating parameters for bus routes.

Syntax

void SetBusRoutes(BUSROUTE_DATA *busroute_inputs);

Parameters

busroute_inputs

Points to an array of BUSROUTE_DATA variables that specify the information to define operating parameters for bus routes: route number, offset, headway, links in route and stations along the route.

SetBusStations

Calls DEFINE_BUSSTATIONS function in etfomm.dll to define operating parameters for bus stations.

Syntax

void SetBusStations(BUSSTATION_DATA *busstation_inputs);

Parameters

busstation_inputs

Points to an array of 99 BUSSTATION_DATA variables that specify the information to define operating parameters for bus stations: link, location on link, blocking code, capacity, mean dwell time and bypass percentage.

SetEvents

Calls DEFINE_EVENTS function in etfomm.dll to define operating parameters for blockage events.

Syntax

void SetEvents(EVENT_DATA *Event_Inputs);

Parameters

Event_Inputs

Points to an array of EVENT_DATA variables that specify the information to define operating parameters for blockage events: link, lane affected, start time, end time and type of event.

SetParkingZones

Calls DEFINE_PARKING_ZONES function in etfomm.dll to define operating parameters for parking zones.

Syntax

void SetParkingZones(PARKING_DATA *Parking_Zone_Inputs);

Parameters

Parking_Zone_Inputs

Points to an array of PARKING_DATA variables that specify the information to define operating parameters for parking zones: frequency and mean duration of parking maneuvers, began and end location of parking zone.

SetIncidents

Calls DEFINE_INCIDENTS function in etfomm.dll to define operating parameters for freeway incidents.

Syntax

void SetIncidents(INCIDENT_DATA *incident_data_inputs);

Parameters

incident_data_inputs

Points to an array of INCIDENT_DATA variables that specify the information to define operating parameters for freeway incidents: link, began and end location, begin and end time, rubbernecking factor, and type of incident.

SetDiversions

Calls DEFINE_DIVERSIONS function in etfomm.dll to define operating parameters for freeway diversions.

Syntax

void SetDiversions(DIVERSION_DATA *Diversion_Inputs);

Parameters

Diversion_Inputs

Points to an array of DIVERSION_DATA variables that specify the information to define operating parameters for freeway diversions: link, location of warning sign, begin and end time, diversion route path ID, percentage of vehicles that will obey the diversion order and the speed at which vehicles should exit the freeway.

SetNodeCoordinates

Calls DEFINE_NODE_COORDINATES function in etfomm.dll to define node X- and Y-coordinates.

Syntax

void SetNodeCoordinates(XY_COORD *xy_coord_inputs);

Parameters

xy_coord_inputs

Points to an array of XY_COORD variables that specify the information to define node X- and Y-coordinates.

SetConditionalTurnpcts

Calls DEFINE_CONDITIONAL_TURNPCTS function in etfomm.dll to define conditional turn percentages for street links.

Syntax

int SetConditionalTurnpcts(COND_TURNPCTS *cond_turnpct_data);

Parameters

cond_turnpct_data

Points to an array of COND_TURNPCTS variables that specify the information to define conditional turn percentages for street links: left, thru, right and diagonal exit percentages for vehicles that entered the link via a left turn, left, thru, right and diagonal exit percentages for vehicles that entered the link via a thru movement, left, thru, right and diagonal exit percentages for vehicles that entered the link via a right turn, and left, thru, right and diagonal exit percentages for vehicles that entered the link via a diagonal movement.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

ProcessFreewayInputs

Calls PROCESS_FREEWAYINPUTS function in etfomm.dll to process passed in freeway link inputs.

Syntax

void ProcessFreewayInputs(void);

ProcessStreetInputs

Calls PROCESS_STREETINPUTS function in etfomm.dll to process passed in street link inputs.

Syntax

void ProcessStreetInputs(void);

SetPHASES

Calls SET_PHASES function in etfomm.dll to indicate current phases.

Syntax

void SetPHASES(int Node, int current_phases[2]);

Parameters

Node

Specifies the node id for which the current phases are updated.

current_phases

An array of 2 int variables that specifies current phases.

SetFVehicle

Calls SET_FVEHICLE_STRUCT function in etfomm.dll to update properties of freeway vehicles in network.

Syntax

int SetFVehicle(VFData *vfdata);

Parameters

vfdata

Points to an array of VFData variables that specify the information to update properties of freeway vehicles in network: driver type, distance to upstream node, speed, acceleration or deceleration rate, desired speed and etc..

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

SetSVehicle

Calls SET_SVEHICLE_STRUCT function in etfomm.dll to update the properties of street vehicles in network.

Syntax

int SetSVehicle(VSData *vsdata);

Parameters

vsdata

Points to an array of VSData variables that specify the information to update the properties of street vehicles in network: driver type, distance to upstream node, speed, acceleration or deceleration rate, desired speed and etc..

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

AddPath

Calls ADD_PATH function in etfomm.dll to add one path.

Syntax

int AddPath(int NofNodes, int *nodes);

Parameters

NofNodes

Specifies number of nodes on the path.

nodes

Points to an array of int variables that specify the node IDs on the path.

Return Value

Returns the path ID of the added path.

GetNumberOfFreewayLinks

Calls GET_NUMBER_OF_FREEWAYLINKS function in etfomm.dll to get the number of freeway links in the network.

Syntax

int GetNumberOfFreewayLinks(void);

Return Value

Returns the number of freeway links in the network.

GetNumberOfStreetLinks

Calls GET_NUMBER_OF_STREETLINKS function in etfomm.dll to get the number of street links in the network.

Syntax

int GetNumberOfStreetLinks(void);

Return Value

Returns the number of street links in the network.

GetNumberOfEntrynodes

Calls GET_NUMBER_OF_ENTRYNODES function in etfomm.dll to get the number of entry nodes.

Syntax

int GetNumberOfEntrynodes(void);

Return Value

Returns the number of entry nodes.

GetNumberOfFTCSignals

Calls GET_NUMBER_OF_FTC_SIGNALS function in etfomm.dll to get the number of fixed-time control signals.

Syntax

int GetNumberOfFTCSignals(void);

Return Value

Returns the number of fixed-time control signals.

GetNumberOfACSignals

Calls GET_NUMBER_OF_AC_SIGNALS function in etfomm.dll to get the number of actuated control signals.

Syntax

int GetNumberOfACSignals(void);

Return Value

Returns the number of actuated control signals.

GetNumberOfRampmeters

Calls GET_NUMBER_OF_RAMPMETERS function in etfomm.dll to get the number of ramp meters.

Syntax

int GetNumberOfRampmeters(void);

Return Value

Returns the number of ramp meters.

SetNumberOfFreewayDetectors

Calls GET_NUMBER_OF_FREEWAY_DETECTORS function in etfomm.dll to get the number of freeway detectors.

Syntax

int GetNumberOfFreewayDetectors(void);

Return Value

Returns the number of freeway detectors.

GetNumberOfStreetDetectors

Calls GET_NUMBER_OF_STREET_DETECTORS function in etfomm.dll to get the number of street detectors.

Syntax

int GetNumberOfStreetDetectors(void);

Return Value

Returns the number of street detectors.

