

UC Berkeley

Research Reports

Title

SmartBRT: A Tool for Simulating, Visualizing, and Evaluating Bus Rapid Transit Systems

Permalink

<https://escholarship.org/uc/item/863303bw>

Author

VanderWerf, Joel

Publication Date

2005-08-01

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

SmartBRT: A Tool for Simulating, Visualizing, and Evaluating Bus Rapid Transit Systems

Joel VanderWerf

**California PATH Research Report
UCB-ITS-PRR-2005-26**

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation, and the United States Department of Transportation, Federal Highway Administration.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California. This report does not constitute a standard, specification, or regulation.

Final Report for Task Order 4400

August 2005

ISSN 1055-1425

SmartBRT:

*A Tool for Simulating, Visualizing, and
Evaluating Bus Rapid Transit Systems*

SmartBRT Team
California PATH
UC Berkeley

August, 2004

This document is an overview of the goals and achievements of the project, and it contains links to the detailed documents and software produced by the SmartBRT project.

*These documents were intended primarily for reading in a web browser. Hypertext versions are available at the project release site:
<http://PATH.Berkeley.EDU/SMARTBRT/Release>. The entire documentation set can be downloaded for local browsing by downloading the tool package: sbrt-tool.zip.*

1 Executive Summary

Evaluation of BRT systems

SmartBRT is designed for modeling and simulating hypothetical transit systems, especially those making use of Bus Rapid Transit (BRT) technologies and policies. SmartBRT can be used to evaluate new technologies and policies that haven't been fully explored in deployed systems.

SmartBRT does not output emissions or cost data. SmartBRT does not calculate the long-term effects of BRT deployment, such as changes in mode choice.

To aid in evaluation, SmartBRT provides features for data gathering and statistics. The model report has a complete list of variables that can be measured and aggregated.

Evaluation of Transit systems

SmartBRT can also be used to evaluate traditional transit system designs. Paramics and other traffic simulation tools have some ability to do this kind of work. However, SmartBRT has an advantage even in cases without any BRT elements: in situations where input data is sparse, there is a corresponding reduction in the effort to specify the system. For instance, if all signal phase structures are hypothetically the same, the same description does not need to be repeated for each signal.

Visualization

Simulation outputs can be used to produce 3D animations. The animated world can be edited to show recognizable landmarks based on digital photographs. However, Paramics was not designed to model vehicle movement with high precision, and so vehicle movement in the animation may not always be realistic.

Applicability of Paramics

We determined that Paramics has some serious limitations for use in simulating BRT. These limitations are discussed in Paramics limitations and problems, section 3.4 of the model report.

Table of Contents

1	Executive Summary.....	2
2	Overview of Project.....	6
2.1	Overview of modeling capabilities.....	6
2.2	Overview of software.....	7
2.3	Continuing and future work.....	7
2.4	The SmartBRT team and website.....	11
3	Model Report.....	12
3.1	Model Overview.....	12
3.1.1	Interaction with Paramics.....	12
3.1.2	Passenger population.....	12
3.1.3	Passenger demand.....	13
3.1.4	Terminals, Stops, and Routes.....	13
3.1.5	Transfers, feeder routes, and shared right-of-way, passenger route choice.....	14
3.1.6	Bus generation and dispatch.....	15
3.1.7	Alighting, boarding, and dwell time.....	15
3.1.8	Signal priority.....	17
3.1.9	Bus movement.....	18
3.1.10	Random number generation.....	19
3.2	Inputs.....	20
3.2.1	Overview.....	20
3.2.2	Conventions.....	20
3.2.2.1	Input file format.....	20
3.2.2.2	Comments in input files.....	21
3.2.2.3	Format of documentation.....	21
3.2.3	BRT Input files.....	22
3.2.3.1	brt_bus_fare_types.....	22
3.2.3.2	brt_bus_type.....	22
3.2.3.3	brt_config.....	23
3.2.3.4	brt_dispatcher.....	23
3.2.3.5	brt_link.....	24
3.2.3.6	brt_link_follow.....	24
3.2.3.7	brt_log.....	25
3.2.3.8	brt_passenger_OD.....	27
3.2.3.9	brt_passenger_mobility_type.....	27
3.2.3.10	brt_passenger_payment_type.....	27
3.2.3.11	brt_route.....	27
3.2.3.12	brt_signal.....	28
3.2.3.13	brt_stop.....	28
3.2.3.14	brt_stop_arrival.....	28
3.2.3.15	brt_terminal.....	29
3.2.3.16	brt_trace.....	29
3.2.3.17	brt_transfer.....	30

3.2.3.18 brt_veh_to_log.....	31
3.3 Outputs.....	32
3.3.1 Statistical outputs.....	32
3.3.2 Passenger traces.....	32
3.3.3 Debugging outputs.....	32
3.3.4 Visualization outputs.....	34
3.4 Implementation of the SmartBRT plug-in.....	36
3.4.1 Paramics limitations and problems.....	36
3.4.1.1 Lateral movement of vehicles.....	36
3.4.1.2 Longitudinal movement of vehicles.....	36
3.4.2 Efficiency and scalability.....	37
3.4.3 Software engineering methodology.....	37
3.4.4 Topics for future work.....	37
4 BRTML Reference.....	39
4.1 Introduction.....	39
4.1.1 The need for BRTML.....	39
4.1.1.1 Key features of BRTML.....	39
4.1.1.2 A quick look at BRMTL.....	41
4.1.1.3 Principles of BRTML.....	43
4.1.2 Using BRTML.....	44
4.1.3 BRTML and Paramics: Two levels of detail.....	45
4.1.4 Numeric Data Types and Units.....	45
4.1.5 FAQ: Frequently Asked Questions about BRTML.....	45
4.2 Global Configuration.....	47
4.2.1 simulation.....	47
4.2.2 veh_to_log.....	49
4.2.3 trace.....	49
4.2.4 log.....	50
4.2.5 variable.....	51
4.2.6 statistic.....	52
4.3 Geometric model used in BRTML.....	52
4.3.1 Relative coordinates.....	53
4.3.2 Junction topology.....	53
4.3.3 Orientation: outbound and inbound.....	53
4.3.4 Subdivision.....	53
4.3.5 Corridors and networks.....	54
4.3.6 Lane numbering.....	54
4.4 Roadway types.....	54
4.4.1 street.....	54
4.4.2 block.....	54
4.4.3 junction.....	56
4.4.4 phase.....	57
4.4.5 major.....	57
4.4.6 medium.....	57
4.4.7 minor.....	58

4.4.8 demand.....	58
4.5 Transit system types.....	58
4.5.1 bus_stop.....	58
4.5.2 terminal.....	60
4.5.3 bus_line.....	61
4.5.4 bus_type.....	62
4.5.5 fare_type.....	63
4.5.6 mobility_type.....	64
4.6 Demand types.....	64
4.6.1 Traffic demand.....	64
4.6.2 Passenger demand.....	65
4.6.2.1 Time-varying passenger arrival rates.....	65
4.6.2.2 passenger_arrivals.....	65
4.6.2.3 arrival.....	65
4.6.3 Passenger OD tables.....	66
4.6.3.1 passenger_demand.....	66
4.6.4 Passenger characteristics.....	66
4.7 Using other data sources.....	66
4.8 Reference Syntax.....	67
4.9 Glossary.....	68
5 Tutorial.....	75
6 User's Manual.....	75
6.1 System requirements.....	75
6.2 Installing Paramics plug-ins.....	75
6.3 Installing platform-independent SmartBRT software tools.....	77
6.4 Troubleshooting.....	77
6.4.1 Using Modeller in Windows.....	77
6.4.2 Using plug-ins on Linux.....	78
6.4.3 Using plug-ins on Windows.....	78
6.4.4 Modeller License Issues.....	78
6.5 Running SmartBRT.....	79
6.5.1 BRTML Tools.....	79
6.5.2 Running SmartBRT in Modeller.....	80
6.5.2.1 Batch runs and other command-line runs.....	80
6.6 SWEditor Manual.....	83
6.7 SmartBRT Software Downloads.....	83
7 The Wilshire Model.....	84
7.1 Network Coding.....	85
7.2 Passenger Demand Model.....	88
7.2.1 Data Collection.....	88
7.2.2 OD Estimation.....	89
7.2.3 Photographic Images.....	89
8 Copyright and License.....	90

SmartBRT:

A Tool for Simulating, Visualizing, and Evaluating Bus Rapid Transit Systems

2 Overview of Project

2.1 Overview of modeling capabilities

SmartBRT is capable of modeling some BRT technologies and policies. Some of the missing modeling features are listed under continuing and future work. Paramics by itself provides very few BRT features at all. Some of the BRT design elements and characteristics that can be modeled in SmartBRT are listed below. Please see the model report for details.

- Bus capacity
- Bus headway and speed limit
- Bus doors: number, width, and load/unload policy
- Low-floor or kneeling buses (as an effect on dwell time)
- Bus docking (as an effect on dwell time)
- Lane restrictions
- Transfers and feeder routes; trunk routes; several bus routes on same street
- Signal priority
- Stops with fare pre-payment
- Fare collection mechanisms (on bus)
- Dead heading—the policy of dispatching buses along an alternate high-speed route

SmartBRT also has some flexibility in modeling non-transit aspects of the corridor:

- Passenger demand at each bus stop (which may vary by time of day) and OD tables
- Characteristics of the passenger population:
 - Fare payment type
 - Mobility level (specifically, as it affects boarding and alighting from buses)
- Traffic demand

- OD Tables
- Turning ratios in junctions

2.2 Overview of software

The SmartBRT software comes in three separate but related components:

- A language for describing transit corridors and simulation experiments involving them. This language (BRTML) allows specification at a more conceptual level than the usual inputs for Paramics Modeller. The SmartBRT software package also include processing tools for this language, as described in the user's manual.
- A set of extensions to Paramics (the extension software is called a *Paramics plug-in*) to add the ability to simulate a wide variety of BRT technologies and policies. Typically, a Paramics plug-in is used to adapt the behavior of Paramics. The SmartBRT plug-in goes further and adds a new layer of simulation involving buses, stops, passengers, and terminals; these entities may have BRT characteristics and behaviors that are not available in standard Paramics. Paramics models are still used for traffic effects.
- A tool for viewing 3D animations of simulation outputs, and for enhancing these animations with photo-quality 3D graphics of buildings, infrastructure, vehicles, and so on. This tool is called SWEditor, the Simulated World Editor.

For performing transit evaluations, only the first component, BRTML, needs to be understood in detail. The BRTML language provides a seamless, unified interface to the simulation capabilities of Paramics extended with the SmartBRT plug-in. Although Paramics and the SmartBRT plug-in each have their own configuration systems—Paramics through the Modeller program or the input files, and the SmartBRT plug-in through its own input files—the BRTML user may generally ignore these configuration systems.

For a detailed look at the relation between the plug-in and Paramics, see the model report. In particular, there is a section (Paramics limitations and problems) discussing some problems using Paramics for bus simulation and the work-arounds we developed.

2.3 Continuing and future work

- *Rigorous testing and calibration for a wide variety of configurations*
 - SmartBRT has been used in a number of special cases. However, most of these cases are not associated with a real corridor that can be used for validation. Also, SmartBRT has not been tested in cases representing the full breadth of transit systems.
 - Data from the SamTrans and VTA projects can be used to

further calibrate the models.

- *Wilshire corridor case study*
 - The SmartBRT project sought to develop a general tool, but at the same time carry out a highly detailed case study of a large, complex urban transit corridor—Wilshire Blvd. in Los Angeles. This work is still in progress, but has yielded a set of processing tools for converting other data formats into BRTML. These tools are site-specific and are not part of this general purpose deliverable.
- *Extended tutorials and case studies*
- *Modeling work*
 - *Passenger route choice*
 - This model would be necessary to support complex networks of bus routes in which passengers must choose between two or more itineraries to reach their destinations.
 - *Signal priority only for designated buses*
 - *Alternate signal priority algorithms*
 - Early green, coordination, etc.
 - *Alternate sensor and communication models as basis for signal priority*
 - Loops, GPS, AVL, DSRC, etc.
 - *Off-line stops*
 - *Queue jump lanes*
 - *Time-varying passenger OD tables*
 - Arrival rates may be time varying in the current version of the software, but OD tables are fixed.
- *Software tools*
 - *Corridor design tool with graphical user interface (GUI)*
 - A “wizard” type of tool that leads the user through a series of questions and outputs a BRTML corridor description. This would be easy to implement, but would not give access to the full expressiveness of BRTML.
 - A general corridor design tool. This would be hard to implement, because of the diversity of

corridors and transit systems.

- *An experimental design and management tool, analogous to Paramics Processor*
 - An experiment may consist of variations in a set of variables: you may be interested in a range of values for both demand variables and for service variables. Each choice of values for these variables requires a separate run. The user specifies what data is gathered from each run, how that data is aggregated across runs, and what plots are generated produced from this data. Currently, this work must all be done manually.
- *Run-time simulation monitor with graphical user interface (GUI)*
 - Currently, there is no useful visualization of a SmartBRT simulation at run-time. Paramics Modeller is not capable of displaying the state of the SmartBRT entities (buses, terminals). SmartBRT works around this problem by providing tracing outputs (see the tracing section of the model report) that display events as they happen. However, this is primarily useful with Paramics Processor, the text-only command-line version of Modeller. To extend Modeller to show SmartBRT information would probably be very difficult, and might be possible only if the developers of Paramics are willing to make changes to their software.
 - An alternative is a two-fold approach:
 - Extend Modeller as possible to show some BRT-related information, possibly using the OpenGL features that have recently been made available in V4 of Paramics. For instance, we could draw text in the display window that shows passenger counts on buses and at stops.
 - Develop a separate program that runs concurrently with Modeller, communicating BRT event

information from Modeller to the new program, and allowing the user to interact with the event data more easily than using the text trace output. Interaction would include “wait for the next event matching specified criteria”, “show most recent event data for specified object”, etc. It might even be possible to allow the user to control the running simulation from the external program, if the Paramics API supports (or can be extended by Quadstone to support) pausing and restarting the simulation.

- This program should also let the user manage outputs and set up complex experiments involving sequences of runs.
- *Address the effect of Paramics vehicle movement problems on visualization*
- Vehicle trajectories output by Paramics are not well suited to high-fidelity visualization: vehicles often move in physically unrealistic ways. Paramics was originally intended for large scale traffic simulations, and so it traded realism for efficiency. In particular:
 - Vehicles sometimes collide, especially in intersections.
 - Vehicles sometimes move sideways.
 - Buses do not completely stop at bus stops. They creep forward and must be periodically translated backwards.
 - Movement is jerky.
- It may be possible to solve the problems by post-processing the output. One possible approach is to use vehicle trajectories as inputs to another simulation, using each trajectory point as a target for a controller, and using a

double integrator for the vehicle dynamics. This would have some degree of physical realism, while smoothing out jerkiness of the Paramics trajectories. Additional preprocessing would be necessary to solve the other problems.

- *More realistic urban visualization*
 - Automatic generation of buildings and road fixtures from corridor design document.
 - Include a library of common buildings and fixtures that could be used as is or extended with digital images.

2.4 The SmartBRT team and website

Team members, contributors, and friends:

- Tunde Balvanyos
- Wes Bethel
- Yonnel Gardes
- Natalia Kourjanskaia
- Hongchao Liu
- Mark Miller
- Jim Misener
- Joao Sousa
- Swekuang Tan
- Wenbin Wei
- Joel VanderWerf
- Yafeng Yin
- Wei-Bin Zhang

The current point of contact for SmartBRT is Joel VanderWerf, vjoel@path.berkeley.edu.

The SmartBRT web site is <http://PATH.Berkeley.EDU/SMARTBRT>.

3 Model Report

This report documents the transit system models used in the SmartBRT Bus Rapid Transit simulation, including parameters and performance measures, It also documents input and output files of the software and some aspects of the implementation of the models as a Paramics plug-in.

3.1 Model Overview

3.1.1 Interaction with Paramics

Paramics itself has limited support for advanced bus systems. However, Paramics provides a fairly flexible API (Application Program Interface) through which additional capabilities can be defined. The code, typically written in a programming language like C, that defines these additional capabilities is called a *plug-in*. The plug-in extends the behavior of Paramics by defining *overload* functions. These are functions defined in the plug-in that Paramics calls at specific points in time, such as at start up, after each time step, or when a bus or other vehicle is created.

To a large extent, the SmartBRT entities do not interact directly with Paramics entities. The passengers and stops defined in SmartBRT are completely unknown to Paramics. Terminals, buses, links, and signals in SmartBRT are correlated with Paramics entities, but maintain separate state that pertains to the BRT application. Buses are based not on the Paramics bus type, but on a type of heavy vehicle.

Much of the behavior of the model is apparent from the parameter descriptions, which are in the section on inputs. The less trivial aspects of the model are discussed in the following sections.

See the section on Implementation for more discussion about implementing the model as a Paramics plug-in.

3.1.2 Passenger population

The passengers that move through the bus system are all drawn from the same population, in the sense that passenger characteristics are sampled from a single set of distributions, regardless of the time or location at which the passenger enters the simulation. Each passenger has a mobility type and a payment type, which play a role in the computation of dwell time. These characteristics are independent of the passenger's origin and destination.

Mobility types are not limited to a predefined list in SmartBRT, but are an input to the simulation defined in the `brt_passenger_mobility_type` input file. Each mobility type specifies the times required to board and alight in both high-floor and low-floor buses. Details are in Section 2.

Payment types are similarly definable in the `brt_passenger_payment_type` input file. Each payment type specifies the time that the form of payment takes, assuming the

bus is equipped for that kind of payment. Each bus type (also definable) specifies a list of payment types that are accepted on buses of that type. If the passenger's payment method is unavailable on a bus, then a default method with its own time setting is used (this can be thought of as paying by cash).

Mobility types are defined for the population as a whole, as are payment types. The two distributions are assumed to be probabilistically independent.

Passengers are logged individually in some detail. See the section under `brt_config` on the `log_passengers` setting.

3.1.3 Passenger demand

Passenger demand is described in terms of arrival rates and OD (Origin-Destination) ratios. These two sets of variables can be adjusted independently.

Each stop has its own arrival rate. Arrival rate can be constant or time-varying in a general way. The initial arrival rate for a stop is the specified in the `frequency` column in the `brt_stop` input file. The frequency for a stop may change any number of times during the simulation, according to the optional `brt_stop_arrival` input file. Each entry in this file determines a time at which the frequency changes and the new frequency. This can be used to model many different demand profiles:

- Spikes in the arrival rate due to scheduled train arrivals, or the end of the school day.
- Changes that persist for long periods, due to rush hour.
- Gradual increase of demand, such as the hours approaching the morning peak.

