Traffic Simulator

Reference Manual

CSE 6730 Group B:

Mary Benage
Vladimir Coxall
Julian Diaz-Ospina
Rajendran Mohan

February 17, 2012

Georgia Institute of Technology

Table of Contents

Introduction

The Traffic Model Simulator is is a software package developed as an academic project for the CSE 6730 course at the Georgia Institute of Technology. This software helps in the analysis of delay times for the evacuation of vehicles from a number of parking lots and their delivery to certain destinations through a network of roads.

The Traffic Model Simulator reads the information about the model from a formatted text file (input.txt), and send the results either to the computer screen or to a file named “Outputfile.txt”. The following are the sections, the input file must have and examples to it.

The Traffic Model is the object which contains the whole model. The first section of the input file provides parameters for the Model and display options. This sections defines the total number of Destinations, Intersections, Parking Lots, and Road Segments in the Model. It will also provide information about the preferred output display and the status of the Trace and Check modes.

In Trace Mode, the model will print the time and the ID of a vehicle every time it enters or exits a Road Segment. in Check Mode, the model will issue a Vehicle to each Destination from each Parking Lot. This mode is used to verify each path in the Model.

```
# 1. TrafficModel
#  DestinationsCount IntersectionsCount ParkingLotsCount RoadSegmentCount
       3                 4                 3                 10

# 1.1 Display Options
#     Trace On(1)/Off(0),   Output File(0)/Screen(1),   CheckMode On(1)/Off(0)
          0                            0                     0
```

The model treats Destinations and Intersections as similar objects. They both are stored in the same list. Therefore, the numbering for both of them is shared. In order to prepare the model, the Destinations should be numbered first.

The required information for each Destination is the average time to complete the exit, assuming there is an exiting road to the final destination. i.e. exits to interstates.

The required information for each Intersection is the number of Phases, the number of Vehicle Queues going out of the Intersection, and the service time and number of Vehicle Queues at each Phase.

```
# 2. TrafficNodes
# Destinations and Intersections are TraffficNodes. They will share the numbering.
# Start numbering the Destinations then the Intersections

# 2.1 Destinations
#       Need one row per Destination
```

```
#       AverageCrossingTime (seconds)
# Node 0
         2.0
# Node 1
         3.0
# Node 2
         1.0

# 2.2 Intersections
#       Need one Block per Intersection

# Node 3
#       PhasesCount, QueuesOut
         1            2
#       Need one row per Phase
#       Service Time, QueuesIn
         60.0           1

# Node 4
#       PhasesCount, QueuesOut
         2              2
#       Need one row per Phase
#       Service Time,  QueuesIn
         100.0          1
         90.0           1

# Node 5
#       PhasesCount, QueuesOut
         2              2
#       Need one row per Phase
#       Service Time,  QueuesIn
         100.0          1
         90.0           2

# Node 6
#       PhasesCount, QueuesOut
         2              2
#       Need one row per Phase
#       Service Time,  QueuesIn
         100.0          1
         90.0           1
```

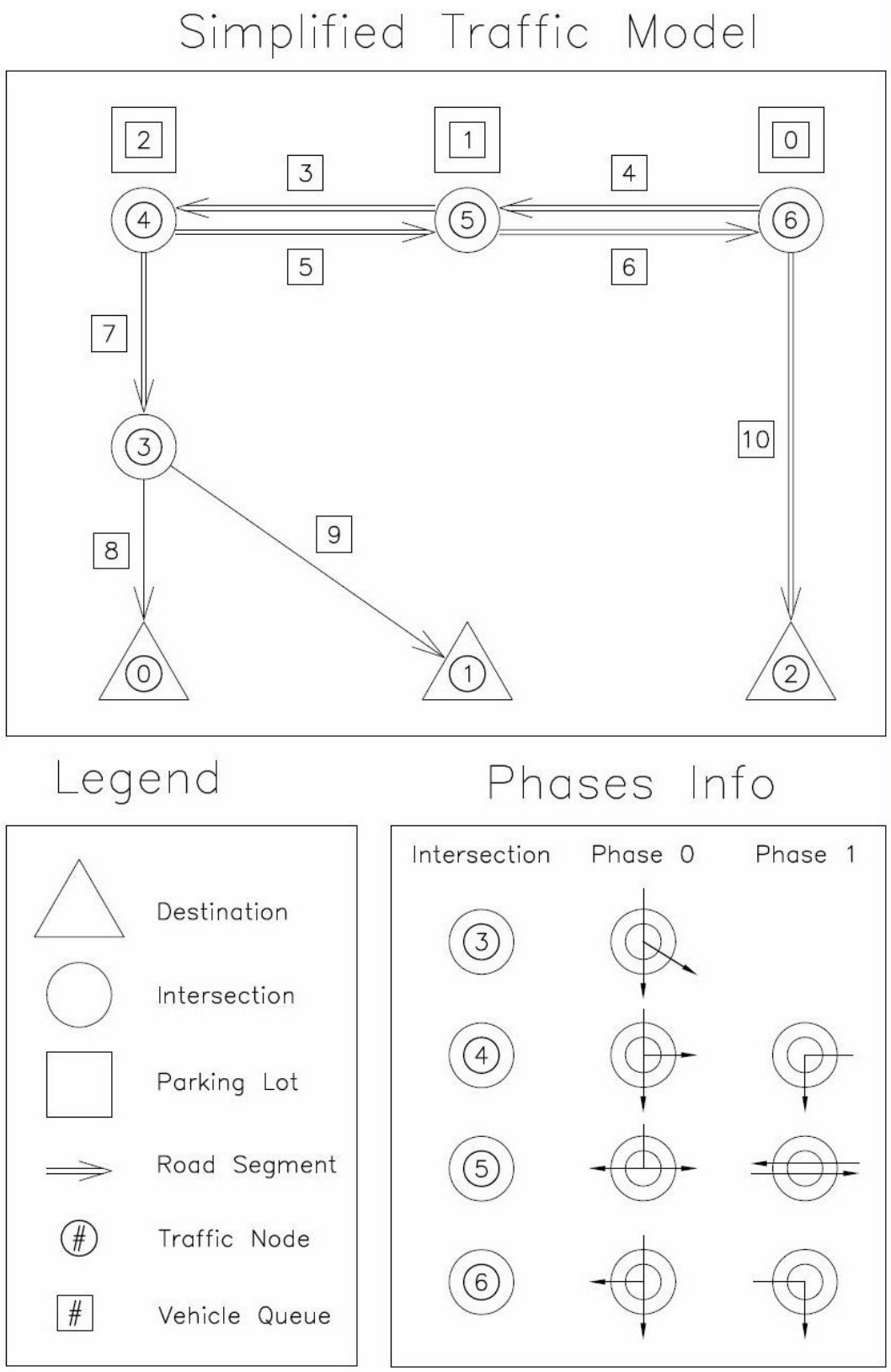
The model also treats Parking Lots and Road Segments as similar objects. They both are stored in the same list. Therefore, the numbering for both of them is shared. In order to prepare the model, the Parking Lots should be numbered first.

The required information for each Parking Lot is the total Capacity, the number of exits, the end Node (Intersection) to which the vehicles are delivered, the Phase which serves the traffic coming out of the Parking Lot, the number of Destinations the vehicles in the Parking Lot may seek, and pair values of Destination IDs and average crossing time through the Node (Intersection) given.

The required information for each Road Segment is the total Capacity, the average time traveling the segment road at average speed, the number of lanes, the start Node (Intersection) from which the vehicles are entering, the end Node to which the vehicles are delivered, the Phase which serves the traffic coming out of the Road Segment, the number of Destinations the Road Segment serves, and pair values of Destination IDs and average crossing time through the end Node given.

```
# 3. VehicleQueues
#     ParkingLots and RoadSegments  are VehicleQueues.
#     They will share the numbering.
#     Start numbering the ParkingLots then the RoadSegments

# 3.1 ParkingLots

#     Need one Line per ParkingLot
#  Capacity, Exits, EndNode, Phase, DestCount, (DestIDs,   AvgTime(s))
#  Vehicle Queue 0
    1000     1      6        0      3          0  2.0  1  2.0  2  3.0
# Vehicle Queue 1
    1800     2      5        0      3          0  3.0  1  3.0  2  3.0
# Vehicle Queue 2
    1100     1      4        0      3          0  3.0  1  3.0  2  2.0

# 3.2 RoadSegments

#       Need one Line per RoadSegment
#  Cap, AvCrTime, Lanes, StaNode, EndNode, Phase, DestCount, (DestIDs, AvgTime)
# Vehicle Queue 3
   50   5.75      2      5        4        1       2          0  3.0  1  3.0
# Vehicle Queue 4
  50   5.75       2     6        5         1       2         0  2.0  1  2.0
# Vehicle Queue 5
  50   5.75       2     4        5         1       1         2   2.0
# Vehicle Queue 6
  50   5.75       2     5        6         1       1         2   3.0
# Vehicle Queue 7
  30   3.45       2     4        3         0       2         0  1.5  1  1.5
# Vehicle Queue 8
  10  2.30        1     3        0         0       1         0  0.0
# Vehicle Queue 9
  20  4.60        1     3        1         0       1         1  0.0
# Vehicle Queue 10
  60  6.90        1     6        2         0       1         2  0.0
```

The simplified model used for the input file described above is is shown in the next figure.



The output file for the example above, with the first lines of the trace removed, is presented here:

```
V Rdy RS 9:    0_991_1 1628.22
V Out D 0:     0_983_0 1623.64
V Out D 0:     0_984_0 1623.65
V In RS 8:     0_993_0 1624.28
V Rdy RS 8:    0_993_0 1626.58
V Out D 0:     0_985_0 1624.35
V In RS 8:     0_994_0 1624.39
V Rdy RS 8:    0_994_0 1626.69
V Out D 0:     0_987_0 1624.43
V In RS 9:     0_995_1 1625.04
V Rdy RS 9:    0_995_1 1629.64
V In RS 9:     0_997_1 1625.12
V Rdy RS 9:    0_997_1 1629.72
V In RS 9:     0_999_1 1625.83
V Rdy RS 9:    0_999_1 1630.43
V Out D 0:     0_990_0 1625.87
V Out D 0:     0_993_0 1626.58
V Out D 0:     0_994_0 1626.69
V Out D 1:     0_988_1 1627.39
V Out D 1:     0_989_1 1627.47
V Out D 1:     0_991_1 1628.22
V Out D 1:     0_995_1 1629.64
V Out D 1:     0_997_1 1629.72
V Out D 1:     0_999_1 1630.43


Destination 0
    Vehicles  Served: 1259
    Cummulative Time: 1.01286e+006
    Average     Time: 804.493


Destination 1
    Vehicles  Served: 1330
    Cummulative Time: 1.08417e+006
    Average     Time: 815.167


Destination 2
    Vehicles  Served: 1311
    Cummulative Time: 964336
    Average     Time: 735.573
```

Libraries and Modules

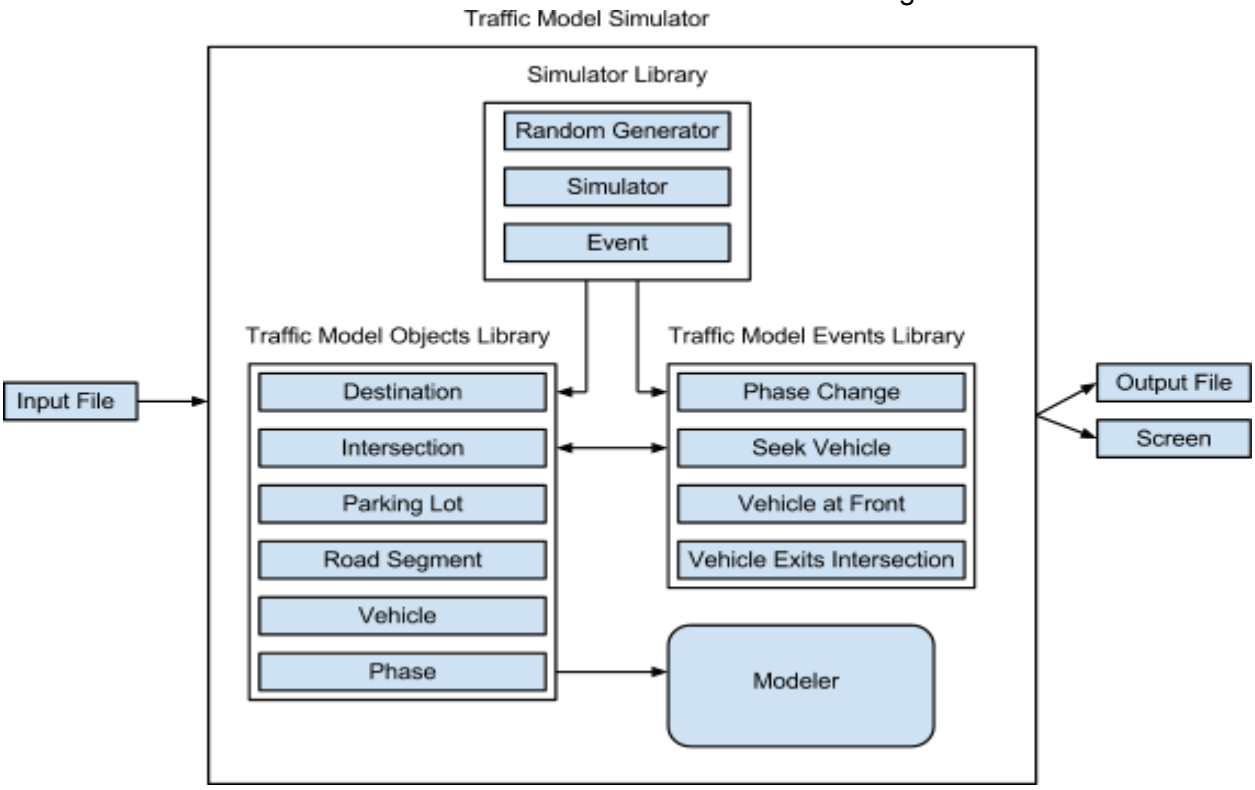
There are four components developed to complete the Traffic Model Simulator. The Modeler or driver program, the Simulator Library, the Traffic Model Objects Library, and the Traffic Model Events Library.

The Simulator Library provides the Random Generator, Linear and Calendar priority queues derived from the basic class Simulator, and the basic Event class. A detailed information about this library is presented later.

The Traffic Model Objects Library provides the Destination, Intersection, Parking Lot, Road Segment, Phase and Vehicle Objects to model the traffic network. A detailed information about this library is presented later.

The Traffic Model Events Library provides the Phase Change, Seek Vehicle, Vehicle at Front, and Vehicle Exits Intersection Events. A detailed information about this library is presented later.

The interaction between these modules is shown in the next figure.



The Modeler

The Modeler (driver program) uses the various TrafficModel objects to create a traffic environment. First, information about the environment in read in from a text file (input.txt). This file contains properties like the number and capacity of parking lots, intersections and service time, a topology of the road & intersection network, and more. The file contains inline comments prefixed with "#" for readability and debugging, describing different sections of the file.

Once the input file has been read in, the modeler creates a machine readable version (MRinput.txt) which is stripped of all comments for environment setup and execution. The modeler proceeds to read in all of these parameters, instantiating ParkingLots, Vehicles, Roads,

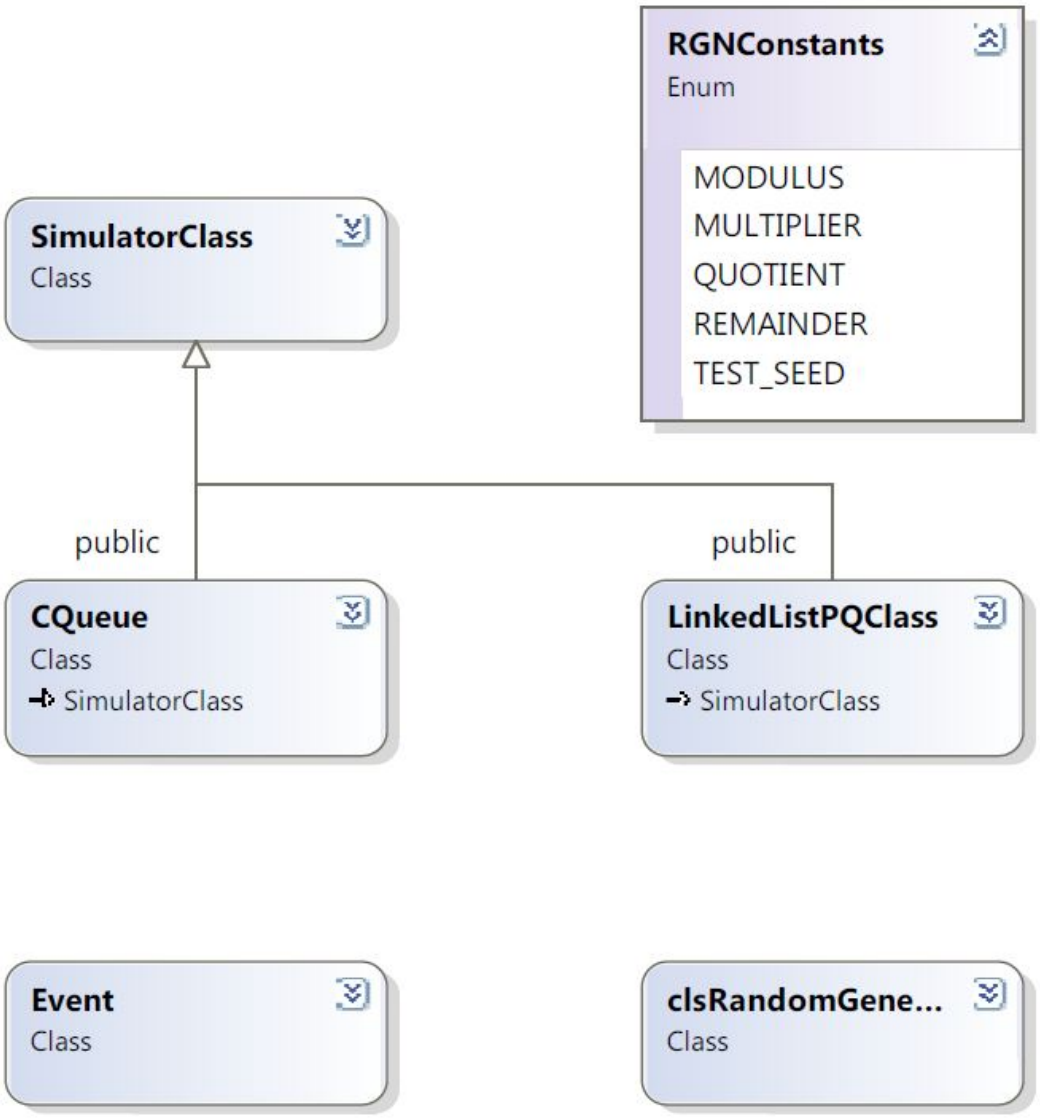
and Intersections with their appropriate properties. Between the Intersection properties and the Road properties, there is some data dependence; Each object needs information about the other class before the other class is instantiated. To solve this, a function called CheckModelConsistency analyzes the newly created environment and checks it against input file for errors.