GetNumberOfBusroutes

Calls GET_NUMBER_OF_BUSROUTES function in etfomm.dll to get the number of bus routes.

Syntax

int GetNumberOfBusroutes(void);

Return Value

Returns the number of bus routes.

GetNumberOfVehicleTypes

Calls GET_NUMBER_OF_VEHICLE_TYPES function in etfomm.dll to get the number of vehicle types.

Syntax

int GetNumberOfVehicleTypes(void);

Return Value

Returns the number of vehicle types.

GetNumberOfEvents

Calls GET_NUMBER_OF_EVENTS function in etfomm.dll to get the number of blockage events.

Syntax

int GetNumberOfEvents();

Return Value

Returns the number of blockage events.

GetNumberOfParkingZones

Calls GET_NUMBER_OF_PARKING_ZONES function in etfomm.dll to get the number of parking zones.

Syntax

int GetNumberOfParkingZones();

Return Value

Returns the number of parking zones.

GetNumberOfIncidents

Calls GET_NUMBER_OF_INCIDENTS function in etfomm.dll to get the number of freeway incidents.

Syntax

int GetNumberOfIncidents(void);

Return Value

Returns the number of freeway incidents.

GetNumberOfDiversions

Calls GET_NUMBER_OF_DIVERSIONS function in etfomm.dll to get the number of freeway diversions.

Syntax

int GetNumberOfDiversions(void);

Return Value

Returns the number of freeway diversions.

GetNumberOfFVehicles

Calls GET_FVEHICLE_STRUCT_SIZE function in etfomm.dll to get the number of freeway vehicles.

Syntax

int GetNumberOfFVehicles(void);

Return Value

Returns the number of freeway vehicles.

GetNumberOfSVehicles

Calls GET_SVEHICLE_STRUCT_SIZE function in etfomm.dll to get the number of street vehicles.

Syntax

int GetNumberOfSVehicles(void);

Return Value

Returns the number of street vehicles.

GetCPUTime

Calls GETCPUTIME function in etfomm.dll to get the elapsed CPU time.

Syntax

float GetCPUTime(void);

Return Value

Returns the elapsed CPU time.

GetElapsedTime

Calls GETELAPSEDTIME function in etfomm.dll to get the elapsed simulation time.

Syntax

float GetElapsedTime(void);

Return Value

Returns the elapsed simulation time.

GetCurrentSimTime

Retrieves SIMPARAMS_mp_SIMTIME variable in etfomm.dll to get the current simulation time.

Syntax

float GetCurrentSimTime(void);

Return Value

Returns the current simulation time.

GetCurrentPeriod

Retrieves SIMPARAMS_mp_TIME_PERIOD variable in etfomm.dll to get the current time period.

Syntax

int GetCurrentPeriod(void);

Return Value

Returns the current time period.

GetTimeStep

Retrieves SIMPARAMS_mp_TIMESTEP variable in etfomm.dll to get the time step value.

Syntax

float GetTimeStep(void);

Return Value

Returns the time step value.

GetNetworkInputs

Calls GET_NETWORK_INPUTS function in etfomm.dll to retrieve the properties of the network.

Syntax

void GetNetworkInputs(NETWORK_INPUTS &Network_Inputs);

Parameters

Network_Inputs[out]

Reference of a NETWORK_INPUTS structure that contains the properties of the network: time period durations, time interval and a flag to skip initialization.

GetFreewayNetworkInputs

Calls GET_FREEWAY_NETWORK_INPUTS function in etfomm.dll to retrieve the properties of freeway networks.

Syntax

void GetFreewayNetworkInputs(FREEWAY_NETWORK_INPUTS &Freeway_Network_Inputs);

Parameters

Freeway_Network_Inputs[out]

Reference of a FREEWAY_NETWORK_INPUTS structure that contains the properties of freeway networks: lag to accelerate or decelerate, friction coefficient, default HOV utilization percentage, car following factors, desired speed distribution table, percentage of cooperative drivers, lane change duration, and multiplier for discretionary lane changes.

GetStreetNetworkInputs

Calls GET_STREET_NETWORK_INPUTS function in etfomm.dll to retrieve the properties of surface street networks.

Syntax

void GetStreetNetworkInputs(STREET_NETWORK_INPUTS &Street_Network_Inputs);

Parameters

Street_Network_Inputs[out]

Reference of a STREET_NETWORK_INPUTS structure that contains the properties of surface street networks: acceptable gaps at signs, amber deceleration table, bus station dwell multiplier, car following factors, desired speed distribution table, percentage of cooperative drivers, lane change duration, left turn jumper and lagger percentages, maximum left and right turning speeds, probability of joining spillback, speed at which a vehicle is considered stopped, speed for vehicles going through a yield sign, driver familiarity with path, multiplier distribution for startup lost time and queue discharge headway, pedestrian distributions, multiplier distribution for short term events.

GetVehicleTypes

Calls GET_VEHICLE_TYPES function in etfomm.dll to retrieve properties of vehicle types.

Syntax

void GetVehicleTypes(VEHICLE_TYPE_DATA *Vehicle_Type_Inputs);

Parameter

Vehicle_Type_Inputs[out]

Points to an array of VEHICLE_TYPE_DATA variables that contains properties of vehicle types: length, headway factor, average occupancy, emergency deceleration, distribution of types within fleets for freeway and street networks.

GetFreewayLinks

Calls GET_FREEWAY_LINKS function in etfomm.dll to retrieve properties of freeway links.

Syntax

int GetFreewayLinks(FREEWAY_LINK *freeway_link_data);

Parameter

freeway_link_data[out]

Points to an array of FREEWAY_LINK variables that contains properties of freeway links: upstream node, downstream node, length, number of full lanes, lane alignments, type of link, auxiliary lanes, added or dropped lanes, mean freeflow speed, car following multiplier, exit percentage, detectors, grade, tilt, curvature, pavement type, lane widths, vehicle-type exclusions, anticipatory lane change inputs, vehicle-type exit multipliers, mean startup time from a rampmeter, HOV lane inputs, through receiving link, offramp receiving link, shoulder width, barriers and datastation settings.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetStreetLinks

Calls GET_STREET_LINKS function in etfomm.dll to retrieve properties of street links.

Syntax

int GetStreetLinks(STREET_LINK *street_link_data);

Parameter

street_link_data[out]

Points to an array of STREET_LINK variables that contains properties of street links: upstream node, downstream node, length, number of full lanes, receiving links, number of left and right pocket lanes, length of left and right pocket lanes, mean freeflow speed, lane alignments, car following multiplier, sight distance, right turn on

red code, pedestrian code, grade, vehicle-type exclusions, turn percentages, vehicle-type specific turn multipliers, and the width of the intersection at the upstream end of the link.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetEntrynodes

Calls GET_ENTRYNODES function in etfomm.dll to retrieve properties of entry nodes.