The OD ratios define what proportion of passengers entering the system at particular origin stop wish to travel to a particular destination stop. Currently, OD ratios cannot vary over time. The syntax used to specify these proportions is described in the section on the `brt_passenger_OD` input file.

3.1.4 Terminals, Stops, and Routes

In our model, a bus route requires at least two terminals and a route that follows stops between them. Terminals are not passenger stops, although they may be placed very close to passenger stops. Terminals control the following aspects of the bus route:

- The number of buses available to be started from this terminal.
- The type of buses that run along the route. A terminal can generate buses of only one bus type.
- Headway used by buses along the route.
- Whether or not buses along this route should wait at each stop to maintain uniform time headways.

The generation of buses by terminals is described in a subsequent section, along with the special case of bus dispatch.

As already noted in 1.3, stop characteristics determine the passenger demand levels. A stop also can be designated to have off-board fare collection: the passengers may pay upon entering the stop, so that there is no delay on boarding.

The route followed by buses on a bus route is determined by a sequence of stops in each direction. The sequence of stops serves to determine both the stops which the buses service and the geographical route followed by buses. A stop may be served by any number of routes.

SmartBRT terminals are associated with Paramics zones, which provide the mechanism used to generate vehicles. The input files which define these entities are `brt_terminal` and `brt_route`.

3.1.5 Transfers, feeder routes, and shared right-of-way, passenger route choice

SmartBRT bus routes can cross and even run along the same roadway. A stop can be served by several routes, or service at the stop can be restricted to just some of the routes. A pair of stops can be designated as transfer stops between two routes even if they are not in the same physical location. These features allow a variety of system designs, such as:

- A BRT route along a corridor with transfers to and from feeder routes along streets crossing the corridor.
- BRT and local service running along the same right-of-way, with the BRT route serving a subset of the local stops.
- Several crossing BRT routes.
- Several BRT routes that funnel into a single busy corridor (a trunk route).
- Transfer areas that are larger than a normal stop and may require a short walk to make the connection (around a corner or through a building).
- Dead-heading—bus dispatch along an alternate high-speed route (using the dispatcher features described in the next section).

SmartBRT associates each bus route with a sequence of links that the bus will follow and a sequence of stops that the bus will service. These are defined in the `brt_link_follow` and `brt_route` files. Transfer stops between routes are defined in the `brt_transfer` file.

The topology of routes can be very complex. However, passengers modeled by SmartBRT may not respond to this complexity in a realistic way. This is because the model of passenger route choice is very simple. Passengers just take the first suitable bus, which means the first bus that will take them either to their destination or to a stop where they can wait for another bus that will take them there. This model would impede a study of local service and BRT service sharing right of way, since passengers might not take the fastest route to their destination. A more intelligent model of passenger route choice would be required, taking into account passengers' estimates of how long a trip will take depending on which bus is available and how long future bus arrivals will take. (For

example, a passenger would need to choose between a local bus that is at the stop and a BRT bus that has not yet arrived.)

3.1.6 Bus generation and dispatch

Normally, a bus route will consist of two terminals at the end of the route, with buses running back and forth between them along a corridor with many stops. As noted above, the number of available buses is specified in advance and a headway time may be specified to prevent two buses from being released too close together. These parameters are listed in the section on the `brt_terminal` file.

However, a bus route operator may wish to run additional buses on a highly traveled segment of the corridor, by inserting and removing buses at locations other than the normal terminals. Or the operator may wish to have some buses travel in only one direction on the corridor and take a faster route (a parallel freeway, for instance) for the return trip.

The former strategy requires defining an additional route that services some subset of the original route's stops, but has terminals located near the ends of the shortened segment. (Since terminals are associated with Paramics zones, and zones must be on dead-end streets, these terminals should be located on cross streets that join the corridor near where the additional service begins or ends.) Having two routes along the same corridor does not interfere with passenger route choice. A passenger will take a bus, regardless of the route it is associated with, as long as taking the bus will get the passenger closer to the passenger's destination.

The accelerated return trip strategy is achieved by specifying the two terminals and the bus travel delay between them in the `brt_dispatcher` file.

3.1.7 Alighting, boarding, and dwell time

Each bus type defines a distinct number of doors of each usage type, alight only, board only, and mixed board and alight. Each door represents the ability to load or unload one passenger at a time, with the duration depending on the passenger characteristics.

At the stop, passengers wait in an orderly queue and are given the option of boarding the bus in the same sequence that they arrived at the stop (they might not want to board the bus if its destination is not useful to them).

On the bus, passengers are seated in first-come-first-served order. The remaining passengers are considered to be standing. Thus, at any point in time, a passenger is in one of three states: seated in a bus, standing in a bus, or waiting at a stop. The total time spent in each of the three states is recorded for each passenger (see the `log_passengers` option in the `brt_config` input file). The seated and standee capacity is determined by the bus type.

The total dwell time of a bus at a stop depends in a complex way on the door arrangement of the bus, the bus's other characteristics, the numbers of boarding and alighting passengers, and their board and alight times. The board and alight times are simple

enough to express in closed form as simple algorithms, discussed below. However, dwell time is too complex to express as a closed form function of inputs. Because of the complexity of the calculation, dwell time is evaluated in the simulation by actually simulating sequences of alight and board events over time. This has the advantage of being more realistic for passenger tracking, since the exact time of boarding and alighting is known.

Algorithms for calculating individual passenger board and alight times can be expressed as the following closed form pseudocode. Note the dependence on whether or not the bus type specifies a low floor. Each passenger mobility type can react to this setting in different ways, as defined in the `brt_passenger_mobility_type` file. A stop with off-board fare collection is recognized as eliminating payment from the board time calculation. The default payment time mentioned in the following is a parameter of the bus type, and typically will be chosen to represent cash payment time.

```
define calculate_passenger_board_time(passenger, stop, bus) as:
  set board_time = 0
  if the bus's type is low_floor
    increment board_time by the passenger's mobility_type's lo_floor_board_time
  else
    increment board_time by the passenger's mobility_type's hi_floor_board_time
  end
  if the stop does not have the fare_at_stop option
    look up the passenger's payment_type
      in the bus's type's fare_collection_table
    if found
      increment board_time by the passenger's payment_type's time
    else
      increment board_time by the bus's type's default_payment_time
    end
  end
end
return board_time
end

define calculate_passenger_alight_time(passenger, stop, bus) as:
  if the bus's type is low_floor
    set alight_time = the passenger's mobility_type's lo_floor_alight_time
  else
    set alight_time = the passenger's mobility_type's hi_floor_alight_time
  end
  return alight_time
end
```

Our model of a bus dwelling at a stop and passengers alighting and boarding is subject to the following assumption and limitations:

- With mixed-use doors, boarding passengers politely wait for alighting passengers to use the doors. Alighters use the doors selfishly regardless of effect on boarders.
- Board times are not “pipelined”, which would mean that the time one passenger uses to pay could (at least partly) overlap with the time the following passenger needs to board.
- Standees remaining on the bus after alighters have alighted will take seats if they can, before boarders can get to them.
- For an approximate model of circulation time of passengers on the bus, the user can increase the alight time inputs or the base dwell time of the bus. There is no separate model of circulation.
- Buses may have more than one door for boarding, for alighting, or for both. In reality, boarders may tend to cluster around a subset of the available boarding doors, and alighters may go to the nearest door even if they have to wait longer. In SmartBRT, however, these phenomena are not modeled, and boarding and alighting passengers choose from the usable doors in the most efficient way possible: as soon as a door is clear for use, some passenger will begin to use it.
- If two or more doors are assigned the same use (boarding, for example), the total door-use time is simply divided equally by the number of doors. In other words, the total board time is calculated as if there were only one door and then divided by the number of doors. This is slightly unrealistic (on the order of one passenger's board time), but greatly simplifies the algorithm. Similar remarks apply to the alight door case.
- Passengers who arrive at the stop while the bus is boarding do not board. If the bus route has been selected to hold buses at stops to maintain headway, however, late arriving passengers may board while the bus is holding.
- Passengers board the first bus that will get them closer to their destination. However, this bus may not be the best choice in terms of their travel time or in terms of their ability to find a seat. Additional modeling work is necessary to support passenger decision (e.g., for local service vs. BRT service).

3.1.8 Signal priority

Signals apply to SmartBRT buses as they do to any other vehicle in the Paramics simulation. However, signals along the BRT route can be designated to adjust their phases to give buses priority. Specifically, signals can extend their green phase for up to a specified maximum number of seconds, if it will allow an approaching bus to pass. Currently, whether the approaching bus is granted an extension does not depend on whether it is BRT service or not. The algorithm is as follows.

If the next signal ahead of the bus supports signal priority on the direction the bus is approaching from, and if the bus is within 200m of the signal, we estimate the bus arrival time at the signal based on the current speed of the bus and the distance of the bus to the signal. The bus requests the signal to extend its green phase to allow the bus to pass through the intersection before the end of the phase. If the current phase is red, the

request is denied. If the requested extension is greater than the specified maximum (the “green hold” parameter of the signal), the request is denied. Otherwise, the request is granted.

Note that the bus makes a new request on every time-step. If the bus slows, and there is room for a longer extension within the maximum, then the originally granted extension will be further extended to provide enough time to cross the intersection.

This is not a model of any particular mechanism of bus sensing, telemetry, communication, or signal control. Rather, it assumes perfect information about the state of the bus and signal and perfect control of and communication with the signal. This model can be thought of as a best case model.

3.1.9 Bus movement

The model of bus movement used in SmartBRT is a hybrid of the vehicle movement models built into Paramics and our own specialized model of bus behavior. The built-in model we chose as the basis for our work is the heavy vehicle model, rather than the bus model, because the latter is designed to work with the simplistic bus route and passenger system built into Paramics.

The behavior of a SmartBRT bus (which we are modeling as a heavy vehicle) depends on whether it is approaching a stop. The bus is considered to be approaching a bus stop when

$$(*) \quad d \leq \frac{v_0^2}{a_{max}}$$

where

d is the distance to the stop,

v_0 is the current velocity, and

a_{max} is the maximum deceleration.

This formula is chosen to ensure that the bus could make the stop using only about half of its maximum deceleration.

When the bus is not approaching a stop, the control mode depends on whether the bus is in free flowing traffic with no speed limitation imposed by traffic. In this case, the bus uses a simple proportional controller to maintain its desired speed, subject to limits imposed by the bus type, the link, and other limits defined by Paramics. In the case when surrounding traffic prevents the bus from traveling at its desired speed, the Paramics car following model takes control. This ensures reasonable traffic flow effects, to the extent that the inherent model in Paramics is reasonable.

When the bus is approaching a stop, the bus must be forced to stop at it, because our SmartBRT bus stops are not known to Paramics. The bus must begin deceleration soon enough, remain stationary while passengers alight and board, and accelerate at a comfortable and realistic level after leaving the stop.

The maximum deceleration is the lesser of the bus type's braking limit, set in the `brt_bus_type` file, and the limit computed by Paramics as the hardest acceptable braking in the current situation. After detecting that the bus is approaching the stop (in other words, that condition (*) is satisfied), the bus decelerates, with an acceleration value determined by the following formula:

$$a = \frac{-v_0^2}{2d}$$

This formula should yield precisely the deceleration needed to stop after distance d , the distance from the bus to the stop. Because of the choice approach threshold discussed above, this value will typically be well below the maximum.

3.1.10 Random number generation

Paramics has its own pseudo-random number generator (PRNG), and the user may set the seed in the Paramics `configuration` file. However, this number sequence is not used by the SmartBRT plug-in. The primary reason for this is that it is often desirable in Monte Carlo simulation to use a different and independently adjustable source of randomness for each random variable. The significant random variables contributed by the SmartBRT plug-in are those that are associated with each stop and control passenger demand: passenger interarrival times, destination choices, and mobility and payment characteristics. Hence we allow the user to supply a seed for each stop, which is used to generate the random variables at that stop.

To generate the sequences, we use a version of the Mersenne Twister algorithm invented by M. Matsumoto and T. Nishimura. ("Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator", *ACM Transactions on Modeling and Computer Simulation*, Vol. 8, No. 1, January 1998, pp 3—30. See also

<http://www.math.keio.ac.jp/~matumoto/emt.html>.)

It is fast, has a long period, has good distribution characteristics, and is ideally suited for Monte Carlo simulation.

3.2 Inputs

3.2.1 Overview

This section describes the parameters that determine the behavior of a single simulation run. These parameters fall into two sets:

- The standard Paramics parameters as contained in the `nodes`, `links`, etc. files. We do not describe these files here, since they are described in the Paramics manuals.
- The SmartBRT-specific parameters, which the SmartBRT plug-in software reads and uses to generate its model of the transit network. These parameters are located in files named with the prefix `brt_` to distinguish them from other input files and from output files. We discuss these files in this chapter.

Note that **the plug-in input files are not the intended user interface for SmartBRT**. They require a fine-grained and literal description of the transit aspects of the model. For instance, the `brt_link` and `brt_link_follow` files require a large quantity of essential but uninteresting detail. For every link, and for every bus route that runs along it, there must be an entry in `brt_link_follow` defining the next link that a bus belonging to that route must take and the destination terminal of the bus. The user typically would prefer simply to specify a list of streets that the bus route runs along, or a list of stops that the bus makes.

The BRTML language and compiler, documented in the User Manual, is a more user-oriented high-level interface that generates the low-level plug-in input files from a high-level BRTML specification file. The BRTML compiler can, for instance, translate the simple block-oriented route description into the appropriate plug-in input files.

This chapter describes the lower level input files for two reasons. First, they represent in a simple way the entire range of configuration possibilities for the plug-in, regardless of use of BRTML. Second, it is sometimes necessary to examine these files to debug a network or to understand a message produced by Paramics or by the plug-in. (The situation is analogous to beginning a course on biology with a discussion of the chemical compounds found in living organisms, even though the bulk of the subject deals with larger-scale systems.)

3.2.2 Conventions

The files that are particular to the SmartBRT plug-in are always given a name beginning with “`brt_`”, to distinguish them from Paramics input files, output files, etc. All files, input and output, associated with a given network are kept in the same directory.

3.2.2.1 Input file format

For simplicity, all SmartBRT input files are formatted as tabular data. Each row (i.e., line in the file) represents an individual object of the same kind, such as a bus stop, a bus type,

a logging switch, or a passenger payment type. Columns (i.e., fields on a line) are separated by whitespace (spaces or tabs), and represent some attribute or characteristic of that object. For example, the `brt_terminal` file might look like this:

```
# zone_id dest_id route_name bus_type headway start_count headway_hold offset
1       2      L1        bus      120      1          0          200.0
2       1      L1        bus      120      0          0          200.0
18      8      L2        bus      120      0          0          200.0
8       18     L2        bus      120      1          0          200.0
```

Each row represents a terminal. The data type and (if that type is numeric) the units of each entry are determined by the column. The first column, “zone_id”, is the Paramics zone in which the terminal is located (zone indexes are defined in the Paramics zones file). The “dest_id” is also a zone id—it’s the destination zone of buses leaving this terminal. Some entries, such as the “L1” and “bus” values, symbolically connect this file, `brt_terminal`, with other SmartBRT input files, which in this case are the files defining bus routes and bus types, respectively. The “L1” refers to a particular route, and the “bus” refers to a particular bus type. The value in the headway column is a time, in seconds. The value in the start_count column is a bus count. The value in the headway_hold column is a boolean (1 is true, 0 is false). The value in the offset column is a distance, in meters. The details of the `brt_terminal` file are discussed in depth in a section of this chapter.

As an aside, we mention three of the ways in which BRTML simplifies these inputs:

- BRTML lets you specify units as part of the data, decreasing ambiguity.
- BRTML puts all input data in one file and uses a consistent notation for references between one object and another, as opposed to, for example, the *ad hoc* use of route names in the `brt_terminal` file.
- BRTML lets you group terminals and define common features (such as, in this case, offset) for all terminals in the group.

3.2.2.2 Comments in input files

The SmartBRT input files may have comments. Comments run from a '#' character to the end of the line. The files generated by the BRTML compiler typically have comments inserted for readability. The first line is often a comment describing the fields of each data line in the file, as in the example above.

3.2.2.3 Format of documentation

In this document, we consider each input file in turn. Since the format of the input files is very consistent, the format of this document is, too. We explain what kind of objects the lines of the file represent, and what characteristic of the objects the fields represent. Much of this documentation is in 9 point courier and is preformatted with space characters. This is because the source of the text is C include files in the SmartBRT plugin source code. To reduce maintenance effort, the text in the include files is simply copied to this document, retaining its original format.

3.2.3 BRT Input files

This section describes, in alphabetical order, every input file that is recognized and read by the SmartBRT plug-in. Paramics input files are described in the Paramics documentation.

3.2.3.1 brt_bus_fare_types

Each bus type has a list of allowed fare types, represented in the `brt_bus_fare_types` file.

The `brt_bus_fare_types` file has the following fields on each line:

```
type_name           = name of bus type
fare_name           = name of a fare type (e.g., "smartcard")
```

There is no limit to the number of fare types, or to the number of types which a particular bus type can accommodate. Simply use a line in this file for each bus type and each fare type that it accommodates. The fare names are also referenced in the `brt_passenger_payment_type` file.

Note: If a bus type allows "default", then passengers who have an invalid payment method will still be able to pay (e.g., with cash), taking the `default_payment_time`. A bus type without "default" can be used to model service that admits only passengers who have the specified payment method(s).

3.2.3.2 brt_bus_type

Each line in the `brt_bus_type` file describes a class of buses (of which many instances may exist in a running simulation).

BRT bus types supplement the heavy vehicle type of Paramics. We must use the latter to provide length, dynamics, and anything else affecting the behavior or appearance in Paramics. However, all other characteristics are under the control of our bus type. There can be more than one of these bus type, allowing several different services (such as BRT and local) with different characteristics.

The `brt_bus_type` file has the following fields on each line:

```
type_name           = name of bus type
low_floor           = 1 if low floor, else 0
capacity            = total cap., standees included (int)
seats               = num. seats (int)
speed_limit         = maximum speed, in m/s (float), 0 to use the
                    limit imposed by the current link
comfort_accel       = comfortable acceleration, in m/s/s (float)
comfort_decel       = comfortable deceleration, in m/s/s (float) (*)
alight_door_count   = number of doors usable for alighting only (**)
board_door_count    = number of doors usable for boarding only
mixed_door_count    = number of doors usable for either purpose
default_pay_time    = added to board time when passenger has none
                    of the types listed in brt_bus_fare_types.
base_dwelling_time  = time added to board/alight times at stop
                    (to open/close doors, etc) (float)
```

(*): `comfort_decel` can be given as positive or negative, and only the absolute value is significant.