Now, with a instantiated and verified environment, simulator and finite event list objects are created and also linked. The simulation is kick started by creating VehicleExit Events. After the simulation has completed, quantitative data such as average travel time and average delay are calculated across all the Exits and the results written to an output file.

Simulator Objects Library

The Simulator Objects Library provides the classes to perform the simulation of the model. The Simulator Class is the interface for any priority queue simulator. There are two implementations of priority queues in this library. A Linked List and a Calender Queue. A Random Number Generator is also implemented in this library. It also provides the Event Class as an interface for any event used with the simulators.

The hierarchy of these classes is shown in the following figure. See Appendix A, for a detailed reference of these classes.



Priority queues

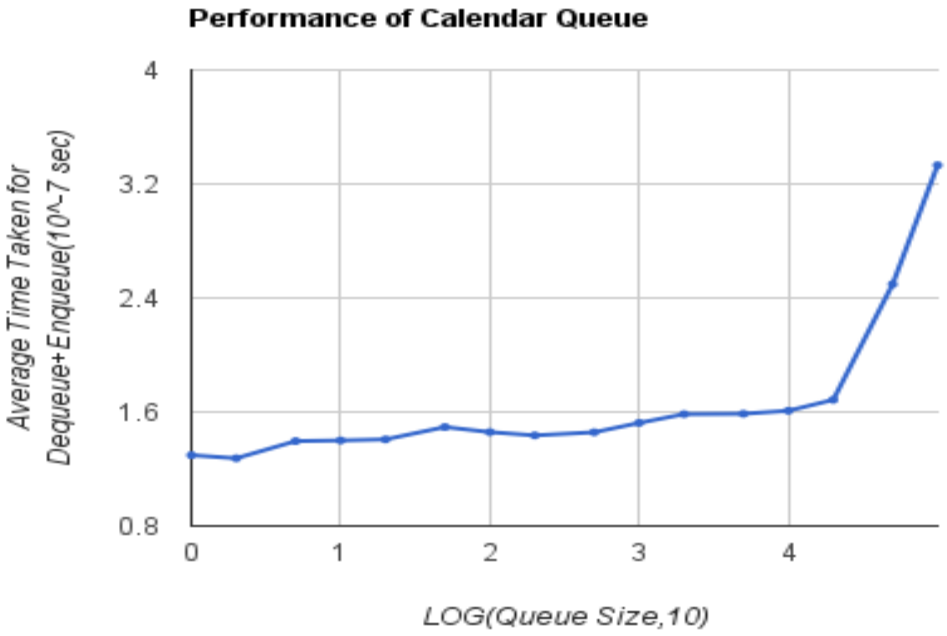
The priority queue used for the purpose of this simulation was of a calendar queue type. The reason that this type of queue was chosen is because of some core functionalities that we required for our code. They are as follows:-

- Ability to store a large number of events
- Ability to return the latest event as quick as possible
- Ability to remove any particular event(not necessarily the latest) from the queue

The last factor was the most instrumental in the creation of the calendar queue as it would allow us to target the location of an event and remove it quickly without the need to adjust the

remaining events.

The queue implemented was based on the paper by Brown(1988). Performance testing was done on the calendar system using the hold model outlined in the paper by Jones(1986) using exponentially distributed timings. Repeated tests were run on the calendar queue and the following timings were obtained from the system.



As can be seen, nearly constant time was maintained up till a queue size of about 20000 after which the timings have spiked. This could be attributed to cache effects since the required memory needs to be fetched from less efficient locations from the computer memory.

Random Generator

The random number generator used for this model was based off the Lehmer Algorithm, also known as the multiplicative congruential algorithm. It is a very commonly used algorithm to generate random numbers and has been well tested to provide a uniform distribution of numbers. The algorithm generates pseudorandom numbers which is desired for repeatability and debugging. The Lehmer Algorithm we used involved four integer parameters; the modulus, the multiplier, the quotient, and the remainder, which remain constant. We implemented the algorithm with values suggested by Lemmis and Park. The values are chosen so that the remainder is less than the quotient, which reduces overflow. The modulus is a well tested prime value and the multiplier is chosen so that it has a full-period, m-1. We also begin with an initial seed. The initial seed will generate the same sequence of numbers, even though the numbers appear random and are uniformly distributed. This is how the models can be debugged and repeated.

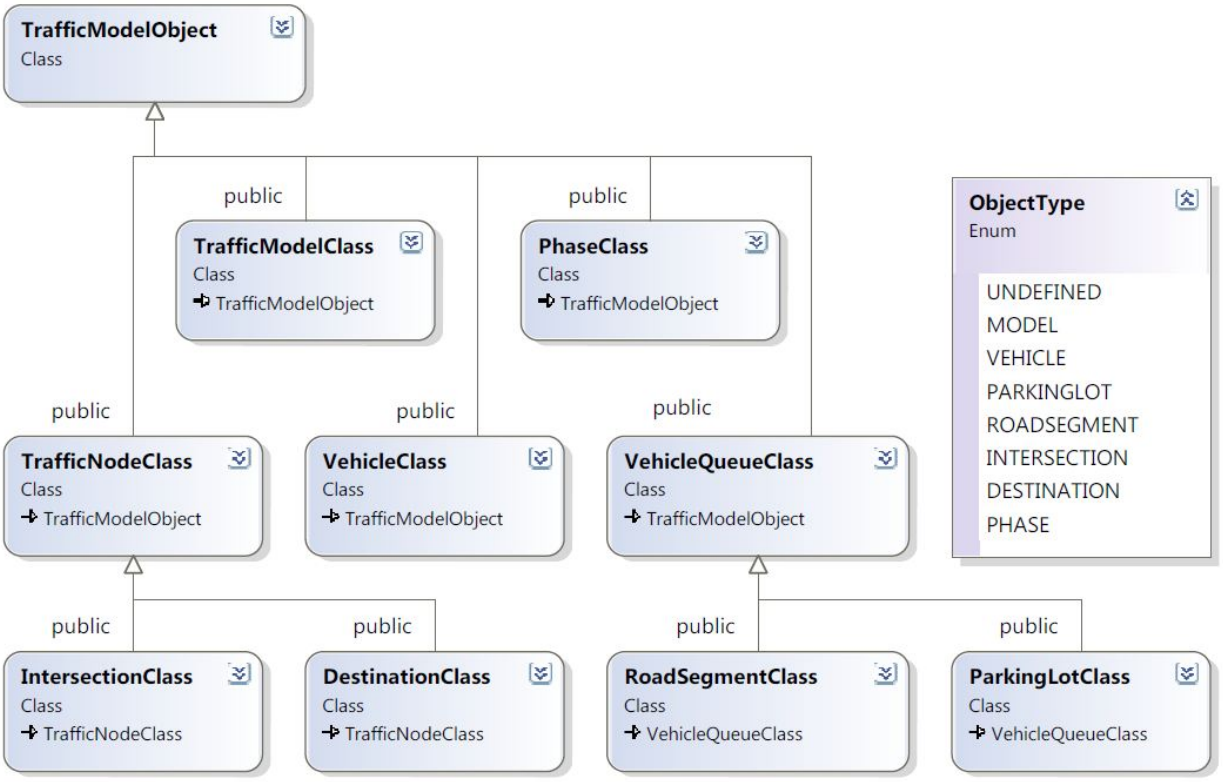
The testing of the Random Number Generator (RNG) was done using the Chi-Squared Test. This is essentially a probability test to determine how far off a value is from the expected, probable value. It is the sum of the squared difference between the calculated number of samples in a bin and the expected number, divided by the expected number. The values from the random number generator are binned into intervals dependent on a given bin size. The results from the Chi-Squared test for our implemented RNG where we used 101 bins so that our degree of freedom is 100 and the values for Chi squared to be less than the 0.1 level of significance or 118.50:

SEED	SAMPLE SIZE	BIN SIZE	Chi-Squared
123	1.00E+03	101	105
123	1.00E+05	101	76
200	5.02E+02	101	76
200	2.00E+03	101	70
1234	1.00E+03	101	70
123456	1.00E+03	101	95
123456	1.00E+06	101	72
123456	1.00E+08	101	69
123456789	1.00E+03	101	79
123456789	1.00E+10	101	11

Therefore we can conclude our Pseudo Random Number Generator generates numbers uniformly and has the appearance of being random.

Traffic Model Objects Library

The Traffic Model Objects Library provide the classes which represent the physical objects of the model. The hierarchy of these classes is shown in the following figure. See Appendix B, for a detailed reference of these classes.



Traffic Model Events Library

The Traffic Model Events Library provide the classes which represent the events in the model. There are basically five events implemented in this library. The Traffic Model Event is the interface for all of the events in the library and is derived from the Event Class of the Simulator Objects library.

The Phase Change Event is the first Event. This event changes the phase at a given time and schedule the Seek Vehicle event and the next Phase Change event.

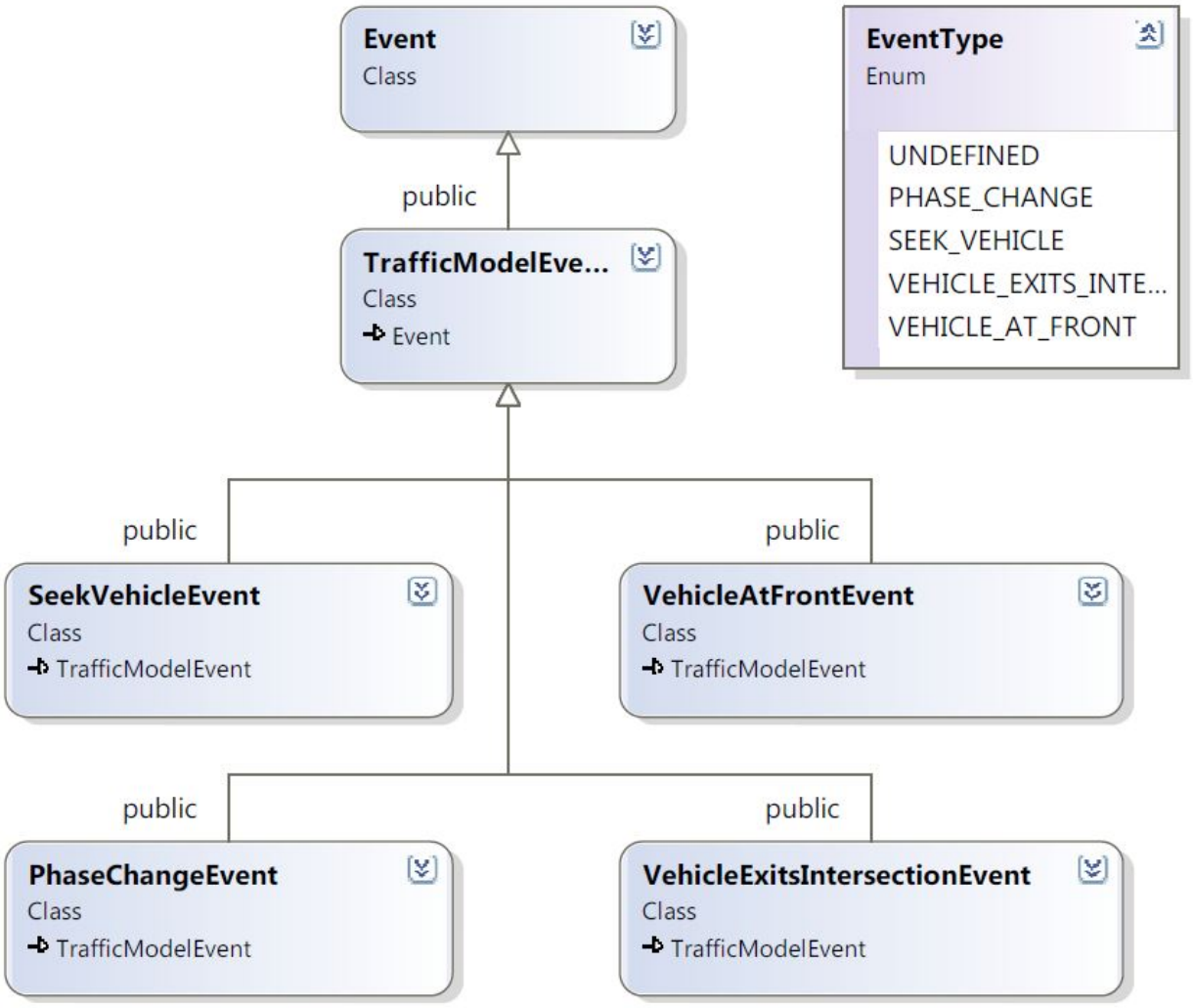
The Seek Vehicle event seeks for vehicles at Vehicle Queue In of the active phase, get a vehicle out of the vehicle queue in if the conditions are given. i.e. There is a vehicle ready to leave the Queue, and the Queue out is not full. This event schedules a Vehicle Exits Intersection event when a vehicle is pulled out of a Vehicle Queue in, and also schedules the next Seek Vehicle event if there are more vehicles ready in the Queue in.

The Vehicle Exits Intersection event passes the vehicle to the Queue out, and schedules a Seek Vehicle event when the intersection is empty. This event may also set the intersection to idle if there are not cars ready in any of the Vehicle Queues in.

The Vehicle at Front event triggers a Phase Change event when the conditions are met.

i.e. The Intersection is in idle mode.

The hierarchy of these classes is shown in the following figure. See Appendix B, for a detailed reference of these classes.



APPENDIX A

Simulator Objects Library Reference

SimulatorObjectsLibrary::clsRandomGenerator Class Reference

Random Generator Class. This object will provide random number uniformly distributed.
`#include <random.h>`

Public Member Functions

- **clsRandomGenerator** (void)
  *Initializes a new instance of the **clsRandomGenerator** class.*
- **clsRandomGenerator** (long Seed)
  *Initializes a new instance of the **clsRandomGenerator** class.*
- **~clsRandomGenerator** (void)
  *Finalizes an instance of the **clsRandomGenerator** class.*
- double **UniformDist** (double a, double b)
  Uniform distance.
- int **intUniformDist** (int a, int b)
  Int uniform distance.
- double **Chi_test** (int samples, int numbins)
  Tests the Random Number Generator using the Chi-test.

Protected Attributes

- long **t**
  The long to process.
- long **seed**
  The seed.

Detailed Description

Random Generator Class. This object will provide random number uniformly distributed.

Definition at line 22 of file random.h.

Constructor & Destructor Documentation

SimulatorObjectsLibrary::clsRandomGenerator::clsRandomGenerator (void) `[inline]`

Initializes a new instance of the **clsRandomGenerator** class.

Definition at line 41 of file random.h.

SimulatorObjectsLibrary::clsRandomGenerator::clsRandomGenerator (long *Seed*) `[inline]`

Initializes a new instance of the **clsRandomGenerator** class.

Parameters:

Seed	The seed.

Definition at line 46 of file random.h.

SimulatorObjectsLibrary::clsRandomGenerator::~clsRandomGenerator (void) `[inline]`

Finalizes an instance of the **clsRandomGenerator** class.

Definition at line 49 of file random.h.

Member Function Documentation

double SimulatorObjectsLibrary::clsRandomGenerator::Chi_test (int *samples*, int *numbins*)

Tests the Random Number Generator using the Chi-test.

Chi Test.

Parameters:

samples	input the number of samples, number of samples must be 1,000 or more.
numbins	input the number of bins, number of bins should be 100 or more.

Returns:

.The Chi value to compare to a table

Parameters:

samples	The number of samples.
numbins	The number of bins.

Returns:

.

Definition at line 50 of file random.cpp.

int SimulatorObjectsLibrary::clsRandomGenerator::intUniformDist (int *a*, int *b*)

Int uniform distance.

Produces a Random Number uniformily distributed. Assumes uniform distribution of values.

Parameters:

a	Lower bound of the interval.
b	Upper bound of the interval.

Returns:

.

Parameters:

a	Lower bound of the interval.
b	Upper bound of the interval.

Returns:

Random number of type int

Definition at line 36 of file random.cpp.

double SimulatorObjectsLibrary::clsRandomGenerator::UniformDist (double *a*, double *b*)

Uniform distance.

Produces a Random Number uniformily distributed. Assumes uniform distribution of values.