Syntax

int GetEntrynodes(int *typedist, int *erlanga, float *minsep, ENTRYNODES_DATA *entrynode_data);

Parameters

typedist[out]

Points to an int variable that holds the type of distribution.

erlanga[out]

Points to an int variable that holds the erlang parameter.

minsep[out]

Points to an int variable that holds the minimum separation between vehicles.

entrynode_inputs[out]

Points to an array of ENTRYNODES_DATA variables that contains properties of entry nodes: flowrate, truck percentage, carpool percentage, percentage of HOV lane violators, lane distribution.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetFTCSignals

Calls GET_FTC_SIGNALS function in etfomm.dll to retrieve operating parameters for fixed-time control signals.

Syntax

int GetFTCSignals(FTC_DATA *ftc_signal_data);

Parameters

ftc_data_inputs[out]

Points to an array of FTC_DATA variables that contains operating parameters for fixed-time control signals: approach links, number of intervals, duration of intervals, offset, and signal codes for each interval.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetACSignals

Calls GET_AC_SIGNALS function in etfomm.dll to retrieve operating parameters for actuated control signals.

Syntax

int GetACSignals(AC_DATA *ac_signal_data);

Parameters

ac_data_inputs[out]

Points to an array of AC_DATA variables that contains operating parameters for actuated control signals: approach links, phase inputs, detectors.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetRampmeters

Calls GET_RAMPMETERS function in etfomm.dll to retrieve operating parameters for ramp meters.

Syntax

int GetRampmeters(RM_DATA *rampmeter_data);

Parameters

rampmeter_inputs[out]

Points to an array of RM_DATA variables that contains operating parameters for ramp meters: type of metering control, onset time, headway for clock-time metering, capacity for demand/capacity metering, speed thresholds for speed control metering, detectors used for metering, detector update interval and two-per-green setting.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetFreewayDetectorData

Calls GET_FREEWAY_DETECTORS function in etfomm.dll to retrieve operating parameters for freeway detectors.

Syntax

void GetFreewayDetectorData(DETECTOR_DATA *Freeway_Detector_Inputs);

Parameters

fdet_inputs[out]

Points to an array of DETECTOR_DATA variables that contains operating parameters for freeway detectors: ID, link, lanes covered, location on the link, length of the sensing area, carryover time, delay time and operation code.

GetStreetDetectorData

Calls GET_STREET_DETECTORS function in etfomm.dll to retrieve operating parameters for street detectors.

Syntax

void GetStreetDetectorData(DETECTOR_DATA *Street_Detector_Inputs);

Parameters

fdet_inputs[out]

Points to an array of DETECTOR_DATA variables that contains operating parameters for street detectors: ID, link, lanes covered, location on the link, length of the sensing area, carryover time, delay time and operation code.

GetBusroutes

Calls GET_BUSROUTES function in etfomm.dll to retrieve operating parameters for bus routes.

Syntax

int GetBusroutes(BUSROUTE_DATA *busroute_data);

Parameters

busroute_inputs[out]

Points to an array of BUSROUTE_DATA variables that contains operating parameters for bus routes: route number, offset, headway, links in route and stations along the route.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetBusstations

Calls GET_BUSSTATIONS function in etfomm.dll to retrieve operating parameters for bus stations.

Syntax

int GetBusstations(BUSSTATION_DATA *busstation_data);

Parameters

busstation_inputs[out]

Points to an array of 99 BUSSTATION_DATA variables that contains operating parameters for bus stations: link, location on link, blocking code, capacity, mean dwell time and bypass percentage.

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetEvents

Calls GET_EVENTS function in etfomm.dll to retrieve operating parameters for blockage events.

Syntax

void GetEvents(EVENT_DATA *Event_Inputs);

Parameters

Event_Inputs[out]

Points to an array of EVENT_DATA variables that contains operating parameters for blockage events: link, lane affected, start time, end time and type of event.

GetParkingZones

Calls GET_PARKING_ZONES function in etfomm.dll to retrieve operating parameters for parking zones.

Syntax

void GetParkingZones(PARKING_DATA *Parking_Zone_Inputs);

Parameters

Parking_Zone_Inputs[out]

Points to an array of PARKING_DATA variables that contains operating parameters for parking zones: frequency and mean duration of parking maneuvers, began and end location of parking zone.

GetIncidents

Calls GET_INCIDENTS function in etfomm.dll to retrieve operating parameters for freeway incidents.

Syntax

void GetIncidents(INCIDENT_DATA *incident_data_inputs);

Parameters

incident_data_inputs[out]

Points to an array of INCIDENT_DATA variables that contains operating parameters for freeway incidents: link, began and end location, begin and end time, rubbernecking factor, and type of incident.

GetDiversions

Calls GET_DIVERSIONS function in etfomm.dll to retrieve operating parameters for freeway diversions.

Syntax

void GetDiversions(DIVERSION_DATA *Diversion_Inputs);

Parameters

Diversion_Inputs[out]

Points to an array of DIVERSION_DATA variables that contains operating parameters for freeway diversions: link, location of warning sign, begin and end time, diversion route path ID, percentage of vehicles that will obey the diversion order and the speed at which vehicles should exit the freeway.

GetNodeCoordinates

Calls GET_NODE_COORDINATES function in etfomm.dll to retrieve node X- and Y-coordinates.

Syntax

void GetNodeCoordinates(XY_COORD *xy_coord_inputs);

Parameters

xy_coord_inputs[out]

Points to an array of XY_COORD variables that contains node X- and Y-coordinates.

GetConditionalTurnpcts

Calls GET_CONDITIONAL_TURNPERCENTAGES function in etfomm.dll to retrieve conditional turn percentages for street links.

Syntax

void GetConditionalTurnpcts(COND_TURNPCTS *cond_turnpct_data);

Parameters

cond_turnpct_data[out]

Points to an array of COND_TURNPCTS variables that contains conditional turn percentages for street links: left, thru, right and diagonal exit percentages for vehicles that entered the link via a left turn, left, thru, right and diagonal exit percentages for vehicles that entered the link via a thru movement, left, thru, right and diagonal exit percentages for vehicles that entered the link via a right turn, and left, thru, right and diagonal exit percentages for vehicles that entered the link via a diagonal movement.

GetFVehicle

Calls GET_FVEHICLE_STRUCT function in etfomm.dll to retrieve properties of freeway vehicles in network.

Syntax

int GetFVehicle(VFData *vfdata);

Parameters

vfdata[out]

Points to an array of VFData variables that contains properties of freeway vehicles in network: driver type, distance to upstream node, speed, acceleration or deceleration rate, desired speed and etc..

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

GetSVehicle

Calls GET_SVEHICLE_STRUCT function in etfomm.dll to retrieve the properties of street vehicles in network.

Syntax

int GetSVehicle(VSData *vsdata);

Parameters

vsdata[out]

Points to an array of VSData variables that contains the properties of street vehicles in network: driver type, distance to upstream node, speed, acceleration or deceleration rate, desired speed and etc..

Return Value

Returns 0 if the function succeeds. If the function fails, the return value is error code from etfomm.

etRunner

WCF_to_HOST_network_input_data

Converts the array that specifies the network properties from WCF Server data type to etfomm acceptable data type.

Syntax

```
NETWORK_INPUTS WCF_to_HOST_network_input_data(
        array<WCF_NETWORK_INPUTS>^ wcf_network_inputs
);
```

Parameters

wcf_network_inputs[in]

Points to an array of variables with type of WCFServer::WCF_NETWORK_INPUTS.

