Contents

Sapporo Software Reference

Version 1.0 α March 1992

Intelligent Traffic Control for Urban Networks

Development of AI Concepts for Traffic Management Systems

Joint Basic Research Project
NTT Data Communications Systems
FZI Forschungszentrum Informatik
Dept. Technical Expert Systems and Robotics
Prof. Dr. U. Rembold

Authors
Bernd Wild, Olaf Dalchow and Bernd Schütze

1 Network Definitions

1.1 Classes

1.1.1 • `node-user-mixin` *Class*

Contains user defined slots which are specific to NODE-CLASS objects.

1.1.2 • `road-user-mixin` *Class*

Contains user defined slots which are specific to ROAD-CLASS objects.

1.1.3 • `road-network-user-mixin` *Class*

Contains user defined slots which are specific to ROAD-NETWORK-CLASS objects.

1.1.4 • `road-section-user-mixin` *Class*

Contains user defined slots which are specific to ROAD-SECTION-CLASS objects.

1.1.5 • `lane-user-mixin` *Class*

Contains user defined slots which are specific to LANE-CLASS objects.

1.1.6 • `identifier-mixin` *Class*

This mixin class contains all identification components necessary for specifying a unique instance.

superclasses	`nil`
number	`:initform 0` `:accessor id-number`
name	`:initform "noname"` `:accessor id-name`
key	`:initform nil` `:accessor key-of`

1.1.7 • `part-of-mixin` *Class*

This mixin class contains the part-of relationship components for network objects, i.e. object i is part of network j.

superclasses	`nil`
part-of	`:initform nil` `:accessor part-of`

1.1.8 • `category-mixin` *Class*

This mixin class contains all category components necessary for traffic engineering specific classification of an instance.

superclasses	`nil`
category	`:initform 0` `:accessor category`

1.1.9 • `demarcation-mixin` *Class*

Most network objects need a description of their start node and end node, e.g. roads, sections, lanes.

superclasses	`nil`
start-node	`:initform nil` `:accessor start-node`

end-node	`:initform nil` `:accessor end-node`

1.1.10 • `node-standard-mixin` *Class*

This class forms the first intergration level for inclusion of component classes. This part of the later Node-class should be left untouched in order to ensure minimum compatibility with other implementations.

superclasses	`identifier-mixin part-of-mixin category-mixin`
x-coordinate	`:initform 0` `:accessor x`
y-coordinate	`:initform 0` `:accessor y`

1.1.11 • `node-class` *Class*

This is the top-level class which is usually instantiated for representing a node object.

superclasses	`node-user-mixin node-standard-mixin`

1.1.12 • `road-network-standard-mixin` *Class*

This class forms the first intergration level for inclusion of component classes. This part of the later ROAD-NETWORK-class should be left untouched in order to ensure minimum compatibility with other implementations.

superclasses	`identifier-mixin part-of-mixin`
neighbouring-nets	`:initform nil` `:accessor all-neighbours`
roads	`:initform nil` `:accessor all-roads`
sections	`:initform nil` `:accessor all-sections`
nodes	`:initform nil` `:accessor all-nodes`

1.1.13 • `road-network-class` *Class*

This is the top-level class which is usually instantiated for representing a road network object.

superclasses road-network-user-mixin road-network-standard-mixin

1.1.14 • `road-standard-mixin` *Class*

This class forms the first intergration level for inclusion of component classes. This part of the later ROAD-class should be left untouched in order to ensure minimum compatibility with other implementations.

superclasses	`identifier-mixin part-of-mixin category-mixin demarcation-mixin`
forward-length	`:initform 0` `:accessor forward-length`
forward-sections	`:initform nil` `:accessor forward-sections`
backward-length	`:initform 0` `:accessor backward-length`
backward-sections	`:initform nil` `:accessor backward-sections`

1.1.15 • `road-class` *Class*

This is the top-level class which is usually instantiated for representing a road object.

superclasses	`road-user-mixin road-standard-mixin`

1.1.16 • `road-section-standard-mixin` *Class*

This class forms the first integration level for inclusion of component classes. This part of the later ROAD-SECTION-class should be left untouched in order to ensure minimum compatibility with other implementations. The slot lanes contains a list of 2 lists: the first sublist indicates the lanes on the start->end side while the second sublists denotes the lanes in the opposite direction, i.e. end->start. The first lane element in a sublists is always the outermost lane, e.g. lane no. 1 while the innermost lane carries the higher number.

superclasses	`identifier-mixin part-of-mixin category-mixin demarcation-mixin`
lanes	`:initform nil` `:accessor lanes`
length	`:initform 0` `:accessor section-length`
start-cross-overs	`:initform nil`

	`:accessor start-cross-overs`
end-cross-overs	`:initform nil` `:accessor end-cross-overs`
max-speed-lorry	`:initform 0` `:accessor max-speed-lorry`
max-speed-car	`:initform 0` `:accessor max-speed-car`

1.1.17 • `road-section-class` *Class*

This is the top-level class which is usually instantiated for representing a road section object.

superclasses	`road-section-user-mixin road-section-standard-mixin`

1.1.18 • `lane-standard-mixin` *Class*

This class forms the first intergration level for inclusion of component classes. This part of the later ROAD-SECTION-class should be left untouched in order to ensure minimum compatibility with other implementations. The slots left-lane and right-lane indicate the neighbouring lane in the same direction, respectively.

superclasses	`identifier-mixin part-of-mixin`
left-lane	`:initform nil` `:accessor left-lane`
right-lane	`:initform nil` `:accessor right-lane`
width	`:initform 0` `:accessor width`
restrictions	`:initform nil` `:accessor restrictions`

1.1.19 • `lane-class` *Class*

This is the top-level class which is usually instantiated for representing a road object.

superclasses	`lane-user-mixin lane-standard-mixin`

1.1.20 • `road-class` categories *Value Set*

The slot CATEGORY of a road-class object contains a list of 3 elements ‘(e1 e2 e3) which can take a value from the following sets:

element	type	value set
e1	lane type	`:single-lane`
		`:multi-lane`
e2	direction	`:south<-north`
		`:south->north`
		`:south<->north`
		`:east->west`
		`:east<-west`
		`:east<->west`
e3	street type	`:main-street`
		`:normal-street`

1.1.21 • `road-section-class` categories *Value Set*

The slot CATEGORY of a road-section-class object contains a list of 3 elements ‘(e1 e2 e3) which can take a value from the following sets:

element	type	value set
e1	lane type	`:single-lane`
		`:multi-lane`
e2	direction	`:south<-north`
		`:south->north`
		`:south<->north`
		`:east->west`
		`:east<-west`
		`:east<->west`
e3	street type	`:main-street`
		`:normal-street`

1.1.22 • `node-class` categories *Value Set*

The slot CATEGORY of a node-class object contains one element e1 which can take a value from the following set:

element	type	value set
e1	node type	`:open-end`
		`:T-crossing`

`:4-crossing`

1.2 Internals

1.2.1 • `print-object ((self identifier-mixin) stream)` *Method*

This generic method is always called when an instance of a network class object is printed on the screen. In contrast to the default method the slot value of the slot NAME is used for printing the object. Usually, this function is never called by the user directly!

self	a network object instance
stream	the output stream

2 Database

2.1 Global Definitions

2.1.1 • `*debug-db*` *Variable*

Global variable for debugging purposes. If nil (default) all protocol messages during loading, parsing and interpreting textual databases are suppressed.

2.1.2 • `*road-networks*` *Variable*

Global variable which contains a list of all instances of road-network-class. This variable is kept for fast access to all road maps currently available for further use. Each Load Map results in a push of a new road network instance onto this list.

2.1.3 • `*network-table*` *Variable*

Global variable containing a hash table of all currently present road networks. Hash keys are network key strings, e.g. "NETWORK-1" while the corresponding hash value is a pointer to the appropriate instance of road-network-class.

2.1.4 • `*default-database-directory*` *Variable*

Global variable which contains a directory string like `"ca:/fzi/robot/bwild/databa-ses/"` indicating where the network databases are store under. Each network database is a subdirectory under `*default-database-directory*` with the name of the appropriate network. Every object type of a network is stored in this subdirectory in a separate text file.