(**): Door counts are in terms of the number of passengers which can board/alight at the same time (a wide door might count as 2).

3.2.3.3 brt_config

The brt_config file allows some global configuration settings, much like the configuration file in Paramics. Each line is a particular setting. The brt_config file has the following fields on each line:

```
key           = string
value        = string
```

Each string is just a sequence of characters without spaces. (Quotes are not needed.)

All keys are allowed, but the plug-in ignores all but the following:

debug (or DEBUG)

Boolean, with 0 as false and all else as true.

Turn on debug mode. The only difference in running in debug mode is that before all the brt plug-in does its setup, the user is given the opportunity to press return or ctrl-C, which is necessary to debug the smartbrt plug-in using the gdb debugger.

This is typically used by the developers of the plug-in, not by users.

log_trajectories (or LOG_TRAJECTORIES)

Boolean, with 0 as false and all else as true.

Output vehicle position data at every time step to the 'vehicles.log' file for use with SWEditor animation playback.

log_passengers (or LOG_PASSENGERS)

Boolean, with 0 as false and all else as true.

Output each passenger's origin, destination, wait time, sitting time, and stand time to the 'passenger.log' file when the passenger exits.

Note that this is more detailed than the statistical outputs available using the statistical logging facility (brt_log).

log_network (or LOG_NETWORK)

Boolean, with 0 as false and all else as true.

Output to 'nodes.log' the position of all nodes. This file has a line for each node with the data "node <name> at x, y, z junction". This is used as an input to animation.

3.2.3.4 brt_dispatcher

The BRT dispatcher augments the release policy of BRT terminals. A terminal that has a dispatcher will send all available bus to a fixed insertion point along the route, after a specified travel delay, rather than release it from the terminal.

The dispatchers are defined in the brt_dispatcher file, which has the following fields on each line:

```
zone_id       = id of the Paramics zone of terminal that buses are
               dispatched from
insert_zone_id = id of the Paramics zone where bus is inserted
               (this zone must have an associated BRTTerminal, which
```

```

        provides the bus type, destination, etc.)
delay      = time to get from orig. terminal to insertion point,
            in seconds (int)

```

Each line defines a dispatch policy. The effect of defining a dispatch policy is as follows. When a bus finishes its route and arrives at the terminal referenced by <zone_id>, it is removed from the terminal. After the specified delay, the bus is released at the insertion terminal in zone <insert_zone_id>, and begins running along the bus route.

Currently, there is no adaptive decision-making (which might be based on observed and projected headways, arrival times, waiting passengers, etc.). A terminal can have only one corresponding insertion point, since there is no decision mechanism to choose the best one. This means that terminals always are paired, with buses running between them. The only difference between a normal pair of terminals and a paired dispatcher terminal and insertion terminal is that, in the latter case, one of the two travel directions is modeled as a simple delay and the other direction is modeled as a bus serving stops, whereas in the normal case both directions are modeled as a bus serving stops.

Typically, this limited form of dispatch will be used in the following way. There are two terminals at the ends of the route which release buses normally. They do not use dispatch. There are one or more additional terminals along the route which use dispatch to increase bus traffic over a shorter segment of the route. A dispatch terminal may be placed at the end of the route or not, as preferred. The corresponding insert zone may be located anywhere along the route. A bus will run along the route from the insert zone to the dispatch terminal. Then it will be held for <delay> seconds, moved to the insert zone, and released from the insert zone. Note that a single terminal with a dispatch policy affects travel in only one direction. It may be useful to have several dispatchers releasing buses in the same or in different directions.

3.2.3.5 brt_link

Each Paramics link along the bus route has some additional information associated with it by the brt_link file, which has the following fields on each line:

```

name       = name of a Paramics link in links file (string)
bus_lane   = lane used by bus (1, 2, 3...)
allow_lane_change = 1 if bus can change into traffic lanes, 0 otherwise
allow_overtake   = 1 if bus can overtake other buses, 0 otherwise
speed_limit     = for bus only, in m/s, float, 0 means the link
                  imposes no limit (but the bus itself may).

```

Note that allow_lane_change and allow_overtake do not currently work. This sort of behavior is very difficult, if not impossible, to implement in a Paramics plug-in. A different approach, which we may take in the future, is to use the network design to force this behavior, by using lane restrictions to the links in question. This technique is currently used to keep the bus in the lane near the bus stop.

3.2.3.6 brt_link_follow

The brt_link_follow file defines the route taken by a bus that serves the named route. It has the following fields on each line:

```

name       = name of link defined in brt_link
route_name = name of bus route
dest_zone_id = id of the destination zone
follow_name = name of the next link on the route

```


The follow_name field defines where the bus goes next after the current link.

3.2.3.7 brt_log

Each line of the brt_log file specifies a kind of measurement and the times at which the measurement is reported. Lines are of the form

```
variable statistic delay period window
```

The variable is one of:

```
-----per stop variables
```

```
headway          bus headways, measured as the bus leaves the stop
wait_time        passenger wait time
left_behind       num pass. left behind
alight_count      num pass. alighting
board_count       num pass. boarding
psgr_interarrival_time
                  time between successive passenger arrivals
alight_time       pass. alight time
board_time        pass. board time
dwell_time        bus dwell time
seg_travel_time   bus travel time on the segment from the previous stop or
                  terminal to this stop
bus_load          num pass. on bus before stopping
```

```
-----per route variables
```

```
bus_travel_time   time it takes for a bus to travel route
psgr_sit_pct      percent of time on bus that passenger is seated
psgr_travel_spd   speed of passenger travel from orig to dest (in m/s)
stop_time         time bus spent at bus stops (same as dwell time but
                  aggregated by route rather than stop)
route_priority    time added to green when bus got priority
```

```
-----per signal variables
```

```
signal_priority   time added to green when bus got priority
signal_wait_time  time bus was stopped at the signal
```

Note that per passenger data is collected separately in the passenger.log file and includes, for each passenger, origin, destination, and wait, stand, and sit times. These times are cumulative for passengers transferring between routes. (The per route variables listed above are not.)

The statistic is one of:

```
value
average
maximum
minimum
variance
count
```

Each line enables measurement of the specified statistic for the specified variable. The delay, period, and window settings affect how data is collected, aggregated, and reported, as explained below.

The values of delay, period, and window are each floating-point numbers or blank. If delay is blank, then period and window must be, too. If period is blank, then window must be, too. Units are seconds.

Currently, a particular statistic for a particular variable can be reported in only one way. In other words, it is an error to have two lines whose first two fields are the same.

The delay specifies the time interval at the beginning of the simulation run during which the variable is ignored. If the delay is omitted, or 0, logging begins as soon as the first data point is observed. This allows the user to set a "warm up" time at the beginning of the simulation that is not included in statistics.

The period specifies the time between reports to the log file (each report is represented by one output line), regardless of when new data is available. If the period is omitted, or 0, the log is written whenever there is new data.

The window specifies the time interval over which the statistic is calculated. If the window is omitted, or 0, cumulative values are reported, except in the case of the "value" statistic, in which case only the latest value is reported. The window setting has no effect on when data is reported, only on the way values are calculated. Note that using the window option with the count statistic results in frequency measurements.

If both delay and window are specified, data will not appear in the log until delay+window seconds have elapsed. Windowing requires a complete window period, exclusive of the warm-up delay.

If there are no lines, nothing is logged. As in all brt_* input files, a # denotes a comment through the end of the line. The variable and statistic fields are case-insensitive.

Each log is written to a separate file, whose name is constructed from the variable and statistic names (e.g., "headway_average.log"). The file consists of a header line describing the fields. On each line, there is a time field plus one field for each object which has the selected "variable". For instance, "headway" is reported for each BRTStop, whereas "bus_travel_time" is reported for each terminal. Here is an example of a brt_log file:

```
# variable statistic delay period window
headway maximum 600 100 0
headway average 600 600 0
headway variance 600 600 5400
```

This file requests the three headway statistics, maximum, average, and variance. All three statistics will ignore the first 600 seconds of simulation ("warm up"). The current maximum is reported to the log file every 100 seconds. The current values of the other statistics are reported every 600 seconds. The maximum and average are reported cumulatively--all values after the warm up delay are aggregated into the reported value of each statistic. The variance is aggregated in windows of 5400 seconds. This limit is desirable because computing variance require keeping all aggregated values in memory.

Here is the resulting "headway_average.log" output file:

```
#time outbound_2 inbound_1 inbound_2 outbound_1
1200.0 20.00 11.00 145.50 142.33
1800.0 134.00 137.67 162.25 156.22
2400.0 142.60 157.33 166.20 159.28
3000.0 153.75 163.62 189.92 164.94
3600.0 170.36 172.75 187.41 197.86
4200.0 180.92 179.44 189.92 188.69
4800.0 175.39 177.22 185.77 182.77
```

Column spacing has been manually adjusted for readability. The columns are not presented in any particular order--the user may want to rearrange them as

desired. Also, later lines in the log may have more columns than earlier lines, because more objects have reported data at that time. In this case, an additional header line is generated to describe the new set of columns.

In cases where no data is available to compute a statistic (for instance, if the window is very small), the value of "NIL" will be reported instead. This is so that all numeric entries in the table can be assumed to be meaningful.

3.2.3.8 brt_passenger_OD

The `brt_passenger_OD` file determines, in combination with the arrival rates specified in `brt_stop` and `brt_stop_arrival`, the passenger demand in the system. The file has the following fields on each line:

```
o_name          = name of origin stop
d_name          = name of destination stop
prob            = probability (float)
```

The probability refers to the proportion of passengers departing from the specified origin who intend to travel to the specified destination. Hence, for a given origin, all listed probabilities must sum to 1--they represent a discrete distribution of destination choices at that origin.

3.2.3.9 brt_passenger_mobility_type

The `brt_passenger_mobility_type` file determines, in combination with the bus parameters, the time it takes passengers to alight and board. Any number of mobility types can be defined in this file. Examples might include "bicycle", "disabled", and "elderly". The file has the following fields on each line:

```
type_name       = name of passenger mobility type
prob            = probability (float)
hi_floor_alight_time = time (in sec.) to alight from hi_floor bus, float
lo_floor_alight_time = time (in sec.) to alight from lo_floor bus, float
hi_floor_board_time  = time (in sec.) to board hi_floor bus, float
lo_floor_board_time  = time (in sec.) to board lo_floor bus, float
```

Note: these represent the physical board/alight motion, not including time for payment, in-bus movements, etc.

3.2.3.10 brt_passenger_payment_type

The `brt_passenger_payment_type` file determines, in combination with the bus parameters, the time it takes boarding passengers to pay their fare. Any number of payment types can be defined in this file. Examples might include "fastpass", "smartcard", "token", etc. The file has the following fields on each line:

```
type_name       = name of passenger payment type
prob            = probability (float)
time            = payment time, assuming bus is equipped (float)
```

Note: the probability distribution is independent from the mobility type distribution.

3.2.3.11 brt_route

The `brt_route` file is used to determine which buses stop at which stops. It can be used to have two bus routes (e.g., local and BRT) on the same ROW. The file has the following fields on each line:

```
route_name      = name of bus route
stop_name       = name of stop
```

The `route_name` is used for correlation with entries in the `brt_terminal` and `brt_link_follow` files. The `stop_name` is used for correlation with entries in the `brt_stop` file.

3.2.3.12 `brt_signal`

Each entry in the `brt_signal` file represents a signalized intersection. Only those intersections whose signals have BRT-specific behavior, such as priority for buses, need to be mentioned in this file.

The `brt_signal` file has the following fields on each line:

<code>node_name</code>	= name of a Paramics node in the nodes file
<code>green_hold</code>	= max time, in seconds, to hold green if it will benefit bus on BRT corridor
<code>hold_phase</code>	= the Paramics phase number of the signal which time is given to (integer), or 0 if no phase gets priority
<code>other_phase</code>	= phase time is taken from
<code>queue_jump_lane</code>	= lane index of queue jump lane (on major route)
<code>queue_jump_time</code>	= time, in seconds, of green phase given to the <code>queue_jump_lane</code>
<code>name</code>	= name of the signal

Currently, queue jumping is not implemented; the two parameters will be ignored, but values must be given to complete the input.

The `name` is used to identify the signal for logging and statistics. It can be any string without spaces. This is useful to make the logging output more readable than it would be if the node name was used. It can also be used for complex junctions that involve more than one node, but should be considered as one intersection for statistical purposes.

3.2.3.13 `brt_stop`

The `brt_stop` defines the locations and characteristics of all the bus stops. The file has the following fields on each line:

<code>name</code>	= name of the stop (string)
<code>link_name</code>	= name of the Paramics link in links file (string) on which this stop is located
<code>dist_to_stop</code>	= in meters, floating point, from start of link
<code>offline</code>	= 1 if stop is off line, 0 if on line
<code>max_num_buses_docked</code>	= number of docking spaces
<code>fare_at_stop</code>	= 1 if paid at stop, 0 if paid on bus
<code>frequency</code>	= freq. of passenger arrival, in passengers per hour (Note: this is the initial frequency, which can be superceded after a certain time according to the <code>brt_stop_arrival</code> file.)
<code>seed</code>	= seed for the arrival sequence and for generating passenger characteristics

The `offline` flag currently has no effect--all stops are on line.

3.2.3.14 `brt_stop_arrival`

The `brt_stop_arrival` file is optional and can be used to fine-tune the passenger arrival rate by time of day. Normally, the `brt_stop` file specifies the frequency for each stop, and this rate stays the same throughout the simulation. When the stop is listed in `brt_stop_arrival`, this rate can change at any number of specified times.

The file has the following fields on each line:

```

stop_name           = name of stop, as in brt_stop
change_time        = time, in minutes (absolute, like brt_time), at
                    which new arrival frequency takes effect
frequency          = rate of passenger arrival, in passengers per hour

```

3.2.3.15 brt_terminal

BRT terminals extend the behavior of Paramics zones. Paramics zones serve as sources and sinks for all kinds of vehicles, including buses. A BRT terminal is associated with a zone that is a source and sink for buses. The BRT terminal's characteristics determine what kind of buses are released, how many buses are available, what destination is assigned to them, what route they will follow, and the headway to be kept between them as they leave the terminal. A terminal can release only one type of bus.

Each terminal corresponds to an endpoint of a bus route. Two distinct bus routes, even if they share all of their right of way, must have different terminals (with different zones as well), though the terminals can be placed close together. Terminals are not passenger stops. A stop may be placed near a terminal, however, to have the effect of a bus station.

The terminals are defined in the brt_terminal file, which has the following fields:

```

zone_index          = id of Paramics zone (integer)
dest_index          = id of destination zone (integer)
route_name          = name of the route that a bus will follow if it is
                    released from this terminal
bus_type            = name of bus type
headway             = headway use for bus release (float)
start_count         = initial size of bus pool at this terminal (int)
headway_hold        = whether to hold buses at stops to maintain headway
                    (1 = true, 0 = false)
offset              = offset from END of link, in m.

```

3.2.3.16 brt_trace

Tracing is a way for the user to see the event history of a simulation. Each line of the brt_trace file is an event name, chosen from the first column of the following table:

event name	data shown	condition when shown
bus	bus state	arrival and departure at terminal/stop, and link change
bus_periodic N	bus state	every N seconds (floating point)
passenger	psgr. state	passenger generation, boarding, alighting, arrival
stop	stop state	arrival and departure of buses and passengers, change in arrival rate
signal	signal state	extension granted
terminal	terminal state	bus arrival and departure
link	topology	start up time

```

general          -          initializing and finalizing the plug-in

dispatch        bus, route,  dispatch of buses and
                and terminal arrival of dispatched buses

```

Each event named in the file switches on tracing of that event type. Note that the `bus_periodic` event also requires a floating point number to specify the period. If there are no lines, no events are traced. As in all `brt_*` input files, lines can be commented out with `'#'`.

Events are output in the order in which they happen, along with the time at which they occur.

Currently, tracing output simply goes to the user's terminal ("standard output", or `stdout`), though it can of course be redirected to a file. Later, there will be an option to write to a file named in the inputs, or even a GUI to parse, display, and query the event history.

Currently, tracing is best used in combination with a pager program like `'less'` or `'more'`. From a Unix command shell, you can run your network with

```
processor-cmd -cmd -clean -netpath `pwd` 2>&1 | less
```

and use the built-in searching ("`grep`") commands to jump (forward or backward) to events matching a pattern. This can be used, for instance, to quickly find all bus arrivals at a particular stop, or to jump to a particular time.

3.2.3.17 brt_transfer

The `brt_transfer` file defines which stops can be used to transfer between which bus routes. In the simulation, this determines if a passenger on a bus alights to wait at a stop for another bus, and it determines whether a passenger at a stop boards an arriving bus. The model of route choice is very simple. Passengers simply take the first available stop (if riding) or bus (if waiting) that brings them either to their destination or to a transfer stop that is closer to their destination. They do not consider the speed of the bus, the directness of the route, or the expected wait time at a stop.

The file has the following fields on each line:

```

curr            = name of the stop where the decision is made
dest           = name of a stop which can be reached by transfer
                from curr (possibly after other transfers)
trans         = name of the stop to wait at (should be curr itself
                or within very easy walking distance of curr) for
                transfer

```

There should be at most one entry for any given `curr` and `dest`. The `"trans"` signifies the most suitable stop at which to transfer, for passengers traveling from the `curr` stop to the `dest` stop, though there may be other stops that are less suitable. It is up to the simulation designer to decide in advance what "suitable" means.

The effect of these inputs, also known collectively as the transfer table, is as follows. First, we consider the case of a passenger waiting at a stop `S1` when a bus arrives. We consider in sequence the remaining stops along the route of the bus. If such a stop, `S2` is the same as the passenger's destination, `D`, the passenger queues to board. If `S2` and `D` are listed in an entry of the transfer table, the passenger queues to board.

Second, we consider the case of a passenger riding a bus upon arrival at a stop, `S`. If `S` is the same as `D`, the passenger's destination, then the passenger decides to alight. If `S` is not the same as `D`, but `S` and `D` are

listed in an entry of the transfer table, along with a stop T, the passenger decides to alight. In the latter case, the passenger starts waiting for a bus at stop T. Note that T may be the same as S, or it may be different. It would be different if it is necessary to cross a street, go around a corner, or otherwise travel a short distance to reach the bus stop for the next bus in the passenger's itinerary.