Parameters:

a	Lower bound of the interval.
b	Upper bound of the interval.

Returns:

.

Parameters:

a	Lower bound of the interval.
b	Upper bound of the interval.

Returns:

Random number of type double

Definition at line 100 of file random.cpp.

Member Data Documentation

long SimulatorObjectsLibrary::clsRandomGenerator::seed `[protected]`

The seed.

Definition at line 29 of file random.h.

long SimulatorObjectsLibrary::clsRandomGenerator::t `[protected]`

The long to process.

Definition at line 26 of file random.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**random.h**
- D:/Training/TM/**random.cpp**

SimulatorObjectsLibrary::Event Class Reference

Event.

`#include <Event.h>`

Inherited by **TrafficModelEventLibrary::TrafficModelEvent**.

Public Member Functions

- **Event** (void)
  *Initializes a new instance of the **Event** class.*
- **~Event** (void)
  *Finalizes an instance of the **Event** class.*
- double **EventTime** (void)
  Gets the event time.
- **Event** * **Next** (void)
  Gets the pointer to the next even in the linked list.
- void **SetNext** (**Event** *nextEvent)
  Set the pointer to the next event.
- virtual void **Run** (void)=0
  Runs this event.
- void **SetEventTime** (double **EventTime**)
  Sets event time.

Protected Attributes

- double **mEventTime**
  Time of the event.
- **Event** * **mNext**
  This member is used to facilitate the use of linked lists. This is a pointer to the next event in the linked list.

Detailed Description

Event.

Definition at line 9 of file Event.h.

Constructor & Destructor Documentation

SimulatorObjectsLibrary::Event::Event (void)

Initializes a new instance of the **Event** class.

Default constructor.

Definition at line 7 of file Event.cpp.

SimulatorObjectsLibrary::Event::~Event (void)

Finalizes an instance of the **Event** class.

Destructor.

Definition at line 13 of file Event.cpp.

Member Function Documentation

double SimulatorObjectsLibrary::Event::EventTime (void)

Gets the event time.

Returns:

.The event time.

Returns:

The event time.

Definition at line 18 of file Event.cpp.

Event * SimulatorObjectsLibrary::Event::Next (void)

Gets the pointer to the next even in the linked list.

Returns:

null if it there is not a next event, else a pointer to the linked list.

Returns:

null if it there is not a next event, else a pointer to the linked list.

Definition at line 26 of file Event.cpp.

virtual void SimulatorObjectsLibrary::Event::Run (void) `[pure virtual]`

Runs this event.

Implemented in **TrafficModelEventLibrary::VehicleExitsIntersectionEvent** (*p.**Error! Bookmark not defined.***), **TrafficModelEventLibrary::TrafficModelEvent** (*p.**Error! Bookmark not defined.***), **TrafficModelEventLibrary::SeekVehicleEvent** (*p.**Error! Bookmark not defined.***), **TrafficModelEventLibrary::VehicleAtFrontEvent** (*p.**Error! Bookmark not defined.***), and **TrafficModelEventLibrary::PhaseChangeEvent** (*p.**Error! Bookmark not defined.***).

void SimulatorObjectsLibrary::Event::SetEventTime (double *EventTime*)

Sets event time.

Parameters:

EventTime	Time of the event.

Parameters:

EventTime	Time of the event.

Reimplemented in **TrafficModelEventLibrary::TrafficModelEvent** (*p.**Error! Bookmark not defined.***).

Definition at line 42 of file Event.cpp.

void SimulatorObjectsLibrary::Event::SetNext (Event * *nextEvent*)

Set the pointer to the next event.

Parameters:

nextEvent	The next event.

Parameters:

nextEvent	[in,out] The next event.

Definition at line 34 of file Event.cpp.

Member Data Documentation

double SimulatorObjectsLibrary::Event::mEventTime `[protected]`

Time of the event.

Definition at line 13 of file Event.h.

Event* SimulatorObjectsLibrary::Event::mNext `[protected]`

This member is used to facilitate the use of linked lists. This is a pointer to the next event in the linked list.

Definition at line 15 of file Event.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**Event.h**
- D:/Training/TM/**Event.cpp**

SimulatorObjectsLibrary::SimulatorClass Class Reference

SimulatorClass Object.
`#include <SimulatorClass.h>`
Inherited by **SimulatorObjectsLibrary::CQueue**, and **SimulatorObjectsLibrary::LinkedListPQClass**.

Public Member Functions

- **SimulatorClass** (void)
  *Initializes a new instance of the **SimulatorClass** class.*
- **~SimulatorClass** (void)
  *Finalizes an instance of the **SimulatorClass** class.*
- virtual bool **AddEvent** (**Event** *__Event__)=0
  Adds an event to the Simulator.
- virtual **Event** * **PopEvent** (**Event** *__Event__)=0
  *Pops the event addressed by **Event**.*
- virtual **Event** * **PopNext** (void)=0
  *Pops the next **Event**.*
- virtual bool **IsEmpty** (void)=0
  Query if this object is empty.

Protected Attributes

- **Event** * **mEventsQueueList**
  *Pointer to an array of **Event** objects.*
- unsigned int **mNumberOfEventsInQueue**
  Number of events in the EventQueueList.

Detailed Description

SimulatorClass Object.

Definition at line 9 of file SimulatorClass.h.

Constructor & Destructor Documentation

SimulatorObjectsLibrary::SimulatorClass::SimulatorClass (void) [inline]

Initializes a new instance of the **SimulatorClass** class.

Definition at line 18 of file SimulatorClass.h.

SimulatorObjectsLibrary::SimulatorClass::~SimulatorClass (void) [inline]

Finalizes an instance of the **SimulatorClass** class.

Definition at line 20 of file SimulatorClass.h.

Member Function Documentation

virtual bool SimulatorObjectsLibrary::SimulatorClass::AddEvent (Event * *Event*) [pure virtual]

Adds an event to the Simulator.

Parameters:

Event	[in,out] If non-null, the event.

Returns:

true if it succeeds, false if it fails.

Implemented in **SimulatorObjectsLibrary::CQueue** (*p.12*), and **SimulatorObjectsLibrary::LinkedListPQClass** (*p.16*).

virtual bool SimulatorObjectsLibrary::SimulatorClass::IsEmpty (void) [pure virtual]

Query if this object is empty.

Returns:

true if empty, false if not.

Implemented in **SimulatorObjectsLibrary::CQueue** (*p.12*), and **SimulatorObjectsLibrary::LinkedListPQClass** (*p.17*).

virtual Event* SimulatorObjectsLibrary::SimulatorClass::PopEvent (Event * *Event*) [pure virtual]

Pops the event addressed by **Event**.

Parameters:

Event	[in,out] If non-null, the event.

Returns:

false if it fails, else returns true

Implemented in **SimulatorObjectsLibrary::CQueue** (*p.13*), and **SimulatorObjectsLibrary::LinkedListPQClass** (*p.17*).

virtual Event* SimulatorObjectsLibrary::SimulatorClass::PopNext (void) [pure virtual]

Pops the next **Event**.

Returns:

Returns pointer to the next event

Implemented in **SimulatorObjectsLibrary::CQueue** (*p.13*), and **SimulatorObjectsLibrary::LinkedListPQClass** (*p.17*).

Member Data Documentation

Event* SimulatorObjectsLibrary::SimulatorClass::mEventsQueueList [protected]

Pointer to an array of **Event** objects.

Definition at line 13 of file SimulatorClass.h.

unsigned int SimulatorObjectsLibrary::SimulatorClass::mNumberOfEventsInQueue [protected]

Number of events in the EventQueueList.

Definition at line 15 of file SimulatorClass.h.

The documentation for this class was generated from the following file:

- D:/Training/TM/**SimulatorClass.h**

SimulatorObjectsLibrary::CQueue Class Reference

Calendar Queue class.
`#include <CQueue.h>`
Inherits **SimulatorObjectsLibrary::SimulatorClass**.

Public Member Functions

- **CQueue** (void)
  *Initializes a new instance of the **CQueue** class and internal structure.*
- **~CQueue** (void)
  *Finalizes the instance of the **CQueue** class.*
- virtual bool **AddEvent** (**Event** *newEvent)
  Adds an event to the Calendar Queue.
- virtual **Event** * **PopEvent** (**Event** *ptrEvent)
  Removes a particular event from the queue.
- virtual **Event** * **PopNext** (void)
  Removes the earliest event from the Calendar Queue.
- virtual bool **IsEmpty** (void)
  Checks if the queue is empty.
- void **print** (void)
  Prints out the current internal structure of the queue.

Protected Attributes

- **Event** ** **bucket**
  The head pointer to the array of buckets.
- int **qsize**
  The number of events currently stored in the queue.
- int **nbucket**
  The number of buckets currently in the queue.
- double **width**
  The width of each bucket ie. the range of timestamps in bucket.
- int **max**
  The maximum number of events that can be stored before resizing.
- int **min**
  The mimimum number of events that can be stored before resizing.
- int **curBucket**
  The current bucket to start searching for the next event.
- double **curBucketLatest**
  The latest time stamp possible to be stored in the current bucket for the current cycle.
- double **prevTime**
  The time stamp of the latest dequeued event.
- bool **resizeEnabled**
  Switch that determines if resize is done if max or min is reached.

Detailed Description

Calendar Queue class.

Definition at line 7 of file CQueue.h.

Constructor & Destructor Documentation

SimulatorObjectsLibrary::CQueue::CQueue (void)

Initializes a new instance of the **CQueue** class and internal structure.

Default constructor.

Definition at line 13 of file CQueue.cpp.

SimulatorObjectsLibrary::CQueue::~CQueue (void)

Finalizes the instance of the **CQueue** class.

Destructor.

Definition at line 20 of file CQueue.cpp.

Member Function Documentation

bool SimulatorObjectsLibrary::CQueue::AddEvent (Event * *newEvent*) `[virtual]`

Adds an event to the Calendar Queue.

Parameters:

newEvent	Pointer to the new event to enqueue

Returns:

True

Parameters:

newEvent	[in,out] Pointer to the new event to enqueue.

Returns:

True.

Implements **SimulatorObjectsLibrary::SimulatorClass** (*p.9*).

Definition at line 71 of file CQueue.cpp.

bool SimulatorObjectsLibrary::CQueue::IsEmpty (void) `[virtual]`

Checks if the queue is empty.

Query if this object is empty.

Returns:

true is qsize=0, false otherwise.

Returns:

true if empty, false if not.

Implements **SimulatorObjectsLibrary::SimulatorClass** (*p.9*).

Definition at line 181 of file CQueue.cpp.

Event * SimulatorObjectsLibrary::CQueue::PopEvent (Event * *ptrEvent*) `[virtual]`

Removes a particular event from the queue.

Parameters:

ptrEvent	[in,out] If non-null, the pointer event.

Returns:

Pointer to the event removed from the queue.

Parameters:

ptrEvent	[in,out] If non-null, the pointer event.

Returns:

Pointer to the event removed from the queue.

Implements **SimulatorObjectsLibrary::SimulatorClass** (*p.9*).

Definition at line 130 of file CQueue.cpp.

Event * SimulatorObjectsLibrary::CQueue::PopNext (void) `[virtual]`

Removes the earliest event from the Calendar Queue.

Pops the next.

Returns:

Pointer to the earliest event in the calendar queue

Returns:

null if it fails, else.

Implements **SimulatorObjectsLibrary::SimulatorClass** (*p.9*).

Definition at line 192 of file CQueue.cpp.

void SimulatorObjectsLibrary::CQueue::print (void)

Prints out the current internal structure of the queue.

Prints this object.

Definition at line 353 of file CQueue.cpp.

Member Data Documentation

Event** SimulatorObjectsLibrary::CQueue::bucket `[protected]`

The head pointer to the array of buckets.

Definition at line 12 of file CQueue.h.

int SimulatorObjectsLibrary::CQueue::curBucket `[protected]`

The current bucket to start searching for the next event.

Definition at line 24 of file CQueue.h.

double SimulatorObjectsLibrary::CQueue::curBucketLatest `[protected]`

The latest time stamp possible to be stored in the current bucket for the current cycle.

Definition at line 26 of file CQueue.h.

int SimulatorObjectsLibrary::CQueue::max `[protected]`

The maximum number of events that can be stored before resizing.

Definition at line 20 of file CQueue.h.

int SimulatorObjectsLibrary::CQueue::min `[protected]`

The mimimum number of events that can be stored before resizing.

Definition at line 22 of file CQueue.h.

int SimulatorObjectsLibrary::CQueue::nbucket `[protected]`

The number of buckets currently in the queue.

Definition at line 16 of file CQueue.h.

double SimulatorObjectsLibrary::CQueue::prevTime `[protected]`

The time stamp of the latest dequeued event.

Definition at line 28 of file CQueue.h.

int SimulatorObjectsLibrary::CQueue::qsize `[protected]`

The number of events currently stored in the queue.

Definition at line 14 of file CQueue.h.

bool SimulatorObjectsLibrary::CQueue::resizeEnabled `[protected]`

Switch that determines if resize is done if max or min is reached.

Definition at line 30 of file CQueue.h.

double SimulatorObjectsLibrary::CQueue::width [protected]

The width of each bucket ie. the range of timestamps in bucket.

Definition at line 18 of file CQueue.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**CQueue.h**
- D:/Training/TM/**CQueue.cpp**

SimulatorObjectsLibrary::LinkedListPQClass Class Reference

Linked list pq class.
`#include <LinkedListPQClass.h>`
Inherits **SimulatorObjectsLibrary::SimulatorClass**.

Public Member Functions

- **LinkedListPQClass** (void)
  *Initializes a new instance of the **LinkedListPQClass** class.*
- **~LinkedListPQClass** (void)
  *Finalizes an instance of the **LinkedListPQClass** class.*
- virtual bool **AddEvent** (**Event** *__Event__)
  Adds an event.
- virtual **Event** * **PopEvent** (**Event** *__Event__)
  *Pops the event described by **Event**.*
- virtual **Event** * **PopNext** (void)
  Pops the next.
- virtual bool **IsEmpty** (void)
  Query if this object is empty.

Detailed Description

Linked list pq class.

Definition at line 8 of file LinkedListPQClass.h.

Constructor & Destructor Documentation

SimulatorObjectsLibrary::LinkedListPQClass::LinkedListPQClass (void)

Initializes a new instance of the **LinkedListPQClass** class.

Default constructor.

Definition at line 8 of file LinkedListPQClass.cpp.

SimulatorObjectsLibrary::LinkedListPQClass::~LinkedListPQClass (void)

Finalizes an instance of the **LinkedListPQClass** class.

Destructor.

Definition at line 14 of file LinkedListPQClass.cpp.

Member Function Documentation

bool SimulatorObjectsLibrary::LinkedListPQClass::AddEvent (Event * *ptrEvent*) `[virtual]`

Adds an event.

Parameters:

Event	[in,out] If non-null, the event.

Returns:

true if it succeeds, false if it fails.

Parameters:

ptrEvent	[in,out] If non-null, the pointer event.

Returns:

true if it succeeds, false if it fails.

Implements **SimulatorObjectsLibrary::SimulatorClass** (*p.9*).

Definition at line 23 of file LinkedListPQClass.cpp.

bool SimulatorObjectsLibrary::LinkedListPQClass::IsEmpty (void) [virtual]

Query if this object is empty.

Returns:

true if empty, false if not.

Returns:

true if empty, false if not.

Implements **SimulatorObjectsLibrary::SimulatorClass** (*p.9*).

Definition at line 116 of file LinkedListPQClass.cpp.

Event * SimulatorObjectsLibrary::LinkedListPQClass::PopEvent (Event * *ptrEvent*) [virtual]

Pops the event described by **Event**.

Pops the event described by ptrEvent.

Parameters:

Event	[in,out] If non-null, the event.

Returns:

null if it fails, else.

Parameters:

ptrEvent	[in,out] If non-null, the pointer event.

Returns:

null if it fails, else.

Implements **SimulatorObjectsLibrary::SimulatorClass** (*p.9*).

Definition at line 95 of file LinkedListPQClass.cpp.

Event * SimulatorObjectsLibrary::LinkedListPQClass::PopNext (void) [virtual]

Pops the next.

Returns:

null if it fails, else.

Returns:

null if it fails, else.

Implements **SimulatorObjectsLibrary::SimulatorClass** (*p.9*).

Definition at line 73 of file LinkedListPQClass.cpp.

The documentation for this class was generated from the following files:

- D:/Training/TM/**LinkedListPQClass.h**
- D:/Training/TM/**LinkedListPQClass.cpp**

APPENDIX B
Traffic Model Objects Library Reference

TrafficModelObjectsLibrary::TrafficModelObject Class Reference

Traffic model object.

`#include <TrafficModelObject.h>`

Inherited by **TrafficModelObjectsLibrary::PhaseClass**, **TrafficModelObjectsLibrary::TrafficModelClass**, **TrafficModelObjectsLibrary::TrafficNodeClass**, **TrafficModelObjectsLibrary::VehicleClass**, and **TrafficModelObjectsLibrary::VehicleQueueClass**.

Public Member Functions

- **TrafficModelObject** (void)
  *Initializes a new instance of the **TrafficModelObject** class.*
- **~TrafficModelObject** (void)
  *Finalizes the instance of the **TrafficModelObject** class.*
- **ObjectType Type** ()
  Return of the type of object.

Protected Attributes

- **ObjectType mType**
  The type of the object.

Detailed Description

Traffic model object.

Definition at line 41 of file TrafficModelObject.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::TrafficModelObject::TrafficModelObject (void)

Initializes a new instance of the **TrafficModelObject** class.

Default constructor.

Definition at line 8 of file TrafficModelObject.cpp.