2.1.5 • `read-filemaker-database-files` (*args*) *Function*

This is the main function for reading textual FileMaker database files and generation of appropriate instances. The arguments are

filename	a file name string describing the input text file
slot-list	a list of slot names. Each element of slot-list corresponds with an item of the input line read from the file filename. The slot names have to be identical to the slot names of the class from which instances are created.
class	For every input line one instance of class will be created.
key-prefix	a symbol which will be used to generate unique hash key strings for the instances that will be created (see also object-key-prefix)
key-slot	the name of the slot of the freshly created instance which gives the ID for the generation of hash key string.

2.1.6 • `get-network-instance-by-name` (*args*) *Function*

This function is a retrieval function for looking-up a network instance by its name slot value. It takes 1 arguments, a name string, and it uses the entries of *NETWORK-TABLE* in order to find the instance.

name	a name string

2.1.7 • `get-road-instance-by-name` *Function*

This function is a retrieval function for looking-up a road instance by its name slot value. It takes 1 argument, a name string, and it uses the entries of *ROAD-NETWORKS* in order to find the instance.

name	a name string

2.1.8 • `get-road-instance` (*args*) *Function*

This function is a retrieval function for looking-up a road instance by its hash key string. It takes 1 argument, a hash key string, and it uses the entries of *ROAD-NETWORKS* in order to find the instance.

key	a hash key string

2.1.9 • `get-section-instance-by-name`*(args)* *Function*

This function is a retrieval function for looking-up a road section instance by its name slot value. It takes 1 argument, a name string, uses the entries of *ROAD-NETWORKS* in order to find the instance.

name	a name string

2.1.10 • `get-section-instance` (*args*) *Function*

This function is a retrieval function for looking-up a road section instance by its hash key string. It takes 1 argument, a hash key string, and it uses the entries of *ROAD-NETWORKS* in order to find the instance.

key	a hash key string

2.1.11 • `get-node-instance-by-name` (*args*) *Function*

This function is a retrieval function for looking-up a node instance by its name slot value. It uses the entries of *ROAD-NETWORKS* in order to find the instance.

name	a name string

2.1.12 • `get-node-instance` (*args*) *Function*

This function is a retrieval function for looking-up a node instance by its hash key string. It takes 1 argument, a hash key string, and it uses the entries of *ROAD-NETWORKS* in order to find the instance.

key	a hash key string

2.1.13 • `data-correction`*(args)* *Function*

This is the main function for doing the data correction of the freshly created instances of a network instance given by net. It calls the appropriate routines as described above. If any objects different from networks, roads, sections and nodes should be read in, this function hash to modified.

net	the network instance whose basic elements should be corrected.

2.1.14 • `dump-to-database` (*args*) *Function*

The main function for saving a network description to file. All objects attached to the network argument object are stored in separate files in a subdirectory entitled by the name of the network under the *DEFAULT-DATABASE-DIRECTORY*. If desired, a different directory can optionally be specified. The storage form is a textual and thus human readable form:

1. line:
<slot-name-1>TAB<slot-name-2>TAB <slot-name-n>LF
all other lanes:
<slot-value-1>TAB<slot-value-2>TAB <slot-value-n>LF.

network	an instance of the network which should be saved.
path-string	a directory string for the network databases. This parameter is optional.

2.1.15 • `load-from-database` (*args*) *Function*

The counterpart to the DUMP-TO-DATABASE routine. This function takes a network name like the subdirectory name of a network database and reads in all files of that subdirectory. By this, new instances of the corresponding network objects are created, their slot values converted from string to number, pointer etc and referenced in order to obtain an efficient internal linked representation of the network. The complete network object is returned and registered in the hash-table of *network-table*.The loaded network becomes automatically the current network, thus drawing commands affect immediately the new network.

net-name	a name string of the network which should be loaded.

path-string	a directory string for the network databases. This parameter is optional.

2.2 Internals

2.2.1 • `network-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'network-`

2.2.2 • `road-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'road-`

2.2.3 • `section-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'section-`

2.2.4 • `node-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'node-`

2.2.5 • `create-and-bind-instance` (*args*) *Function*

This is an internal function called by read-data-file. It takes an input line with a number of tokens, separated by tabulators and terminated by a carriage return. An instance of the given class is created and the tokens are stored into the slots of the instance in accordance with the slot name list as given by slot-list. As result a pointer to the new instance is returned. The arguments are

input-line	a line of text tokens, separated by TABs and terminated by a CR.
slot-list	a list of slot names. Each element of slot-list corresponds with an item of the input line read from the file filename. The slot names have to be identical to the slot names of the class from which instances are created.
class	For every input line one instance of class will be created.

2.2.6 • `read-database-files` *Function*

User interface function for the loading of textual databases. The function pops up a multiple accept menu where the user can specify pathnames for the different text database files. The default values given are <user-homedir>/Mac/{networks roads sections nodes}. The function takes no arguments.

2.2.7 • `build-node-key` (*args*) *Function*

This function takes a string of a number (as obtained by reading a textual database file) and returns a valid hash key string consisting of the appropriate object prefix and the number, e.g. NODE-21.

number	a number string

2.2.8 • `build-section-key` (*args*) *Function*

This function takes a string of a number (as obtained by reading a textual database file) and returns a valid hash key string consisting of the appropriate object prefix and the number, e.g. SECTION-21.

number	a number string

2.2.9 • `build-road-key` (*args*) *Function*

This function takes a string of a number (as obtained by reading a textual database file) and returns a valid hash key string consisting of the appropriate object prefix and the number, e.g. ROAD-21.

number	a number string

2.2.10 • `build-network-key` (*args*) *Function*

This function takes a string of a number (as obtained by reading a textual database file) and returns a valid hash key string consisting of the appropriate object prefix and the number, e.g. NETWORK-21.

number	a number string

2.2.11 • `correction` (*args*) *Generic*

The definition of the generic function CORRECTION. After the input and creation of network objetcs the slots of every object normally contain strings. The task of this function is the parsing of the slot value strings into the proper slot value types, e.g. numbers, lists, pointer to objects, etc. It lies in the responsibility of the user to define an own correction method for every network object class which should be read in from text database files. CORRECTION takes 2 arguments:

candidate	the instance whose slots should be parsed and corrected.
candidate-net	the network instance to which the candidate object belongs.

2.2.12 • `correction` (*args*) *Method*

Makes the appropriate corrections for the slots of a recently created instance of the road-network-class. The 2nd argument is dummy.

candidate	an instance of road-network-class
candidate-net	an instance of road-network-class (dummy)

2.2.13 • `correction` (*args*) *Method*

Makes the appropriate corrections for the slots of a recently created instance of the road-class.

candidate	an instance of road-class
candidate-net	an instance of road-class

2.2.14 • `correction` (*args*) *Method*

Makes the appropriate corrections for the slots of a recently created instance of the road-SECTION-class.

candidate	an instance of road-SECTION-class
candidate-net	an instance of road-SECTION-class

2.2.15 • `correction` (*args*) *Method*

Makes the appropriate corrections for the slots of a recently created instance of the NODE-class.

candidate	an instance of NODE-class
candidate-net	an instance of NODE-class

2.2.16 • `parse-string-to-list-of-instances` (*args*) *Function*

An internal function which is called by correction methods. It takes a reference to a hash table, a key-prefix and an input string. The function destructures the input string into a list of elements which are then used to generate together with the key-prefix a valid hash key string for the hash values of the hash table. The final action is the look-up of the instances by hashing and the returning of a list consisting of pointers to the appropriate instances. For example, an input string “12 43 63” belonging to a road instance will be parsed into ‘(ROAD-12 ROAD-43 ROAD-63) and finally dereferenced into something like ‘(<#ROAD-CLASS Road-12 365623774> <#ROAD-CLASS Road-43 36562226674> <#ROAD-CLASS Road-63 3656774474>).

table	a hash table containing all instances to which the dereferenced input elements should point.
key-prefix	the prefix symbol necessary to construct together with the parsed input element a valid hash key string.
string	a string of input elements separated by blank spaces.