3.2.3.18 brt_veh_to_log

The `brt_veh_to_log` file designates which vehicle, if any, is to be logged in detail (trajectory data suitable for 3D visualization), along with its surrounding vehicles.

The `brt_veh_to_log` file has one line with the following fields:

<code>vehicle_type</code>	the type of vehicle to be selected (integer)
<code>vehicle_number</code>	select the n-th vehicle of the type (integer)
<code>distance</code>	only log vehicles within this distance from selected vehicle, in meters (float)

For example, the following `brt_veh_to_log` file will log all vehicles within 1000 meters of the 10th vehicle of type 1:

#	Type to log	Vehicle number to log	Area to log
1	1	10	1000

The output goes to the file `vehicles.log`. The format is:

<code>time</code>	simulation time, in seconds (float)
<code>vehicle_ptr</code>	integer which uniquely identifies the vehicle
<code>vehicle_type</code>	type of vehicle (integer)
<code>x</code>	x coordinate of vehicle (float)
<code>y</code>	y coordinate of vehicle (float)
<code>z</code>	z coordinate of vehicle (float)
<code>bearing</code>	bearing of vehicle (float)
<code>gradient</code>	gradient of vehicle (float)

3.3 Outputs

Outputs are currently stored in the network directory, along with inputs. (This may change in the future.) All outputs generated by SmartBRT have the `.log` suffix, to distinguish them from inputs. Note that some of the input files have names ending in `_log`.

3.3.1 Statistical outputs

Statistical outputs are described in the section on the `brt_log` input file.

3.3.2 Passenger traces

Each passenger is considered as an individual, with its own identity and characteristics, namely:

- origin and destination (bus stops),
- mobility type, and
- payment type.

In addition, as a passenger travels through the system, a record is kept of three cumulative durations:

- time spent waiting in queues,
- time spent seated on buses, and
- time spent standing on buses.

When the passenger finishes its trip, this data is logged to the `passenger.log` file, assuming that the `log_passengers` flag has been turned on in the `brt_config` input file. The output lists the names of the origin and destination stops (but not the names of any transfer stops the passenger may have visited) and the total time spent waiting at stops (including transfer stops), seated on buses, and standing on buses.

An example of the output is given below:

```
#orig      dest      wait      sit      stand
stop4      stop5      80        112      0
stop1      stop2      140       124      0
stop5      stop6      214       79       0
stop4      stop6      15        188      0
stop4      stop6      111       193      0
stop2      stop3      57        102      0
stop2      stop3      104       101      0
stop2      stop3      162       106      0
stop2      stop3      182       105      0
stop1      stop3      119       228      0
```

3.3.3 Debugging outputs

Debugging outputs are specified in the `brt_trace` input file, which is described in the section in input files. The tracing output is somewhat self-descriptive. Here is an example

of one event from the event history of a simulation.

```
At 707.0 sec: Bus departed from stop:
      id 9
      type B1
      route R1
      link m5:m4
      lane 1
      dist to link end 180.4 m
      dist to stop 393.4 m
      seated 4: 31 36 39 41
      standee 0:
      speed 0.4 m/s
      nearby stop stop5
      dist_to_signal 573.4 m
      next signal m2
      depart time 707.0 s
      mode car following
```

This particular event is a bus event. Bus events are reported if the user has requested either “bus” or “bus_periodic” in the `brt_trace` input file. The data has the following meaning:

Event name	Value
id	The id of the bus.
type	The bus type (as used in <code>brt_terminal</code> and <code>brt_bus_type</code>)
route	The bus route (as used in <code>brt_terminal</code> and <code>brt_route</code>)
link	The name (as used in Paramics) of the link in which the bus is currently located.
lane	The number of the lane in which the bus is currently located.
dist to link end	Distance from bus's current position to the end of the link, in meters.
dist to stop	Distance from bus's current position to the next stop, if any, in meters.
seated	Number of seated passengers (in this case 4), followed by a list of the passenger ids.
standee	Number of standee passengers (in this case 0), followed by a list of the passenger ids.

speed	Bus's current speed in meters/second.
nearby stop	Name of the next stop on the route, if any.
dist to signal	Distance from bus's current position to the next signalized intersection, if any, in meters.
next signal	The name (as used in Paramics) of the node where the next signal is located.
depart time	If the bus is at a stop, this is the estimated time in the future at which the bus will leave the stop, after exchanging passengers. If the bus is not at a stop, this is the time at which the bus most recently left a stop or (if none) the terminal.
mode	One of three modes: approaching bus stop stopped at bus stop car following These modes are described in the section on bus movement.

As noted in the section describing the `brt_trace` input file, using trace output in conjunction with pattern searches is an easy way to perform queries on past events in the simulation, or to wait for future events. For example:

- "the next event for bus 6"
- "the 7th bus generated at zone 2"
- "the next arrival at stop stop5"

It would be desirable to develop a GUI that has the same query capabilities, but without the need for the user to be comfortable with command-line tools and pattern matching using regular expressions.

3.3.4 Visualization outputs

The plug-in can produce vehicle trajectory data suitable (in form, at least) for 3D visualization. This is activated by turning on the `log_trajectories` flag in the `brt_config` file. The selection of which vehicle to log is made in the `brt_veh_to_log` file, described in Chapter 2. The specified vehicle and all vehicles within the specified distance from it are logged (by selecting a reasonable distance, the file size can be kept fairly small). For example, the following `brt_veh_to_log` file will log all vehicles within 1000 meters of the 10th vehicle of type 1:

```
# Type to log      Vehicle number to log      Area to log
1                  10                          1000
```

The output goes to the file `vehicles.log`. The file has the following fields on each line:

```
time              simulation time, in seconds (float)
vehicle_ptr       integer which uniquely identifies the vehicle
vehicle_type      type of vehicle (integer)
x                 x coordinate of vehicle (float)
y                 y coordinate of vehicle (float)
z                 z coordinate of vehicle (float)
bearing           bearing of vehicle (float)
gradient          gradient of vehicle (float)
```

In addition, because of internal coordinate transformations in Paramics, it is useful to output a file of node positions in the same coordinate frame as the vehicle trajectories. This is activated by turning on the `log_network` flag in the `brt_config` file. The format of the resulting output goes to the `nodes.log` file and contains the positions of all nodes. For each node, the file has a line with the data

```
| node <name> at x, y, z junction
```

3.4 Implementation of the SmartBRT plug-in

Although the source code to the plug-in is not part of the current software release, there are some implementation issues that are important to discuss.

3.4.1 Paramics limitations and problems

There are a number of limitations and problems in Paramics that we have had to work around one way or another in developing our plug-in.

3.4.1.1 Lateral movement of vehicles

We found the interface that Paramics provides for controlling vehicle lateral movements to be unreliable, and as a result the plug-in supports bus stops by using some rather artificial network design, described in this section.

Our bus stops are unknown to Paramics, because we are not using the overly limited built-in bus and bus stop functionality. Our buses are, from the point of view of Paramics, nothing more than heavy vehicles. Consequently, our buses will not stop at our bus stops without some special effort. We have not found it possible to reliably use the Paramics API to force the bus to change to the lane adjacent to the stop. We got around this limitation by using lane restrictions on the links which have bus stops to keep the bus in the correct lane to make the stop. Since lane restrictions apply to entire links, a city block with a bus stop is subdivided into at least three links to allow for this restriction near the stop (the third link has to do with non-buses being allowed to make right turns in the bus lane).

This problem is one reason why the plug-in cannot be easily used without using the BRTML compiler to generate networks that have appropriate restrictions. Lane restrictions are automatically generated by the BRTML compiler based on the user's choice of bus lane for each block.

Unfortunately, this work-around introduces a new problem: buses cannot pass each other (or pass cars) while they are in the restricted area around the bus stop. This behavior is not realistic.

3.4.1.2 Longitudinal movement of vehicles

Another consequence of using our own bus stops, unknown to Paramics, is that it is necessary to override the Paramics longitudinal model to force a bus to stop at a stop, wait there while passengers are boarding and alighting, and start up again. However, one problem with longitudinal movement is that a bus stopped at a bus stop is not recognized by Paramics to be stopped for any legitimate reason, and so Paramics keeps trying to make the bus move forward. As a workaround, we must reset the velocity to 0 at every time step while the bus is (according to our model) stopped, and also we must subtract a certain amount from its position along the link to account for the distance that Paramics has already advanced it.

3.4.2 Efficiency and scalability

We have made an effort to use efficient, scalable data structures, so that the user will not have the unpleasant experience of adding one too many of a certain kind of object and experiencing a failure or unpredictable behavior caused by the plug-in. In very few cases does the plug-in have a hard-coded limit to the number of entities of any type. We achieve this by making widespread use of hash tables for associations between objects or attributes of objects. Hash tables not only have variable size, but also have the additional benefit of fast lookups. Accessing entries in a hash table is efficient and scalable in the sense that, as the table grows, the access time does not grow or grows very little.

Rather than use the “LT” hashing library suggested in the Paramics documentation, however, we use a public domain implementation called “st.c” which was written by Peter Moore at UC Berkeley. It is widely used (for instance, as the hash implementation in the Ruby programming language), and has an important advantage over LT: table keys can be strings as well as pointers or integers. This feature lets us use route names, bus type names, etc. as keys for table entries. This simplifies using symbolic references between the plug-in input files.

In the broader implementation picture, there are aspects of the plug-in that have been implemented in an efficient way to preserve simplicity and ensure reliability. It remains to be seen whether this will be a problem for larger networks, though our experience so far suggests not.

3.4.3 Software engineering methodology

Although the plug-in is coded in C, we have followed the standards of object-oriented, modular programming for functions and variables. Data members of structures are generally not accessed directly outside of the “class” in which they are defined, the first argument to a function is usually thought of as the “object” on which the “method” is being called, public functions are distinguishable by name, private functions are static to the file in which they are used, function names are chosen to clearly indicate the “class” to which they belong, and so on. These rules, though not universally followed, make the code easier to understand and maintain. In addition, all documentation for a “class” or input file is contained in the header file in which it is defined or the functions to read it are defined. This documentation is useful to the programmer, but additionally can easily be extracted and inserted in a document as it is in this one.

3.4.4 Topics for future work

Future work on the SmartBRT plug-in might include:

- Passenger route choice model.
- Signal priority only for designated buses.
- Off line stops.(*)
- Queue jump lanes.(*)

- Time-varying passenger OD tables.

The topics marked with a (*) have more to do with network design than with the plug-in itself, and most of the work would be on the BRTML compiler, rather than the plug-in.

4 BRTML Reference

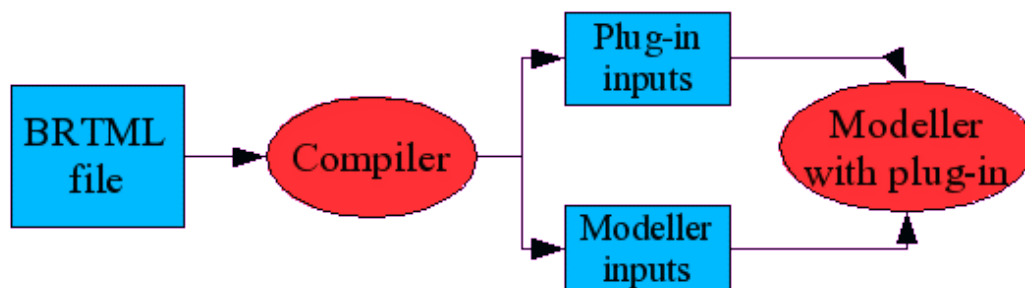
4.1 Introduction

BRTML is a language for modeling transit corridors and for specifying simulation experiments involving them. The BRTML compiler translates a model expressed in this language to a set of input files that can be read by Paramics Modeller in combination with the SmartBRT plug-in. We think of a BRTML model as kind database describing all aspects of the project, from transit system characteristics to the design of the experiment to be carried out in simulation.

4.1.1 The need for BRTML

The Modeller GUI is quite powerful, but it cannot be used to configure the SmartBRT plug-in. SmartBRT represents buses, terminals, passengers, bus routes, bus stops, and so on in a way that Modeller is mostly unaware of. The design of these bus system components can be quite complex. Modeller can be extended only in limited ways to allow the user to input a small set of scalar parameters, but not to design something as complex as a BRT corridor.

Paramics input files are not meant to be edited by hand, nor are the plug-in inputs (see SmartBRT Model Report for details on these files). The only solution to this problem, since Modeller is not open source, is preprocessing: we provide the user with a way of specifying the corridor and compiling this specification into input files for Modeller and the SmartBRT plug-in. This process is diagrammed in the illustration below. This approach has the advantage of unifying all the modeling concepts in a single, seamless system. The user doesn't need to know anything about Paramics Modeller to use this system.



4.1.1.1 Key features of BRTML

Concise:

BRTML is optimized for concise description of corridors. Since a corridor is an essentially linear structure (with curves), geometrical parameters are relative, rather than absolute. The position, shape, and connectivity of roadway segments is specified in terms of adjoining segments. The basic geometry of each successive section of roadway can be described simply in terms of length and curvature. This leads to a disadvantage of BRTML—the difficulty of representing models in absolutely specified coordinates. However, for typical evaluation problems, such a high degree of geometrical accuracy may not be necessary. For a quick what-if study, such data may not even be available.

(Example. In the Wilshire modeling project, the BRTML file size (~450K) is about one fourth that of the resulting set of files that are input to Paramics and the SmartBRT plug-in. The ratio would be more extreme, except for the high level of precision and detail in the BRTML file, in which, in which most elements are modeled individually, due to the level of detail available in the original input data. A quick what-if study might have a ratio closer to 1 to 10 or more.)

Consistent syntax:

Basic principles apply to all parts of a BRTML document. For example:

- There is standard syntax for number formats and units (including speed, distance, time, and date).
- All items can be given names.
- Item names can be used in unambiguous references from other parts of the document (e.g., the bus type that is used along a bus route). There is a standard syntax for references.
- Items can be grouped in meaningful ways (e.g., the city blocks that are served by a bus line). Grouping also makes the document more precise since attributes can be defined once and shared by the group.

Hierarchical:

A key feature of BRTML is inheritance of attributes controlled by the hierarchy of items. In some respects, the description of a transit system falls naturally into a hierarchical pattern.

Geographical units range from large (city or even county jurisdictions) to medium-sized (a downtown region, or segments of a corridor with varying design principles or usage patterns) to small (a city block, or a junction). Characteristics of the larger unit are generally passed on to the smaller units, with local exceptions (for instance, a corridor may have three traffic lanes and one bus lane in both directions, except for the lightly used segments at each end and a region in the middle that is undergoing construction).

Other hierarchies are not geographical, but abstract: terminals belong to a bus route, for example. Routes belong to agencies. Another example: stops do not belong to a single route, but may be grouped by service characteristics: all stops in a certain group have “high” passenger arrival rate and pre-paid fares, but all stops in another group have “low” arrival rate and no pre-paid fares. For quickly defining a corridor, this is preferable to

setting all characteristics individually.

Note that hierarchies in BRTML are not exclusive: a city block can belong to one hierarchy which defines the lane structure (number of lanes in each direction, turn lanes, bus lanes, etc.) and to another hierarchy which defines traffic demand characteristics. In this sense, BRTML supports inheritance of attributes from multiple parents. In other words, an item can belong to multiple groups, each of which supplies to the item attributes of a different kind.

Familiar:

Although the syntax of BRTML is new, the semantics is adapted from the standard file system semantics used on Unix and Windows computers. Also, the notation for references using paths of item names is similar to the path notation used by these operating systems.

Flexible:

Working within a few simple rules, and knowing the meaning of the basic item types and their attributes, the corridor designer is free to introduce new types and arrange items using multiple hierarchies in ways that clearly express the intent of the designer. Types such as “jurisdiction” are not built into BRTML, but are chosen by the user. Outside of the set of basic types supplied by BRTML and documented here, the designer is free to use any types.

Ultimately, what is significant to the simulation is not the exact conceptual structure of the BRTML input, but the association of attributes to items and also the containment relation described by the hierarchy.

For instance, organizing bus types by manufacturer is of value to the designer, especially because some attributes are likely to be shared by and can be defined at once for all types from a certain manufacturer rather than for each bus type individually. The definition of bus types can, in this way, clearly express the designer's intent. Ultimately, however, the simulation depends only on the attributes assigned to each bus type, whether they were assigned by inheritance or individually.

4.1.1.2 A quick look at BRMTL

The following fragment of a BRTML document describes certain aspects of a junction. The rest of the BRTML document describes other junctions, bus types, terminals, demands, and so on.

```
junction 4th and Main
  legs:
    .4th.S [def]
    .Main.b3 [def]
    .4th.N [def]
    .Main.b4 [def]
  phases:
    legs:^legs
    phase east west
```

length:30 sec
red length:5 sec
major
 from:legs[1]
 to:legs[3]
major
 from:legs[3]
 to:legs[1]
medium
 from:legs[1]
 to:legs[0]
medium
 from:legs[3]
 to:legs[2]
minor
 from:legs[2]
 to:legs[1]
minor
 from:legs[0]
 to:legs[3]
phase **protected left from east and west**
length:10 sec
red length:5 sec
major
 from:legs[1]
 to:legs[2]
major
 from:legs[3]
 to:legs[0]
minor
 from:legs[0]
 to:legs[3]
minor
 from:legs[1]
 to:legs[0]
minor
 from:legs[2]
 to:legs[1]
minor
 from:legs[3]
 to:legs[2]
phase **north south**
length:10 sec
red length:5 sec
major
 from:legs[0]
 to:legs[2]
major
 from:legs[2]
 to:legs[0]
medium
 from:legs[0]
 to:legs[3]

```
medium
  from:legs[2]
  to:legs[1]
minor
  from:legs[0]
  to:legs[1]
minor
  from:legs[2]
  to:legs[3]
```

Note that the `[def]` hyperlinks (`.4th.S`, `.4th.N`, `.Main.b3`, `.Main.b4`) are not functional because they refer to parts of the original document outside of the fragment listed here. Also, all of the `legs[i]` hyperlinks have been disabled, since they link outside the fragment. Normally, the HTML generated from BRTML has many hyperlinks.