TrafficModelObjectsLibrary::TrafficModelObject::~TrafficModelObject (void)

Finalizes the instance of the **TrafficModelObject** class.

Destructor.

Definition at line 14 of file TrafficModelObject.cpp.

Member Function Documentation

ObjectType TrafficModelObjectsLibrary::TrafficModelObject::Type (void)

Return of the type of object.

Gets the type.

Returns:

mType

Returns:

.

Definition at line 21 of file TrafficModelObject.cpp.

Member Data Documentation

ObjectType TrafficModelObjectsLibrary::TrafficModelObject::mType `[protected]`

The type of the object.

Definition at line 45 of file TrafficModelObject.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**TrafficModelObject.h**
- D:/Training/TM/**TrafficModelObject.cpp**

TrafficModelObjectsLibrary::TrafficModelClass Class Reference

Traffic Model Class.

`#include <TrafficModelClass.h>`

Inherits **TrafficModelObjectsLibrary::TrafficModelObject**.

Public Member Functions

- **TrafficModelClass** (void)
  *Initializes a new instance of the **TrafficModelClass**.*
- **~TrafficModelClass** (void)
  *Finalizes the instance of the **TrafficModelClass**.*
- **TrafficModelClass** (int Destinations, int Intersections, int ParkingLots, int RoadSegments)
  *Initializes a new instance of the **TrafficModelClass** with the given number of elements in the traffic.*
- **DestinationClass** * **AddDestination** (double AverageCrossingTime)
  Creates a new Destination and adds its pointer to the pointers list of Traffic Nodes of the Model.
- **IntersectionClass** * **AddIntersection** (int Phases, int QueuesOut)
  Creates a new Intersection and adds its pointer to the pointers list of Traffic Nodes of the Model.
- **ParkingLotClass** * **AddParkingLot** (int Capacity, int Exits, int EndTraficNode, int Phase, int Destinations)
  Creates a new Parking Lot and adds its pointer to pointers list of Vehicle Queues of the Model.
- **RoadSegmentClass** * **AddRoadSegment** (int Capacity, double AverageTravelTime, int Lanes, int StartTrafficNode, int EndTrafficNode, int Phase, int Destinations)
  Creates a new Road Segment and adds its pointer to pointers list of Vehicle Queues of the Model.
- **TrafficNodeClass** * **GetTrafficNode** (int Index)
  Gets a pointer to a Traffic Node.
- **VehicleQueueClass** * **GetVehicleQueue** (int Index)
  Gets the pointer to a Vehicle Queue.
- int **DestinationsCount** ()
  Gets the Destinations count.
- int **IntersectionsCount** ()
  Gets the Intersections count.
- int **ParkingLotsCount** ()
  Gets the Parking Lots count.
- int **RoadSegmentsCount** ()
  Gets the Road Segments count.
- int **NodesCount** ()
  Gets the Traffic Nodes count.
- int **VehicleQueuesCount** ()
  Gets the Vehicle Queues count.
- bool **Trace** (void)
  Gets the Trace state. If Trace is true, intermediate output is produced during the simulation.
- void **setTrace** (bool bTrace)
  Sets a the Trace state.
- void **setCheck** (bool bCheck)
  Sets a Check mode.
- bool **Check** (void)

Gets the Check state. If Check is true, The Parking Lots capacity is set to the number of destinations, and a vehicle per destination will be issued form each Parking Lot.

- void **setOutputStream** (ostream *OutputStream)
  Sets an output stream.
- ostream * **getOutputStream** (void)
  Gets the output stream.
- void **setSimulator** (**SimulatorClass** ***Simulator**)
  Sets a Simulator.
- **SimulatorClass** * **Simulator** (void)
  Gets the Simulator.
- void **setRandomGenerator** (**clsRandomGenerator** ***RandomGenerator**)
  Sets a Random Number Generator.
- **clsRandomGenerator** * **RandomGenerator** (void)
  Gets the random generator.

Protected Attributes

- int **mDestinations**
  Number of Destinations in the Model.
- int **mIntersections**
  Number of Intersections in the Model.
- int **mParkingLots**
  Number of Parking Lots in the Model.
- int **mRoadSegments**
  Number of Road Segments in the Model.
- int **mNodesCount**
  Counter for the Traffic Nodes in the Model. When the Model is fully set up, the Traffic Nodes count should be equal to the sum of Destinations plus Intersections.
- int **mVehicleQueuesCount**
  Counter for the Vehicle Queues. When the Model is fully set up, the Vehicle Queues count should be equal to the sum of Parking Lots and Road Segments.
- **TrafficNodeClass** ** **mNodesList**
  List of pointers to the Traffic Nodes.
- **VehicleQueueClass** ** **mVehicleQueuesList**
  List of pointers to the Vehicle Queues.
- **SimulatorClass** * **mSimulator**
  Pointer to a Simulator.
- **clsRandomGenerator** * **mRandomGenerator**
  Pointer to the Random Number Generator.
- bool **mTrace**
  Boolean value to indicate weather or not to produce intermediate output.
- bool **mCheck**
  Boolean value to indicate the Traffic Model is in Check mode.
- ostream * **mOutputStream**
  Pointer to the output stream to be used.

Detailed Description

Traffic Model Class.

Definition at line 8 of file TrafficModelClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::TrafficModelClass::TrafficModelClass (void) `[inline]`

Initializes a new instance of the **TrafficModelClass**.

Definition at line 53 of file TrafficModelClass.h.

TrafficModelObjectsLibrary::TrafficModelClass::~TrafficModelClass (void) `[inline]`

Finalizes the instance of the **TrafficModelClass**.

Definition at line 56 of file TrafficModelClass.h.

TrafficModelObjectsLibrary::TrafficModelClass::TrafficModelClass (int *Destinations*, int *Intersections*, int *ParkingLots*, int *RoadSegments*)

Initializes a new instance of the **TrafficModelClass** with the given number of elements in the traffic.

Initializes a new instance of the **TrafficModelClass**.

Parameters:

Destinations	Number of Destinations
Intersections	Number of Intersections
ParkingLots	Number of Parking Lots
RoadSegments	Number of Road Segments

Parameters:

Destinations	Number of Destinations.
Intersections	Number of Intersections.
ParkingLots	Number of Parking Lots.
RoadSegments	Number of Road Segments.

Definition at line 19 of file TrafficModelClass.cpp.

Member Function Documentation

DestinationClass * TrafficModelObjectsLibrary::TrafficModelClass::AddDestination (double *AverageCrossingTime*)

Creates a new Destination and adds its pointer to the pointers list of Traffic Nodes of the Model.

Parameters:

AverageCrossingT	The average time a vehicle spend thru the exit road. It may be zero if there is

ime	not exit road

Returns:

null if it fails, else a pointer to the new Destination

Parameters:

AverageCrossingTime	The average time a vehicle spend thru the exit road. It may be zero if there is not exit road.

Returns:

null if it fails, else a pointer to the new Destination.

Definition at line 48 of file TrafficModelClass.cpp.

IntersectionClass * TrafficModelObjectsLibrary::TrafficModelClass::AddIntersection (int *Phases*, int *QueuesOut*)

Creates a new Intersection and adds its pointer to the pointers list of Traffic Nodes of the Model.

Parameters:

Phases	The number of Phases in the Intersection
QueuesOut	The number of Vehicle Queues going out of the Intersection

Returns:

null if it fails, else a pointer to the new Intersection

Parameters:

Phases	The number of Phases in the Intersection.
QueuesOut	The number of Vehicle Queues going out of the Intersection.

Returns:

null if it fails, else a pointer to the new Intersection.

Definition at line 71 of file TrafficModelClass.cpp.

ParkingLotClass * TrafficModelObjectsLibrary::TrafficModelClass::AddParkingLot (int *Capacity*, int *Exits*, int *EndTrafficNode*, int *Phase*, int *Destinations*)

Creates a new Parking Lot and adds its pointer to pointers list of Vehicle Queues of the Model.

Parameters:

Capacity	The number of vehicles than can be parked in the Parking Lot
Exits	The number of exits
EndTrafficNode	The index in the Traffic Nodes pointers list corresponding to the Traffic Node outside the Parking Lot
Phase	The index in the Phases pointers list corresponding to the Phase which evacuates Vehicles from the Parking Lot
Destinations	The number of Destinations available to the Vehicles in the Parking Lot

Returns:

null if it fails, else a pointer to the new Parking Lot

Parameters:

Capacity	The number of vehicles than can be parked in the Parking Lot.
Exits	The number of exits.
EndTrafficNode	The index in the Traffic Nodes pointers list corresponding to the Traffic Node outside the Parking Lot.
Phase	The index in the Phases pointers list corresponding to the Phase which

	evacuates Vehicles from the Parking Lot.
Destinations	The number of Destinations available to the Vehicles in the Parking Lot.

Returns:

null if it fails, else a pointer to the new Parking Lot.

Definition at line 100 of file TrafficModelClass.cpp.

RoadSegmentClass * TrafficModelObjectsLibrary::TrafficModelClass::AddRoadSegment (int *Capacity*, double *AverageTravelTime*, int *Lanes*, int *StartTrafficNode*, int *EndTrafficNode*, int *Phase*, int *Destinations*)

Creates a new Road Segment and adds its pointer to pointers list of Vehicle Queues of the Model.

Parameters:

Capacity	The number of vehicles than can ocupy the Road Segment
AverageTravelTime	Time to travel de Road Segment at average speed
Lanes	The number of lanes
StartTrafficNode	The index in the Traffic Nodes pointers list corresponding to the Traffic Node where the Road Segment Starts
EndTrafficNode	The index in the Traffic Nodes pointers list corresponding to the Traffic Node where the Road Segment Ends
Phase	The index in the Phases pointers list corresponding to the Phase which evacuates Vehicles from the Road Segment
Destinations	The number of Destinations served by the Road Segment

Returns:

null if it fails, else a pointer to the new Road Segment

Parameters:

Capacity	The number of vehicles than can ocupy the Road Segment.
AverageTravelTime	Time to travel de Road Segment at average speed.
Lanes	The number of lanes.
StartTrafficNode	The index in the Traffic Nodes pointers list corresponding to the Traffic Node where the Road Segment Starts.
EndTrafficNode	The index in the Traffic Nodes pointers list corresponding to the Traffic Node where the Road Segment Ends.
Phase	The index in the Phases pointers list corresponding to the Phase which evacuates Vehicles from the Road Segment.
Destinations	The number of Destinations served by the Road Segment.

Returns:

null if it fails, else a pointer to the new Road Segment.

Definition at line 132 of file TrafficModelClass.cpp.

bool TrafficModelObjectsLibrary::TrafficModelClass::Check (void) `[inline]`

Gets the Check state. If Check is true, The Parking Lots capacity is set to the number of destinations, and a vehicle per destination will be issued form each Parking Lot.

Returns:

true if it succeeds, false if it fails

Definition at line 182 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::DestinationsCount () [inline]

Gets the Destinations count.

Returns:

The number of Destinations in the Model

Definition at line 134 of file TrafficModelClass.h.

ostream* TrafficModelObjectsLibrary::TrafficModelClass::getOutputStream (void) [inline]

Gets the output stream.

Returns:

null if it fails, else the output stream

Definition at line 192 of file TrafficModelClass.h.

TrafficNodeClass* TrafficModelObjectsLibrary::TrafficModelClass::GetTrafficNode (int *Index*) [inline]

Gets a pointer to a Traffic Node.

Parameters:

Index	Zero-based index of the pointer in the list of Traffic Nodes

Returns:

Pointer to the requested traffic node, NULL if fail

Definition at line 122 of file TrafficModelClass.h.

VehicleQueueClass* TrafficModelObjectsLibrary::TrafficModelClass::GetVehicleQueue (int *Index*) [inline]

Gets the pointer to a Vehicle Queue.

Parameters:

Index	Zero-based index of the pointer in the list of Vehicle Queues

Returns:

null if it fails, else the pointer to a Vehicle Queue

Definition at line 129 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::IntersectionsCount () [inline]

Gets the Intersections count.

Returns:

The number of Intersections in the Model

Definition at line 139 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::NodesCount () [inline]

Gets the Traffic Nodes count.

Returns:

The number of Traffic Nodes in the Model. For consistency, this value should be equal to the sum of Destinations and Intersections

Definition at line 154 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::ParkingLotsCount () [inline]

Gets the Parking Lots count.

Returns:

The number of Parking Lots in the Model

Definition at line 144 of file TrafficModelClass.h.

clsRandomGenerator* TrafficModelObjectsLibrary::TrafficModelClass::RandomGenerator (void) [inline]

Gets the random generator.

Returns:

null if it fails, else the pointer to the Random umber Generator

Definition at line 212 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::RoadSegmentsCount () [inline]

Gets the Road Segments count.

Returns:

The number of Road Segments in the Model

Definition at line 149 of file TrafficModelClass.h.

void TrafficModelObjectsLibrary::TrafficModelClass::setCheck (bool *bCheck*) [inline]

Sets a Check mode.

Parameters:

bCheck	True to set the Model in Check mode

Definition at line 174 of file TrafficModelClass.h.

void TrafficModelObjectsLibrary::TrafficModelClass::setOutputStream (ostream * *OutputStream*) [inline]

Sets an output stream.

Parameters:

OutputStream	[in,out] If non-null, stream to write data to

Definition at line 187 of file TrafficModelClass.h.

void TrafficModelObjectsLibrary::TrafficModelClass::setRandomGenerator (clsRandomGenerator * *RandomGenerator*) [inline]

Sets a Random Number Generator.

Parameters:

RandomGenerator	[in,out] If non-null, the random number generator

Definition at line 207 of file TrafficModelClass.h.

void TrafficModelObjectsLibrary::TrafficModelClass::setSimulator (SimulatorClass * *Simulator*) [inline]

Sets a Simulator.

Parameters:

Simulator	[in,out] If non-null, the simulator

Definition at line 197 of file TrafficModelClass.h.

void TrafficModelObjectsLibrary::TrafficModelClass::setTrace (bool *bTrace*) [inline]

Sets a the Trace state.

Parameters:

bTrace	True to produce intermediate output, false to omit it

Definition at line 169 of file TrafficModelClass.h.

SimulatorClass* TrafficModelObjectsLibrary::TrafficModelClass::Simulator (void) [inline]

Gets the Simulator.

Returns:

null if it fails, else the pointer to the Simulator

Definition at line 202 of file TrafficModelClass.h.

bool TrafficModelObjectsLibrary::TrafficModelClass::Trace (void) [inline]

Gets the Trace state. If Trace is true, intermediate output is produced during the simulation.

Returns:

True/false

Definition at line 164 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::VehicleQueuesCount () [inline]

Gets the Vehicle Queues count.

Returns:

The number of Vehicle Queues in the Model. For consistency, this value should be equal to the sum of Parking Lots and Road Segments

Definition at line 159 of file TrafficModelClass.h.

Member Data Documentation

bool TrafficModelObjectsLibrary::TrafficModelClass::mCheck [protected]

Boolean value to indicate the Traffic Model is in Check mode.

Definition at line 46 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::mDestinations [protected]

Number of Destinations in the Model.

Definition at line 13 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::mIntersections [protected]

Number of Intersections in the Model.

Definition at line 16 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::mNodesCount [protected]

Counter for the Traffic Nodes in the Model. When the Model is fully set up, the Traffic Nodes count should be equal to the sum of Destinations plus Intersections.

Definition at line 25 of file TrafficModelClass.h.

TrafficNodeClass** TrafficModelObjectsLibrary::TrafficModelClass::mNodesList [protected]

List of pointers to the Traffic Nodes.

Definition at line 31 of file TrafficModelClass.h.

ostream* TrafficModelObjectsLibrary::TrafficModelClass::mOutputStream [protected]

Pointer to the output stream to be used.

Definition at line 49 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::mParkingLots [protected]

Number of Parking Lots in the Model.

Definition at line 19 of file TrafficModelClass.h.

clsRandomGenerator* TrafficModelObjectsLibrary::TrafficModelClass::mRandomGenerator [protected]

Pointer to the Random Number Generator.

Definition at line 40 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::mRoadSegments [protected]

Number of Road Segments in the Model.

Definition at line 22 of file TrafficModelClass.h.

SimulatorClass* TrafficModelObjectsLibrary::TrafficModelClass::mSimulator [protected]

Pointer to a Simulator.

Definition at line 37 of file TrafficModelClass.h.

bool TrafficModelObjectsLibrary::TrafficModelClass::mTrace [protected]

Boolean value to indicate weather or not to produce intermediate output.

Definition at line 43 of file TrafficModelClass.h.

int TrafficModelObjectsLibrary::TrafficModelClass::mVehicleQueuesCount [protected]

Counter for the Vehicle Queues. When the Model is fully set up, the Vehicle Queues count should be equal to the sum of Parking Lots and Road Segments.

Definition at line 28 of file TrafficModelClass.h.

VehicleQueueClass** TrafficModelObjectsLibrary::TrafficModelClass::mVehicleQueuesList [protected]

List of pointers to the Vehicle Queues.

Definition at line 34 of file TrafficModelClass.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**TrafficModelClass.h**
- D:/Training/TM/**TrafficModelClass.cpp**

TrafficModelObjectsLibrary::VehicleClass Class Reference

VehicleClass object.
`#include <VehicleClass.h>`
Inherits **TrafficModelObjectsLibrary::TrafficModelObject**.