2.2.17 • `parse-string-to-list-of-categories` (*args*) *Function*

Similar to the function PARSE-STRING-TO-LIST-OF-INSTANCES this internal function takes an input string of number elements separated by blank spaces and parses them into a list of road network categorie keywords as given by a category alist. For example, the input string “4 2 3” would be translated into ‘(:single-lane :east<->west :normal-street).

string	a string of input elements separated by blank spaces.
cat-alist	a list of lists (one for each input element of an input string) of alists of the form (<key-number> . <category-keyword>).

2.2.18 • `set-object-length` (*args*) *Generic*

An internal function for calculation of the length slot of various network objects in dependance of the x- and y-coordinates of their start- and end-nodes.

object	the instance whose length slot should be computed.

2.2.19 • `set-object-length` (*args*) *Method*

An internal function for calculation of the length slot of road section objects in dependance of the x- and y-coordinates of their start- and end-nodes.

object	an instance of road-section-class whose length slot should be computed.

2.2.20 • `set-object-length` (*args*) *Method*

An internal function for calculation of the length slot of road objects in dependance of the x- and y-coordinates of their start- and end-nodes.

object	an instance of road-class whose length slot should be computed.

3 Density Calculus

3.1 Classes

3.1.1 • `interval-range-mixin` *class*

This mixin class contains the information about the intervals which describe the qualitative density values and the traffic states used in the density calculus, such as the borders of the range of density, speed and flow.

superclasses	`nil`
min-density	`:initform 0` `:accessor density-min-d`
max-density	`:initform 0` `:accessor density-max-d`
min-speed	`:initform 0` `:accessor density-min-v`
max-speed	`:initform 0` `:accessor density-max-v`
min-flow	`:initform 0` `:accessor density-min-f`
max-flow	`:initform 0` `:accessor density-max-f`

3.1.2 • `density-calculus-user-mixin` *Class*

Contains user defined slots which are specific to DENSITY-CALCULUS-CLASS objects.

3.1.3 • `fundamental-diagram-user-mixin` *Class*

Contains user defined slots which are specific to FUNDAMENTAL-DIAGRAM-CLASS objects.

3.1.4 • `density-value-user-mixin` *Class*

Contains user defined slots which are specific to DENSITY-VALUE-CLASS objects.

3.1.5 • `traffic-state-user-mixin` *Class*

Contains user defined slots which are specific to TRAFFIC-STATE-CLASS objects.

3.1.6 • `density-calculus-standard-mixin` *Class*

This class forms the first intergration level for inclusion of component classes of the density calculus. This part of the later DENSITY-CALCULUS-class should be left untouched.

superclasses	`identifier-mixin`
fundamental-diagram	`:initform nil` `:accessor fundamental-diagram`
density-value-keys	`:initform nil` `:accessor density-value-keys`
sorted-density-value-list	`:initform nil` `:accessor sorted-density-value-list`
sorted-traffic-state-list	`:initform nil` `:accessor sorted-traffic-state-list`
border-speed-table	`:initform nil` `:accessor border-speed-table`
new-density-zone-table	`:initform nil` `:accessor new-density-zone-table`
rule-sets	`:initform nil` `:accessor rule-sets`

3.1.7 • `density-calculus-class` *Class*

This is the top-level class which is usually instantiated for representing a density calculus object.

superclasses	`density-calculus-user-mixin` `density-calculus-standard-mixin`

3.1.8 • `fundamental-diagram-standard-mixin` *Class*

This class forms the first intergration level for inclusion of component classes of the density calculus. This part of the later DENSITY-CALCULUS-class should be left untouched.

superclasses	`identifier-mixin category-mixin`
interval-of-measurement	`:initform 0` `:accessor interval`
observation-point	`:initform "noname"` `:accessor observation-point`
maximum-density	`:initform 0` `:accessor max-d`
maximum-flow	`:initform 0` `:accessor max-f`
maximum-speed	`:initform 0` `:accessor max-v`
value-descriptors	`:initform nil` `:accessor all-value-descriptors`
value-list	`:initform nil` `:accessor value-list`

3.1.9 • `fundamental-diagram-class` *Class*

This is the top-level class which is usually instantiated for representing a fundamental diagram object.

superclasses	`fundamental-diagram-user-mixin` `fundamental-diagram-standard-mixin`

3.1.10 • `density-value-standard-mixin` *Class*

This class contains the description of the qualitative density values used in the density calculus. This part of the later DENSITY-VALUE-class should be left untouched.

superclasses	`identifier-mixin part-of-mixin` `interval-range-mixin`
mean-density	`:initform 0` `:accessor mean-d`
mean-flow	`:initform 0` `:accessor mean-f`
mean-speed	`:initform 0` `:accessor mean-v`

3.1.11 • `density-value-class` *Class*

This is the top-level class which is usually instantiated for representing a density value object.

superclasses	`density-value-user-mixin` `density-value-standard-mixin`

3.1.12 • `traffic-state-standard-mixin` *Class*

This class contains the description of the traffic states used in the density calculus to describe a traffic situation. This part of the later TRAFFIC-STATE-class should be left untouched.

superclasses	`identifier-mixin part-of-mixin` `interval-range-mixin`
density-values	`:initform nil` `:accessor attached-density-values`

3.1.13 • `traffic-state-class` *Class*

This is the top-level class which is usually instantiated for representing a traffic state object.

superclasses	`traffic-state-user-mixin` `traffic-state-standard-mixin`

3.2 Interface Functions

3.2.1 • `read-fundamental-diagram-from-FileMaker-db` (*args*) *Function*

This function reads a textual FileMaker database file containing the information of a fundamental diagram and generates the appropriate instance. The arguments are

network	the current network instance on which the density calculus will be attached to after its creation based on the inforamtion given by a fundamental diagram
fundamental-diagram-name	a string

3.2.2 • `get-density-calculus-instance-by-name` (*args*) *Function*

This function is a retrieval function for looking-up the density calculus instance by its name slot value. It takes 1 arguments, a name string, and it uses the entries of *ROAD-NETWORKS* in order to find the network instance the density calculus is attached to.

name	a name string

3.2.3 • `get-density-calculus-instance` (*args*) *Function*

This function is a retrieval function for looking-up a density calculus instance by its hash key string. It takes 1 argument, a hash key string, and it uses the entries of *ROAD-NETWORKS* in order to find the network instance the density calculus is attached to.

key	a hash key string

3.2.4 • `get-fundamental-diagram-instance-by-name` (*args*) *Function*

This function is a retrieval function for looking-up the fundamental diagram instance by its name slot value. It takes 1 arguments, a name string, and it uses the entries of *ROAD-NETWORKS* in order to find the network instance the fundamental diagram of a density calculus is attached to.

name	a name string

3.2.5 • `get-fundamental-diagram-instance` (*args*) *Function*

This function is a retrieval function for looking-up a fundamental diagram instance by its hash key string. It takes 1 argument, a hash key string, and it uses the entries of *ROAD-NET-

WORKS* in order to find the network instance the fundamental diagram of a density calculus is attached to.

key	a hash key string

3.2.6 • `get-traffic-state-instance-by-name` (*args*) *Function*

This function is a retrieval function for looking-up the traffic state instance by its name slot value. It takes 1 arguments, a name string, and it uses the entries of *ROAD-NETWORKS* in order to find the network instance this traffic state of a density calculus is attached to.

name	a name string

3.2.7 • `get-traffic-state-instance` (*args*) *Function*

This function is a retrieval function for looking-up a traffic state instance by its hash key string. It takes 1 argument, a hash key string, and it uses the entries of *ROAD-NETWORKS* in order to find the network instance the traffic state of a density calculus is attached to.

key	a hash key string

3.2.8 • `get-density-value-instance-by-name` (*args*) *Function*

This function is a retrieval function for looking-up the density value instance by its name slot value. It takes 1 arguments, a name string, and it uses the entries of *ROAD-NETWORKS* in order to find the network instance this qualitative density value of a density calculus is attached to.