The meaning of this fragment is partly apparent and partly obscure. It's easy to see that:

- there is a junction at “4th and Main”,
- the junction has four legs—two are part of 4th Street and two are part of Main Street,
- the signal at the junction has three phases, “east west”, “protected left from east and west”, and “north south”,
- the phases each have a given length and red length, in seconds (the meaning of these two concepts is the same as in Paramics),
- associated with each phase are major, medium, and minor motions (terminology as in Paramics) from one given leg to another.

Also, from these observations it is apparent that indentation is significant—more precisely that is is closely related with the “has a” relationship between, for example, a phase and its signal timing attributes, or a junction and its parts.

What's not obvious is how the parts are linked together. What does `.Main.b3` refer to? What does `legs[2]` refer to? What is `legs: ^legs`? These questions about how the BRTML language works are answered in the rest of this introduction. Details about specific constructs, such as `junction` and `phase`, are in subsequent sections.

4.1.1.3 Principles of BRTML

The example in the previous section hints at how BRTML documents are arranged. We list some basic principles here, leaving precise formal definitions for later:

- The key to understanding BRTML is reading the hierarchical structure from the pattern of indentation of the lines.
 - Each line of text defines a coherent unit of the model.
 - Depth of indentation denotes the depth of a unit in the hierarchy.
- There are three kinds of lines that we have seen so far:
 - attribute: a line consisting of a word or short phrase followed

by a colon, followed by more text (shown in green in the example). An attribute associates a name, such as “phase length” with a value, which can be another item, or a value of a particular type, such as a time or a distance.

- item: a line consisting of a type (such as `junction`) followed (optionally) by a name for the item. The item may also have an indented list of attributes and subitems.
- item *link* or *reference*: an expression like “.Main.b4” or “legs [0]” that refers to another item.
- A BRTML document is built up out of these items and attributes.
- White space, especially indentation, delimits the parts of BRTML syntax.
- Attributes that are not specifically recognized for some purpose are not rejected. You can use this to, for example, insert a comment attribute (perhaps explaining why a certain parameter value was chosen). A more advanced use would be to add additional information to the model that is not used by SmartBRT, but might be used by another program (such as cost calculation). It helps to think of a BRTML model as a database.
- Items have names. Typically, the name is not used to convey information about the item (it is considered poor form to encode characteristics in the name—use attributes instead), but it is used as a reference target for other parts of the document. For instance, there is no inherent need to assign a name to a street. However, doing so allows one to refer to the street in, for instance, the design of a bus route.
- Items are listed in order, and the order is sometimes significant (the “legs” of a junction are listed in clockwise order, and this ordering contributes to the definition of the topology of the corridor).
- Attributes are unique within an item—for example, an attribute like “color” would signify *the* color of the item it is attached to.
- Subitems need not be unique within their parent—for example, a “phase” type of item might have a list of several “major” subitems (and also “minor” and “medium” subitems). If two subitems need to be distinguished so that they can be referred to from elsewhere, they can be given distinct names, or they can be referred to numerically, using the subitem index number.
- In most cases, the depth of a subitem is not important, so grouping of subitems does not affect meaning, except that it can be used to define attributes for the group. The mechanism of grouping and inheritance is really just a way to conveniently organize attributes that are shared among many items.

4.1.2 Using BRTML

Running the BRTML compiler and using the compiler's output to run Paramics simulations is discussed in the section on using BRTML.

4.1.3 BRTML and Paramics: Two levels of detail

Paramics describes networks in terms of nodes and links. The designer must manually break up the roadway into links in order to model varying link parameters, such as number of lanes, turn lanes, and lane restrictions. BRTML make some of these decisions for the user, letting the user concentrate on transit issues. The designer works at the level of “blocks”, which the BRTML compiler automatically subdivides into three links in each of the two directions. If the user specifies the number of turn lanes in one direction, for example, then the appropriate link is defined to have the required number of lanes.

4.1.4 Numeric Data Types and Units

BRTML is fairly flexible in allowing the user to provide explicit units along with numeric values. The glossary lists formats for each category:

- Boolean
- Distance
- Speed
- Acceleration
- Time
- Day
- Angle
- Lane number

4.1.5 FAQ: Frequently Asked Questions about BRTML

Q: Can BRTML be used with other simulation engines, besides Paramics?

A: The BRTML *language* is not dependent on Paramics, though some of its terms and parameters are obviously influenced by Paramics (e.g., “major”, “medium”, and “minor” signal phases, “annotations”, “drive side”). The BRTML *compiler* currently targets Paramics. A translator from BRTML to another simulation engine could be written if the simulation engine supports the same set of features that are provided by Paramics and the SmartBRT plug-in.

Q: Is BRTML necessary to use the SmartBRT plug-in?

A: Strictly speaking, no. BRTML was developed to manage the complexity in the SmartBRT inputs, both the Paramics inputs and the plug-in specific inputs. It is possible, though difficult, to develop a model by directly constructing these input files. The Paramics inputs could be constructed using the Modeller interface. The SmartBRT plug-in inputs would have to be constructed manually, with careful reference to the Paramics input files (since the plug-in files need to refer to link names, node names, and other names defined in the Paramics input files).

Q: Does BTRML have a GUI (graphical user interface)?

A: BRTML documents are accessed as text files using a text editor and can be viewed in a web browser. However, their syntax is simple and expressive, and users can use their favorite GUI text editor to edit them, rather than be forced into an unfamiliar GUI.

There is a GUI program for running the BRTML compiler and starting simulations based on its output. See the user's manual for details.

Note that, regardless of user interface, understanding the conceptual structure of BRTML documents remains essential, and so the GUI would only provide an alternate visual representation of this conceptual structure. Also, the text representation of models is still essential with a GUI, since even with a GUI one must save documents to files, and structured text files (such as BRTML) are better than binary files since they can be edited by humans and also processed by programs.

It is possible to configure a text editor to recognize BRTML syntax and to color items and attributes to make them more visually recognizable, as one would do with program source code in a programming editor. We have done this with the NEdit editor, for example. Also, the BRTML compiler can output HTML which incorporates syntax coloring and hyperlinks from an item to a referenced item and back to all referrers. For example, see the Wilshire BRTML file (warning: this is a large file). This HTML file was created from the Wilshire BRTML file using the included utility software. See the user's manual for details.

Q: Is BRTML related to XML?

A: No. BRTML is based on a generic hierarchical modeling language, SuperML. (We document BRTML and SuperML together, since their integration is seamless from the user's point of view.) XML serves better as a text markup language than as a modeling language, since the fundamental XML data type is the character string type, and so an additional layer of processing is required anyway to treat XML documents as transit system models. Powerful modeling features, such as complex attributes, grouping, references, and multiple inheritance, are built into BRTML. BRTML does share some general ideas with XML: hierarchical structure of items and sub-items and the association of attributes with items. (However, attributes are not just character strings, as in XML, but can be other items. This ability is important in designing complex items like traffic signals that contain and refer to other items.)

Q: Is BRTML a programming language?

A: No. It is a *model specification language*, or a *modeling language*, for short. A BRTML file is a static representation of the various components in a simulation and the parameters for running an experiment involving those components. BRTML does not have variables, loops, function calls, etc.

Q: How do I use format X with BRTML?

A: This topic is discussed in the section on data sources.

Q: Can BRTML documents be extended to include X data?

A: Yes. As long as you use your own item types, or use new attribute names with existing item types, you may include other data in your BRTML documents as you wish. Doing so does not require any changes to the BRTML software—you just use the items and attributes in your document. The data represented in those items and attributes are available through the BRTML API. This feature could be used to, for instance, perform further post-processing of experimental data (such as cost or emission calculations) or to specify plotting of outputs. We cannot guarantee, however, that any particular item types or attribute names are reserved and will not be used by BRTML itself.

Note however that BRTML does not have very good support for tables. Currently, tables (for example, OD tables) are handled using a separate attribute for each row of the table.

4.2 Global Configuration

BRTML offers a number of settings that apply to the entire simulation, affect the meaning of other parts of the document, or determine how the simulation outputs are produced.

These include:

- Seeds for the Paramics PRNG.
- Time frame and step.
- Debugging settings.
- Logging settings.
- Units.

These settings are contained in items of the types described below.

4.2.1 simulation

There can be any number of items of this type. The item name is not significant. There may be more than one items of this type; in this case, if two items each specify different values for a particular attribute, the value specified in the last item is used. There are no subitems of particular significance to the BRTML compiler. An item of this type can occur anywhere in the structure of a document, though it is typical to put it at the top level (i.e., indented flush left), and to use only one **simulation** item.

Attribute	Value	Default	Meaning
<i>name</i> :	character string	SmartBRT Prototype Network	Used for display.

Attribute	Value	Default	Meaning
<i>Short name:</i>	character string, must be legal file name on your platform	SmartBRT	Used for display and file names.
<i>annotate stops:</i>	Boolean	false	In Modeller visualization, display text strings with the names of a stop, terminal, or junction at the location of the entity. See also, e.g., the color attributes of a bus stop item and blocks.
<i>annotate terminals:</i>	Boolean	false	
<i>annotate junctions:</i>	Boolean	false	
<i>Start time:</i>	Time	0:00	Time of day at which simulation starts.
<i>Start day:</i>	Day	Monday	Day of week on which simulation starts.
<i>simulation time:</i>	Time	1:00	Duration of simulation
<i>timestep:</i>	float	1.0	Granularity of time.
<i>timestep detail:</i>	character string	⁻¹	See Paramics manual.
<i>seed:</i>	integer	⁻²	Seed for Paramics PRNG.
<i>Units:</i>	character string	metric	Sets the <i>default</i> unit system for all values. Currently, only metric is supported, but individual values can be specified in feet if explicit units are given.
<i>Drive side:</i>	character string: left or right	⁻³	Currently, only right hand drive is supported by SmartBRT.

¹ Same as Paramics default.

² Same as Paramics default.

³ Same as Paramics default.

Attribute	Value	Default	Meaning	
<i>Debug:</i>	Boolean	false	Configure SmartBRT plug-in. See the model report for details. See also <code>veh_to_log</code> .	Debug mode.
<i>log network:</i>	Boolean	false		For animation.
<i>log trajectories:</i>	Boolean	false		For animation.
<i>log passengers:</i>	Boolean	false		Detailed passenger logging.

4.2.2 `veh_to_log`

This item selects which vehicle will be logged when the `log trajectories` attribute of the **simulation** item is true. BRTML allows any number of vehicles to be selected for logging, but the current version of the plug-in only logs the first selected vehicle. Details of the feature, and explanation of output format, are in the model report.

There can be any number of items of this type. The item name is not significant. There are no significant subitems. An item of this type can occur anywhere in the structure of a document, though it is typical to put it at the top level or in a **simulation** item.

Attribute	Value	Default	Meaning
<i>Type:</i>	bus or car	bus	Type of vehicle to be selected for logging.
<i>number:</i>	positive integer	1	Ordinal of vehicle (in order of creation of vehicles of that type)
<i>distance:</i>	Distance	1000 meters	Log all vehicles within this distance from selected vehicle.

4.2.3 `trace`

This item is used to turn on tracing output, which is useful for debugging a simulation, or simply for observing the event history of the simulation. Each attribute of a trace item turns on tracing of a specific kind of event. Tracing is discussed in more detail in the model report, under debugging outputs and `brt_trace`. See also the discussion in running `smarbrt`.

There can be any number of items of this type. The item name is not significant. There are no significant subitems. An item of this type can occur anywhere in the structure of a document, though it is typical to put it at the top level.

Attribute	Value	Default	Meaning
<i>bus:</i>	Boolean	false	Trace bus events.
<i>bus periodic:</i>	Time or Boolean	false	Trace bus at fixed time interval.
<i>terminal:</i>	Boolean	false	Trace terminal events.
<i>passenger:</i>	Boolean	false	Trace passenger events.
<i>Stop:</i>	Boolean	false	Trace bus stop events.
<i>Link:</i>	Boolean	false	Trace link events.
<i>signal:</i>	Boolean	false	Trace traffic signal events.
<i>dispatch:</i>	Boolean	false	Trace bus dispatcher events.

4.2.4 log

The **log** item specifies statistical logging. It has no significant attributes of its own (though of course it may provide attributes to be inherited by subitems). The subitems of significance are described in the table below.

There can be any number of items of this type. The item name is not significant. An item of this type can occur anywhere in the structure of a document, though it is typical to put it at the top level.

Subitem type	Significance of subitem name	Significance of subitem order
variable	Signifies variable to measure. Name must be one of following, also described in the brt_log section of the model report: headway wait time left behind alight count board count psgr interarrival time alight time board time dwell time seg travel time bus load bus travel time psgr sit pct psgr travel spd stop time route priority signal priority signal wait time	none

4.2.5 variable

The **variable** items each specify a variable to observe over time, possibly for many different objects (e.g., buses) in the simulation. It has no significant attributes of its own (though of course it may provide attributes to be inherited by subitems). The subitems of significance are described in the table below. A **variable** item must be an indirect child of a **log** item.

Subitem type	Significance of subitem name	Significance of subitem order
statistic	Signifies statistic to collect. Name must be one of the following, also described in the <code>brt_log</code> section of the model report: value average maximum minimum variance count	none

4.2.6 statistic

The **statistic** items each specify a statistic to apply to a variable. The **statistic** items have no significant subitems, but do have significant attributes, listed in the table below. A **statistic** item must be an indirect child of a **variable** item. Details of the feature, and explanation of output format, are in the model report.

Attribute	Value	Default	Meaning
<code>delay:</code>	Time or none	none	Time delay before collecting data for the statistic, none means no delay.
<code>period:</code>	Time or when available	when available	Time between reports to the log file.
<code>window:</code>	Time or cumulative	cumulative	Time interval over which the statistic is calculated.

[Back to BRTML Reference](#)

4.3 Geometric model used in BRTML

Before trying to understand the types of items that are used to specify roadway geometry (street, block, junction) and the types of items that refer to these items (**bus_line**, terminal, demand, **passenger_demand**, bus_stop), it's important to understand their conceptual basis.

4.3.1 Relative coordinates

Roadways are defined in relative coordinates, rather than absolute ones. This means that each segment of the roadway (that is, each **block**) is given a length and a curvature by the user, but most blocks are not directly assigned (x,y,z) coordinates or heading angle. The user picks one block to “pin down” in the absolute coordinate system, and chooses the (x,y,z) and heading of the beginning of the block. The adjoining blocks get their coordinates based on these values and the length and curvature of this block, and so on throughout the network.

4.3.2 Junction topology

Currently, all junction angles are right angles.

The order of legs in junctions is key to defining the way two roadways intersect. Each of the two roadways has two legs that touch the junction. This information is not in itself enough to assign coordinates to all the blocks. Either of the roadways could be rotated 180 degrees and still have a legitimate junction. That's where the order given for the legs is used. In the definition of the **junction**, the legs are listed in *clockwise* order. This ensures that there is a unique interpretation of the junction.

4.3.3 Orientation: outbound and inbound

It's necessary to talk in some way about the direction of travel along a corridor. For example, which direction of travel does a particular bus stop serve? Compass directions are not useful for describing how a bus or stop is oriented on the corridor, because a street may change directions. Local street numbers are not useful because the transit corridor may change streets. Instead we use the fact that a corridor has an essentially linear shape, and we use two fairly arbitrary terms for orientation: inbound and outbound. The outbound orientation is defined by the order of blocks as they are defined: reading down the list is the same as moving outbound. The inbound orientation is the reverse.

4.3.4 Subdivision

A single **block** translates to several links in Paramics. The purpose of this translation is to simplify the model from the point of view of the transit system designer. A block defines behavior in both directions of travel (inbound and outbound), such as number of lanes, turn lanes, and lane restrictions (a Paramics link is for just one direction of travel). Also, a block has within itself three subdivisions. The middle subdivision has the lane usage characteristics defined by the user. However, the subdivision containing the end of the block must be different to allow cars to make right turns using the bus lane. The and subdivision containing the start of the block (if there is a bus stop there) must have lane restrictions to force the bus to be in the curb lane (we have been unable to impose a lateral bus control model on Paramics).

These subdivisions are carried out behind the scenes, and usually not of interest to the user of SmartBRT. However, it is good to be aware that select lane restrictions may apply

only to the middle part of the block for which they are selected.

4.3.5 Corridors and networks

Although BRTML has been designed as a *corridor* language, it can be used for some more complex networks, such as crossing corridors. More precisely, the current version of BRTML can represent loop-free networks—networks in which any two points are connected by only one route.

4.3.6 Lane numbering

Lane numbering in BRTML is consistent with Paramics: lane numbers are counted starting with the curb lane as lane 1. This is confusing if you are used to lane 1 being the lane nearest the median. For this reason, BRTML has an alternative: you may use *negative* lane numbers to count from the median, starting with -1.

4.4 Roadway types

4.4.1 street

Streets are used to group blocks and to assign an ordering to them. The order of blocks that are found among the indirect children of the street determines the order in which the blocks are placed along the roadway. The street item itself has no significant attributes, but of course it may be assigned attributes to be inherited by all of its blocks. This is useful if, for instance, all blocks have the same number of lanes, or the same length. If some of the characteristics vary along the roadway, grouping the blocks by section can reduce redundancy.

Attribute	Value	Default	Meaning
-	-	-	-

4.4.2 block

Blocks are the basic unit of roadway construction, as far as BRTML is concerned. (A block typically translates to a sequence of several links in two directions in Paramics, however—see the Geometry section.) A block can represent an ordinary city block, or a more general roadway section with constant curvature and through lane count. Note that a block represents both directions of traffic.

Attribute	Value	Default	Meaning
<i>length:</i>	Distance	-	Length of the block measured along center line.

Attribute	Value	Default	Meaning
<i>lanes:</i>	Integer	0	Number of lanes in each direction. If these two numbers differ, use the (in/out)bound thru lanes attributes.
<i>Bus lane:</i>	Lane number		Positive means the number is counted from curb side, starting with 1, as is standard in Paramics. Negative means the absolute value of the number is counted from the median side.
<i>allow lane change:</i>	Boolean	false	Can the bus change into traffic lanes?
<i>allow overtake:</i>	Boolean	false	Can the bus overtake other buses?
<i>speed limit:</i>	Speed	Default is the limit imposed by the link itself (on all traffic)	Speed limit for buses.
<i>outbound color:</i> <i>outbound colour:</i>	Color name (as defined by Paramics)	green	Color used for outbound annotations, such as bus stop names. See also simulation .
<i>inbound color:</i> <i>inbound colour:</i>	Color name (as defined by Paramics)	yellow	Color used for inbound annotations, such as bus stop names. See also simulation .
<i>flow:</i>	Floating point number, in vehicles/hour	0 vehicles/hour	Flow of traffic entering the network on this block, if the block is at the edge of the network.