Public Member Functions

- **VehicleClass** (void)
  Default constructor.
- **VehicleClass** (**TrafficModelObject** *TrafficModel, int Destination, int IssuedBy, int ID)
  ***VehicleClass** constructor with parameters.*
- **~VehicleClass** (void)
  ***VehicleClass** Destructor.*
- **VehicleClass** * **Next** ()
  *Gets the next **VehicleClass** object.*
- int **getDestination** ()
  *Gets the destination of the **VehicleClass** object.*
- void **setNext** (**VehicleClass** ***mNext**)
  *Sets a pointer to the next **VehicleClass** object.*
- double **getStartTime** (void)
  *Gets the **VehicleClass** object's start time.*
- void **setStartTime** (double dblStartTime)
  *Sets a **VehicleClass** object's start time.*
- double **getTravelTime** (void)
  *Gets the travel time of the **VehicleClass** object.*
- void **setTravelTime** (double dblTravelTime)
  *Sets a travel time for the **VehicleClass** object.*
- char * **ToString** (void)
  Convert this object into a string representation.

Protected Attributes

- **VehicleClass** * **mNext**
  A pointer to the next Vehicle object.
- int **mDestination**
  Index of the destination for the vehicle.
- **TrafficModelObject** * **mTrafficModel**
  A pointer to the the traffic model.
- double **mStartTime**
  Starting time for a vehicle.
- double **mTravelTime**
  Time the vehicle has taken to travel.
- int **mIssuedBy**
  Index of the ParkingLot that issued the Vehicle.
- int **mID**
  ID of the Vehicle.

Detailed Description

VehicleClass object.

Definition at line 8 of file VehicleClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::VehicleClass::VehicleClass (void)

Default constructor.

Definition at line 25 of file VehicleClass.cpp.

TrafficModelObjectsLibrary::VehicleClass::VehicleClass (TrafficModelObject * *TrafficModel*, int *Destination*, int *IssuedBy*, int *ID*)

VehicleClass constructor with parameters.

Parameters:

TrafficModel	[in,out] If non-null, pointer to the traffic model.
Destination	Integer ID for the Destination for the vehicle.
IssuedBy	ID/Index of the parking lot that issued the **VehicleClass** object.
ID	The identifier.

Parameters:

TrafficModel	[in,out] If non-null, pointer to the traffic model.
Destination	Integer ID for the Destination for the vehicle.
IssuedBy	ID/Index of the parking lot that issued the **VehicleClass** object.
ID	The identifier.

Definition at line 34 of file VehicleClass.cpp.

TrafficModelObjectsLibrary::VehicleClass::~VehicleClass (void)

VehicleClass Destructor.

Destructor.

Definition at line 55 of file VehicleClass.cpp.

Member Function Documentation

int TrafficModelObjectsLibrary::VehicleClass::getDestination ()

Gets the destination of the **VehicleClass** object.

Gets the destination.

Returns:

Integer: Destination ID

Returns:

The destination.

Definition at line 62 of file VehicleClass.cpp.

double TrafficModelObjectsLibrary::VehicleClass::getStartTime (void)

Gets the **VehicleClass** object's start time.

Returns:

Double:Starttime

double TrafficModelObjectsLibrary::VehicleClass::getTravelTime (void)

Gets the travel time of the **VehicleClass** object.

Gets the travel time.

Returns:

The travel time as a double

Returns:

The travel time.

Definition at line 102 of file VehicleClass.cpp.

VehicleClass * TrafficModelObjectsLibrary::VehicleClass::Next (void)

Gets the next **VehicleClass** object.

Returns:

If there is no next **VehicleClass** object returns null

Returns:

If there is no next **VehicleClass** object returns null.

Definition at line 78 of file VehicleClass.cpp.

void TrafficModelObjectsLibrary::VehicleClass::setNext (VehicleClass * *Vehicle*)

Sets a pointer to the next **VehicleClass** object.

Sets a next.

Parameters:

mNext	[in,out] If non-null, a pointer to the next **VehicleClass** object

Parameters:

Vehicle	[in,out] If non-null, the vehicle.

Definition at line 70 of file VehicleClass.cpp.

void TrafficModelObjectsLibrary::VehicleClass::setStartTime (double *StartTime*)

Sets a **VehicleClass** object's start time.

Sets a start time.

Parameters:

dblStartTime	start time (double)

Parameters:

StartTime	The start time.

Definition at line 86 of file VehicleClass.cpp.

void TrafficModelObjectsLibrary::VehicleClass::setTravelTime (double *TravelTime*)

Sets a travel time for the **VehicleClass** object.

Sets a travel time.

Parameters:

dblTravelTime	Travel time as a double

Parameters:

TravelTime	Time of the travel.

Definition at line 94 of file VehicleClass.cpp.

char * TrafficModelObjectsLibrary::VehicleClass::ToString (void)

Convert this object into a string representation.

Returns:

Returns a string representation of the **VehicleClass** object describing attributes like start time, and issue id

Returns:

This object as a char*.

Definition at line 110 of file VehicleClass.cpp.

Member Data Documentation

int TrafficModelObjectsLibrary::VehicleClass::mDestination [protected]

Index of the destination for the vehicle.

Definition at line 15 of file VehicleClass.h.

int TrafficModelObjectsLibrary::VehicleClass::mID [protected]

ID of the Vehicle.

Definition at line 25 of file VehicleClass.h.

int TrafficModelObjectsLibrary::VehicleClass::mIssuedBy [protected]

Index of the ParkingLot that issued the Vehicle.

Definition at line 23 of file VehicleClass.h.

VehicleClass* TrafficModelObjectsLibrary::VehicleClass::mNext [protected]

A pointer to the next Vehicle object.

Definition at line 13 of file VehicleClass.h.

double TrafficModelObjectsLibrary::VehicleClass::mStartTime [protected]

Starting time for a vehicle.

Definition at line 19 of file VehicleClass.h.

TrafficModelObject* TrafficModelObjectsLibrary::VehicleClass::mTrafficModel [protected]

A pointer to the the traffic model.

Definition at line 17 of file VehicleClass.h.

double TrafficModelObjectsLibrary::VehicleClass::mTravelTime [protected]

Time the vehicle has taken to travel.

Definition at line 21 of file VehicleClass.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**VehicleClass.h**
- D:/Training/TM/**VehicleClass.cpp**

TrafficModelObjectsLibrary::PhaseClass Class Reference

Phase class.

`#include <PhaseClass.h>`

Inherits **TrafficModelObjectsLibrary::TrafficModelObject**.

Public Member Functions

- **PhaseClass** (void)
  *Initializes a new instance of the **PhaseClass** class.*
- **PhaseClass** (int Index, int QueuesIn, double AvgServiceTime)
  *Initializes a new instance of the **PhaseClass** class.*
- **~PhaseClass** (void)
  *Finalizes an instance of the **PhaseClass** class.*
- void **AddVehicleQueueIn** (**TrafficModelObject** *QueueIn)
  Adds the given vehicle queue in to vehicles queues in list of the phase.
- **TrafficModelObject** * **GetVehicleQueueIn** (int Index)
  Gets the vehicle queue in at the given index in the phases vehicle queues in list.
- double **GetAverageServiceTime** (void)
  Gets the average service time for the current phase.
- int **GetIndexInNode** (void)
  Gets the index of the node (destination or intersection) which contains the phase.
- int **GetVehicleQueuesIn** (void)
  Gets the number of vehicle queues being serve by the Phase.
- int **GetVehicleQueuesInCount** (void)
  Gets count of vehicles queues in the phase.

Protected Attributes

- int **mVehicleQueuesIn**
  The number of vehicle queues delivering vehicles to the intersection.
- int **mVehicleQueuesInCount**
  Counter for the number of vehicle queues in.
- **TrafficModelObject** ** **mVehicleQueuesInList**
  List of pointers to the vehicles queues delivering vehicles to the intersection.
- bool **mActive**
  Show the state of the phase.
- double **mAverageServiceTime**
  The interval of time at which the phase is going to run.
- int **mIndexInNode**
  The index in the phases pointers list of the node.

Detailed Description

Phase class.

Definition at line 8 of file PhaseClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::PhaseClass::PhaseClass (void)

Initializes a new instance of the **PhaseClass** class.

Constructor.

Definition at line 35 of file PhaseClass.cpp.

TrafficModelObjectsLibrary::PhaseClass::PhaseClass (int *Index*, int *QueuesIn*, double *AvgServiceTime*)

Initializes a new instance of the **PhaseClass** class.

Parameters:

Index	Index of the pointer to this Phase in the pointers to phases list in the intersection .
QueuesIn	The number of vehicle queues that are being served by the phase.
AvgServiceTime	The interval of time at which the phase is going to run.

Definition at line 19 of file PhaseClass.cpp.

TrafficModelObjectsLibrary::PhaseClass::~PhaseClass (void)

Finalizes an instance of the **PhaseClass** class.

Destructor.

Definition at line 38 of file PhaseClass.cpp.

Member Function Documentation

void TrafficModelObjectsLibrary::PhaseClass::AddVehicleQueueIn (TrafficModelObject * *QueueIn*)

Adds the given vehicle queue in to vehicles queues in list of the phase.

Adds a vehicle queue in.

Parameters:

QueueIn	Vehicle queue being added to th elist to pointers list of the vehicle queue.

Parameters:

QueueIn	[in,out] If non-null, the queue in.

Definition at line 72 of file PhaseClass.cpp.

double TrafficModelObjectsLibrary::PhaseClass::GetAverageServiceTime (void)

Gets the average service time for the current phase.

Gets the average service time.

Returns:

The average service time.

Returns:

The average service time.

Definition at line 43 of file PhaseClass.cpp.

int TrafficModelObjectsLibrary::PhaseClass::GetIndexInNode (void)

Gets the index of the node (destination or intersection) which contains the phase.

Gets the index in node.

Returns:

The index in node.

Returns:

The index in node.

Definition at line 85 of file PhaseClass.cpp.

TrafficModelObject * TrafficModelObjectsLibrary::PhaseClass::GetVehicleQueueIn (int *Index*)

Gets the vehicle queue in at the given index in the phases vehicle queues in list.

Gets a vehicle queue in.

Parameters:

Index	Index of the pointer.

Returns:

null if it fails, else pointer to the vehicle queue in requested.

Parameters:

Index	Zero-based index of the.

Returns:

null if it fails, else the vehicle queue in.

Definition at line 56 of file PhaseClass.cpp.

int TrafficModelObjectsLibrary::PhaseClass::GetVehicleQueuesIn (void)

Gets the number of vehicle queues being serve by the Phase.

Gets the vehicle queues in.

Returns:

The vehicle queues in.

Returns:

The vehicle queues in.

Definition at line 93 of file PhaseClass.cpp.

int TrafficModelObjectsLibrary::PhaseClass::GetVehicleQueuesInCount (void)

Gets count of vehicles queues in the phase.

Gets the vehicle queues in count.

Returns:

The vehicle queues in count.

Returns:

The vehicle queues in count.

Definition at line 101 of file PhaseClass.cpp.

Member Data Documentation

bool TrafficModelObjectsLibrary::PhaseClass::mActive [protected]

Show the state of the phase.

Definition at line 23 of file PhaseClass.h.

double TrafficModelObjectsLibrary::PhaseClass::mAverageServiceTime [protected]

The interval of time at which the phase is going to run.

Definition at line 26 of file PhaseClass.h.

int TrafficModelObjectsLibrary::PhaseClass::mIndexInNode [protected]

The index in the phases pointers list of the node.

Definition at line 29 of file PhaseClass.h.

int TrafficModelObjectsLibrary::PhaseClass::mVehicleQueuesIn [protected]

The number of vehicle queues delivering vehicles to the intersection.

Definition at line 14 of file PhaseClass.h.

int TrafficModelObjectsLibrary::PhaseClass::mVehicleQueuesInCount [protected]

Counter for the number of vehicle queues in.

Definition at line 17 of file PhaseClass.h.

TrafficModelObject** TrafficModelObjectsLibrary::PhaseClass::mVehicleQueuesInList [protected]

List of pointers to the vehicles queues delivering vehicles to the intersection.

Definition at line 20 of file PhaseClass.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**PhaseClass.h**
- D:/Training/TM/**PhaseClass.cpp**

TrafficModelObjectsLibrary::TrafficNodeClass Class Reference

TrafficNodeClass Object.
`#include <TrafficNodeClass.h>`
Inherits **TrafficModelObjectsLibrary::TrafficModelObject**.
Inherited by **TrafficModelObjectsLibrary::DestinationClass**, and **TrafficModelObjectsLibrary::IntersectionClass**.

Public Member Functions

- **TrafficNodeClass** (void)
  *Initializes a new instance of the **TrafficNodeClass** class.*
- **~TrafficNodeClass** (void)
  *Finalizes an instance of the **TrafficNodeClass** class.*
- **TrafficModelObject** * **TrafficModel** (void)
  Get a pointer to the TrafficModel object.
- int **PhasesCount** (void)
  Gets the phases count of an intersection.
- int **ActivePhaseIndex** (void)
  Gets the index of the active phase of an intersection.
- **PhaseClass** * **ActivePhase** (void)
  Gets a pointer to the active phase object.
- **PhaseClass** * **GetPhase** (int Index)
  Gets a phase object at location "index".
- double **ChangePhase** (void)
  Changes the Active Phase and returns the Time when the next phase change must occur.
- bool **IsIdle** (void)
  Query if this object is idle.
- bool **IsEmpty** (void)
  Query if this object is empty.
- **Event** * **NextPhaseChangeScheduledEvent** (void)
  Gets the next phase change scheduled event.
- void **SetNextPhaseChangeScheduledEvent** (**Event** *ptrEvent)
  Sets a next phase change scheduled event.
- double **SetActivePhase** (int Index)
  Sets an active phase.
- double **SetNextActivePhase** (void)
  Sets the next active phase.
- void **VehicleIn** (void)
  Vehicle in.
- void **VehicleOut** (void)
  Vehicle out.
- virtual **TrafficModelObject** * **GetVehicleQueueOut** (int Destination)=0
  Gets a vehicle queue out.
- int **VehicleQueuesOutCount** (void)
  Gets the vehicle queues out count.

- void **AddVehicleQueueOut** (**TrafficModelObject** *QueueOut)
  Adds a vehicle queue out.
- virtual void **VehicleIn** (**VehicleClass** *Vehicle)=0
  Vehicle in.
- virtual bool **VehicleQueueOutIsFull** (int Destination)=0
  Vehicle queue out is full.

Protected Attributes

- int **mPhasesCount**
  Counter for the Number of phases in the TrafficModel object.
- int **mPhasesMax**
  Maximum number of phases.
- **PhaseClass** ** **mPhasesList**
  *Pointer to an array of **PhaseClass** objects.*
- unsigned int **mActivePhaseIndex**
  The index of the active phase.
- int **mVehiclesIn**
  The number of vehicles coming into the TrafficNode object.
- int **mVehicleQueuesOut**
  The vehicle queues out.
- int **mVehicleQueuesOutCount**
  The number of VehicleQueues outgoing to the intersection.
- **TrafficModelObject** ** **mVehicleQueuesOutList**
  Pointer to an array of the VehicleQueues outgoing to the intersection.
- **TrafficModelObject** * **mTrafficModel**
  Pointer to the TrafficModel object.
- bool **mIdle**
  Flag indicating the status of the Intersection.
- **Event** * **mNextPhaseChangeScheduledEvent**
  Event queue for all listeners interested in NextPhaseChangeScheduled events.
- int **mIndexInList**
  List of indicies of all incoming queues to the intersection.

Detailed Description

TrafficNodeClass Object.

Definition at line 9 of file TrafficNodeClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::TrafficNodeClass::TrafficNodeClass (void)

Initializes a new instance of the **TrafficNodeClass** class.

Default constructor.

Definition at line 12 of file TrafficNodeClass.cpp.

TrafficModelObjectsLibrary::TrafficNodeClass::~TrafficNodeClass (void)

Finalizes an instance of the **TrafficNodeClass** class.

Destructor.

Definition at line 18 of file TrafficNodeClass.cpp.

Member Function Documentation

PhaseClass * TrafficModelObjectsLibrary::TrafficNodeClass::ActivePhase (void)

Gets a pointer to the active phase object.

Gets the active phase.

Returns:

null if it fails

Returns:

null if it fails, else.

Definition at line 124 of file TrafficNodeClass.cpp.

int TrafficModelObjectsLibrary::TrafficNodeClass::ActivePhaseIndex (void)

Gets the index of the active phase of an intersection.

Gets the active phase index.

Returns:

Integer.

Returns:

.

Definition at line 116 of file TrafficNodeClass.cpp.

void TrafficModelObjectsLibrary::TrafficNodeClass::AddVehicleQueueOut (TrafficModelObject * *QueueOut*)

Adds a vehicle queue out.

Parameters:

QueueOut	[in,out] If non-null, the queue out.

Parameters:

QueueOut	[in,out] If non-null, the queue out.

Definition at line 156 of file TrafficNodeClass.cpp.

double TrafficModelObjectsLibrary::TrafficNodeClass::ChangePhase (void)

Changes the Active Phase and returns the Time when the next phase change must occur.

Returns:

Time of duration of the phase.

Definition at line 25 of file TrafficNodeClass.cpp.

PhaseClass * TrafficModelObjectsLibrary::TrafficNodeClass::GetPhase (int *Index*)

Gets a phase object at location "index".

Parameters:

Index	Zero-based index of the Phase objects

Returns:

null if it fails, else the phase.

Parameters:

Index	Zero-based index of the Phase objects.

Returns:

null if it fails, else the phase.

Reimplemented in **TrafficModelObjectsLibrary::IntersectionClass** (*p.38*).

Definition at line 134 of file TrafficNodeClass.cpp.

virtual TrafficModelObject* TrafficModelObjectsLibrary::TrafficNodeClass::GetVehicleQueueOut (int *Destination*) `[pure virtual]`

Gets a vehicle queue out.

Parameters:

Destination	Destination for the.

Returns:

null if it fails, else the vehicle queue out.