name	a name string

3.2.9 • `get-density-value-instance` (*args*) *Function*

This function is a retrieval function for looking-up a density value instance by its hash key string. It takes 1 argument, a hash key string, and it uses the entries of *ROAD-NETWORKS* in order to find the network instance the qualitative density value of a density calculus is attached to.

key	a hash key string

3.2.10 • `create-density-calculus` (*args*) *Method*

The main function for creating a new density calculus. It uses the information given in a fundamental diagram which will be attached to the network and the inforamtion given by the user in order to create several qualitative density values und some traffic states.The function returns the density calculus instance.

network	the network instance the new density calculus will be attached to

3.2.11 • `update-density-calculus` (*args*) *Function*

The main function for updating an existing density calculus attached to a network. After checking the modified ranges of the density values and traffic states all the tables and lists used in the density calculus are updated. Also the slots of the network instance connected with the density calculus are updated. The function returns the density calculus instance with updated slot-values.

density-calculus	the density calculus instance to be updated
network	the network instance the density calculus is be attached to

3.2.12 • `dump-density-calculus-to-database` (*args*) *Function*

The main function for saving a density calculus to file. All objects attached to the density calculus argument object are stored in separate files in a subdirectory entitled by the "{name of the network}/DENSITY-CALCULI/ {name of thew density calculus}" under the *DEFAULT-DATABASE-DIRECTORY*. The storage form is a textual and thus human readable form:

1. line:
<slot-name-1>TAB<slot-name-2>TAB <slot-name-n>LF
all other lanes:
<slot-value-1>TAB<slot-value-2>TAB <slot-value-n>LF.

network	an network instance.
density-calculus	an instance of the density calculus which should be saved.
path-string	a directory string for the network databases. This parameter is optional.

3.2.13 • `load-density-calculus-from-database` (*args*) *Function*

The counterpart to the DUMP-DENSITY-CALCULUS-TO-DATABASE routine. This function takes a network instance and the name of a subdirectory named "DENSITY-CALCU-

LI” of a subdirectory of a network database. It reads all files of one of its subdirectories. By this, new instances of the corresponding density calculus objects are created, their slot values converted from string to number, pointer etc and referenced in order to obtain an efficient internal linked representation of the density calculus. Finally, all the complete density calculus objects are attached to the network. The function returns the density calculus instance. The loaded density calculus becomes automatically the current density calculus.

network	an instance of the current network the density calculus will be attached to.
dc-directory	a directory string for a density calculus directory stored in the network databases.

3.2.14 • `load-fundamental-diagram-from-density-calculus` (*args*) *Function*

This function takes a network instance and the name of a subdirectory named “DENSITY-CALCULI” of a subdirectory of a network database. It reads one file of one of its subdirectories. By this, the new instance of the corresponding fundamental diagram object is created, its slot values converted from string to number, pointer etc and referenced. Finally, the complete density calculus object is attached to the network. The function returns the fundamental diagram instance.

network	an instance of the current network the fundamental diagram will be attached to
dc-directory	a directory string for a density calculus directory stored in the network databases.

3.2.15 • `get-border-speed` (*args*) *Function*

This function reads the speed [m/s] of the border between two different density zones from the border-speed-table of the currently used density calculus.

density-value-I	the key of the qualitative density value being in the zone before the border
density-value-II	the key of the qualitative density value being in the zone after the border

---

3.2.16 • `new-intermedium-zone-p` (*args*) *Function*

This function reads from the new-density-zone-table of the currently used density calculus whether a new density zone could be created between two density zones or not.

density-value-new	the key of the qualitative density value being in the zone that might be created at the border between both other zones

density-value-I	the key of the qualitative density value being in the zone before the border
density-value-II	the key of the qualitative density value being in the zone after the border

3.3 Internals

3.3.1 • `density-calculus-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'density-calculus-`

3.3.2 • `fundamental-diagram-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'fundamental-diagram-`

3.3.3 • `density-value-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined with exception of one instance having the unique key ´stop.

Default value: `'d-`

3.3.4 • `traffic-state-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'traffic-state-`

3.3.5 • `create-and-bind-value-item` (*args*) *Function*

This is an internal function called by read-value-file. It takes an input line with a number of tokens, separated by tabulators and terminated by a carriage return. As result a list with numbers is returned. The arguments are

input-line	a line of text tokens, separated by TABs and terminated by a CR.
number-of-values	a number

3.3.6 • `read-value-file` (*args*) *Function*

This function reads the for textual FileMaker database file representing a table with values from fundamental-diagrams, profiles etc. and generates an appropriate instance. The storage form is a textual and thus human readable form:

1. line: information

<slot-value-1>TAB<slot-value-2>TAB <slot-value-n>LF

2. line: value-descriptors

<value-name-1>TAB<value-name-2>TAB <value-name-n>LF

all other lanes:

<value-1>TAB<value-2>TAB <value-n>LF.

The arguments are

filename	a file name string describing the input text file
slot-list	a list of slot names. Each element of slot-list corresponds with an item of the input line read from the file filename. The slot names have to be identical to the slot names of the class from which instances are created.
class	For every input line one instance of class will be created.
key-prefix	a symbol which will be used to generate unique hash key strings for the instances that will be created (see also object-key-prefix)
key-slot	the name of the slot of the freshly created instance which gives the ID for the generation of hash key string.

3.3.7 • `build-traffic-state-key` (*args*) *Function*

This function takes a string of a number (as obtained by reading a textual database file) and returns a valid hash key string consisting of the appropriate object prefix and the number, e.g. ´TRAFFIC-STATE-8.

number	a number string

3.3.8 • `build-density-value-key` (*args*) *Function*

This function takes a string of a number (as obtained by reading a textual database file) and returns a valid hash key string consisting of the appropriate object prefix and the number, e.g. ´D-4.

number	a number string

3.3.9 • `build-fundamental-diagram-key` (*args*) *Function*

This function takes a string of a number (as obtained by reading a textual database file) and returns a valid hash key string consisting of the appropriate object prefix and the number, e.g. FUNDAMENTAL-DIAGRAM-2.

number	a number string

3.3.10 • `build-density-calculus-key` (*args*) *Function*

This function takes a string of a number (as obtained by reading a textual database file) and returns a valid hash key string consisting of the appropriate object prefix and the number, e.g. ´DENSITY-CALCULUS-1.

number	a number string

3.3.11 • `correction` (*args*) *Method*

Makes the appropriate corrections for the slots of a recently created instance of the fundamental-diagram-class. The 2nd argument is dummy.

candidate	an instance of fundamental-diagram-class
candidate-net	an instance of road-network-class (dummy)

3.3.12 • `complete-fundamental-diagram-value-list` (*args*) *Method*

This internal function completes the list of values given by a fundamental diagram. If this list only contains (density flow)-tuples, the speed value will be calculated and added to the tuple. Accordingly the value descriptors of the fundamental diagram are modified.

fundamental-diagram	the fundamental diagram instance

3.3.13 • `set-fundamental-diagram-maxima` (*args*) *Method*

An internal function for calculation of the maximum density, flow and speed of the values stored in the value list of the used fundamental diagram.

fundamental-diagram	the fundamental diagram instance

3.3.14 • `check-fundamental-diagram-boundary-conditions` (*args*) *Method*

This function checks the boundary conditions which a correct fundamental diagram must satisfy. The check includes only the simple tests.

fundamental-diagram	the fundamental diagram instance

3.3.15 • `create-density-value` (*args*) *Function*

An internal function for creation of a density value instance.

density-value-number	a number
density-value-name	the name of the density value

3.3.16 • `create-density-value-table` (*args*) *Function*

An internal function for creation of the hash table containing all density values of the density calculus. According to the number of density values the instances of density value objects are created.

number-of-density-values	a number

3.3.17 • `create-density-value-list` (*args*) *Function*

This function creates a sorted list with all density value instances, which will be ascendingly sorted by their density.

density-value-table	the hash table having all density value instances

3.3.18 • `set-density-value-means` (*args*) *Method*

This function calculates the mean values of density, flow and speed intervals desribing a qualitative density value. These values will be used in the calculation of the border speed between density zones and the split calculation.