Return Value

Returns a variable with type of etRunner::NETWORK_INPUTS.

WCF_to_HOST_freeway_network_input_data

Converts the array that specifies the properties of freeway network from WCF Server data type to etfomm acceptable data type.

Syntax

```
FREEWAY_NETWORK_INPUTS WCF_to_HOST_freeway_network_input_data(
        array<WCF_FREEWAY_NETWORK_INPUTS>^ wcf_freeway_network_inputs
);
```

Parameters

wcf_freeway_network_inputs[in]

Points to an array of variables with type of WCFServer::WCF_FREEWAY_NETWORK_INPUTS.

Return Value

A variable with type of etRunner::FREEWAY_NETWORK_INPUTS.

WCF_to_HOST_street_network_input_data

Converts the array that specifies the properties of surface street network from WCF Server data type to etfomm acceptable data type.

Syntax

```
STREET_NETWORK_INPUTS WCF_to_HOST_street_network_input_data(
        array<WCF_STREET_NETWORK_INPUTS>^ wcf_street_network_inputs
);
```

Parameters

wcf_street_network_inputs[in]

Points to an array of variables with type of WCFServer::WCF_STREET_NETWORK_INPUTS.

Return Value

A variable with type of etRunner::STREET_NETWORK_INPUTS.

WCF_to_HOST_Vehicle_Type_Inputs

Converts the array that specifies the properties of vehicle types from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_Vehicle_Type_Inputs(
        VEHICLE_TYPE_DATA* Vehicle_Type_Inputs,
        array<WCF_VEHICLE_TYPE_DATA>^ wcf_vdi
);
```

Parameters

Vehicle_Type_Inputs[out]

Points to an array of variables with type of etRunner::VEHICLE_TYPE_DATA.

wcf_vdi[in]

Points to an array of variables with type of WCFServer::WCF_STREET_NETWORK_INPUTS.

WCF_to_HOST_freeway_link_data

Converts the array that specifies the properties of freeway links from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_freeway_link_data(
    FREEWAY_LINK* freeway_link_data,
    array<Wcf_freeway_link>^ wcf_fwl
);
```

Parameters

freeway_link_data[out]

Points to an array of variables with type of etRunner::FREEWAY_LINK.

wcf_fwl[in]

Points to an array of variables with type of WCFServer::Wcf_freeway_link.

WCF_to_HOST_street_link_data

Converts the array that specifies the properties of street links from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_street_link_data(
    STREET_LINK* street_link_data,
    array<Wcf_street_link>^ wcf_sl
);
```

Parameters

street_link_data[out]

Points to an array of variables with type of etRunner::STREET_LINK.

wcf_sl[in]

Points to an array of variables with type of WCFServer::Wcf_street_link.

WCF_to_HOST_entry_node_data

Converts the array that specifies the properties of entry nodes from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_entry_node_data(
    ENTRYNODES_DATA* entrynode_inputs,
    array <WCF_ENTRYNODES_DATA> ^ wcf_entry_node
);
```

Parameters

entrynode_inputs[out]

Points to an array of variables with type of etRunner::ENTRYNODES_DATA.

wcf_entry_node[in]

Points to an array of variables with type of WCFServer::WCF_ENTRYNODES_DATA.

WCF_to_HOST_ftc_data_inputs

Converts the array that specifies the operating parameters for fixed-time control signals from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_ftc_data_inputs(
        FTC_DATA* ftc_data_inputs,
        array<WCF_FTC_DATA>^ wcf_ftcs
);
```

Parameters

ftc_data_inputs[out]

Points to an array of variables with type of etRunner::FTC_DATA.

wcf_ftcs[in]

Points to an array of variables with type of WCFServer::WCF_FTC_DATA.

WCF_to_HOST_ac_data_inputs

Converts the array that specifies the operating parameters for actuated control signals from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_ac_data_inputs(
        AC_DATA* ac_data_inputs,
        array<WCF_AC>^ wcf_acl
);
```

Parameters

ac_data_inputs[out]

Points to an array of variables with type of etRunner::AC_DATA.

wcf_acl[in]

Points to an array of variables with type of WCFServer::WCF_AC.

WCF_to_HOST_rampmeter_inputs

Converts the array that specifies the operating parameters for ramp meters from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_rampmeter_inputs(
        RM_DATA* rampmeter_inputs,
        array<WCF_RM_DATA>^ wcf_rampmeter_inputs
);
```

Parameters

rampmeter_inputs[out]

Points to an array of variables with type of etRunner::RM_DATA.

wcf_rampmeter_inputs[in]

Points to an array of variables with type of WCFServer::WCF_RM_DATA.

WCF_to_HOST_det_inputs

Converts the array that specifies the operating parameters for detectors from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_det_inputs(
        DETECTOR_DATA* det_inputs,
        array<WCF_DETECTOR_DATA>^ wcf_det_inputs
);
```

Parameters

det_inputs[out]

Points to an array of variables with type of etRunner::DETECTOR_DATA.

wcf_det_inputs[in]

Points to an array of variables with type of WCFServer::WCF_DETECTOR_DATA.

WCF_to_HOST_busroute_inputs

Converts the array that specifies the operating parameters for bus routes from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_busroute_inputs(
        BUSROUTE_DATA* busroute_inputs,
        array<WCF_BUSROUTE_DATA>^ wcf_busroute_inputs
);
```

Parameters

busroute_inputs[out]

Points to an array of variables with type of etRunner::BUSROUTE_DATA.

wcf_busroute_inputs[in]

Points to an array of variables with type of WCFServer::WCF_BUSROUTE_DATA.

WCF_to_HOST_busstation_inputs

Converts the array that specifies the operating parameters for bus stations from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_busstation_inputs(
        BUSSTATION_DATA* busstation_inputs,
        array<WCF_BUSSTATION_DATA>^ wcf_busstation_inputs
);
```

Parameters

busstation_inputs[out]

Points to an array of variables with type of etRunner::BUSSTATION_DATA.

wcf_busstation_inputs[in]

Points to an array of variables with type of WCFServer::WCF_BUSSTATION_DATA.

WCF_to_Host_event_data_inputs

Converts the array that specifies the operating parameters for blockage events from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_Host_event_data_inputs(
        EVENT_DATA* event_data_inputs,
        array<WCF_EVENT_DATA>^ wcf_event_data_inputs
);
```

Parameters

event_data_inputs[out]

Points to an array of variables with type of etRunner::EVENT_DATA.

wcf_event_data_inputs[in]

Points to an array of variables with type of WCFServer::WCF_EVENT_DATA.

WCF_to_Host_parking_data_inputs

Converts the array that specifies the operating parameters for parking zones from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_Host_parking_data_inputs(
        PARKING_DATA* parking_data_inputs,
        array<WCF_PARKING_DATA>^ wcf_parking_data_inputs
);
```

Parameters

parking_data_inputs[out]

Points to an array of variables with type of etRunner::PARKING_DATA.

wcf_parking_data_inputs[in]

Points to an array of variables with type of WCFServer::WCF_PARKING_DATA.