Implemented in **TrafficModelObjectsLibrary::DestinationClass** (*p.35*), and **TrafficModelObjectsLibrary::IntersectionClass** (*p.39*).

bool TrafficModelObjectsLibrary::TrafficNodeClass::IsEmpty (void)

Query if this object is empty.

Returns:

true if empty, false if not.

Returns:

true if empty, false if not.

Definition at line 177 of file TrafficNodeClass.cpp.

bool TrafficModelObjectsLibrary::TrafficNodeClass::IsIdle (void)

Query if this object is idle.

Returns:

true if idle, false if not.

Returns:

true if idle, false if not.

Definition at line 39 of file TrafficNodeClass.cpp.

Event * TrafficModelObjectsLibrary::TrafficNodeClass::NextPhaseChangeScheduledEvent (void)

Gets the next phase change scheduled event.

Returns:

null if it fails, else.

Returns:

null if it fails, else.

Definition at line 55 of file TrafficNodeClass.cpp.

int TrafficModelObjectsLibrary::TrafficNodeClass::PhasesCount (void)

Gets the phases count of an intersection.

Gets the phases count.

Returns:

Integer - Number of phases of an intersection

Returns:

.

Definition at line 107 of file TrafficNodeClass.cpp.

double TrafficModelObjectsLibrary::TrafficNodeClass::SetActivePhase (int *Index*)

Sets an active phase.

Sets a phase to active state and returns the time of duration of the phase.

Parameters:

Index	Zero-based index of the.

Returns:

.

Parameters:

Index	Zero-based index of the active phase.

Returns:

Time of duration of the phase.

Definition at line 75 of file TrafficNodeClass.cpp.

double TrafficModelObjectsLibrary::TrafficNodeClass::SetNextActivePhase (void)

Sets the next active phase.

Sets the next phase to active state and returns the time of duration of the phase.

Returns:

.

Returns:

Time of duration of the phase.

Definition at line 89 of file TrafficNodeClass.cpp.

void TrafficModelObjectsLibrary::TrafficNodeClass::SetNextPhaseChangeScheduledEvent (Event * *ptrEvent*)

Sets a next phase change scheduled event.

Parameters:

ptrEvent	[in,out] If non-null, the pointer event.

Parameters:

ptrEvent	[in,out] If non-null, the pointer event.

Definition at line 63 of file TrafficNodeClass.cpp.

TrafficModelObject * TrafficModelObjectsLibrary::TrafficNodeClass::TrafficModel (void)

Get a pointer to the TrafficModel object.

Gets the traffic model.

Returns:

null if it fails, else.

Returns:

null if it fails, else.

Definition at line 142 of file TrafficNodeClass.cpp.

void TrafficModelObjectsLibrary::TrafficNodeClass::VehicleIn (void)

Vehicle in.

virtual void TrafficModelObjectsLibrary::TrafficNodeClass::VehicleIn (VehicleClass * *Vehicle*) [pure virtual]

Vehicle in.

Parameters:

Vehicle	[in,out] If non-null, the vehicle.

Implemented in **TrafficModelObjectsLibrary::DestinationClass** (*p.35*), and **TrafficModelObjectsLibrary::IntersectionClass** (*p.39*).

void TrafficModelObjectsLibrary::TrafficNodeClass::VehicleOut (void)

Vehicle out.

Definition at line 148 of file TrafficNodeClass.cpp.

virtual bool TrafficModelObjectsLibrary::TrafficNodeClass::VehicleQueueOutIsFull (int *Destination*) [pure virtual]

Vehicle queue out is full.

Parameters:

Destination	Destination for the.

Returns:

true if it succeeds, false if it fails.

Implemented in **TrafficModelObjectsLibrary::DestinationClass** (*p.35*), and **TrafficModelObjectsLibrary::IntersectionClass** (*p.39*).

int TrafficModelObjectsLibrary::TrafficNodeClass::VehicleQueuesOutCount (void)

Gets the vehicle queues out count.

Returns:

.

Returns:

.

Definition at line 169 of file TrafficNodeClass.cpp.

Member Data Documentation

unsigned int TrafficModelObjectsLibrary::TrafficNodeClass::mActivePhaseIndex [protected]

The index of the active phase.

Definition at line 21 of file TrafficNodeClass.h.

bool TrafficModelObjectsLibrary::TrafficNodeClass::mIdle [protected]

Flag indicating the status of the Intersection.

Definition at line 36 of file TrafficNodeClass.h.

int TrafficModelObjectsLibrary::TrafficNodeClass::mIndexInList [protected]

List of indicies of all incoming queues to the intersection.

Definition at line 41 of file TrafficNodeClass.h.

Event* TrafficModelObjectsLibrary::TrafficNodeClass::mNextPhaseChangeScheduledEvent [protected]

Event queue for all listeners interested in NextPhaseChangeScheduled events.

Definition at line 38 of file TrafficNodeClass.h.

int TrafficModelObjectsLibrary::TrafficNodeClass::mPhasesCount [protected]

Counter for the Number of phases in the TrafficModel object.

Definition at line 14 of file TrafficNodeClass.h.

PhaseClass** TrafficModelObjectsLibrary::TrafficNodeClass::mPhasesList [protected]

Pointer to an array of **PhaseClass** objects.

Definition at line 18 of file TrafficNodeClass.h.

int TrafficModelObjectsLibrary::TrafficNodeClass::mPhasesMax [protected]

Maximum number of phases.

Definition at line 16 of file TrafficNodeClass.h.

TrafficModelObject* TrafficModelObjectsLibrary::TrafficNodeClass::mTrafficModel [protected]

Pointer to the TrafficModel object.

Definition at line 33 of file TrafficNodeClass.h.

int TrafficModelObjectsLibrary::TrafficNodeClass::mVehicleQueuesOut [protected]

The vehicle queues out.

Definition at line 27 of file TrafficNodeClass.h.

int TrafficModelObjectsLibrary::TrafficNodeClass::mVehicleQueuesOutCount [protected]

The number of VehicleQueues outgoing to the intersection.

Definition at line 29 of file TrafficNodeClass.h.

TrafficModelObject** TrafficModelObjectsLibrary::TrafficNodeClass::mVehicleQueuesOutList [protected]

Pointer to an array of the VehicleQueues outgoing to the intersection.

Definition at line 31 of file TrafficNodeClass.h.

int TrafficModelObjectsLibrary::TrafficNodeClass::mVehiclesIn [protected]

The number of vehicles coming into the TrafficNode object.

Definition at line 24 of file TrafficNodeClass.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**TrafficNodeClass.h**
- D:/Training/TM/**TrafficNodeClass.cpp**

TrafficModelObjectsLibrary::DestinationClass Class Reference

Destination class.
`#include <DestinationClass.h>`
Inherits **TrafficModelObjectsLibrary::TrafficNodeClass**.

Public Member Functions

- **DestinationClass** (void)
  Initializes a new instance of the Destination Class.
- **DestinationClass** (int Index, **TrafficModelObject** ***TrafficModel**, double AverageCrossingTime)
  Initializes a new instance of the Destination Class.
- **~DestinationClass** (void)
  *Finalizes an instance of the **DestinationClass** class.*
- int **GetVehiclesServed** (void)
  Gets Number of vehicles served.
- double **GetCummulativeTime** (void)
  Gets the cummulative time of travel for the vehicle.
- double **GetAverageTime** (void)
  Gets the average time.
- virtual **TrafficModelObject** * **GetVehicleQueueOut** (int Destination)
  Gets the vehicle queue which serves the destination given.
- virtual void **VehicleIn** (**VehicleClass** *Vehicle)
  Adds the vehicle given to the destination.
- virtual bool **VehicleQueueOutIsFull** (int Destination)
  Vehicle queue out is full.

Protected Attributes

- double **mAverageCrossingTime**
  The average crossing time for a vehicle.
- int **mVehiclesServed**
  A counter for the number of vehilces served.
- double **mCummulativeTime**
  Cummulative Time used to keep track of vehicle travel time.

Detailed Description

Destination class.

Definition at line 7 of file DestinationClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::DestinationClass::DestinationClass (void) `[inline]`

Initializes a new instance of the Destination Class.

Definition at line 22 of file DestinationClass.h.

TrafficModelObjectsLibrary::DestinationClass::DestinationClass (int *Index*, TrafficModelObject * *TrafficModel*, double *AverageCrossingTime*)

Initializes a new instance of the Destination Class.

Parameters:

Index	Index of the destination in the traffic node list.
TrafficModel	pointer to the Model.
AverageCrossingTime	Time of the average crossing.

Parameters:

Index	Index of the destination in the traffic node list.
TrafficModel	[in,out] pointer to the Model.
AverageCrossingTime	Time of the average crossing.

Definition at line 16 of file DestinationClass.cpp.

TrafficModelObjectsLibrary::DestinationClass::~DestinationClass (void) `[inline]`

Finalizes an instance of the **DestinationClass** class.

Definition at line 32 of file DestinationClass.h.

Member Function Documentation

double TrafficModelObjectsLibrary::DestinationClass::GetAverageTime (void)

Gets the average time.

Returns:
The average time.

Returns:
The average time.

Definition at line 40 of file DestinationClass.cpp.

double TrafficModelObjectsLibrary::DestinationClass::GetCummulativeTime (void) `[inline]`

Gets the cummulative time of travel for the vehicle.

Returns:
The the cummulative travel time.

Definition at line 42 of file DestinationClass.h.

virtual TrafficModelObject* TrafficModelObjectsLibrary::DestinationClass::GetVehicleQueueOut (int *Destination*) `[inline, virtual]`

Gets the vehicle queue which serves the destination given.

Parameters:

Destination	Destination of the vehicle.

Returns:

always null. there are no vehicle queues going aout of a destination.

Implements **TrafficModelObjectsLibrary::TrafficNodeClass** (*p.27*).

Definition at line 54 of file DestinationClass.h.

int TrafficModelObjectsLibrary::DestinationClass::GetVehiclesServed (void) `[inline]`

Gets Number of vehicles served.

Returns:

The number of the number of vehicles served in the Model.

Definition at line 37 of file DestinationClass.h.

void TrafficModelObjectsLibrary::DestinationClass::VehicleIn (VehicleClass * *Vehicle*) `[virtual]`

Adds the vehicle given to the destination.

Parameters:

Vehicle	pointer to the vehicle reaching the destination.

Parameters:

Vehicle	[in,out] pointer to the vehicle reaching the destination.

Implements **TrafficModelObjectsLibrary::TrafficNodeClass** (*p.29*).

Definition at line 55 of file DestinationClass.cpp.

virtual bool TrafficModelObjectsLibrary::DestinationClass::VehicleQueueOutIsFull (int *Destination*) `[inline, virtual]`

Vehicle queue out is full.

Parameters:

Destination	Destination of the vehicle.

Returns:

Always return false, since it is the destination and reached the end. There is no Vehicle Out Queue.

Implements **TrafficModelObjectsLibrary::TrafficNodeClass** (*p.30*).

Definition at line 66 of file DestinationClass.h.

Member Data Documentation

double TrafficModelObjectsLibrary::DestinationClass::mAverageCrossingTime [protected]

The average crossing time for a vehicle.

Definition at line 12 of file DestinationClass.h.

double TrafficModelObjectsLibrary::DestinationClass::mCummulativeTime [protected]

Cummulative Time used to keep track of vehicle travel time.

Definition at line 18 of file DestinationClass.h.

int TrafficModelObjectsLibrary::DestinationClass::mVehiclesServed [protected]

A counter for the number of vehilces served.

Definition at line 15 of file DestinationClass.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**DestinationClass.h**
- D:/Training/TM/**DestinationClass.cpp**

TrafficModelObjectsLibrary::IntersectionClass Class Reference

Intersection class.
`#include <IntersectionClass.h>`
Inherits **TrafficModelObjectsLibrary::TrafficNodeClass**.

Public Member Functions

- **IntersectionClass** (void)
  *Initializes a new instance of the **IntersectionClass** class.*
- **IntersectionClass** (int Index, **TrafficModelObject** *__TrafficModel__, int PhasesMax, int QueuesOut)
  *Initializes a new instance of the **IntersectionClass** class.*
- **~IntersectionClass** (void)
  *Finalizes an instance of the **IntersectionClass** class.*
- **PhaseClass** * **AddPhase** (int QueuesIn, double AverageServiceTime)
  Creates a phase and returns the pointer to it.
- **PhaseClass** * **GetPhase** (int Index)
  Gets the pointer at the given index in the pointer phases list of the intersection.
- virtual **TrafficModelObject** * **GetVehicleQueueOut** (int Destination)
  Gets the vehicle queue going out of the intersection which serves the destination given.
- virtual void **VehicleIn** (**VehicleClass** *Vehicle)
  Adds the vehicle given to the intersection.
- virtual bool **VehicleQueueOutIsFull** (int Destination)
  Checks if the vehicle queue going out of the intersection and serving the given destination is full.

Detailed Description

Intersection class.

Definition at line 5 of file IntersectionClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::IntersectionClass::IntersectionClass (void)

Initializes a new instance of the **IntersectionClass** class.

Definition at line 13 of file IntersectionClass.cpp.

TrafficModelObjectsLibrary::IntersectionClass::IntersectionClass (int *Index*, TrafficModelObject * *TrafficModel*, int *PhasesMax*, int *QueuesOut*)

Initializes a new instance of the **IntersectionClass** class.

Parameters:

Index	Index of the intersection in the traffic node list.
TrafficModel	pointer to the model.
PhasesMax	The maximum number of phases for the current intersection.
QueuesOut	The number of vehicle Queues going out of the intersection.

Parameters:

Index	Index of the intersection in the traffic node list.
TrafficModel	[in,out] pointer to the model.
PhasesMax	The maximum number of phases for the current intersection.
QueuesOut	The number of vehicle Queues going out of the intersection.

Definition at line 24 of file IntersectionClass.cpp.

TrafficModelObjectsLibrary::IntersectionClass::~IntersectionClass (void) `[inline]`

Finalizes an instance of the **IntersectionClass** class.

Definition at line 22 of file IntersectionClass.h.

Member Function Documentation

PhaseClass * TrafficModelObjectsLibrary::IntersectionClass::AddPhase (int *QueuesIn*, double *mAverageServiceTime*)

Creates a phase and returns the pointer to it.

Parameters:

QueuesIn	The number of queues delivering cars to the intersection at the current phase.
AverageServiceTime	The interval of time at which the phase is going to run .

Returns:

null if it fails, returns pointers of current created phase.

Parameters:

QueuesIn	The number of queues delivering cars to the intersection at the current phase.
mAverageServiceTime	Time of the average service.

Returns:

null if it fails, returns pointers of current created phase.

Definition at line 53 of file IntersectionClass.cpp.

PhaseClass * TrafficModelObjectsLibrary::IntersectionClass::GetPhase (int *Index*)

Gets the pointer at the given index in the pointer phases list of the intersection.

Parameters:

Index	Index of the pointer to the required phase.

Returns:

null if it fails, else returns the pointer to the Phase.

Parameters:

Index	Index of the pointer to the required phase.

Returns:

null if it fails, else returns the pointer to the Phase.

Reimplemented from **TrafficModelObjectsLibrary::TrafficNodeClass** (*p.27*).

Definition at line 77 of file IntersectionClass.cpp.

TrafficModelObject * TrafficModelObjectsLibrary::IntersectionClass::GetVehicleQueueOut (int *Destination*) [virtual]

Gets the vehicle queue going out of the intersection which serves the destination given.

Parameters:

Destination	Destination of the vehicle using the intersection.

Returns:

null if it fails, else the pointer to the vehicle queue serving the given destination.

Parameters:

Destination	Destination of the vehicle using the intersection.

Returns:

null if it fails, else the pointer to the vehicle queue serving the given destination.

Implements **TrafficModelObjectsLibrary::TrafficNodeClass** (*p.27*).

Definition at line 91 of file IntersectionClass.cpp.

void TrafficModelObjectsLibrary::IntersectionClass::VehicleIn (VehicleClass * *Vehicle*) [virtual]

Adds the vehicle given to the intersection.

Parameters:

Vehicle	pointer to the vehicle reaching the intersection.

Parameters:

Vehicle	[in,out] pointer to the vehicle reaching the intersection.

Implements **TrafficModelObjectsLibrary::TrafficNodeClass** (*p.29*).

Definition at line 106 of file IntersectionClass.cpp.

bool TrafficModelObjectsLibrary::IntersectionClass::VehicleQueueOutIsFull (int *Destination*) [virtual]

Checks if the vehicle queue going out of the intersection and serving the given destination is full.

Parameters:

Destination	Destination of the vehicle.

Returns:

The full status of the vehicle queue serving the destination out of the current intersection.

Parameters:

Destination	Destination of the vehicle.

Returns:

The full status of the vehicle queue serving the destination out of the current intersection.

Implements **TrafficModelObjectsLibrary::TrafficNodeClass** (*p.30*).

Definition at line 125 of file IntersectionClass.cpp.

The documentation for this class was generated from the following files:

- D:/Training/TM/**IntersectionClass.h**
- D:/Training/TM/**IntersectionClass.cpp**

TrafficModelObjectsLibrary::VehicleQueueClass Class Reference

Vehicle queue class.

`#include <VehicleQueueClass.h>`

Inherits **TrafficModelObjectsLibrary::TrafficModelObject**.

Inherited by **TrafficModelObjectsLibrary::ParkingLotClass**, and **TrafficModelObjectsLibrary::RoadSegmentClass**.