density-value	the density value object

3.3.19 • `create-traffic-state` (*args*) *Function*

An internal function for creation of a traffic-state instance.

traffic-state-number	a number
traffic-state-key	a string
traffic-state-name	a string

3.3.20 • `create-traffic-state-table` (*args*) *Function*

An internal function for creation of the hash table containing all traffic states of the density calculus. According to the names of traffic states the instances oftraffic state objects are created.

traffic-state-name-list	a list with names for new traffic states

3.3.21 • `create-traffic-state-list` (*args*) *Function*

This function creates a sorted list with all traffic state instances, which will be ascendingly sorted by their density.

traffic-state-table	the hash table having all traffic state instances

3.3.22 • `set-traffic-state-borders` (*args*) *Method*

This function determines the minima and maxima of the density, flow and speed intervals describing a traffic state according to the list of attached qualitative density values.

traffic-state	the traffic state object

3.3.23 • `update-density-value` (*args*) *Method*

This function updates the modified density, flow and speed intervals describing a qualitative density value. It uses the actual given intervals of density and the values from the given fundamental diagram for corrections.

density-value	the density value object
fundamental-diagram	the fundamental diagram object used in the current density calculus

3.3.24 • `update-density-value-table` (*args*) *Method*

An internal function for updating the density, flow and speed intervals of all qualitative density values stored in the density value table. The function uses the values of the fundamental diagram attached to the density calculus.

density-calculus	the density calculus object

3.3.25 • `update-density-value-list` (*args*) *Method*

An internal function for resorting the list with the density value instances ordered by their density intervals.

density-calculus	the density calculus object

3.3.26 • `update-traffic-state-table` (*args*) *Function*

An internal function for updating the density, flow and speed intervals of all trafic states stored in the traffic state table. The function uses the attched density values stored in the density value table.

traffic-state-table	the hash table having all traffic state instances
density-value-table	the hash table having all density value instances

3.3.27 • `update-traffic-state-list` (*args*) *Function*

An internal function for resorting the list with the traffic state instances ordered by their density intervals.

density-calculus	the density calculus object

3.3.28 • `update-ts-to-dv-attachment` (*args*) *Function*

This function modifies the attachment of density values to traffic states. If a density value will be part of an other traffic state, the slots of the traffic states and the density value will be modified according to this change.

traffic-state-table	the hash table having all traffic state instances
density-value-table	the hash table having all density value instances

3.3.29 • `create-border-speed-table (`*args*`)` *Function*

This main fanction calculates a table, BORDER-SPEED-TABLE, containing the speed of the border between two density zones with different qualitative density value. The value of the border speed is ´NIL, if both density values are equal. This table will be attached to the actual density calculus.

density-value-table the hash table having all density value instances

3.3.30 • `create-new-density-zone-table (`*args*`)` *Function*

This main fanction calculates a table, NEW-DENSITY-ZONE-TABLE, containing lists of the keys of all qualitative density values wihich can be inserted at the border between two density zones with different qualitative density value according to the rules used in our density calculus implicitly (later on using the rules explicitly). The value of the list is ´NIL, if both density values are equal. This table will be attached to the actual density calculus.

density-value-table the hash table having all density value instances

4 The Simulator Object Classes

4.1 Classes

4.1.1 • `dynamic-standard-mixin` *class*

The standard mixin for all dynamic objects contains all informations necessary for the simulation.

superclasses	`identifier-mixin part-of-mixin`
time-of-next-event	`:initform infinity` `:accessor time-of-next-event`
next-event	`:initform nil` `:accessor next-event`
event-list	`:initform nil` `:accessor event-list`

4.1.2 • `dynamic-crossing-mixin` *class*

This mixin class contains the additional information specific for crossing objects, such as the connected cycle object, a list of the actual density values at the connected links and a ratiolist of the actual turning-ratios.

superclasses	`nil`
cycle-object	`:initform nil` `:accessor cycle-object`
density-list	`:initform nil` `:accessor density-list`
ratio-list	`:initform nil` `:accessor ratio-list`

4.1.3 • `dynamic-endpoint-mixin` *class*

This mixin class contains the specific additional information for endpoint objects, such as the actual quantitative and qualitative source value of the connected source and a list of the actual density values at the connected links.

superclasses	`nil`
actual-source-value	`:initform nil` `:accessor actual-source-value`
density-list	`:initform nil` `:accessor density-list`
qualitative-source-profile	`:initform nil` `:accessor qualitative-source-profile`

4.1.4 • `dynamic-link-mixin` *class*

This mixin class contains the specific additional information for link objects, which are the start object and the end object of the link.

superclasses	`nil`
start-object	`:initform nil` `:accessor start-object`
end-object	`:initform nil` `:accessor end-object`

4.1.5 • `sim-crossing-class` *class*

This class collects all informations about crossing objects.

superclasses	`dynamic-crossing-mixin` `dynamic-standard-mixin`

4.1.6 • `sim-endpoint-class` *class*

This class collects all informations about endpoint objects.

superclasses	`dynamic-endpoint-mixin` `dynamic-standard-mixin`

4.1.7 • `sim-link-class` *class*

This class collects all informations about link objects.

superclasses	`dynamic-link-mixin` `dynamic-standard-mixin`

4.1.8 • `coordinator-class` *class*

A coordinator controls the simulation and needs therefore a list of all simulation objects and a sorted list of the next simulation timepoint for every object.

superclasses	`identifier-mixin part-of-mixin`
tN-list	`:initform nil` `:accessor tN-list`
component-list	`:initform nil` `:accessor component-list`
start-time	`:initform nil` `:accessor start-time`
end-time	`:initform nil` `:accessor end-time`

4.1.9 • `situation-class` *class*

A situation describes the traffic on the network for one specific timepoint.

superclasses	`identifier-mixin part-of-mixin`
situation-time	`:initform nil` `:accessor situation-time`
situation-list	`:initform nil` `:accessor situation-list`

4.2 Interface Functions

4.2.1 • `*sim-coordinator*` *Variable*

The global variable, that contains the actual simulation-controller object.

4.2.2 • `*sim-situation*` *Variable*

The global variable, that contains a traffic situation. Used for creating, saving and loading of traffic situations. The simulation has to be started with a situation object.

4.2.3 • `simulation-install` *(args)* *Method*

This method creates a simulation controller, stored in the global variable `*sim-coordinator*`, creates all simulation objects and binds them to the simulation controller.

network	the current network instance on which the simulation shall run

4.2.4 • `start-sim` *(args)* *Method*

This method causes a simulation controller to start a simulation run for a given time horizon, starting with a given traffic situation at the situations time. In this method also the basic monitoring routines are started.

simulator	the simulation controller object
situation	the situation object for the start situation and start time
Th	the value of the time horizon to be simulated (in seconds)

4.2.5 • `compute-traffic-state` *(args)* *Generic*

The description of the generic functions for the computation of a traffic situation at one timepoint, that must be in the simulated time range of the simulation controller object. The time can be given as an absolute number of seconds, a relative number of seconds (after the simulation started time) or a time-string. The result will be a situation object.

simulator	the simulation controller object
time	the time parameter (number or string)
&key relative	the optional parameter for time identification default: nil

4.2.6 • `compute-traffic-state` *(args)* *Method*

This method computes the traffic situation at a timepoint given in the parameter time, if it is of type number. If the optional parameter relative is True, the number means the relative time (in seconds) to the simulation start time, otherwise the number stays for the absolute time (in seconds) after a global start-time.

simulator	the simulation controller object
time	a number
&key relative	True, if time is relative, nil otherwise default: nil

4.2.7 • `compute-traffic-state` *(args)* *Method*

This method computes the traffic situation at a timepoint given in the parameter time, if it is of the type string. The optional parameter relative is ignored, the given time always is an absolute time.

simulator	the simulation controller object
time	a string
&key relative	is ignored

4.2.8 • `compute-simobject-state` *(args)* *Generic*

The description of the generic functions for computing a traffic state of one simulation object, out of his simulation event list at a given time, that must be in the simulation time range.