WCF_to_Host_incident_data_inputs

Converts the array that specifies the operating parameters for freeway incidents from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_Host_incident_data_inputs(
        INCIDENT_DATA* incident_data_inputs,
        array<WCF_INCIDENT_DATA>^ wcf_incident_data_inputs
);
```

Parameters

incident_data_inputs[out]

Points to an array of variables with type of etRunner::INCIDENT_DATA.

wcf_incident_data_inputs[in]

Points to an array of variables with type of WCFServer::WCF_INCIDENT_DATA.

WCF_to_Host_diversion_data_inputs

Converts the array that specifies the operating parameters for freeway diversions from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_Host_diversion_data_inputs(
        DIVERSION_DATA* diversion_data_inputs,
        array<WCF_DIVERSION_DATA>^ wcf_diversion_data_inputs
);
```

Parameters

diversion_data_inputs[out]

Points to an array of variables with type of etRunner::DIVERSION_DATA.

wcf_diversion_data_inputs[in]

Points to an array of variables with type of WCFServer::WCF_DIVERSION_DATA.

WCF_to_Host_diversion_data_inputs

Converts the array that specifies node X- and Y-coordinates from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_xy_coord_inputs(
        XY_COORD* xy_coord_inputs,
        array<WCF_XY_COORD>^ wcf_xy_coord_inputs
);
```

Parameters

xy_coord_inputs[out]

Points to an array of variables with type of etRunner::XY_COORD.

wcf_xy_coord_inputs[in]

Points to an array of variables with type of WCFServer::WCF_XY_COORD.

WCF_to_HOST_cond_turnpct_data

Converts the array that specifies conditional turn percentages for street links from WCF Server data type to etfomm acceptable data type.

Syntax

```
void WCF_to_HOST_cond_turnpct_data(
        COND_TURNPCTS* cond_turnpct_data,
        array<WCF_COND_TURNPCTS>^ wcf_cond_turnpct_data
);
```

Parameters

cond_turnpct_data[out]

Points to an array of variables with type of etRunner::COND_TURNPCTS.

wcf_cond_turnpct_data[in]

Points to an array of variables with type of WCFServer::WCF_COND_TURNPCTS.

Processing Functions

These functions retrieve corresponding component values from etfomm, print them to text files if print flag is set, and upload the values to WCF Server.

Syntax

```
void ProcessNetworkInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessFNetworkInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessSNetworkInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessVTypeInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessFreewayLinksData(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessStreetLinksData(std::ofstream &outputFile, IService1 ^proxy, bool *change_phase_flag, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessEntryNodes(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessFTCSignals(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessRampMeter(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessFDetectorInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessSDetectorInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessBusRouteInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessBusStationInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessEvents(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
```

void ProcessParkingZones(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessIncidentInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessDiversionInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessCoordInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessCondTurnpctInputs(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessFVehicleData(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);
void ProcessSVehicleData(std::ofstream &outputFile, IService1 ^proxy, etFommInterface *etFommIF, int updatedFlag = 0);

Parameters

outputFile

Reference of the text file for output.

proxy

Points to the WCF Service instance.

etFommIF

Points to the etFommInterface instance.

updatedFlag

Indicates which component is updated. Default value of 0 means all components are updated.

change_phase_flag

This parameter is only for processing the properties of street links. It indicates whether to reset current phases for the actuated control signals on street links.

Updating Functions

These functions retrieve corresponding modified component values from WCF Server, print them to text files if print flag is set, and reset the values in etfomm.

Syntax

void UpdateNetworkInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateFNetworkInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateSNetworkInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateVTypeInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateFreewayLinkData(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateStreetLinkData(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateEntryNodes(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateFTCSignals(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateACSignals(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateRampMeters(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateFDetectorInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateSDetectorInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateBusRouteInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateBusStationInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateEvents(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateParkingZones(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateIncidentInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateDiversionInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);

void UpdateCoordInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateCondTurnpctInputs(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateFVehicles(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);
void UpdateSVehicles(std::ofstream &outputFile, IService1^ proxy, etFommInterface *etFommIF);

Parameters

outputFile

Reference of the text file for output.

proxy

Points to the WCF Service instance.

etFommIF

Points to the etFommInterface instance.

APIProcessClientRequest

Waits for API Client inputs, and calls appropriate updating function based on the specific API Client input.

Syntax