Public Member Functions

- **VehicleQueueClass** (void)
  *Initializes a new instance of the **VehicleQueueClass** class.*
- **~VehicleQueueClass** (void)
  *Finalizes an instance of the **VehicleQueueClass** class.*
- virtual **VehicleClass** * **VehicleOut** (double Time)=0
  VehicleOut.
- void **VehicleIn** (**VehicleClass** *Vehicle)
  VehicleIn.
- int **VehiclesReady** (void)
  Gets the vehicles ready.
- int **Exits** (void)
  Gets the exits.
- int **Capacity** (void)
  Check the number of exits of a VehicleQueue object.
- bool **FrontRowClear** (void)
  Checks whether the head of the queue is clear.
- void **SetDestination** (int Index, int DestinationID, double IntersectionCrossingTime)
  Sets a destination for the VehicleQueue.
- void **AddVehicleReady** (**VehicleClass** *Vehicle)
  Adds a vehicle to the VehicleQueue if the queue has space. If not, an event is scheduled latert on to add the vehicle.
- **TrafficNodeClass** * **EndTrafficNode** (void)
  Returns the end traffic node of a VehicleQueue.
- **PhaseClass** * **Phase** (void)
  Gets a pointer to the current phase (object) of the VehicleQueue.
- bool **IsFull** (void)
  Query if this VehicleQueue objeect is full.
- double **GetIntersectionCrossingTime** (int Destination)
  Gets an intersection crossing time.
- bool **IsValidDestination** (int Destination)
  Query if 'Destination' is valid destination.
- bool **HasVehiclesReady** (void)
  Query if this VehicleQueue object has vehicles ready.
- int **GetNextVehicleDestination** (void)
  Gets the next vehicle destination.
- double **GetAverageTravelTime** (void)
  Gets the average travel time.

- int **IndexInList** (void)
  Gets the index in list.

Protected Attributes

- **TrafficModelObject** * **mTrafficModel**
  ffic model object. Acts as a container for other simulation objects like Vehicles
- int **mVehiclesinQueue**
  Number of vehicle in a queue (road) at the current instant.
- int **mVehiclesReady**
  Number of vehicles ready to be processed off the queue.
- int **mCapacity**
  Capacity (in vehicles) of the VehicleQueue.
- int **mExits**
  The number of exits of a vehicle queue.
- int **mDestinations**
  The number of destinations that a VehicleQueue serves.
- int * **mDestinationIDs**
  A pointer to an array of Destination IDs that are served by the VehicleQueue.
- int **mDestinationCount**
  The number of destinations that the queue(road) serves.
- double * **mIntersectionCrossingTime**
  Average crossing time of the Intersection.
- double **mAverageTravelTime**
  Average time that it takes a vehicle to travel the length of the road.
- **VehicleClass** * **mVehicleReadyFront**
  The number of vehicles that are ready at the front of the queue to be processed.
- **VehicleClass** * **mVehicleReadyLast**
  The number of vehicles that are "ready" at the end of the queue.
- int **mEndTrafficNodeIndex**
  ID of the ending traffic node that is incident to the VehicleQueue.
- **TrafficNodeClass** * **ptrEndTrafficNode**
  A pointer to the ending traffic node (incident to the VehicleQueue)
- int **mPhaseIndex**
  Indicator of the current phase (starts from zero)
- **PhaseClass** * **mPhase**
  A pointer to the phase of the VehicleQueue.
- int **mIndexInList**
  List of index ins.

Detailed Description

Vehicle queue class.

Definition at line 7 of file VehicleQueueClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::VehicleQueueClass::VehicleQueueClass (void)

Initializes a new instance of the **VehicleQueueClass** class.

Default constructor.

Definition at line 11 of file VehicleQueueClass.cpp.

TrafficModelObjectsLibrary::VehicleQueueClass::~VehicleQueueClass (void)

Finalizes an instance of the **VehicleQueueClass** class.

Destructor.

Definition at line 21 of file VehicleQueueClass.cpp.

Member Function Documentation

void TrafficModelObjectsLibrary::VehicleQueueClass::AddVehicleReady (VehicleClass * *Vehicle*)

Adds a vehicle to the VehicleQueue if the queue has space. If not, an event is scheduled latert on to add the vehicle.

Parameters:

Vehicle	[in,out] If non-null, the vehicle.

Parameters:

Vehicle	[in,out] If non-null, the vehicle.

Definition at line 70 of file VehicleQueueClass.cpp.

int TrafficModelObjectsLibrary::VehicleQueueClass::Capacity (void)

Check the number of exits of a VehicleQueue object.

Gets the capacity.

Returns:

Integer representing number of exits of a VehicleQueue

Returns:

.

Definition at line 36 of file VehicleQueueClass.cpp.

TrafficNodeClass * TrafficModelObjectsLibrary::VehicleQueueClass::EndTrafficNode (void)

Returns the end traffic node of a VehicleQueue.

Returns:

null if it fails

Returns:

null if it fails.

Definition at line 90 of file VehicleQueueClass.cpp.

int TrafficModelObjectsLibrary::VehicleQueueClass::Exits (void)

Gets the exits.

Returns:

.

Returns:

.

Definition at line 44 of file VehicleQueueClass.cpp.

bool TrafficModelObjectsLibrary::VehicleQueueClass::FrontRowClear (void)

Checks whether the head of the queue is clear.

Determines if we can front row clear.

Returns:

true if clear, false if not clear.

Returns:

true if it succeeds, false if it fails.

Definition at line 52 of file VehicleQueueClass.cpp.

double TrafficModelObjectsLibrary::VehicleQueueClass::GetAverageTravelTime (void)

Gets the average travel time.

Returns:

Double representing the average travel time.

Returns:

The average travel time.

Definition at line 209 of file VehicleQueueClass.cpp.

double TrafficModelObjectsLibrary::VehicleQueueClass::GetIntersectionCrossingTime (int *Destination*)

Gets an intersection crossing time.

Parameters:

Destination	Destination for the Vehicle

Returns:

The intersection crossing time.

Parameters:

Destination	Destination for the Vehicle.

Returns:

The intersection crossing time.

Definition at line 125 of file VehicleQueueClass.cpp.

int TrafficModelObjectsLibrary::VehicleQueueClass::GetNextVehicleDestination (void)

Gets the next vehicle destination.

Returns:
The next vehicle destination.

Returns:
The next vehicle destination.

Definition at line 194 of file VehicleQueueClass.cpp.

bool TrafficModelObjectsLibrary::VehicleQueueClass::HasVehiclesReady (void)

Query if this VehicleQueue object has vehicles ready.

Query if this object has vehicles ready.

Returns:
true if vehicles ready, false if not.

Returns:
true if vehicles ready, false if not.

Definition at line 158 of file VehicleQueueClass.cpp.

int TrafficModelObjectsLibrary::VehicleQueueClass::IndexInList (void) `[inline]`

Gets the index in list.

Returns:
Integer represting the object's index in the list.

Definition at line 165 of file VehicleQueueClass.h.

bool TrafficModelObjectsLibrary::VehicleQueueClass::IsFull (void)

Query if this VehicleQueue objeect is full.

Query if this object is full.

Returns:
true if full, false if not.

Returns:
true if full, false if not.

Definition at line 106 of file VehicleQueueClass.cpp.

bool TrafficModelObjectsLibrary::VehicleQueueClass::IsValidDestination (int *Destination*)

Query if 'Destination' is valid destination.

Parameters:

Destination	Destination for the Vehicle

Returns:
true if valid destination, false if not.

Parameters:

Destination	Destination for the Vehicle.

Returns:

true if valid destination, false if not.

Definition at line 143 of file VehicleQueueClass.cpp.

PhaseClass * TrafficModelObjectsLibrary::VehicleQueueClass::Phase (void)

Gets a pointer to the current phase (object) of the VehicleQueue.

Returns:

null if it fails, else.

Definition at line 98 of file VehicleQueueClass.cpp.

void TrafficModelObjectsLibrary::VehicleQueueClass::SetDestination (int *Index*, int *DestinationID*, double *IntersectionCrossingTime*)

Sets a destination for the VehicleQueue.

Parameters:

Index	Zero-based index of destination
DestinationID	Identifier for the destination.
IntersectionCrossingTime	Time of the intersection crossing.

Parameters:

Index	Zero-based index of destination.
DestinationID	Identifier for the destination.
IntersectionCrossingTime	Time of the intersection crossing.

Definition at line 175 of file VehicleQueueClass.cpp.

void TrafficModelObjectsLibrary::VehicleQueueClass::VehicleIn (VehicleClass * *Vehicle*)

VehicleIn.

Parameters:

Vehicle	[in,out] If non-null, the vehicle.

Parameters:

Vehicle	[in,out] If non-null, the vehicle.

Definition at line 186 of file VehicleQueueClass.cpp.

virtual VehicleClass* TrafficModelObjectsLibrary::VehicleQueueClass::VehicleOut (double *Time*) [pure virtual]

VehicleOut.

Parameters:

Time	The time.

Returns:

Method that processes a vehicle at time=Time.

Implemented in **TrafficModelObjectsLibrary::RoadSegmentClass** (*p.53*), and **TrafficModelObjectsLibrary::ParkingLotClass** (*p.51*).

int TrafficModelObjectsLibrary::VehicleQueueClass::VehiclesReady (void)

Gets the vehicles ready.

Returns:

.

Returns:

.

Definition at line 28 of file VehicleQueueClass.cpp.

Member Data Documentation

double TrafficModelObjectsLibrary::VehicleQueueClass::mAverageTravelTime [protected]

Average time that it takes a vehicle to travel the length of the road.

Definition at line 31 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mCapacity [protected]

Capacity (in vehicles) of the VehicleQueue.

Definition at line 19 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mDestinationCount [protected]

The number of destinations that the queue(road) serves.

Definition at line 27 of file VehicleQueueClass.h.

int* TrafficModelObjectsLibrary::VehicleQueueClass::mDestinationIDs [protected]

A pointer to an array of Destination IDs that are served by the VehicleQueue.

Definition at line 25 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mDestinations [protected]

The number of destinations that a VehicleQueue serves.

Definition at line 23 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mEndTrafficNodeIndex [protected]

ID of the ending traffic node that is incident to the VehicleQueue.

Definition at line 38 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mExits [protected]

The number of exits of a vehicle queue.

Definition at line 21 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mIndexInList [protected]

List of index ins.

Definition at line 47 of file VehicleQueueClass.h.

double* TrafficModelObjectsLibrary::VehicleQueueClass::mIntersectionCrossingTime [protected]

Average crossing time of the Intersection.

Definition at line 29 of file VehicleQueueClass.h.

PhaseClass* TrafficModelObjectsLibrary::VehicleQueueClass::mPhase [protected]

A pointer to the phase of the VehicleQueue.

Definition at line 44 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mPhaseIndex [protected]

Indicator of the current phase (starts from zero)

Definition at line 42 of file VehicleQueueClass.h.

TrafficModelObject* TrafficModelObjectsLibrary::VehicleQueueClass::mTrafficModel [protected]

ffic model object. Acts as a container for other simulation objects like Vehicles

Definition at line 13 of file VehicleQueueClass.h.

VehicleClass* TrafficModelObjectsLibrary::VehicleQueueClass::mVehicleReadyFront [protected]

The number of vehicles that are ready at the front of the queue to be processed.

Definition at line 33 of file VehicleQueueClass.h.

VehicleClass* TrafficModelObjectsLibrary::VehicleQueueClass::mVehicleReadyLast [protected]

The number of vehicles that are "ready" at the end of the queue.

Definition at line 35 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mVehiclesinQueue [protected]

Number of vehicle in a queue (road) at the current instant.

Definition at line 15 of file VehicleQueueClass.h.

int TrafficModelObjectsLibrary::VehicleQueueClass::mVehiclesReady [protected]

Number of vehicles ready to be processed off the queue.

Definition at line 17 of file VehicleQueueClass.h.

TrafficNodeClass* TrafficModelObjectsLibrary::VehicleQueueClass::ptrEndTrafficNode [protected]

A pointer to the ending traffic node (incident to the VehicleQueue)

Definition at line 40 of file VehicleQueueClass.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**VehicleQueueClass.h**
- D:/Training/TM/**VehicleQueueClass.cpp**

TrafficModelObjectsLibrary::ParkingLotClass Class Reference

Parking lot class.
`#include <ParkingLotClass.h>`
Inherits **TrafficModelObjectsLibrary::VehicleQueueClass**.

Public Member Functions

- **ParkingLotClass** (void)
  *Initializes a new instance of the **ParkingLotClass** class.*
- **ParkingLotClass** (int Index, **TrafficModelObject** *TrafficModel, int **Capacity**, int **Exits**, int Intersection, int **Phase**, int Destinations)
  *Initializes a new instance of the **ParkingLotClass** class.*
- **~ParkingLotClass** (void)
  *Finalizes an instance of the **ParkingLotClass** class.*
- virtual **VehicleClass** * **VehicleOut** (double Time)
  Gets the vehicle out of the parking lot and passes it to the intersection .

Detailed Description

Parking lot class.

Definition at line 7 of file ParkingLotClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::ParkingLotClass::ParkingLotClass (void)

Initializes a new instance of the **ParkingLotClass** class.

Definition at line 12 of file ParkingLotClass.cpp.

TrafficModelObjectsLibrary::ParkingLotClass::ParkingLotClass (int *Index*, TrafficModelObject * *TrafficModel*, int *Capacity*, int *Exits*, int *EndTrafficNodeIndex*, int *Phase*, int *Destinations*)

Initializes a new instance of the **ParkingLotClass** class.

Parameters:

Index	Index of the pointer to the parking lot into the Vehicle Queue list of the Model
TrafficModel	Pointer to the model.
Capacity	The capacity of the parking lot.
Exits	The number of exit for the parking lot.
Intersection	Traffic node index corresponding the intersection outside the parking lot (End node).
Phase	The phase that serves the traffic coming out of the parking lot.
Destinations	The number of destinations of the vehicles in the parking lot.

Parameters:

Index	Index of the pointer to the parking lot into the Vehicle Queue list of the Model.
TrafficModel	[in,out] Pointer to the model.
Capacity	The capacity of the parking lot.
Exits	The number of exit for the parking lot.
EndTrafficNodeIndex	The end traffic node index.
Phase	The phase that serves the traffic coming out of the parking lot.
Destinations	The number of destinations of the vehicles in the parking lot.

Definition at line 28 of file ParkingLotClass.cpp.

TrafficModelObjectsLibrary::ParkingLotClass::~ParkingLotClass (void)

Finalizes an instance of the **ParkingLotClass** class.

Destructor.

Definition at line 61 of file ParkingLotClass.cpp.

Member Function Documentation

VehicleClass * TrafficModelObjectsLibrary::ParkingLotClass::VehicleOut (double *Time*) [virtual]

Gets the vehicle out of the parking lot and passes it to the intersection .

Vehicle out.

Parameters:

Time	The time that the vehicle will take to cross the intersection ahead .

Returns:

null if it fails, returns a pointer to the vehicle issued by the parking lot.

Parameters:

Time	The time.

Returns:

null if it fails, else.

Implements **TrafficModelObjectsLibrary::VehicleQueueClass** (*p.46*).

Definition at line 70 of file ParkingLotClass.cpp.

The documentation for this class was generated from the following files:

- D:/Training/TM/**ParkingLotClass.h**
- D:/Training/TM/**ParkingLotClass.cpp**

TrafficModelObjectsLibrary::RoadSegmentClass Class Reference

Road segment class.
`#include <RoadSegmentClass.h>`
Inherits **TrafficModelObjectsLibrary::VehicleQueueClass**.

Public Member Functions

- **RoadSegmentClass** (void)
  *Initializes a new instance of the **RoadSegmentClass** class.*
- **RoadSegmentClass** (int Index, **TrafficModelObject** *TrafficModel, int **Capacity**, double AverageTravelTime, int Lanes, int StartTrafficNode, int **EndTrafficNode**, int **Phase**, int DestinationsCount)
  *Initializes a new instance of the **RoadSegmentClass** class.*
- **~RoadSegmentClass** (void)
  *Finalizes an instance of the **RoadSegmentClass** class.*
- virtual **VehicleClass** * **VehicleOut** (double Time)
  Gets the vehicle out of the road segment and passes it to the intersection .

Protected Attributes

- int **mStartTrafficNodeIndex**
  The index of the node at which the road segment starts.
- **TrafficNodeClass** * **ptrStartTrafficNode**
  The pointer to the node that starts the road segment.

Detailed Description

Road segment class.

Definition at line 7 of file RoadSegmentClass.h.

Constructor & Destructor Documentation

TrafficModelObjectsLibrary::RoadSegmentClass::RoadSegmentClass (void)

Initializes a new instance of the **RoadSegmentClass** class.

Definition at line 12 of file RoadSegmentClass.cpp.

TrafficModelObjectsLibrary::RoadSegmentClass::RoadSegmentClass (int *Index*, TrafficModelObject * *TrafficModel*, int *Capacity*, double *AverageTravelTime*, int *Lanes*, int *StartTrafficNode*, int *EndTrafficNode*, int *Phase*, int *Destinations*)

Initializes a new instance of the **RoadSegmentClass** class.

Parameters:

Index	Index of the pointer to the Road Segment into the Vehicle Queue list of the Model.
TrafficModel	Pointer to the model.
Capacity	The capacity of the road segment.
AverageTravelTime	The average time is takes a vehicle to cross the road segment freely.
Lanes	The number of lanes for the road segment.
StartTrafficNode	The node at which the road segment starts.
EndTrafficNode	The node at which the road segment ends.
Phase	The phase at which the vehicle queue in is going to be served at the intersection ahead.
DestinationsCount	The number of destinations the road segment serves.