simulator	the simulation controller object
objectkey	the key of a simulation object
time	the time parameter (number or string)
&key relative	the optional parameter for time identification default: nil

4.2.9 • `compute-simobject-state` *(args)* *Method*

This method computes the traffic state of a simulation object at a timepoint, given in the parameter time, if it is of the type number. If the optional parameter relative is True, the number means the relative time (in seconds) to the simulation start time, otherwise the number stays for the absolute time (in seconds) after a global start-time.

simulator	the simulation controller object
objectkey	the key of a simulation object
time	a number

&key relative	True, if time is relative, nil otherwise default: nil

4.2.10 • `compute-simobject-state` *(args)* *Method*

This method computes the traffic state of a simulation object at a timepoint, given in the parameter time, if it is of the type string. The optional parameter relative is ignored, the given time always is an absolute time.

simulator	the simulation controller object
objectkey	the key of a simulation object
time	a string
&key relative	is ignored

4.3 Internals

4.3.1 • `crossing-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'cross-`

4.3.2 • `endpoint-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'endp-`

4.3.3 • `link-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'link-`

4.3.4 • `cont-key-prefix` *Variable*

In order to generate unique hash key strings for storing of instances in hash tables consisting of a string concatenation of a prefix and a ID number a prefix symbol can be defined.

Default value: `'cont-`

4.3.5 • `create-sim-object` *(args)* *Generic*

The description of the generic functions for instanciating, initializing and referencing of a simulation object, for a given network object (node or lane).

object	the corresponding static object
network	the current network object

4.3.6 • `create-sim-object` *(args)* *Method*

This method creates a corresponding simulation object for a node object, It creates a crossing object, if the given object is not of the category `:open-end`, and creates an endpoint object, if the given object is of the category `:open-end`.

object	the corresponding static object of type node
network	the current network object

4.3.7 • `create-sim-object` *(args)* *Method*

This method creates a corresponding simulation object for a lane object. It creates a link object.

object	the corresponding static object of type lane
network	the current network object

4.3.8 • `get-endp-name` *(args)* *Macro*

Macro-function, that returns the key of the corresponding endpoint object to a given node object.

object	the node object

4.3.9 • `get-cross-name` *(args)* *Macro*

Macro-function, that returns the key of the corresponding crossing object to a given node object.

object	the node object

4.3.10 • `get-link-name` *(args)* — *Macro*

Macro-function, that returns the key of the corresponding link object to a given lane object.

object	the lane object

4.3.11 • `simstep` *(args)* — *Method*

All events in the simulation controllers tN-list, that have to be processed at the same given time are processed by this method.

simulator	the simulation controller object
time	the simulation timepoint to be processed

4.3.12 • `build-tN-list` *(args)* — *Method*

To initialize the simulation controllers tN-list, all simulation objects are called to compute their next (first) event timepoint, which are then stored and sorted by the time returned in the tN-list.

simulator	the simulation controller object

4.3.13 • `tN-insert` *(args)* — *Method*

To update the simulation controllers tN-list, this method deletes the entry of the given object in the tN-list and inserts a new entry of this object with the new given time at the right position of the ordered tN-list. If the given time is `ìnfinity, no new entry is inserted.

simulator	the simulation controller object
object	the object key of a simulation object
time	the time of the objects next event

4.3.14 • `select` *(args)* — *Function*

Returns True, if the second list, consisting of a simulation object key and a time-number, is strictly lower than the second list. Used to install a fixed order of simulation objects.

list1	first list to compare
list2	second list to compare

4.3.15 • `OUTPUT` *(args)* *Generic*

Definition of the generic functions for computing an simulation objects event including the computation of messages for other connected objects. In the OUTPUT-method the actual processed event is added to the objects event-list. The time is always relative time after the simulation start in seconds.

object	the simulation object
time	the simulation time

4.3.16 • `OUTPUT` *(args)* *Method*

Computation of events at crossings. The new calculation of density zones at the connected links and generation of messages for the links whose density zones at the links ends have changed.

object	the simulation object of type node
time	the simulation time

4.3.17 • `OUTPUT` *(args)* *Method*

Computation of events at endpoints. Generation of messages to the outgoing links.Events, that are caused by incoming links do not invoke any messages.

object	the simulation object of type endpoint
time	the simulation time

4.3.18 • `OUTPUT` *(args)* *Method*

Computation of events at links. Generation of messages to the connected nodes or endpoints.

object	the simulation object of type link
time	the simulation time

4.3.19 • `INT-TRANS` *(args)* *Generic*

Definition of the generic function for the determination of the next internal events at simulation objects. The time is always relative time after the simulation start in seconds.

object	the simulation object
time	the simulatoin time

4.3.20 • `INT-TRANS` *(args)* *Method*

Generation of the next internal event at crossings, that is the time of the next cycle change.

object	the simulation object of type crossing
time	the simulatoin time

4.3.21 • `INT-TRANS` *(args)* *Method*

Generation of the next internal event at endpoints, that is the time of the next change of the connected sources qualitative density value.

object	the simulation object of type endpoint
time	the simulatoin time

4.3.22 • `INT-TRANS` *(args)* *Method*

Generation of the next internal event for links, that is the intersection of two density zone borders, or the arrival of a density zone border at the links end.

object	the simulation object of type link
time	the simulatoin time

4.3.23 • `EXT-TRANS` *(args)* *Generic*

Definition of the generic functions for the processing of external events at simulation objects. The time is always relative time after the simulation start time in seconds.

object	the simulation object of type link
time	the simulatoin time
message	the list containing the external event

4.3.24 • `EXT-TRANS` *(args)* *Method*

Processing of external events at crossings which are lists of a connected link and its new density value. Every external event will cause a new computation of the density values of all connected links at the same time.

object	the simulation object of type crossing
time	the simulatoin time
message	the list containing the external event

4.3.25 • EXT-TRANS *(args)* *Method*

Processing of external events at endpoints, which are lists of a connected link and its new density value.

object	the simulation object of type endpoint
time	the simulatoin time
message	the list containing the external event

4.3.26 • EXT-TRANS *(args)* *Method*

Processing of external events at links, which are lists of a connected crossing or endpoint an its new density value at the links end. A new density zone of the given density value is installed at the links end.

object	the simulation object of type link
time	the simulatoin time
message	the list containing the external event

4.3.27 • `set-object-state` *(args)* *Generic*

Definition of the generic functions for the initialization of the simulation objects before a simulation run. This means initializing the event-list and other slots.

object	the simulation object
init-state	the initial state description

4.3.28 • `set-object-state` *(args)* *Method*

This method sets an initial state of a crossing object.

object	the simulation object of type crossing
init-state	the initial state description

4.3.29 • `set-object-state` *(args)* *Method*

This method sets the initial state of an endpoint object.

object	the simulation object of type endpoint
init-state	the initial state description

4.3.30 • `set-object-state` *(args)* *Method*

This method sets the initial state of a link object.

object	the simulation object of type link
init-state	the initial state description

4.3.31 • `get-state-at-timepoint` *(args)* *Generic*

Definition of the generic function for the computation of an object state at a given timepoint. The state is computed out of the objects event-list.

object	the simulation object
time	the (relative) time after the simulation start (in seconds)

4.3.32 • `get-state-at-timepoint` *(args)* *Method*

Computation of a crossing state at a given timepoint. The result is the last event description before the given timepoint.

object	the simulation object of type crossing
time	the (relative) time after the simulation start (in seconds)

4.3.33 • `get-state-at-timepoint` *(args)* *Method*

Computation of an endpoint state at a given timepoint. The result is the last event description before the given time and the actual source-value.

object	the simulation object of type endpoint
time	the (relative) time after the simulation start (in seconds)

4.3.34 • `get-state-at-timepoint` *(args)* *Method*

Computation of a link state at a given timepoint. The result is the interpolated density zone situation on the link at this time.

object	the simulation object of type link
time	the (relative) time after the simulation start (in seconds)