Attribute	Value	Default	Meaning
<i>X:, y:, z:</i>	Floating point number, in arbitrary units	-	Exactly one block is assigned absolute coordinates using these attributes. The remaining blocks get their coordinates by their relation to this one block.
<i>heading:</i>	Angle	-	

4.4.3 junction

Junctions are very complex items. They define the traffic signal logic (or stop-sign control), the arrangement of entering roadways, and turning patterns of the ambient traffic in the simulation.

Attribute	Value	Default	Meaning
<i>Legs:</i>	Orphan item	-	List of the blocks that touch this junction. The order must be clockwise, starting with any one leg. If one leg is absent (in a T-junction), use a “-” as the item instead.
<i>green hold:</i>	Time	0 seconds	Maximum time which the green phase will be extended by to let a bus through (signal priority).
<i>demands:</i>	Orphan item	-	List of demand items, defining the turning ratios for traffic flowing through the junction.
<i>Stop control:</i>	Boolean	false	Is this junction controlled by an all-way stop sign? (A true value is incompatible with the phases attribute.)
<i>phase offset:</i>	Floating point number	0	Timing offset of entire phase structure from start of simulation.
<i>phases:</i>	Orphan item	-	List of phase items.

Junctions are automatically annotated by the SmartBRT compiler for the Paramics

visualization, assuming that annotations have been turned on in the global settings.

4.4.4 phase

A phase item describes an individual phase of the traffic signal at a **junction**.

Attribute	Value	Default	Meaning
<i>length:</i>	Time	30 seconds	The length of the phase.
<i>max length:</i>	Time	length*2 seconds	The maximum length of the phase, including green extension.
<i>red length:</i>	Time	5 seconds	The length of the red phase.

The subitems of the phase item define the allowed movements during the phase. These subitems are of type **major**, **medium**, and **minor**, and are discussed below.

4.4.5 major

A major movement is for traffic that has primary right of way during a phase.

Attribute	Value	Default	Meaning
<i>From:</i>	block reference	–	The leg of the junction from which traffic approaches on this movement.
<i>to:</i>	block reference	–	The leg of the junction to which traffic departs on this movement.

4.4.6 medium

A medium movement is for traffic that has right of way, yielding to major movements, during a phase.

Attribute	Value	Default	Meaning
<i>From:</i>	block reference	–	The leg of the junction from which traffic approaches on this movement.
<i>to:</i>	block reference	–	The leg of the junction to which traffic departs on this movement.

4.4.7 minor

A minor movement is for traffic that has right of way, yielding to major and medium movements, during a phase.

Attribute	Value	Default	Meaning
<i>from:</i>	block reference	–	The leg of the junction from which traffic approaches on this movement.
<i>to:</i>	block reference	–	The leg of the junction to which traffic departs on this movement.

4.4.8 demand

Demand items define the the turning ratios for traffic flowing through the junction. The values can be given on any scale (percent, for example). Regardless of the scale, only the ratios are significant. In other words. There is one demand item for each incoming traffic stream, and the demand item defines what fraction of that stream turns left or right, or continues through the junction.

Attribute	Value	Default	Meaning
<i>from:</i>	block reference	–	The leg of the junction from which traffic approaches.
<i>left:</i>	Floating point number	0	Ratio of traffic that turns left.
<i>right:</i>		0	Ratio of traffic that turns right.
<i>thru:</i>		1	Ratio of traffic that continues through.

4.5 Transit system types

4.5.1 bus_stop

Items of the `bus_stop` type represent stops along bus routes. Passengers arrive at bus stops, wait, and select an arriving bus to board, depending on whether it serves their destination. Bus stops may serve one or more bus routes—the mechanism for specifying the routes served is described below. The recognized attributes of bus stops are:

Attribute	Value	Default	Meaning
<i>orientation:</i>	Orientation	outbound	The direction of travel along the corridor served by the stop. (This setting also affects coloring used in annotations.)
<i>block:</i>	block reference	-	The block at which the stop is located.
<i>offset:</i>	Distance	50 meters	A positive value means the offset of the stop from the near end of block. A negative value means the offset is measured from end of block.
<i>lines:</i>	Orphan item	-	List of bus lines which serve this stop. (One of two alternate ways to associate bus lines with bus stops—the alternative is to list the stops under the “stops” attribute of the bus line item.)
<i>offline:</i>	Boolean	false	Not yet supported in SmartBRT plug-in.
<i>max docked:</i>	Positive number	1	The maximum number of buses that can dock at the stop and load/unload passengers at the stop.
<i>prepaid:</i>	Boolean	false	Do passengers pay at the stop, rather than on the bus?
<i>frequency:</i>	Floating point number	0	Frequency of passenger arrival, in passengers per hour. Note that this frequency describes the initial state of the bus stop. The frequency may vary by time of day.

Attribute	Value	Default	Meaning
<i>Seed:</i>	Integer	0	Seed for PRNG that controls passenger arrival at this stop.

Bus stops are often grouped under a parent item that contains all bus stops for a particular bus route. This parent item might, by convention, be given the type `bus_stops`, but that type is not internally defined and the designer may use any type that is not built in.

Note on annotation. Paramics allows some simple annotation of the visual display with text strings located at certain coordinates. BRTML makes this easier to use by, optionally, using user-defined names of items such as bus stops and automatically calculating coordinates at which to place the annotation. Turning annotation on or off is easy -- see the globals section. For bus stops, the annotation text colors are either given default values or chosen based on attributes specified for the block on which the stop is located. The default for inbound is yellow. The default for outbound is green.

4.5.2 terminal

Each bus route must have at least two terminals, plus possibly additional terminals used for bus dispatch. A terminal is *not* also a stop—the designer should explicitly place a stop near the terminal if that behavior is desired. The terminals should be subitems of the `bus_line` item, and therefore inherit from this `bus_line` item. Typically, the bus type attribute is inherited in this way rather than specified for each terminal independently. Bus dispatch is described in more detail in the model report.

Attribute	Value	Default	Meaning
<i>route name:</i>	Text string	-	Name of the route (usually inherited from the <code>bus_line</code> item).
<i>block:</i>	block reference	-	The block at which the terminal is located.
<i>offset:</i>	Distance	50 meters	A positive value means the offset of the stop from the near end of block. A negative value means the offset is measured from end of block.

Attribute	Value	Default	Meaning
<i>bus type:</i>	Orphan item	-	The orphan item must have a unique item, which is the name of the bus_type item that is emitted from this terminal.
<i>number of buses:</i>	Integer	0	Initial number of buses available to leave from this terminal.
<i>headway:</i>	Floating point number	5 minutes	Minimum time interval between release of buses.
<i>dispatch:</i>	Orphan item	-	If provided, the dispatch item must have <i>to</i> and <i>delay</i> attributes—see below.
<i>(in dispatch orphan item)</i>	<i>to:</i>	terminal reference	The insertion terminal to which buses are sent from this terminal to begin travel on the corridor.
	<i>delay:</i>	Time	The delay incurred by off-corridor travel to the insertion terminal.

Terminals are annotated for visualization by the BRTML compiler in much the same way as stops (see above).

4.5.3 bus_line

A bus line comprises a physical route along streets, a list of bus stops, and some terminals.

Attribute	Value	Default	Meaning
<i>route name:</i>	Text string	-	Name of the route, used for informing user and for annotations in the visualization.

Attribute	Value	Default	Meaning	
<i>route:</i>	Orphan item	-	An item which must have two attributes listing the blocks along the inbound and outbound portions of the route. See below.	
<i>(in route orphan item)</i>	<i>inbound:</i>	Orphan item	-	List of block items, in order, along which the bus travels when following the inbound direction on this line.
	<i>outbound:</i>	Orphan item	-	List of block items, in order, along which the bus travels when following the outbound direction on this line.
<i>stops:</i>	Orphan item	-	List of bus_stop items that are served by this route. (This is an alternative to using the <i>lines:</i> attribute of the stops.)	

4.5.4 bus_type

A bus type item (that is an item whose type is “bus_type”) defines a class of buses that can be selected to run along a bus route. Bus type characteristics are discussed in more detail in the model report.

Attribute	Value	Default	Meaning
<i>low floor:</i>	Boolean	false	Is the bus designed to have a low floor or to be able to kneel to improve boarding?
<i>capacity:</i>	Number	0	Maximum occupancy of bus.
<i>seats:</i>	Number	0	Maximum seated occupancy of bus.
<i>comfort accel:</i>	Acceleration	2 m/s/s	The maximum rate of acceleration that is comfortable on this bus.

Attribute	Value	Default	Meaning
<i>comfort decel:</i>	Acceleration	2 m/s/s	The maximum rate of deceleration that is comfortable on this bus.
<i>alight doors:</i>	Number	0	Number of doors available for alighting.
<i>board doors:</i>	Number	0	Number of doors available for boarding.
<i>mixed doors:</i>	Number	1	Number of doors available for alighting or boarding.
<i>default pay time:</i> (OR) <i>default payment time:</i>	Time	2 seconds	Time to pay if passenger doesn't have specialized payment methods and must use default method (typically cash).
<i>Base dwell time:</i>	Time	3 seconds	Estimate of typical time at stop, not including boarding and alighting. (May include opening and closing doors, for instance.)
<i>speed limit:</i>	Speed	65 mph	Maximum speed of bus. (Speed is also constrained by the roadway and traffic.)
<i>length:</i>	Distance	-	Physical dimensions of the bus. <i>(Not currently used. All buses have the same dimensions in Paramics.)</i>
<i>height:</i>			
<i>width:</i>			
<i>Fare options:</i>	Orphan item	-	List of fare_type items that are accepted for payment on this bus.

4.5.5 fare_type

A fare type represents a type of payment that certain passengers can use on certain buses. A bus may accept more than one fare type. The corridor designer is free to define any fare types, such as smart cards, tokens, electronic payment, etc. Fare type affects only board time in the simulation. A passenger may have only one fare type (although passengers are assumed to be able to make a default payment on any bus if their fare type is not on the

“fare options” list of the bus type. The default payment time depends on the bus.)

Attribute	Value	Default	Meaning
<i>pay time:</i>	Time	0 seconds	Time to make this kind of payment, if the bus is capable of accepting it.
<i>Prob:</i>	Floating point number	0	Probability that any given passenger can make this kind of payment.

4.5.6 mobility_type

A mobility type represents a set of mobility limitations that certain passengers may have, such as being in a wheelchair or carrying a bicycle. Each mobility type can have its own effects on boarding and alighting, and this effect may differ depending on whether the bus is a low floor bus or a kneeling bus. A passenger may have only one mobility type.

Attribute	Value	Default	Meaning
<i>hi floor alight:</i>	Time	0 seconds	Time to alight from a high floor bus.
<i>hi floor board:</i>	Time	0 seconds	Time to board a high floor bus.
<i>lo floor alight:</i>	Time	0 seconds	Time to alight from a low floor bus.
<i>lo floor board:</i>	Time	0 seconds	Time to board a low floor bus.
<i>Prob:</i>	Floating point number	0	Probability that any given passenger has this kind of mobility.

4.6 Demand types

4.6.1 Traffic demand

Traffic demand is discussed in the roadway types section, under **block** (the flow attribute) and **demand**.

4.6.2 Passenger demand

Passenger demand is more complex in BRTML than traffic demand, because it is expected to be more central to the planning and analysis of BRT systems. As described under **bus_stop**, each bus stop may be assigned an initial arrival frequency. This number is independent of the OD (origin-destination) ratios, and also the arrival rate may change during the course of the simulation.

4.6.2.1 Time-varying passenger arrival rates

Passenger arrival may be defined to change at specified times during the day using the following BRTML items.

4.6.2.2 passenger_arrivals

This item contains items of type **arrival**, discussed below. It has no significant attributes. (There may be more than one item of this type, but that has the same effect as combining all their subitems under a single item of this type.)

4.6.2.3 arrival

An arrival item specifies the time at which a stop's passenger arrival rate changes to a new value.

Attribute	Value	Default	Meaning
<i>stop:</i>	bus_stop reference	-	The bus stop to which this setting applies.
<i>day:</i>	Day	Same as the simulation start day.	The day on which the change occurs.
<i>time:</i>	Time	Same as the simulation start time.	The time of day at which the change occurs.
<i>frequency:</i>	Floating point number	0	Frequency of passenger arrival, in passengers per hour.

4.6.3 Passenger OD tables

4.6.3.1 passenger_demand

This item defines the origin-destination ratios of generated passengers. (There may be more than one item of this type, but that has the same effect as combining all their subitems under a single item of this type.) This item is essentially a large matrix whose rows and columns are indexed by bus stops. Unfortunately, this kind of structure is not yet well represented in BRTML. It's easiest to explain this type of item using an example:

```
passenger_demand

out1: .L1 stops.outbound.origin
out2: .L1 stops.outbound.3rd
out3: .L1 stops.outbound.6th
out4: .L1 stops.outbound.destination

in1: .L1 stops.inbound.origin
in2: .L1 stops.inbound.6th
in3: .L1 stops.inbound.3rd
in4: .L1 stops.inbound.destination

to:      out1  out2  out3  out4  in1  in2  in3  in4
from out1:  0   90   5   5   0   0   0   0
from out2:  0   0   50  50   0   0   0   0
from out3:  0   0   0  100   0   0   0   0
from out4:  0   0   0   0   0   0   0   0
from in1:   0   0   0   0   0   0   0  100
from in2:   0   0   0   0   0   0  50  50
from in3:   0   0   0   0   0   0   0  100
from in4:   0   0   0   0   0   0   0   0
```

The out1, out2... and in1, in2,... attributes are simply short local names for the stops. In this form the table is easy to read. For example, 90% of passengers from out1 (that is, the origin stop on the outbound direction of route L1) desire to travel to out2. The numbers in a row do not need to add up to 100, in which case they are prorated accordingly to preserve their ratios.

Note that a passenger's destination stop does not have to be on the same bus route as the origin stop. The details of transfers are explained in the model report.

4.6.4 Passenger characteristics

Two characteristics of passengers affect calculations of board and alight times: fare type and mobility type. These two characteristics are generated from discrete distributions that are independent of the arrival distributions. The distributions are defined by the “prob” attributes in the **fare_type** and **mobility_type** items.

4.7 Using other data sources

BRTML may be used as the design language for a transit system. This approach works well when the parameters of the system are understood in imprecise terms (for instance,

no absolute node coordinates), or the amount of precisely given data is small enough for manual entry into a BRTML document. In many cases, however, large amounts of data may already be available in external files which have a format that is beyond the user's control (e.g., output of a data collection activity, or of a simulation). Using these data sources in a BRTML specification can be difficult, requiring either tedious and error-prone data entry or intelligent but difficult programming.

In the SmartBRT project, we developed translators for several external formats. Note that each of these translators produces only a fragment of a BRTML document. It is also necessary to write code to integrate these fragments into a consistent whole.

Our program starts with a base BRTML model, specifying global simulation parameters, bus types, passenger types, fare types, and so on. These are not derived from external data files but chosen by the user, perhaps based on some knowledge of the corridor and bus system. Then several programs read from external files, read the BRTML model, and contribute some additional information to it, such as demand tables or signal designs. The data sources are:

- TRAFFIX simulation outputs, with traffic flow data and some signal design data.
- Manually coded CAD drawings, with geometrical data, stop locations, street names, etc. We developed a specialized notation for manually coding this data. The manual coder can quickly type in this shorthand, and the shorthand is precise enough to be read by software.
- Spreadsheets of passenger demand data. Our software performs OD estimation.

One of the difficulties of this process is that there are varying levels of precision and completeness in the inputs, which must be smoothed over. Another is that sometimes different data sources are not consistent with each other, and they have to be reconciled. It is difficult to do these intelligent manipulations by software in a way that is general enough to apply to more than one corridor.

These programs are all defined in terms of the BRTML API. They are not included in the current release of the BRTML software since they are, to a large extent, specific to the Wilshire corridor. Some of the data formats involved are not standard and will not be of use to other users.

[Back to BRTML Reference](#)

4.8 Reference Syntax

BRTML has a consistent syntax for referring from one part of a document to another. References are necessary for many reasons: for example, if an item is defined within one group, a reference is needed to make it part of another group. Also, attribute values are sometimes allowed to be references to items. Detailed and precise discussion of that syntax is out of the scope of the current document.

4.9 Glossary

This document is the glossary of terms used in the BRTML tutorial and reference.

Model

A mathematical representation of some physical entity or process, described in terms of equations, logical rules, parameters, etc. In BRTML, a transit system model is expressed as a document in the BRTML language.

Document

In the BRTML language, a document is a text file that represents a model. The distinction between the document and the model is due to the possibility of many documents expressing the same model. The syntax rules applied to the document determine the correctness of the document and, if correct, the way in which textual elements are to be interpreted as mathematical or conceptual elements of the model.

BRTML language

The primary input format for using SmartBRT. It is translated by the BRTML compiler into files that are read by Paramics and the SmartBRT plug-in during execution of a simulation. Actually, “BRTML language” is redundant, since “L” stands for language; however, we use this phrase to emphasize the “L”, and distinguish from the tools associated with BRTML.

BRTML compiler

The program that translates documents in the BRTML language to files that are read by Paramics and the SmartBRT plug-in during execution of a simulation.

BRTML API

The Application Program Interface to a BRTML model. Knowledge of the BRTML API is not needed for ordinary use of BRTML and SmartBRT: designing corridors, running simulations, processing outputs, etc. The BRTML API is not documented in this release of SmartBRT.

The BRTML API is used by the BRTML compiler and by programs that construct BRTML documents based on other input files (see data sources). A program uses this API to build a BRTML model, and then automatically render it to a document, or to parse a document into a model so that the program can operate on it at the level of modeling elements rather than syntactical elements. This approach is much easier than forcing programs to generate the document directly in correct BRTML syntax.

Item

An item in a BRTML model is defined by its type, its name, and its attributes. An item may also be related to other items, in the following ways:

- An item may have a list of child items (or subitems).
- An item may be listed among the children of another item, in which case we say that the other item is the parent item.
- An item may be referenced as an attribute of another item.

Items that have a common parent are listed in sequence, and their order is significant. The order is, in some cases, interpreted by the BRTML compiler as defining the physical or logical order of entities in the simulation, such as the logical order of stops along bus route, or the physical order of blocks along a corridor.

An item can represent something in the simulation such as a bus type or a terminal, but does not need to. There are items that are used for defining the variables that are to be measured and logged, for instance. The user can introduce items (of new types, if desired) whose only purpose is grouping, or even just documentation.