```
int APIProcessClientRequest(
        IService1^ proxy,
        etFommInterface *etFommIF,
        const std::string& dataPath,
        const std::string& fileName,
        int &updatedFlag,
        std::string &prefix
);
```

Parameters

Proxy[in]

Points to the WCF Service instance.

etFommIF[in]

Points to the etFommInterface instance.

dataPath[in]

Reference of a string to specify the path of the output file.

filename[in]

Reference of a string to specify the default name of the output file.

updatedFlag[out]

Reference of an int variable that indicates which component is updated. Default value of 0 means all components are updated.

prefix[out]

Reference of a string variable that indicates the prefix for the output file.

WCFServer

SetServerNetworkInput

Defines the array in WCF Server that specifies the network properties.

Syntax

void SetServerNetworkInput(array <WCF_NETWORK_INPUTS> ^ wcf_network_inputs);

Parameters

wcf_network_inputs

Points to an array of WCF_NETWORK_INPUTS that specifies the network properties: time period durations, time interval and a flag to skip initialization.

GetServerNetworkInput

Retrieves the array in WCF Server that specifies the network properties.

Syntax

array <WCF_NETWORK_INPUTS>^ GetServerNetworkInput();

Return Value

Return a pointer to an array of WCF_NETWORK_INPUTS that specifies the network properties: time period durations, time interval and a flag to skip initialization.

SetServerFreewayNetworkInput

Defines the array in WCF Server that specifies the properties of freeway networks.

Syntax

void SetServerFreewayNetworkInput(

array <WCF_FREEWAY_NETWORK_INPUTS> ^ wcf_freeway_network_inputs

);

Parameters

wcf_freeway_network_inputs

Points to an array of WCF_FREEWAY_NETWORK_INPUTS that specifies properties of freeway networks: lag to accelerate or decelerate, friction coefficient, default HOV utilization percentage, car following factors, desired speed distribution table, percentage of cooperative drivers, lane change duration, and multiplier for discretionary lane changes.

GetServerFreewayNetworkInput

Retrieves the array in WCF Server that specifies the properties of freeway networks.

Syntax

array <WCF_FREEWAY_NETWORK_INPUTS>^ GetServerFreewayNetworkInput();

Return Value

Return a pointer to an array of WCF_FREEWAY_NETWORK_INPUTS that specifies properties of freeway networks: lag to accelerate or decelerate, friction coefficient, default HOV utilization percentage, car following factors, desired speed distribution table, percentage of cooperative drivers, lane change duration, and multiplier for discretionary lane changes.

SetServerStreetNetworkInput

Defines the array in WCF Server that specifies the properties of surface street networks.

Syntax

void SetServerStreetNetworkInput(array <WCF_STREET_NETWORK_INPUTS> ^ wcf_street_network_inputs);

Parameters

wcf_street_network_inputs

Points to an array of WCF_STREET_NETWORK_INPUTS that specifies properties of surface street networks: acceptable gaps at signs, amber deceleration table, bus station dwell multiplier, car following factors,

desired speed distribution table, percentage of cooperative drivers, lane change duration, left turn jumper and lagger percentages, maximum left and right turning speeds, probability of joining spillback, speed at which a vehicle is considered stopped, speed for vehicles going through a yield sign, driver familiarity with path, multiplier distribution for startup lost time and queue discharge headway, pedestrian distributions, multiplier distribution for short term events.

GetServerStreetNetworkInput

Retrieves the array in WCF Server that specifies the properties of surface street networks.

Syntax

array <WCF_STREET_NETWORK_INPUTS>^ GetServerStreetNetworkInput();

Return Value

Return a pointer to an array of WCF_STREET_NETWORK_INPUTS that specifies properties of surface street networks: acceptable gaps at signs, amber deceleration table, bus station dwell multiplier, car following factors, desired speed distribution table, percentage of cooperative drivers, lane change duration, left turn jumper and lagger percentages, maximum left and right turning speeds, probability of joining spillback, speed at which a vehicle is considered stopped, speed for vehicles going through a yield sign, driver familiarity with path, multiplier distribution for startup lost time and queue discharge headway, pedestrian distributions, multiplier distribution for short term events.

SetServerVehicleTypeInputs

Defines the array in WCF Server that specifies the properties of vehicle types.

Syntax

void SetServerVehicleTypeInputs(array <WCF_VEHICLE_TYPE_DATA> ^ wcf_Vehicle_Type_Inputs);

Parameters

wcf_Vehicle_Type_Inputs

Points to an array of WCF_VEHICLE_TYPE_DATA that specifies properties of vehicle types: length, headway factor, average occupancy, emergency deceleration, distribution of types within fleets for freeway and street networks.

GetServerVehicleTypeInputs

Retrieves the array in WCF Server that specifies the properties of vehicle types.

Syntax

array <WCF_VEHICLE_TYPE_DATA>^ GetServerVehicleTypeInputs();

Return Value

Return a pointer to an array of WCF_VEHICLE_TYPE_DATA that specifies properties of vehicle types: length, headway factor, average occupancy, emergency deceleration, distribution of types within fleets for freeway and street networks.

SetServerFreewayData

Defines the array in WCF Server that specifies the properties of freeway links.

Syntax

void SetServerFreewayData(array <Wcf_freeway_link> ^ wcf_freeway_links);

Parameters

wcf_freeway_links

Points to an array of Wcf_freeway_link that specifies the properties of freeway links: upstream node, downstream node, length, number of full lanes, lane alignments, type of link, auxiliary lanes, added or dropped lanes, mean freeflow speed, car following multiplier, exit percentage, detectors, grade, tilt, curvature, pavement type, lane widths, vehicle-type exclusions, anticipatory lane change inputs, vehicle-type exit multipliers, mean startup time from

a rampmeter, HOV lane inputs, through receiving link, offramp receiving link, shoulder width, barriers and datastation settings.

GetServerFreewayData

Retrieves the array in WCF Server that specifies the properties of freeway links.

Syntax

array <Wcf_freeway_link>^ GetServerFreewayData();

Return Value

Return a pointer to an array of Wcf_freeway_link that specifies the properties of freeway links: upstream node, downstream node, length, number of full lanes, lane alignments, type of link, auxiliary lanes, added or dropped lanes, mean freeflow speed, car following multiplier, exit percentage, detectors, grade, tilt, curvature, pavement type, lane widths, vehicle-type exclusions, anticipatory lane change inputs, vehicle-type exit multipliers, mean startup time from a rampmeter, HOV lane inputs, through receiving link, offramp receiving link, shoulder width, barriers and datastation settings.

SetServerStreetData

Defines the array in WCF Server that specifies the properties of street links.

Syntax

void SetServerStreetData(array <Wcf_street_link> ^ wcf_street_links);

Parameters

wcf_street_links

Points to an array of Wcf_street_link that specifies the properties of street links: upstream node, downstream node, length, number of full lanes, receiving links, number of left and right pocket lanes, length of left and right pocket lanes, mean freeflow speed, lane alignments, car following multiplier, sight distance, right turn on red code, pedestrian code, grade, vehicle-type exclusions, turn percentages, vehicle-type specific turn multipliers, and the width of the intersection at the upstream end of the link.

GetServerStreetData

Retrieves the array in WCF Server that specifies the properties of street links.

Syntax

array <Wcf_street_link>^ GetServerStreetData();

Return Value

Return a pointer to an array of Wcf_street_link that specifies the properties of street links: upstream node, downstream node, length, number of full lanes, receiving links, number of left and right pocket lanes, length of left and right pocket lanes, mean freeflow speed, lane alignments, car following multiplier, sight distance, right turn on red code, pedestrian code, grade, vehicle-type exclusions, turn percentages, vehicle-type specific turn multipliers, and the width of the intersection at the upstream end of the link.

SetServerEntryNodeData

Defines the array in WCF Server that specifies the properties of entry nodes.

Syntax

void SetServerEntryNodeData(array <WCF_ENTRYNODES_DATA> ^ wcf_entry_nodes);

Parameters