4.3.35 • `split-calculation` *(args)* *Function*

This function computes the new density zones at the connected links after an external event.

ratiolist	the actual turning ratios for the crossing
linkvalues	the actual density zones at the connected links
inlinks	list of the incoming links
outlinks	list of the outgoing links

4.3.36 • `compute-crossing-messages` *(args)* *Function*

This function computes the output messages of a crossing-object for the changed links

old-dens-list	the old link value description for all connected links
new-dens-list	the new calculated link value description for all connected links
sending-object-key	the key of the sending crossing-object

4.3.37 • `ask-for-ratiolist` *(args)* *Method*

This is an interface-function to the cycle-controller and returns a ratiolist that describes the turning ratios on a given crossing at a given time

cross	the crossing object for which the ratiolist shall be computed
time	the time at which the ratiolist shall be computed

4.3.38 • `ask-for-next-cycle-change-time` *(args)* *Method*

This is an interface-function to the cycle-controller and returns the timepoint of the next change in the cycle for a given crossing object (time relative to `*actual-time*`)

cross	the crossing object for which the next cycle-change-time shall be computed
time	time-offset for the desired timepoint

4.3.39 • `add-new-density-zone-at-link-end` *(args)* *Function*

This function processes the messages from neighbouring objects of a link, e.g. ´(“CROSS-8” “STOP”). It computes the new density zone list on the link at the timepoint of the arrival of that message by adding new density zones at start or end of that link. The result is the interpolated density zone situation on the link at the time of the arrival of the message.

link	the simulation object of type link
old-event	the description of the actual, old link state having the time of the last event, the corresponding density zone list, and the descriptor of this event
next-event	the message of an event from a neigbouring object containing sending-object-key and a density value

time	the (relative) time of next-event (in seconds)

4.3.40 • `compute-link-messages` *(args)* *Function*

This function computes the list of output messages of a link object, e.g. ´(“CROSS-8” (“LINK-1” “STOP”), by checking start and end of the new density zone list. These messages will be sent to neigbouring objects.

old-event	the description of the actual, old link state having the time of the last event, the corresponding density zone list, and the descriptor of this event
new-event	the corresponding description of the new link state
sending-object-key	the key of the link object

4.3.41 • `compute-next-link-event` *(args)* *Function*

This function computes the new density zone list on a link at the timepoint of the next internal event using the description of the old-event and the density-calculus. It returns list with the new event, if an internal event can be calculated. If no internal event can be derived, a list with the time-point ´INFINITY and the old density zone list is returned.

old-event	the description of the actual, old link state having the time of the last event, the corresponding density zone list, and the descriptor of this event

4.3.42 • `insert-new-density-zones` *(args)* *Function*

This function tests whether new density zones having maximal flow can be inserted in the density zone list of a link object and returns the new density zone list modified according to the insert-(rules-)table of the density calculus.

old-link-state-description	the actual density zone list of the link
density-value-with-max-flow	the key of the density value having the interval of highest traffic flow (and lower density)

4.3.43 • `interpolate-next-link-state` *(args)* *Function*

This function updates the density zone list of a link object using the actual density zone list of the link and the border-speed-table of the density calculus. It computes the new density zone list which results after a given time step.

old-link-state-description	the actual density zone list of the link
time-step	the (relative) time step of next-event (in seconds)

4.3.44 • `get-qualitative-density-value` *(args)* *Method*

This function computes the current qualitative density value at a certain time by using the qualitative profile of the simulation object of type endpoint or by using the quantitive profile of the source attatched to this object. It returns the key of the current density value.

endpoint	the simulation object of type endpoint
time	the current time (absolute or relative in seconds)
old-density-value-key	the key of the old density value describing the state of the source of the simulation object of type endpoint
relative	relative time (t) or not (nil)

4.3.45 • `get-next-profile-change-time` *(args)* *Method*

This function determines the point of time at which the current denstity value describing asimulation object of type endpoint will change to a new value. It uses the qualitative profile or it calculates the new density value from the quantitative values from the source object attached to the endpoint object. It returns a list containing the time and the key of the new density value.

endpoint	the simulation object of type endpoint
time	the current time (absolute or relative in seconds)
old-density-value-key	the key of the old density value describing the state of the attached simulation object of type endpoint
relative	relative time (t) or not (nil)

4.3.46 • `get-next-profile-change-time-from-profile` *(args)* *Generic*

Definition of the generic functions for the calculation of the point of time at which the current density value describing the state of the source of the simulation object of type endpoint will change to a new qualitative value. These functions use the quantitative profile of the network object of type source or sink.

source&sink-object	the network object of type source or sink
time	the current time (absolute or relative in seconds)
old-density-value-key	the key of the old density value describing the state of the source of the simulation object of type endpoint
relative	relative time (t) or not (nil)

4.3.47 • `get-next-profile-change-time-from-profile` *(args)* *Method*

This method uses an unknown format of the point of time.

source&sink-object	the network object of type source or sink
time	the current time
old-density-value-key	the key of the old density value describing the state of the source of the simulation object of type endpoint
relative	relative time (t) or not (nil)

4.3.48 • `get-next-profile-change-time-from-profile` *(args)* *Method*

This method uses a number as format of the point of time.

source&sink-object	the network object of type source or sink
time	the current time (absolute or relative in seconds)
old-density-value-key	the key of the old density value describing the state of the source of the simulation object of type endpoint
relative	relative time (t) or not (nil)

4.3.49 • `get-next-profile-change-time-from-profile` *(args)* *Method*

This method uses a string as format of the point of time.

source&sink-object	the network object of type source or sink
time	the current time
old-density-value-key	the key of the old density value describing the state of the source of the simulation object of type endpoint
relative	relative time (t) or not (nil)

4.3.50 • `create-qualitative-profile` *(args)* *Method*

This function computes the qualitative profile of a simulation object of type endpoint using the quantitative profile of the source&sink object the endpoint object is attached to. The qualitative profile contains the points of time, at which the density values change, and the key of those density values.

endpoint	the simulation object of type endpoint
start-density-value-key	the key of the density value starting the qualitative profile, if the transformation of the value starting the quantitative profile is ambiguous

4.3.51 • `transform-profile-value` *(args)* *Function*

This function transforms a quantitative profile value, like “0.12 cars/m”, into an adequate qualitative density value, like “D-5”.

quantitative-profile-value	the profile value(s) as a list

old-density-value	the key of the density value the new the density value will be next to, if the transformation of the profile value is ambiguous
profile-category	the category the profile value belongs to (flow, flow&speed or flow&occupation-ratio)

4.3.52 • `get-next-density-value-from-qualitative-profile` *(args)Generic*

Definition of the generic functions for the calculation of the point of time at which the current density value describing the state of the source of the simulation object of type endpoint will change to a new qualitative value. These functions use the qualitative profile of the endpoint object.

endpoint	the simulation object of type endpoint
time	the current time (absolute or relative in seconds)
relative	relative time (t) or not (nil)

4.3.53 • `get-next-density-value-from-qualitative-profile` *(args)Method*

This method uses an unknown format of the point of time.

endpoint	the simulation object of type endpoint
time	the current time
relative	relative time (t) or not (nil)

4.3.54 • `get-next-density-value-from-qualitative-profile` *(args)Method*

This method uses a number as format of the point of time.

endpoint	the simulation object of type endpoint
time	the current time (absolute or relative in seconds)
relative	relative time (t) or not (nil)

4.3.55 • `get-next-density-value-from-qualitative-profile` *(args)Method*

This method uses a string as format of the point of time.

endpoint	the simulation object of type endpoint
time	the current time
relative	relative time (t) or not (nil)

5 The Monitor

5.1 Interface Functions

5.1.1 • `show-fundamental-diagram` *(args)* *Method*

This method creates a window, into which the fundamental diagram will be displayed. A margin choice facility is added to the window. Thereby, the user can select one of three relations to be displayed in a diagram (traffic density & traffic flow, traffiv dansity & mean speed or traffic flow & mean speed). In addition to that, the density values can be displayed in those diagrams.

fundamental diagram	the density calculus object of type fundamental diagram,

5.1.2 • `show-profile` *(args)* *Method*

This method creates a window, into which the profile of an endpoint object in the simulation network will be displayed. A margin choice facility is added to the window. Thereby, the user can select one of the given profile values (traffic flow, mean speed or occupation ratio) to be displayed in a diagram. In addition to that, the density values describing the profile qualitatively will displayed in those diagrams.

endpoint	the simulation object of type endpoint,
with-qualitative-profile	indicates that both the qualitative and the quantitative description of the profile should be drawn.