Item type

The type of an item is used in the item definition and referred to in the document simply by a string with no spaces or special characters: only letters, numbers, and underscores are allowed. A type can be a built-in type or a user-defined type).

Built-in item type

Some item types are recognized by the BRTML compiler for special purposes, such as the **simulation** type. These types are documented in the BRTML reference and related files.

User-defined item type

The user can introduce new types simply by using them (no special declaration is needed). Items of those types can be used for grouping other items (which may be of a built-in type or of a user-defined type) and assigning attributes through inheritance. For example, the **block** type is built-in, but the user may find it useful to group them into items of type **neighborhood**, and to group those into items of type **jurisdiction**.

Item definition

In a BRTML document, an item is defined in the following way:

```
item_type the item name
```

Subsequent indented lines define the attributes and child items. Contrast item reference.

Note that the item name can have spaces, but the item type cannot.

Item reference

An item may be referred to from a different part of the document. References are built using a special reference syntax, involving the characters ^ (up a level), . (down a level), [n] (down to n-th item), item names, and attribute names.

Reference syntax is explained here. Contrast item definition.

References may occur in two kinds of places:

- In attribute values.
- In the list of child items of an item. The item thereby becomes a secondary parent of the referenced child.

Reference target

The item to which a reference refers.

Attribute

An attribute of an item is a property that has a name, unique among all attributes of the item, and a value, which may be another item or a string that can be interpreted as a value of a specific type, such as a distance or a time.

In a BRTML document, an attribute is denoted by the second line below:

```

item name
    attribute: value

```

The attribute belongs to the item because it is listed below it with a greater indentation, and there is no intervening lines of text with a higher indentation. Attributes can also be assigned to an item by inheritance from a parent item.

Parent

The *primary* parent of an item i_1 is the item i_2 which, in the document, has less indentation than i_1 , is defined before i_1 , and is not textually separated from i_1 by another item of less indentation than i_1 . We refer to i_1 as the child or subitem of i_2 .

An item may have additional, *secondary* parents. A secondary parent is associated with an item by referring to the item, as in the following example:

```

item parent 1
    item child
item parent 2
    ^parent 1.child

```

The second line defines the child item as a child of the first (and primary) parent. The fourth line references the child and makes it a child of the second parent.

Child

Also known as subitem or direct child. See parent. See also indirect child.

Indirect child

An indirect child of an item is a child of the item or, recursively, a child of an indirect child of the item. For example:

```
item parent
  item child 1
    item child 2
      item child 3
        item child 4
```

All of the “child n” items are indirect children of the parent, but only child 1 is a direct child.

In practice, this concept is more important than direct child, because attributes are inherited through any number of parent-child relations. Most items that place significance on their children do not distinguish between direct and indirect children. This practice enables flexible grouping using intermediate items that can provide additional attributes.

Child index

Each children of a parent item has a unique index, starting with 0, and increasing by 1 for each successive child at that level:

```
item parent
  item child with index 0
  item child with index 1
    item child with index 0
  item child with index 2
  item child with index 3
```

Note that numbering starts over in each indented level. Using the index to refer to

children is not usually the best practice: it is easily broken if new children are inserted or existing ones removed or if a child is moved up or down a level in the hierarchy. It's generally better to use items names.

Item group

Loosely speaking, all the children of an item i_j and, recursively, children of members of the group. These are the items to which passes on its attributes by inheritance. Grouping is a useful way to assign a default value (which can be overridden) of an attribute to a large class of items.

Inheritance

Inheritance in BRTML refers to an attribute that is passed from a parent to a child item. Unless the child assigns a different value to the attribute, the attribute will have the same value in the child as in the parent.

Orphan item

An item that is defined as the value of an attribute. (This is different from an item that is simply referred to by the value of an attribute.) The notation is:

```
attr:
  a1: value1
  a2: value2
  item sub1
  item sub2
```

The item whose attributes are “a1” and “a2” and whose subitems are “sub1” and “sub2” is the orphan item in this example. Unlike most items, orphan items do not have a type or name. They only have attributes and subitems. Using an orphan is essentially the same as defining an item elsewhere and using a reference to it as the value of the attribute, as in this example, which is more or less equivalent to the one above:

```
attr: some other item
...
...
...
```



```
item some other item

  a1: value1

  a2: value2

  item sub1

  item sub2
```

However, in this example, “some other item” inherits any attributes from its parent.

Orphan items are useful for two reasons:

- An orphans does not have parents, and it does not inherit from the item above it in the hierarchy. An orphan starts with a clean slate of attributes.
- An orphan keeps the definition of the new item close (within the document) to the attribute whose value it is.

PRNG

Pseudo-random number generator. Paramics Modeller and the SmartBRT plug-in have distinct sets of seeds. The latter has a distinct seed for each random variable, such as passenger inter-arrival time at each bus stop. PRNGs in SmartBRT are discussed in the model report.

Boolean

Boolean values for attributes are the following (case insensitive):

- Truth: true, yes, on, 1
- Falsity: false, no, off, 0

Distance

Distances are real numbers and can be specified in the following units:

- m, meter, meters
- km, kilometer, kilometers
- f, foot, feet
- mi, mile, miles

Spaces between value and unit designation are ignored.

If units are omitted, meters are assumed.

If the distance is interpreted as an offset, negative numbers are allowed.

Speed

Speeds are real numbers and can be specified in m/s, mph, or kph.

Acceleration

Accelerations are real numbers and can be specified only in m/s/s.

Time

Duration or time of day (duration since midnight). In a BRTML document, recognized formats are:

- 50.25 seconds
- 90 minutes
- 1.5 hours
- 1:30
- 1:30:15

Abbreviations allowed for seconds are: s, sec, second

Abbreviations allowed for minutes are: min, minute

Abbreviations allowed for hours are: h, hour

If units are omitted, and no colons are present, seconds are assumed.

Spaces between value and unit designation are ignored.

Day

Weekday. In a BRTML document, recognized weekday names are:

Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday, Sunday

Angle

Units currently allowed for angles are degrees, with the following forms:

- 1 degree
- 2.5 deg
- 45 degrees

Lane number

A lane number is a positive or negative integer. A positive lane number is counted from curb side, starting with 1, as is standard in Paramics. A negative lane number is counted from the median side, starting with -1 and decreasing.

Orientation

Direction of travel. Each street has two directions of travel, which we refer to as outbound and inbound. There are also some synonyms:

- outbound: out, down
- inbound: in, up

Outbound and inbound are given meaning by the order of blocks as they are listed in the street. Outbound is the order obtained by reading down this list. Inbound is the reverse order.

5 Tutorial

This document follows a sequence of examples leading up to a simple case study. It uses these examples to explain the construction of BRTML models and the interpretation of their outputs using the SmartBRT plug-in for Paramics.

This document is part of the SmartBRT project.

THE TUTORIAL IS INCOMPLETE DUE TO FAULTY PARAMICS LICENSES—WE ARE CURRENTLY UNABLE TO DEVELOP OR TEST ANY PARAMICS MODELS OR PLUG-INS.

6 User's Manual

6.1 System requirements

The SmartBRT software runs on Windows, SPARC/Solaris, and Linux/x86. We recommend Pentium III or equivalent, 128M memory, and 100Mb free disk space.

SmartBRT requires licensed installations of both Paramics Modeller and Paramics Programmer.

6.2 Installing Paramics plug-ins

Instructions for all platforms are in the document entitled `ParamicsV4-ProgrammerUserGuide.pdf` (which we do not distribute or link to here, as required by Quadstone).

Briefly, you should get the appropriate plug-in file from the download page, unpack it, and put it into your plug-in directory. Here are the relevant instructions quoted (with minor corrections) from the Programmer User Guide.:

1. Place the DLL/SO file in the default plugins directory i.e.

`<PARAMICSHOME>\plugins\<PLATFORM>`

i.e.

`C:\Program Files\Paramics\plugins\windows`

Or

`/home/username/Paramics/plugins/sparc/`

2. Place the full path to the plugin in the default plugins load file i.e.

`<PARAMICSHOME>\plugins\<PLATFORM>\plugins`

i.e.

`C:\Program Files\Paramics\plugins\windows\plugins`

Or

`/home/username/Paramics/plugins/sparc/plugins`

3. Place the name of the Plugin in a network specific file called programming. The new programming files can also be application and platform specific i.e.

`programming
programming.modeller
programming.modeller.windows`

The Plugin named in the programming file should be placed in the Paramics directory [for example, `C:\Program Files\Paramics`]. Programming files are really designed to let you specify specific plugins for specific networks.

In addition using a programming file gives you the option to specifically enable or disable a QPX function from being called improving execution performance and run times for your plugins.

Some “programming” files are automatically created by the BRTML compiler, but you are not required to use them.

You may select any one of the three options. The first two are simpler, but imply that the plug-in will be active even for non-SmartBRT networks. If you are using Paramics for

work that is not based on SmartBRT, you should either use method (3) or use method (1) or (2) and remove the SmartBRT plug-in from the plugins directory before running a non-SmartBRT network.

Note that, on Windows, there are two plug-in files in the download. The file named `smartbrt-mod.dll` is used for Modeller (Paramics with concurrent visualization). The file named `smartbrt-proc.dll` other is used for Processor, the program for executing batch runs. Use of this program with SmartBRT is described in the section on running SmartBRT. These files are automatically created by the BRTML compiler.

Troubleshooting the programming files.

- Note that the programming files for Processor are confusingly named “simulator” as in `programming.simulator` and `programming.simulator.windows` rather than `programming.modeller` and `programming.modeller.windows`.
- If you are using version 4.x of Paramics or earlier and are working on Linux or Solaris, file names are case sensitive, and you must capitalize each word in the file name, such as `Programming.Modeller.Linux`.
- On Linux, Paramics V4 is not able to load plugin files. We hope this will be fixed in V5.
- In all cases, there must be a blank line after the plug-in name in the programming file.

Back to the User's Manual.

6.3 Installing platform-independent SmartBRT software tools

Tools such as the BRTML compiler are written in the platform independent Ruby programming language. First you must install Ruby (and, for SPARC/Linux, you must separately install Fox and FXRuby)—see the brief instructions on the download page.

Now that ruby is installed, installing the SmartBRT tools follows the same steps on all platforms. Simply download the `sbrt-tool.zip` file and unpack it wherever you wish. It contains documents from this web site as well as the program `SmartBRT-tool.rb`. You can run this program by double clicking it, assuming ruby was installed correctly. This program is documented in the Running SmartBRT section of the User's Manual.

6.4 Troubleshooting

6.4.1 Using Modeller in Windows

Modeller hangs while loading a large network in Windows.

The progress meter may say something like “Allocating cost tables: 68%”, and the

application becomes unresponsive. This is recognized by Quadstone as a problem with Exceed, but fortunately it has an easy fix:

The reason for the problem relates to a conflict between Exceed and Windows XP. We have noticed this occur and have worked around the problem by giving the user the option to remove the progress bar and so enable the network to initialise.

There are two ways to do this:

1) load a new or demo network and then ensure that the Tools>>Options>>Progress Bar is off. Saving the network will save the option to the ParamicsApp.cfg file and next time Paramics is initialised and a network is loaded the progress bar will not be used, circumventing the conflict.

2) Manually edit the ParamicsApp.cfg file and change the 'genopt 16 1' line to 'genopt 16 0', switching off the Progress Bar option via the initialisation file.

The effects will be the same.

6.4.2 Using plug-ins on Linux

Regardless of which mechanism used (“programming” files, the plugins directory, the plugins file), Modeller (or Processor) doesn't seem to find the plugin.

This problem has been reported to Quadstone and recognized as a bug to be fixed in Paramics V5. It is apparently not possible, therefore, to run SmartBRT on Linux, with Paramics V4 or earlier.

6.4.3 Using plug-ins on Windows

The programming file refers to a plug-in, but the plug-in is not loaded by Modeller (or Processor).

Check that the line in the file ends with a Windows-style line termination. Many text editors have commands to do this.

6.4.4 Modeller License Issues

If Modeller starts up with the message “This licence for Paramics is supplied purely for

use as a demonstration copy...”, then your license file is incorrect or out of date, or your HASP key is not correctly installed.

6.5 Running SmartBRT

6.5.1 BRTML Tools

The tool for working with BRTML is written in the platform-independent Ruby programming language. First you must install Ruby (and, for Solaris or Linux, you must separately install Fox and FXRuby)—see the brief instructions on the download page.

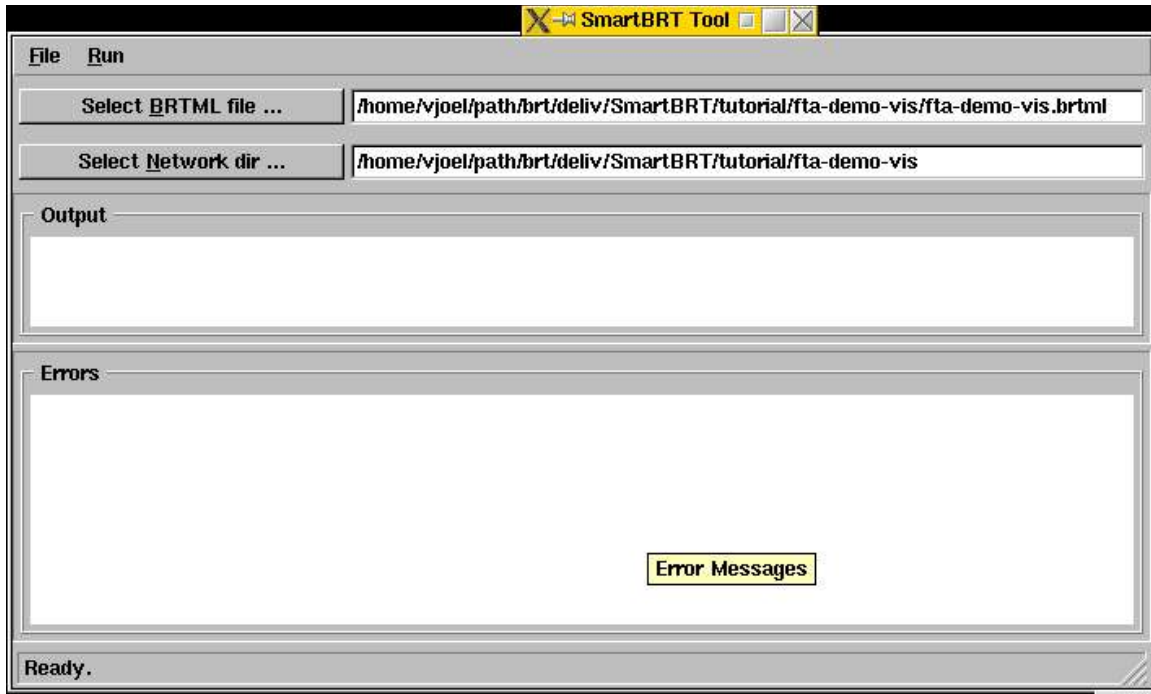
With Ruby installed, installing the SmartBRT tools follows the same steps on all platforms. Simply download the sbrt-tool.zip file and unpack it wherever you wish. It contains the documents from this web site as well as the program SmartBRT-tool.rb. You can run this program by double clicking it, assuming Ruby was installed correctly. Feel free to make a shortcut to this program and place it anywhere. However, do not move the SmartBRT-tool.rb file itself outside of its directory. On windows, clicking on SmartBRT-tool.rbw (note the “w” in the extension) will avoid showing the DOS window. (Otherwise, the two programs are the same.)

You can use this program to:

- compile a BRTML document into a network directory that Paramics (along with the SmartBRT plug-in) can use to run your simulation,
- export a BRTML document as an HTML document, which can be opened in a web browser, and
- start Paramics Modeller on the network directory you have generated using SmartBRT-tool.rb.

If you wish to use SmartBRT-tool.rb to start Modeller, you must make sure that your system's PATH environment variable lists the directory containing the Paramics executables. The procedure for setting this up varies from system to system. It may have been set automatically during the Paramics installation process.

Operation of this program is simple. The main window is shown below.



Use the “Select BRTML file” button, or type in the box next to the button, to select the BRTML file you wish to work with. Use the “Select Network dir” button, or type in the box, to select the directory where the generated network files will be placed so that they can be loaded by Paramics with the SmartBRT plug-in. The File menu has commands to compile the BRTML file to the network directory and to export an HTML version of the BRTML file. The Run menu has the “Start Modeller” command to run Paramics Modeller on the selected network directory. Output and error messages from Modeller are shown in the boxes below.

The functionality of this GUI program can also be accessed from the command line. For syntax help, run it with the following option:

```
ruby SmartBRT-tool.rb --help
```

6.5.2 Running SmartBRT in Modeller

The SmartBRT-tool.rb program is not necessary to execute a SmartBRT network. You can simply run a SmartBRT network in Modeller as you would normally run a simulation—simply use the File/Open menu command to open the directory in which your network files are located. (The BRTML document must have been processed using SmartBRT-tool.rb, first.) Opening the network directly in Modeller is equivalent to the “Start Modeller” command in SmartBRT-tool.rb.

6.5.2.1 Batch runs and other command-line runs

Modeller is useful for a single run at a time, with concurrent visualization. Modeller is not useful in the following situations:

- You want to perform an experiment consisting of many runs.
- You need fast simulation, without the overhead of visualization.
- You want to run in trace mode and use a “pager” or other filter program to process the text-only output of event data, as described in the model report, in the section on debugging output.
- You want to save the trace output to a file.

The Paramics suite comes with a program that is intended for use in these situations: Paramics Processor. Processor has a GUI for defining a sequence of scenarios, executing them, and gathering their outputs. Processor can also be run from the command line to execute a single run without starting up any GUI. Processor is described in the Paramics manual “Processor User Guide”, which should be part of your Paramics installation.

Running Paramics Processor

Unfortunately, the Processor GUI does not recognize the SmartBRT inputs, and it cannot be extended. We hope, in future work, to develop a comparable program for SmartBRT. In the meantime, we use the Processor command line program. This is executed as follows:

```
processor-cmd -netpath dir
```

Where `dir` is the directory of the network. Note: for some reason, you cannot use simply “.” for `dir`, if you have already changed into the directory where your network files are located. In that case, you must supply a longer path, either the absolute path (`\my\simulation\example`) or `..\example`. (This bug has been reported to Quadstone.)

You can get help on options for the processor command by typing

```
processor -h
```

One other useful option is `-clean`, which automatically removes the annoying paralock file.