wcf_entry_nodes

Points to an array of WCF_ENTRYNODES_DATA that specifies the properties of entry nodes: flowrate, truck percentage, carpool percentage, percentage of HOV lane violators, lane distribution.

GetServerEntryNodeData

Retrieves the array in WCF Server that specifies the properties of entry nodes.

Syntax

array <WCF_ENTRYNODES_DATA>^ GetServerEntryNodeData();

Return Value

Return a pointer to an array of WCF_ENTRYNODES_DATA that specifies the properties of entry nodes: flowrate, truck percentage, carpool percentage, percentage of HOV lane violators, lane distribution.

SetServerFTCSignalData

Defines the array in WCF Server that specifies the properties of operating parameters for fixed-time control signals.

Syntax

void SetServerFTCSignalData(array<WCF_FTC_DATA> ^ wcf_ftc);

Parameters

wcf_ftc

Points to an array of WCF_FTC_DATA that specifies the operating parameters for fixed-time control signals: approach links, number of intervals, duration of intervals, offset, and signal codes for each interval.

GetServerFTCSignalData

Retrieves the array in WCF Server that specifies the properties of operating parameters for fixed-time control signals.

Syntax

array <WCF_FTC_DATA>^ GetServerFTCSignalData();

Return Value

Return a pointer to an array of WCF_FTC_DATA that specifies the operating parameters for fixed-time control signals: approach links, number of intervals, duration of intervals, offset, and signal codes for each interval.

SetServerACData

Defines the array in WCF Server that specifies the properties of operating parameters for actuated control signals.

Syntax

void SetServerACData(array <WCF_AC>^ wcf_acl);

Parameters

wcf_acl

Points to an array of WCF_AC that specifies the operating parameters for actuated control signals: approach links, phase inputs, detectors.

GetServerACData

Retrieves the array in WCF Server that specifies the properties of operating parameters for actuated control signals.

Syntax

array <WCF_AC>^ GetServerACData();

Return Value

Return a pointer to an array of WCF_AC that specifies the operating parameters for actuated control signals: approach links, phase inputs, detectors.

SetServerRampmeterInputs

Defines the array in WCF Server that specifies the properties of operating parameters for ramp meters.

Syntax

void SetServerRampmeterInputs(array <WCF_RM_DATA>^ wcf_rampmeter_inputs);

Parameters

wcf_rampmeter_inputs

Points to an array of WCF_RM_DATA that specifies the operating parameters for ramp meters: type of metering control, onset time, headway for clock-time metering, capacity for demand/capacity metering, speed thresholds for speed control metering, detectors used for metering, detector update interval and two-per-green setting.

GetServerRampmeterInputs

Retrieves the array in WCF Server that specifies the properties of operating parameters for ramp meters.

Syntax

array <WCF_RM_DATA>^ GetServerRampmeterInputs();

Return Value

Return a pointer to an array of WCF_RM_DATA specifies the operating parameters for ramp meters: type of metering control, onset time, headway for clock-time metering, capacity for demand/capacity metering, speed thresholds for speed control metering, detectors used for metering, detector update interval and two-per-green setting.

SetServerFreewayDetectorInputs

Defines the array in WCF Server that specifies the properties of operating parameters for freeway detectors.

Syntax

void SetServerFreewayDetectorInputs(array <WCF_DETECTOR_DATA>^ wcf_fdet_inputs);

Parameters

wcf_fdet_inputs

Points to an array of WCF_DETECTOR_DATA that specifies the operating parameters for freeway detectors: ID, link, lanes covered, location on the link, length of the sensing area, carryover time, delay time and operation code.

GetServerFreewayDetectorInputs

Retrieves the array in WCF Server that specifies the properties of operating parameters for freeway detectors.

Syntax

array <WCF_DETECTOR_DATA>^ GetServerFreewayDetectorInputs();

Return Value

Return a pointer to an array of WCF_DETECTOR_DATA that specifies the operating parameters for freeway detectors: ID, link, lanes covered, location on the link, length of the sensing area, carryover time, delay time and operation code.

SetServerStreetDetectorInputs

Defines the array in WCF Server that specifies the properties of operating parameters for street detectors.

Syntax

void SetServerStreetDetectorInputs(array <WCF_DETECTOR_DATA>^ wcf_sdet_inputs);

Parameters

wcf_sdet_inputs

Points to an array of WCF_DETECTOR_DATA that specifies the operating parameters for street detectors: ID, link, lanes covered, location on the link, length of the sensing area, carryover time, delay time and operation code.

GetServerStreetDetectorInputs

Retrieves the array in WCF Server that specifies the properties of operating parameters for street detectors.

Syntax

array <WCF_DETECTOR_DATA>^ GetServerStreetDetectorInputs();

Return Value

Return a pointer to an array of WCF_DETECTOR_DATA that specifies the operating parameters for street detectors: ID, link, lanes covered, location on the link, length of the sensing area, carryover time, delay time and operation code.

SetServerBusRouteInputs

Defines the array in WCF Server that specifies the properties of operating parameters for bus routes.

Syntax

void SetServerBusRouteInputs(array <WCF_BUSROUTE_DATA>^ wcf_busroute_inputs);

Parameters

wcf_busroute_inputs

Points to an array of WCF_BUSROUTE_DATA that specifies the operating parameters for bus routes: route number, offset, headway, links in route and stations along the route.

GetServerBusRouteInputs

Retrieves the array in WCF Server that specifies the properties of operating parameters for bus routes.

Syntax

array <WCF_BUSROUTE_DATA>^ GetServerBusRouteInputs();

Return Value

Return a pointer to an array of WCF_BUSROUTE_DATA that specifies the operating parameters for bus routes: route number, offset, headway, links in route and stations along the route.

SetServerBusStationInputs

Defines the array in WCF Server that specifies the properties of operating parameters for bus stations.

Syntax

void SetServerBusStationInputs(array <WCF_BUSSTATION_DATA>^ wcf_busstation_inputs);

Parameters

wcf_busstation_inputs

Points to an array of WCF_BUSSTATION_DATA that specifies the operating parameters for bus stations: link, location on link, blocking code, capacity, mean dwell time and bypass percentage.

GetServerBusStationInputs

Retrieves the array in WCF Server that specifies the properties of operating parameters for bus stations.

Syntax

array <WCF_BUSSTATION_DATA>^ GetServerBusStationInputs();

Return Value

Return a pointer to an array of WCF_BUSSTATION_DATA that specifies the operating parameters for bus stations: link, location on link, blocking code, capacity, mean dwell time and bypass percentage.

SetServerEventData

Defines the array in WCF Server that specifies the properties of operating parameters for blockage events.

Syntax

void SetServerEventData(array<WCF_EVENT_DATA>^ wcf_event_inputs);

Parameters

wcf_event_inputs

Points to an array of WCF_EVENT_DATA that specifies the operating parameters for blockage events: link, lane affected, start time, end time and type of event.

GetServerEventData

Retrieves the array in WCF Server that specifies the properties of operating parameters for blockage events.

Syntax

array<WCF_EVENT_DATA>^ GetServerEventData();

Return Value

Return a pointer to an array of WCF_EVENT_DATA that specifies the operating parameters for blockage events: link, lane affected, start time, end time and type of event.

SetServerParkingData

Defines the array in WCF Server that specifies the properties of operating parameters for parking zones.

Syntax

void SetServerParkingData(array<WCF_PARKING_DATA>^ wcf_parking_inputs);

Parameters

wcf_parking_inputs

Points to an array of WCF_PARKING_DATA that specifies the operating parameters for parking zones: frequency and mean duration of parking maneuvers, began and end location of parking zone.

GetServerParkingData

Retrieves the array in WCF Server that specifies the properties of operating parameters for parking zones.

Syntax

array<WCF_PARKING_DATA>^ GetServerParkingData();

Return Value

Return a pointer to an array of WCF_PARKING_DATA that specifies the operating parameters for parking zones: frequency and mean duration of parking maneuvers, began and end location of parking zone.

SetServerIncidentData_Inputs

Defines the array in WCF Server that specifies the properties of operating parameters for freeway incidents.

Syntax