5.1.3 • `show-border-speed-table` *(args)* *Method*

This method creates a window, into which the border speed table used in the current density calculus will be displayed. This table contains the border speed between two density zones with different density values. A margin choice facility is added to the window. Thereby, the user can select the unit of the items in that table (using m/s or km/h).

density-calculus	the density calculus object

5.1.4 • `show-new-density-zone-table` *(args)* *Method*

This method creates a window, into which the new density zone table used in the current density calculus will be displayed. This table contains the lists of density values a density zone might have, which will be inserted between two other zones according to the actual “insertion rule”. A margin choice facility is added to the window. Thereby, the user can select one of three actions: change the used “insertion rule”, display the actual “insertion rule” or display the new density zone table created by that “insertion rule”.

density-calculus	the density calculus object

5.1.5 • `show-event-list` *(args)* *Method*

This method creates a window, into which the event list of a simulation object of type link will be displayed. This event list has been created during a simulation run. It describes the bidimensional development of the density zones on the lane represented by the link objekt. A margin choice facility is added to the window. Thereby, the user can select one of several views on the event list: a preview of the whole list, the first or last part of the list or moving forward and backward through the list by selecting the next or the previous section of the event-list.

link	the simulation object of type link

5.1.6 • `show-link-profile` *(args)* *Method*

This method creates a window, into which a qualitative profile will be displayed. The qualitative profile is created by analysing the event list of a simulation object of type link. The profile will be created for each point on the lane represented by the simulation object link. A margin choice facility is added to the window. Thereby, the user can select one of several views on the qualitative profile: a preview of the whole profile, the first or last part of it or moving forward and backward through the profile by selecting the next or the previous section of the profile. In addition to that, one can select one of four quantitative intervals (traffic density, traffic flow, mean speed or occupation ratio) to represent the displayed (qualitative) density values. In order to compare qualitative to quantative values, a numeric profile as a reference profile will be displayed, too.

at-start	indicates that the qualitative profile at the start of the link should be displayed
at-end	indicates that the qualitative profile at the end of the link should be displayed
at-position	the point (in traffic direction) on the lane at which the qualitative profile should be displayed
reference-profile	a list with the type of value (speed, flow or occupation ratio) and a list representing a quantitative profile at the selected point

5.2 Internals

5.2.1 • `display-profile` *(args)* *Method*

This method displays the profile of an endpoint object in the simulation network using the profile of the attached source object.

endpoint	the simulation object of type endpoint
display-value	the key word of the qualitative value which should be displayed (flow, speed or occupation-ratio)
with-qualitative-profile	indicates that both the qualitative and the quantitative description of the profile should be drawn
with-raster	indicates that a raster should be drawn in the background of the profile or not
with-borderline	indicates that a borderline should be drawn around the profile
start-value-x	the start value on the x-axis
start-value-y	the start value on the y-axis
end-value-x	the end value on the x-axis
end-value-y	the end value on the y-axis
step-x	the step between two marks on the x-axis
step-y	the step between two marks on the y-axis
axis-raster-scale-x	the ratio between step-length and raster-step-length on the x-axis
axis-raster-scale-y	the ratio between step-length and raster-step-length on the y-axis
window-margin	a number

pixel-margin	a number
stream	the output medium

5.2.2 • `display-event-list` *(args)* *Method*

This method displays the event list of a simulation object of type link which has been created during the simulation run. The event list is a description of the bidimensional development of the density zones on the link object.

link	the simulation object of type link
modified-event-list	modification of the event list attached to the link object containing only the events of the section of the list which will be displayed
with-labels	indicates that the density zones would be marked by a label containing the symbol of the qualitative density value
with-borderline	indicates that a borderline would be drawn around the event list
with-history	indicates that a history would be drawn to describe which density value is attached ot which pattern, gray-level or color
start-value-x	the start value on the x-axis
start-value-y	the start value on the y-axis
end-value-x	the end value on the x-axis
end-value-y	the end value on the y-axis
step-x	the step between two marks on the x-axis
step-y	the step between two marks on the y-axis
axis-raster-scale-x	the ratio between step-length and raster-step-length on the x-axis
axis-raster-scale-y	the ratio between step-length and raster-step-length on the y-axis
window-margin	a number
pixel-margin	a number
stream	the output medium

5.2.3 • `display-link-profile` *(args)* *Method*

This method displays the qualitative profile created during the simulation run somewhere on the lane, which is represented by a simulation object of type link. The qualitative profile describes the development of the qualitative density values at the selected point. In addition to that, a numeric profile will be displayed as a reference profile.

link	the simulation object of type link
modified-event-list	modification of the event list attached to the link object containing only the events of a section of the list
reference-profile	a list representing a quantitative profile at the selected point
at-start	indicates that the qualitative profile at the start of the link should

	be displayed
at-end	indicates that the qualitative profile at the end of the link should be displayed
at-position	the point (in traffic direction) on the lane at which the qualitative profile should be displayed
display-value	the key word of the qualitative value which should be displayed (flow, speed or occupation-ratio)
reference-value	the key word of the numeric value which should be displayed (flow, speed or occupation-ratio)
with-raster	indicates that a raster should be drawn in the background of the profile or not
with-borderline	indicates that a borderline should be drawn around the profile
start-value-x	the start value on the x-axis
start-value-y	the start value on the y-axis
end-value-x	the end value on the x-axis
end-value-y	the end value on the y-axis
step-x	the step between two marks on the x-axis
step-y	the step between two marks on the y-axis
axis-raster-scale-x	the ratio between step-length and raster-step-length on the x-axis
axis-raster-scale-y	the ratio between step-length and raster-step-length on the y-axis
window-margin	a number
pixel-margin	a number
stream	the output medium

5.2.4 • `display-fundamental-diagram` *(args)* *Method*

This method displays the fundamental diagram used in the density calculus. It displays the quantitaive coherence between the traffic quantities traffic flow, traffic density and mean speed, which transformed into an qualitative coherence in the density calculus the fundamental diagram is attached to.

fundamental diagram	the density calculus object of type fundamental diagram
display-value	the key word for the function which should be displayed (density&flow, density&speed or flow&speed)
with-raster	indicates that a raster should be drawn in the background of the profile or not
with-borderline	indicates that a borderline should be drawn around the profile
start-value-x	the start value on the x-axis
start-value-y	the start value on the y-axis
end-value-x	the end value on the x-axis
end-value-y	the end value on the y-axis

step-x	the step between two marks on the x-axis
step-y	the step between two marks on the y-axis
axis-raster-scale-x	the ratio between step-length and raster-step-length on the x-axis
axis-raster-scale-y	the ratio between step-length and raster-step-length on the y-axis
window-margin	a number
pixel-margin	a number
stream	the output medium

5.2.5 • `display-border-speed-table` *(args)* *Method*

This method displays the border speed table used in the current density calculus. This table contains the border speed between two density zones with different density values. The unit of the displayed speed values will be m/s or km/h.

density-calculus	the density calculus object
displayed-unit	the unit of the values displayed in the table (:m-s or :km-h)
stream	the output medium

5.2.6 • `show-new-density-zone-table` *(args)* *Method*

This method displays the new density zone table used in the current density calculus. This table contains the lists of density values a density zone might have, which will be inserted between two existing zones. According to the “insertion rule” etiher only zones having the density value, which represents optimal traffic flow, will be inserted or zones having density values “between” the density values of the two existing density zones.

density-calculus	the density calculus object
density-value-key-list	indicates that only one density value for a zone to be inserted is allowed or all density values according to certain restrictions are allowed (nil)
stream	the output medium

6 Function Index