Parameters:

Index	Index of the pointer to the Road Segment into the Vehicle Queue list of the Model.
TrafficModel	[in,out] Pointer to the model.
Capacity	The capacity of the road segment.
AverageTravelTime	The average time is takes a vehicle to cross the road segment freely.
Lanes	The number of lanes for the road segment.
StartTrafficNode	The node at which the road segment starts.
EndTrafficNode	The node at which the road segment ends.
Phase	The phase at which the vehicle queue in is going to be served at the intersection ahead.
Destinations	The destinations.

Definition at line 28 of file RoadSegmentClass.cpp.

TrafficModelObjectsLibrary::RoadSegmentClass::~RoadSegmentClass (void)

Finalizes an instance of the **RoadSegmentClass** class.

Definition at line 55 of file RoadSegmentClass.cpp.

Member Function Documentation

VehicleClass * TrafficModelObjectsLibrary::RoadSegmentClass::VehicleOut (double *Time*) `[virtual]`

Gets the vehicle out of the road segment and passes it to the intersection .

Parameters:

Time	The time that the vehicle will take to cross the intersection ahead .

Returns:

null if it fails, returns a pointer to the vehicle issued by the road segment.

Parameters:

Time	The time that the vehicle will take to cross the intersection ahead .

Returns:

null if it fails, returns a pointer to the vehicle issued by the road segment.

Implements **TrafficModelObjectsLibrary::VehicleQueueClass** (*p.46*).

Definition at line 68 of file RoadSegmentClass.cpp.

Member Data Documentation

int TrafficModelObjectsLibrary::RoadSegmentClass::mStartTrafficNodeIndex [protected]

The index of the node at which the road segment starts.

Definition at line 13 of file RoadSegmentClass.h.

TrafficNodeClass* TrafficModelObjectsLibrary::RoadSegmentClass::ptrStartTrafficNode [protected]

The pointer to the node that starts the road segment.

Definition at line 15 of file RoadSegmentClass.h.

The documentation for this class was generated from the following files:

- D:/Training/TM/**RoadSegmentClass.h**
- D:/Training/TM/**RoadSegmentClass.cpp**

APPENDIX C

Traffic Model Events Library Reference

TrafficModelEventLibrary::TrafficModelEvent Class Reference

Traffic Model Event Class.

`#include <TrafficModelEvent.h>`

Inherits **SimulatorObjectsLibrary::Event**.

Inherited by **TrafficModelEventLibrary::PhaseChangeEvent**, **TrafficModelEventLibrary::SeekVehicleEvent**, **TrafficModelEventLibrary::VehicleAtFrontEvent**, and **TrafficModelEventLibrary::VehicleExitsIntersectionEvent**.

Public Member Functions

TrafficModelEvent (void)

*Initializes a new instance of the **TrafficModelEvent** class.*

~TrafficModelEvent (void)

*Finalizes an instance of the **TrafficModelEvent** class.*

EventType Type (void)

Returns the type of event.

virtual void **Run** (void)=0

Runs this object.

void **SetEventTime** (double **EventTime**)

Sets the time of event.

Protected Attributes

EventType mType

The type of the event.

Detailed Description

Traffic Model Event Class.

Definition at line 17 of file TrafficModelEvent.h.

Constructor & Destructor Documentation

TrafficModelEventLibrary::TrafficModelEvent::TrafficModelEvent (void)

Initializes a new instance of the **TrafficModelEvent** class.

Definition at line 9 of file TrafficModelEvent.cpp.

TrafficModelEventLibrary::TrafficModelEvent::~TrafficModelEvent (void)

Finalizes an instance of the **TrafficModelEvent** class.

Definition at line 15 of file TrafficModelEvent.cpp.

Member Function Documentation

virtual void TrafficModelEventLibrary::TrafficModelEvent::Run (void) [pure virtual]

Runs this Event.

Implements **SimulatorObjectsLibrary::Event** (*p.Error! Bookmark not defined.*).

Implemented in **TrafficModelEventLibrary::VehicleExitsIntersectionEvent** (*p.9*), **TrafficModelEventLibrary::SeekVehicleEvent** (*p.7*), **TrafficModelEventLibrary::VehicleAtFrontEvent** (*p.12*), and **TrafficModelEventLibrary::PhaseChangeEvent** (*p.5*).

void TrafficModelEventLibrary::TrafficModelEvent::SetEventTime (double *EventTime*)

Sets the time of event.

Parameters:

EventTime	Time of the event

Parameters:

EventTime	Time of the event.

Reimplemented from **SimulatorObjectsLibrary::Event** (*p.Error! Bookmark not defined.*).

Definition at line 22 of file TrafficModelEvent.cpp.

EventType TrafficModelEventLibrary::TrafficModelEvent::Type (void)

Returns the type of event.

Gets the type.

Returns:

mType

Returns:

.

Definition at line 30 of file TrafficModelEvent.cpp.

Member Data Documentation

EventType TrafficModelEventLibrary::TrafficModelEvent::mType [protected]

The type of the event.

Definition at line 23 of file TrafficModelEvent.h.

The documentation for this class was generated from the following files:

D:/Training/TM/**TrafficModelEvent.h**
D:/Training/TM/**TrafficModelEvent.cpp**

TrafficModelEventLibrary::PhaseChangeEvent Class Reference

Phase change event.

`#include <PhaseChangeEvent.h>`

Inherits **TrafficModelEventLibrary::TrafficModelEvent**.

Public Member Functions

PhaseChangeEvent (void)

*Initializes a new instance of the **PhaseChangeEvent** class.*

PhaseChangeEvent (**IntersectionClass** *Intersection, double **EventTime**)

*Initializes a new instance of the **PhaseChangeEvent** class.*

~PhaseChangeEvent (void)

*Finalizes an instance of the **PhaseChangeEvent** class.*

virtual void **Run** (void)

Runs this event.

Protected Attributes

IntersectionClass * **mIntersection**

pointer to the intersection at which this event occurs.

Detailed Description

Phase change event.

Definition at line 9 of file PhaseChangeEvent.h.

Constructor & Destructor Documentation

TrafficModelEventLibrary::PhaseChangeEvent::PhaseChangeEvent (void)

Initializes a new instance of the **PhaseChangeEvent** class.

Default constructor.

Definition at line 16 of file PhaseChangeEvent.cpp.

TrafficModelEventLibrary::PhaseChangeEvent::PhaseChangeEvent (IntersectionClass * *Intersection*, double *EventTime*)

Initializes a new instance of the **PhaseChangeEvent** class.

Constructor.

Parameters:

Intersection	Pointer to the intersection where the event is going to occur.
EventTime	Time of the event.

Parameters:

Intersection	[in,out] If non-null, the intersection.
EventTime	Time of the event.

Definition at line 22 of file PhaseChangeEvent.cpp.

TrafficModelEventLibrary::PhaseChangeEvent::~PhaseChangeEvent (void)

Finalizes an instance of the **PhaseChangeEvent** class.

Destructor.

Definition at line 31 of file PhaseChangeEvent.cpp.

Member Function Documentation

void TrafficModelEventLibrary::PhaseChangeEvent::Run (void) [virtual]

Runs this event.

Runs this Event.

Implements **TrafficModelEventLibrary::TrafficModelEvent** (*p.3*).

Definition at line 34 of file PhaseChangeEvent.cpp.

Member Data Documentation

IntersectionClass* TrafficModelEventLibrary::PhaseChangeEvent::mIntersection [protected]

pointer to the intersection at which this event occurs.

Definition at line 14 of file PhaseChangeEvent.h.

The documentation for this class was generated from the following files:

D:/Training/TM/**PhaseChangeEvent.h**
D:/Training/TM/**PhaseChangeEvent.cpp**

TrafficModelEventLibrary::SeekVehicleEvent Class Reference

Seek vehicle event.

`#include <SeekVehicleEvent.h>`

Inherits **TrafficModelEventLibrary::TrafficModelEvent**.

Public Member Functions

SeekVehicleEvent (void)

*Initializes a new instance of the **SeekVehicleEvent** class.*

~SeekVehicleEvent (void)

*Finalizes an instance of the **SeekVehicleEvent** class.*

SeekVehicleEvent (**IntersectionClass** *Intersection, **VehicleQueueClass** *VehicleQueueIn, double **EventTime**)

*Initializes a new instance of the **SeekVehicleEvent** class.*

virtual void **Run** (void)

Runs this object.

Protected Attributes

IntersectionClass * **mIntersection**

The intersection.

VehicleQueueClass * **mVehicleQueueIn**

The vehicle queue in.

Detailed Description

Seek vehicle event.

Definition at line 9 of file SeekVehicleEvent.h.

Constructor & Destructor Documentation

TrafficModelEventLibrary::SeekVehicleEvent::SeekVehicleEvent (void)

Initializes a new instance of the **SeekVehicleEvent** class.

Definition at line 14 of file SeekVehicleEvent.cpp.

TrafficModelEventLibrary::SeekVehicleEvent::~SeekVehicleEvent (void)

Finalizes an instance of the **SeekVehicleEvent** class.

Definition at line 19 of file SeekVehicleEvent.cpp.

TrafficModelEventLibrary::SeekVehicleEvent::SeekVehicleEvent (IntersectionClass * *Intersection*, VehicleQueueClass * *VehicleQueueIn*, double *EventTime*)

Initializes a new instance of the **SeekVehicleEvent** class.

Parameters:

Intersection	[in,out] If non-null, the intersection.
VehicleQueueIn	[in,out] If non-null, the vehicle queue in.
EventTime	Time of the event.

Parameters:

Intersection	[in,out] If non-null, the intersection.
VehicleQueueIn	[in,out] If non-null, the vehicle queue in.
EventTime	Time of the event.

Definition at line 28 of file SeekVehicleEvent.cpp.

Member Function Documentation

void TrafficModelEventLibrary::SeekVehicleEvent::Run (void) [virtual]

Runs this object.

Runs this Event.

Implements **TrafficModelEventLibrary::TrafficModelEvent** (*p.3*).

Definition at line 39 of file SeekVehicleEvent.cpp.

Member Data Documentation

IntersectionClass* TrafficModelEventLibrary::SeekVehicleEvent::mIntersection [protected]

The intersection.

Definition at line 14 of file SeekVehicleEvent.h.

VehicleQueueClass* TrafficModelEventLibrary::SeekVehicleEvent::mVehicleQueueIn [protected]

The vehicle queue in.

Definition at line 16 of file SeekVehicleEvent.h.

The documentation for this class was generated from the following files:

D:/Training/TM/**SeekVehicleEvent.h**
D:/Training/TM/**SeekVehicleEvent.cpp**

TrafficModelEventLibrary::VehicleExitsIntersectionEvent Class Reference

The object vehicle exits an intersection event.
`#include <VehicleExitsIntersectionEvent.h>`
Inherits **TrafficModelEventLibrary::TrafficModelEvent**.

Public Member Functions

VehicleExitsIntersectionEvent (void)
*Constructor for a new instance of the **VehicleExitsIntersectionEvent** class.*

VehicleExitsIntersectionEvent (**VehicleClass** *Vehicle, **TrafficNodeClass** *Intersection, **VehicleQueueClass** *VehicleQueueOut, double **EventTime**)
*Initializes a new instance of the **VehicleExitsIntersectionEvent** class.*

~VehicleExitsIntersectionEvent (void)
Destructor of a VehicleExitIntersections Event.

virtual void **Run** (void)
Process and execute an Event object.

Protected Attributes

VehicleClass * **mVehicle**
The number of vehicles.

TrafficNodeClass * **mIntersection**
The ID of the intersection.

VehicleQueueClass * **mVehicleQueueIn**
The number of vehicle queues feeding into the intersection.

VehicleQueueClass * **mVehicleQueueOut**
The number of vehicle queue feeding out of the intersection.

Detailed Description

The object vehicle exits an intersection event.

Definition at line 10 of file VehicleExitsIntersectionEvent.h.

Constructor & Destructor Documentation

TrafficModelEventLibrary::VehicleExitsIntersectionEvent::VehicleExitsIntersectionEvent (void)

Constructor for a new instance of the **VehicleExitsIntersectionEvent** class.

Definition at line 16 of file VehicleExitsIntersectionEvent.cpp.

TrafficModelEventLibrary::VehicleExitsIntersectionEvent::VehicleExitsIntersectionEvent (VehicleClass * *Vehicle*, TrafficNodeClass * *Intersection*, VehicleQueueClass * *VehicleQueueOut*, double *EventTime*)

Initializes a new instance of the **VehicleExitsIntersectionEvent** class.

Parameters:

Vehicle	[in,out] Pointer to the vehicle.
Intersection	[in,out] Pointer to the intersection
VehicleQueueOut	[in,out] Pointer to the outgoing vehicle queue
EventTime	Time of the event.

Parameters:

Vehicle	[in,out] Pointer to the vehicle.
Intersection	[in,out] Pointer to the intersection.
VehicleQueueOut	[in,out] Pointer to the outgoing vehicle queue.
EventTime	Time of the event.

Definition at line 24 of file VehicleExitsIntersectionEvent.cpp.

TrafficModelEventLibrary::VehicleExitsIntersectionEvent::~VehicleExitsIntersectionEvent (void)

Destructor of a VehicleExitIntersections Event.

Destructor.

Definition at line 36 of file VehicleExitsIntersectionEvent.cpp.

Member Function Documentation

void TrafficModelEventLibrary::VehicleExitsIntersectionEvent::Run (void) `[virtual]`

Process and execute an Event object.

Runs this Event.

Implements **TrafficModelEventLibrary::TrafficModelEvent** (*p.3*).

Definition at line 39 of file VehicleExitsIntersectionEvent.cpp.

Member Data Documentation

TrafficNodeClass* TrafficModelEventLibrary::VehicleExitsIntersectionEvent::mIntersection `[protected]`

The ID of the intersection.

Definition at line 17 of file VehicleExitsIntersectionEvent.h.

VehicleClass* TrafficModelEventLibrary::VehicleExitsIntersectionEvent::mVehicle `[protected]`

The number of vehicles.

Definition at line 15 of file VehicleExitsIntersectionEvent.h.

VehicleQueueClass* TrafficModelEventLibrary::VehicleExitsIntersectionEvent::mVehicleQueueIn [protected]

The number of vehicle queues feeding into the intersection.

Definition at line 19 of file VehicleExitsIntersectionEvent.h.

VehicleQueueClass* TrafficModelEventLibrary::VehicleExitsIntersectionEvent::mVehicleQueueOut [protected]

The number of vehicle queue feeding out of the intersection.

Definition at line 21 of file VehicleExitsIntersectionEvent.h.

The documentation for this class was generated from the following files:

D:/Training/TM/**VehicleExitsIntersectionEvent.h**
D:/Training/TM/**VehicleExitsIntersectionEvent.cpp**

TrafficModelEventLibrary::VehicleAtFrontEvent Class Reference

VehicleAtFronEvent Class.

`#include <VehicleAtFrontEvent.h>`

Inherits **TrafficModelEventLibrary::TrafficModelEvent**.

Public Member Functions

VehicleAtFrontEvent (void)

*Initializes a new instance of the **VehicleAtFrontEvent** class.*

VehicleAtFrontEvent (**VehicleQueueClass** *VehicleQueue, **VehicleClass** *Vehicle, double **EventTime**)

*Initializes a new instance of the **VehicleAtFrontEvent** class.*

~VehicleAtFrontEvent (void)

*Destructor for **VehicleAtFrontEvent** class.*

virtual void **Run** (void)

*Execute the **VehicleAtFrontEvent** object.*

Protected Attributes

VehicleQueueClass * **mVehicleQueue**

Queue of VehicleQueueClass objects.

VehicleClass * **mVehicle**

VehicleClass object.

Detailed Description

VehicleAtFronEvent Class.

Definition at line 9 of file VehicleAtFrontEvent.h.

Constructor & Destructor Documentation

TrafficModelEventLibrary::VehicleAtFrontEvent::VehicleAtFrontEvent (void)

Initializes a new instance of the **VehicleAtFrontEvent** class.

Definition at line 15 of file VehicleAtFrontEvent.cpp.

TrafficModelEventLibrary::VehicleAtFrontEvent::VehicleAtFrontEvent (VehicleQueueClass * *VehicleQueue*, VehicleClass * *Vehicle*, double *EventTime*)

Initializes a new instance of the **VehicleAtFrontEvent** class.

Parameters:

VehicleQueue	[in,out] If non-null, queue of vehicles.
Vehicle	[in,out] If non-null, the vehicle.
EventTime	Time of the event.

Parameters:

VehicleQueue	[in,out] If non-null, queue of vehicles.
Vehicle	[in,out] If non-null, the vehicle.
EventTime	Time of the event.

Definition at line 24 of file VehicleAtFrontEvent.cpp.

TrafficModelEventLibrary::VehicleAtFrontEvent::~VehicleAtFrontEvent (void)

Destructor for **VehicleAtFrontEvent** class.

Destructor.

Definition at line 35 of file VehicleAtFrontEvent.cpp.

Member Function Documentation

void TrafficModelEventLibrary::VehicleAtFrontEvent::Run (void) [virtual]

Execute the **VehicleAtFrontEvent** object.

Runs this Event.

Implements **TrafficModelEventLibrary::TrafficModelEvent** (*p.3*).

Definition at line 41 of file VehicleAtFrontEvent.cpp.

Member Data Documentation

VehicleClass* TrafficModelEventLibrary::VehicleAtFrontEvent::mVehicle [protected]

VehicleClass object.

Definition at line 16 of file VehicleAtFrontEvent.h.

VehicleQueueClass* TrafficModelEventLibrary::VehicleAtFrontEvent::mVehicleQueue [protected]

Queue of VehicleQueueClass objects.

Definition at line 14 of file VehicleAtFrontEvent.h.

The documentation for this class was generated from the following files:

D:/Training/TM/**VehicleAtFrontEvent.h**
D:/Training/TM/**VehicleAtFrontEvent.cpp**