void SetServerIncidentData_Inputs(array<WCF_INCIDENT_DATA>^ wcf_incident_data_inputs);

Parameters

wcf_incident_data_inputs

Points to an array of WCF_INCIDENT_DATA that specifies the operating parameters for freeway incidents: link, began and end location, begin and end time, rubbernecking factor, and type of incident.

GetServerIncidentData_Inputs

Retrieves the array in WCF Server that specifies the properties of operating parameters for freeway incidents.

Syntax

array<WCF_INCIDENT_DATA>^ GetServerIncidentData_Inputs();

Return Value

Return a pointer to an array of WCF_INCIDENT_DATA that specifies the operating parameters for freeway incidents: link, began and end location, begin and end time, rubbernecking factor, and type of incident.

SetServerDiversionData

Defines the array in WCF Server that specifies the properties of operating parameters for freeway diversions.

Syntax

void SetServerDiversionData(array<WCF_DIVERSION_DATA>^ wcf_diversion_inputs);

Parameters

wcf_diversion_inputs

Points to an array of WCF_DIVERSION_DATA that specifies the operating parameters for freeway diversions: link, location of warning sign, begin and end time, diversion route path ID, percentage of vehicles that will obey the diversion order and the speed at which vehicles should exit the freeway.

GetServerDiversionData

Retrieves the array in WCF Server that specifies the properties of operating parameters for freeway diversions.

Syntax

array<WCF_DIVERSION_DATA>^ GetServerDiversionData();

Return Value

Return a pointer to an array of WCF_DIVERSION_DATA that specifies the operating parameters for freeway diversions: link, location of warning sign, begin and end time, diversion route path ID, percentage of vehicles that will obey the diversion order and the speed at which vehicles should exit the freeway.

SetServerXYCoordInputs

Defines the array in WCF Server that specifies node X- and Y-coordinates.

Syntax

void SetServerXYCoordInputs(array <WCF_XY_COORD>^ wcf_xy_coord_inputs);

Parameters

wcf_xy_coord_inputs

Points to an array of WCF_XY_COORD that specifies node X- and Y-coordinates.

GetServerXYCoordInputs

Retrieves the array in WCF Server that specifies node X- and Y-coordinates.

Syntax

array <WCF_XY_COORD>^ GetServerXYCoordInputs();

Return Value

Return a pointer to an array of WCF_XY_COORD that specifies node X- and Y-coordinates.

SetServerCondTurnpctData

Defines the array in WCF Server that specifies conditional turn percentages for street links.

Syntax

void SetServerCondTurnpctData(array <WCF_COND_TURNPCTS> ^ wcf_cond_turnpct_data);

Parameters

wcf_cond_turnpct_data

Points to an array of WCF_COND_TURNPCTS that specifies define conditional turn percentages for street links: left, thru, right and diagonal exit percentages for vehicles that entered the link via a left turn, left, thru, right and diagonal exit percentages for vehicles that entered the link via a thru movement, left, thru, right and diagonal exit percentages for vehicles that entered the link via a right turn, and left, thru, right and diagonal exit percentages for vehicles that entered the link via a diagonal movement.

GetServerCondTurnpctData

Retrieves the array in WCF Server that specifies conditional turn percentages for street links.

Syntax

array <WCF_COND_TURNPCTS>^ GetServerCondTurnpctData();

Return Value

Return a pointer to an array of WCF_COND_TURNPCTS that specifies define conditional turn percentages for street links: left, thru, right and diagonal exit percentages for vehicles that entered the link via a left turn, left, thru, right and diagonal exit percentages for vehicles that entered the link via a thru movement, left, thru, right and diagonal exit percentages for vehicles that entered the link via a right turn, and left, thru, right and diagonal exit percentages for vehicles that entered the link via a diagonal movement.

SetNumberOfConnectedClients

Specifies the number of connected clients in WCF Server.

Syntax

void SetNumberOfConnectedClients(int count);

Parameters

count

Specified number of connected clients.

GetNumberOfConnectedClients

Retrieves the number of connected clients in WCF Server.

Syntax
int GetNumberOfConnectedClients();
Return Value
Return the number of connected clients in WCF Server.

SetClientState
Specifies the client state in WCF Server.
Syntax
void SetClientState(int s);
Parameters
s
Specified the client state.

GetClientState
Retrieves the client state in WCF Server.
Syntax
int GetClientState();
Return Value
Return the client state.

SetServerTimestep
Specifies the value of time step in WCF Server.
Syntax
void SetServerTimestep(float t);
Parameters
t
Specified the value of time step.

GetServerTimestep
Retrieves the value of time step in WCF Server.
Syntax
float GetServerTimestep();
Return Value
Return the value of time step.

SetAPITimestepInterval
Specifies the value in WCF Server that specifies the time step interval for etfomm to pause and wait for client inputs.
Syntax
void SetAPITimestepInterval(float s);
Parameters
s
Specified the value of the time step interval.

GetAPITimestepInterval
Retrieves the value in WCF Server that specifies the time step interval for etfomm to pause and wait for client inputs.
Syntax
float GetAPITimestepInterval();
Return Value
Return the value of the time step interval.

SetServerFVehicleData

Defines the array in WCF Server that specifies the properties of freeway vehicles.

Syntax

void SetServerFVehicleData(array<WCF_VFData>^ wcf_fveh);

Parameters

wcf_fveh

Points to an array of WCF_VFData variables that specifies the properties of freeway vehicles: driver type, distance to upstream node, speed, acceleration or deceleration rate, desired speed and etc..

GetServerFVehicleData

Retrieves the array in WCF Server that specifies the properties of freeway vehicles.

Syntax

array<WCF_VFData>^ GetServerFVehicleData();

Return Value

Return a pointer to an array of WCF_VFData variables that specifies the properties of freeway vehicles: driver type, distance to upstream node, speed, acceleration or deceleration rate, desired speed and etc..

SetServerSVehicleData

Defines the array in WCF Server that specifies the properties of street vehicles.

Syntax

void SetServerSVehicleData(array<WCF_VSData>^ wcf_sveh);

Parameters

wcf_sveh

Points to an array of WCF_VFData variables that specifies the properties of street vehicles: driver type, distance to upstream node, speed, acceleration or deceleration rate, desired speed and etc..

GetServerSVehicleData

Retrieves the array in WCF Server that specifies the properties of street vehicles.

Syntax

array<WCF_VSData>^ GetServerSVehicleData();

Return Value

Return a pointer to an array of WCF_VFData variables that specifies the properties of street vehicles: driver type, distance to upstream node, speed, acceleration or deceleration rate, desired speed and etc..

SetServerPathData

Defines the array in WCF Server that specifies the node IDs of one path.

Syntax

void SetServerPathData(array<int>^ nodes_list);

Parameters

nodes_list

Points to an array of int variables that specifies the node IDs of one path.

SetServerPathData

Retrieves the array in WCF Server that specifies the node IDs of one path.

Syntax

array <int> ^ GetServerPathData();

Return Value

Return a pointer to an array of int variables that specifies the node IDs of one path.

SetServerNewVehicleData

Defines the array in WCF Server that specifies the properties of new vehicles.

Syntax

void SetServerNewVehicleData(array<NewVehicle> ^ nv);

Parameters

nv

Points to an array of NewVehicle variables that specifies the properties of new vehicles: time, associated node, path ID, driver type, fleet type, vehicle type.

GetServerNewVehicleData

Retrieves the array in WCF Server that specifies the properties of new vehicles.

Syntax

array <NewVehicle> ^ GetServerNewVehicleData();

Return Value

Return a pointer to an array of NewVehicle variables that specifies the properties of new vehicles: time, associated node, path ID, driver type, fleet type, vehicle type.

SetServerSignalData

Defines the array in WCF Server that specifies the properties of signals.

Syntax

void SetServerSignalData(array<Signal> ^ sig);

Parameters

sig

Points to an array of Signal variables that specifies the properties of signals: node ID, interval, duration.

GetServerSignalData

Retrieves the array in WCF Server that specifies the properties of signals.

Syntax

array <Signal> ^ GetServerSignalData();

Return Value

Return a pointer to an array of Signal variables that specifies the properties of signals: node ID, interval, duration.