Processor and the plug-in

Note also that you may need to inform Paramics where your plug-in is located. Recall from the plug-in installation instructions that there are three ways to install a plug-in. If you have chosen the first or second way, there is no additional work needed. If you have chosen the third way, the use of “programming files” in your network directory, you will need a special programming file to use the plug-in with Processor, as opposed to Modeller. The word “simulator” must replace “modeller” in the file name (this inconsistency has been reported to Quadstone). For example, on Windows, you might have a file called

```
programming.simulator.windows
```

which has a single line in it:

```
smartbrt-processor.dll
```

This tells Processor to use the Processor version of the SmartBRT plug-in when you run this network. On Windows, these files are automatically created by the BRTML compiler. On Linux and Solaris, the current version of Paramics (V4) does not use programming files correctly.

Using Processor output

If you have enabled tracing in the SmartBRT plug-in, the plug-in will produce a substantial amount of informative output along with a small amount of output from Processor itself. Tracing shows events, such as bus arrival, passenger boarding, signal priority granting, and so on, as they occur. Tracing can be enabled through BRTML (or directly in the plug-in input file `brt_trace`)—see the BRTML reference for details.

The output can be piped into another program. For instance,

```
processor-cmd -netpath dir 2>&1 | more
```

(This may not work on Windows 95/98/ME.) This does two things with output. First, the `2>&1` causes error messages to be sent out along with normal output. Second, the `| more` causes all of this output to be read by the “pager” program called `more`, which exists on both Windows and Unix-like systems. It allows you to scroll through the output rather than lose it as it goes past. A better program called “less” also allows you to search for patterns, such as “the arrival of a bus at the stop at 3rd and Main streets”, or “the next signal priority event”. Note that these two operations on simulation output are not part of Paramics or SmartBRT—they are standard features of Windows NT/2000/XP and Unix/Linux. See your OS manual for details.

To save output to a file, use the following command line:

```
processor-cmd -netpath dir 2>&1 > file
```

where `file` is the name of your file.

Processor and visualization

Processor does not have concurrent visualization, but this does not affect the ability of SmartBRT to generate outputs for off-line visualization using SWEditor. The procedure is the same as when using SmartBRT with Modeller. See *Connecting to SWEditor* for details.

Scripting multiple runs

Using the command-line access described above, it is easy to use either DOS scripts or Unix shell scripts to navigate through a sequence of directories and run the simulation specified in each one. It is beyond the scope of this document to explain this process. We expect future developments to make it easy for users to design and execute experiments with multiple runs, and scripting will be unnecessary except for advanced users.

6.6 SWEditor Manual

The SWEditor manual is provided in an additional file.

6.7 SmartBRT Software Downloads

This page links to the software provided with the SmartBRT project deliverables and to the additional free and commercial software needed to run SmartBRT. The user's manual, with installation instructions, can be browsed here, and is also included with the “SmartBRT platform independent software and documentation package”, below. Items relevant to each platform, Windows, SPARC, and GNU/Linux, are marked with **W** **S** **L** respectively. Please take note of the copyright text at the bottom of this page. For help with this web site, please contact Joel VanderWerf, vjoel@path.berkeley.edu.

http://www.paramics-online.com	Paramics Modeller, Processor, Programmer, and other tools. Licenses for both Modeller and Programmer are required for SmartBRT. The Paramics suite is a commercial traffic modeling package. W S L	
smartbrt.dll.zip	Paramics plug-in for Windows. W	To install, follow standard instructions in Paramics documentation for your platform. See also the user's manual.
smartbrt.so.gz	Paramics plug-in for Solaris/SPARC. S	
smartbrt.so.gz	Paramics plug-in for Linux/86. L	
sbbrt-tool.zip	SmartBRT platform independent software tools and documentation package. W Follow installation instructions in the user's manual. Includes all the documents from this web site.	

ruby-1.8.1	<p>Ruby Windows installer. W</p> <p>Please select the latest version, at least ruby181-13.exe. You may select ruby182-XX.exe, but only if it is not labelled “RC”, or “Release Candidate”.</p> <p>Run the installer program and follow the instructions. You may choose any location for installation directory. The installer includes the Fox/FXRuby GUI library; please do not disable the FXRuby option.</p>	<p>Ruby is a programming language which SmartBRT depends on. It is free software with a license that is compatible with commercial uses. Ruby's home page is http://www.ruby-lang.org</p> <p>Fox is a GUI library, and FXRuby is Fox binding for ruby. See http://www.fox-toolkit.com and http://www.fxruby.org.</p>
ruby-1.8.1.tar.gz	Ruby source for Solaris and Linux. S L	
Fox Toolkit	Fox source for Solaris and Linux. (Use the latest fox-1.0.xx tarball. Do not use fox-1.1 or fox-1.2.) S L	
FXRuby	FXRuby source for Solaris and Linux. S L	
SWEditor V1.zip	<p>SWEditor installer for Windows only W</p> <p>To install, unpack the zip file, open the folder, and run the installer in side, following the instructions.</p> <p>Note that the zip file size is about 115Mb.</p>	<p>SWEditor is a tool for designing 3D visualizations of transit corridors, complete with near-photo quality buildings, vehicles, signs and so on. It can be used to play animations of simulations generated by SmartBRT.</p> <p>SWEditor is not necessary for running simulations and performing analyses.</p>

7 The Wilshire Model

The data collection and modeling effort covered the Wilshire corridor from the Santa Monica terminal to downtown Los Angeles. We performed the following tasks:

- Manual coding of the road geometry, intersection characteristics, and lane configurations from CAD drawings.
- Processing of outputs from a TRAFFIX simulation to provide phase information and turning counts for most of the traffic signals. (We wrote a program to automate this task.)
- Observation of passenger board and alight counts and estimation of OD tables for

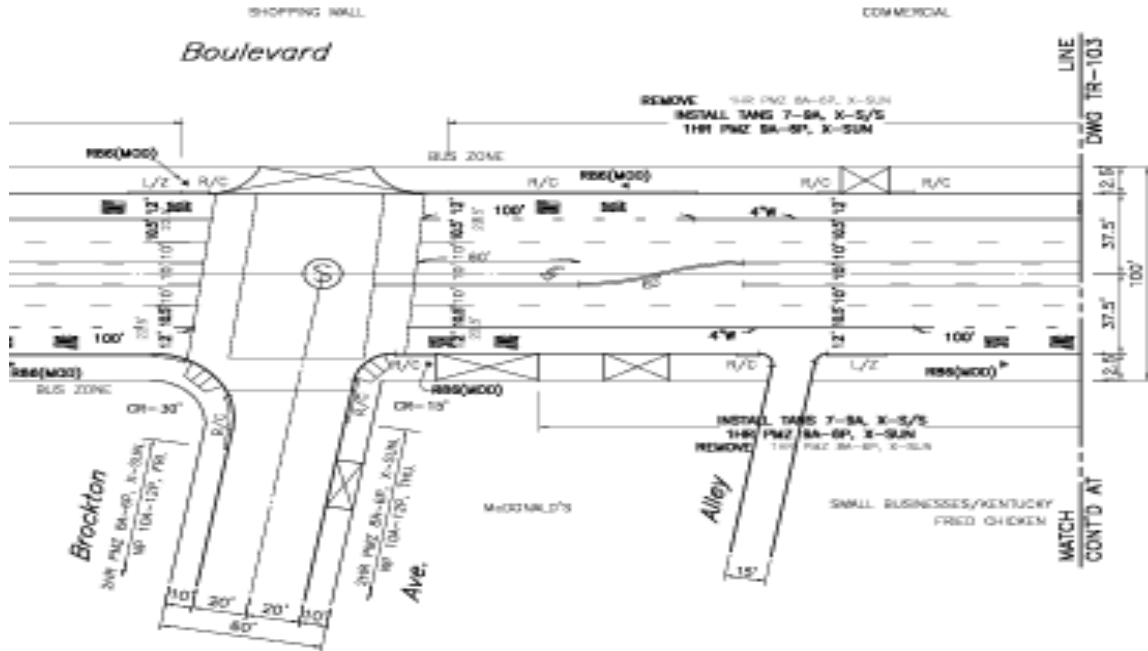
several combinations of time of day and day of week. (This work was done for the BRT Deployment Planning project.)

- Photographic imagery of many recognizable features along the corridor, which are now incorporated into the SWEditor 3D visual model.
- Discussions with LACMTA to define scheduling and bus insertion operational procedures, bus characteristics, bus stop locations, signal priority design, etc.
- Additional corridor details were filled in from road maps and personal observations.

Further details on these tasks are given below. Note that the software used to convert formats may or may not be useful for other networks, depending on which data formats are available; the software can be provided to users of SmartBRT on request.

7.1 Network Coding

We entered data from CAD drawings into a simple shorthand form, and we wrote a program to translate from that form into the more complex BRTML required for the SmartBRT simulation. An example of the shorthand is shown below:



This fragment of a CAD drawing translates (manually) into the following encoded data:

```

-
3w 3e m
220
+
80
Brockton Ave. sw
signal
1s20 1n20
-
3w 3e wltl
90

```

Then, this data file is read as one input (among several) to a program which generated BRTML. The data includes bus stop locations (if available), link lengths and curvatures, lane counts and widths, turn lane configurations, street names.

The TRAFFIX data was produced by a consultant for LACMTA using the TRAFFIX simulation program. We were given data for both AM and PM conditions. Sample TRAFFIX data is shown below:

```

*****
**
Intersection #25 Centinela Ave /Wishire Blvd
*****
**
Cycle (sec):          60          Critical Vol./Cap. (X):          0.557
Loss Time (sec):      0 (Y+R = 7 sec) Average Delay (sec/veh):          9.4
Optimal Cycle:        53          Level Of Service:          A
*****
**
Approach:      North Bound      South Bound      East Bound      West Bound
Movement:      L - T - R      L - T - R      L - T - R      L - T - R
-----|-----|-----|-----|
|-----|
Control:        Split Phase      Split Phase      Permitted      Permitted
Rights:         Include          Include          Include          Include
Min. Green:     21  21  21      0  0  0      32  32  32      32  32
32
Lanes:          0  1  0  1  0      0  0  1! 0  0      1  0  1  1  0      1  0  1  1  0
-----|-----|-----|-----|
|-----|
Volume Module: >> Count Date: 13 May 1999 <<
Base Vol:       156  14  85  16  4  18      2 1421  127  78 1122
19
Growth Adj:     1.00 1.00  1.00  1.00 1.00  1.00  1.00 1.00  1.00  1.00 1.00
1.00
Initial Bse:    156  14  85  16  4  18      2 1421  127  78 1122
19
User Adj:        1.03 1.03  1.03  1.03 1.03  1.03  1.03 1.03  1.03  1.03 1.03
1.03
PHF Adj:         0.95 0.95  0.95  0.95 0.95  0.95  0.95 0.95  0.95  0.95 0.95
0.95
PHF Volume:     169  15  92  17  4  20      2 1541  138  85 1216
21
Reduct Vol:      0  0  0      0  0  0      0  0  0      0  0
0
Reduced Vol:    169  15  92  17  4  20      2 1541  138  85 1216
21
PCE Adj:         1.00 1.00  1.00  1.00 1.00  1.00  1.00 1.00  1.00  1.00 1.00
1.00
MLF Adj:         1.00 1.00  1.00  1.00 1.00  1.00  1.00 1.00  1.00  1.00 1.00
1.00
Final Vol.:     169  15  92  17  4  20      2 1541  138  85 1216
21
-----|-----|-----|-----|
|-----|
Saturation Flow Module:
Sat/Lane:       1900 1900  1900  1900 1900  1900  1900 1900  1900  1900 1900
1900
Adjustment:     0.97 0.89  0.89  0.87 0.87  0.87  0.16 1.00  1.00  0.12 1.01
1.01

```

Lanes:	1.00	0.14	0.86	0.41	0.10	0.49	1.00	1.84	0.16	1.00	1.97
	0.03										
Final Sat.:	1835	236	1446	689	162	811	297	3503	314	220	3786
	65										
----- ----- ----- -----											

Capacity Analysis Module:											
Vol/Sat:	0.09	0.06	0.06	0.02	0.02	0.02	0.01	0.44	0.44	0.39	0.32
	0.32										
Crit Moves:	****					****		****			
Green/Cycle:	0.35	0.35	0.35	0.03	0.03	0.03	0.62	0.62	0.62	0.62	0.62
	0.62										
Volume/Cap:	0.26	0.18	0.18	0.71	0.71	0.71	0.01	0.71	0.71	0.63	0.52
	0.52										
Delay/Veh:	14.1	13.6	13.6	63.4	63.4	63.4	4.5	9.0	9.0	16.4	6.7
	6.7										
User DelAdj:	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	1.00										
AdjDel/Veh:	14.1	13.6	13.6	63.4	63.4	63.4	4.5	9.0	9.0	16.4	6.7
	6.7										
DesignQueue:	4	0	2	1	0	1	0	22	2	1	17
	0										

**											

Our program parses this data and extracts information relevant to signal timing and vehicle turning movements for each intersection. This information is then encoded in the single BRTML document that describes the corridor.

7.2 Passenger Demand Model

7.2.1 Data Collection

We collected data during Metro Rapid’s weekday peak periods for average and heavy passenger loads, that is, between 7am and 10am for morning peak and 4pm and 7pm for afternoon peak. Data was collected during morning and afternoon peak periods on two typical weekdays (Monday and Thursday) and represented average and crush passenger volume load days, respectively. Data was collected by observation as to the number of passengers boarding and alighting each bus and the passenger load for each bus upon its departure from the bus stop together with the bus arrival and departure time and bus identification number, that is, the bus’ run number. The focus was on that part of the Wilshire-Whittier Metro Rapid line between Santa Monica and Downtown Los Angeles. During the morning peak periods we collected data on Metro Rapid buses heading in the west direction while in the afternoon we collected data on Metro Rapid buses heading in the east direction. We were not able to collect data at each of the twenty Metro Rapid bus stops during each of the designated directional peak periods; however, most bus stops were covered.

Due to limited resources, data was collected only on either day for half of the 20-mile corridor. Therefore, in order to obtain a complete estimate of boarding and alighting passengers at each individual stop, a Monday/Thursday ridership ratio was used to convert the recorded passenger counts of one day to the estimates of the counts for the other day. The ratio was about 1.1457, calculated based on observed passenger counts at

six stops where data were collected on both days. In addition, adjustments were made where appropriate to deal with issues such as missing bus counts and late arrivals or early departures of data collectors.

7.2.2 OD Estimation

Given total numbers of boarding/alighting passengers at individual stops, estimating O-D trip tables was essentially the same problem as the conventional O-D distribution problem with trip generation/attraction in traffic zones in the conventional four-step travel demand analysis procedure. Since all necessary data to calibrate the models was not available, we adopted a simple O-D estimation procedure based on the assumption that number of alighting passengers is proportionally divided among all the possible origins according to the respective number of boarding at these origins.

An example of an OD table resulting from this process is shown below.

eastbound on Monday, 4:00-5:00																		
	Ocean/Col	4th	14th	Bundy	Barrington	VA Hospit	Westwood	Santa Mor	Beverly	Robertson	La Cienega	Fairfax	La Brea	Crenshaw	Western	Normandie	Vermont	Alvarado
Ocean/Col	0	3	0.331858	0.155174	0.086621	0.059727	0.475857	0.095967	0.080685	0.067739	0.498734	0.149731	0.100983	0.161396	1.058763	0.495726	0.432896	0.351684
4th	0	0	7.168142	3.351752	1.871019	1.290095	10.2785	2.072877	1.742787	1.463171	10.77265	3.234192	2.181234	3.486149	22.86927	10.70768	9.350553	7.596379
14th	0	0	0	2.493075	1.391688	0.959589	7.64528	1.541832	1.296307	1.088325	8.012832	2.405632	1.622429	2.593042	17.01045	7.964506	6.955059	5.650282
Bundy	0	0	0	0	1.650672	1.138162	9.068018	1.828757	1.537541	1.290855	9.503969	2.853305	1.924353	3.07559	20.17599	9.44665	8.249351	6.701763
Barrington	0	0	0	0	0	1.092968	8.707946	1.756141	1.476489	1.239598	9.126587	2.740007	1.847941	2.953465	19.37484	9.071544	7.921787	6.435651
VA Hospit	0	0	0	0	0	0	4.824396	0.972941	0.818008	0.686765	5.056332	1.518025	1.0238	1.636285	10.73409	5.025837	4.388846	3.565494
Westwood	0	0	0	0	0	0	0	8.731486	7.341065	6.163248	45.37715	13.62324	9.187914	14.68455	96.33121	45.10348	39.38692	31.99789
Santa Mor	0	0	0	0	0	0	0	0	3.040452	2.552635	18.79387	5.642344	3.805362	6.081907	39.89754	18.68053	16.3129	13.25258
Beverly	0	0	0	0	0	0	0	0	0	1.945427	14.32328	4.300171	2.900162	4.635173	30.40691	14.2369	12.43247	10.10012
Robertson	0	0	0	0	0	0	0	0	0	0	12.03459	3.613056	2.436751	3.894529	25.54825	11.96201	10.44591	8.486245
La Cienega	0	0	0	0	0	0	0	0	0	0	0	7.920295	5.341679	8.53732	56.00514	26.22231	22.89881	18.60297
Fairfax	0	0	0	0	0	0	0	0	0	0	0	0	2.627392	4.199219	27.54704	12.89787	11.26315	9.150171
La Brea	0	0	0	0	0	0	0	0	0	0	0	0	0	2.061374	13.52269	6.331492	5.529018	4.491769
Crenshaw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15.86869	7.42992	6.488227	5.271029
Western	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25.22828	22.03077	17.89777
Normandie	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42.41335	34.45656
Vermont	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45.12721
Alvarado	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Witmer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5th	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nowhere	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

7.2.3 Photographic Images

A sample of the collection of images taken along the corridor is shown below:



8 Copyright and License

Components of the SmartBRT software that were developed at California PATH are subject to the following copyright notice:

Copyright (c)2001-2004 The Regents of the University of California (Regents). All Rights Reserved.

Permission to use, copy, modify, and distribute this software and its documentation for educational, research, and not-for-profit purposes, without fee and without a signed licensing agreement, is hereby granted, provided that the above copyright notice, this paragraph and the following two paragraphs appear in all copies, modifications, and distributions.

Contact The Office of Technology Licensing, UC Berkeley, 2150 Shattuck Avenue, Suite 510, Berkeley, CA 94720-1620, (510) 643-7201, for commercial licensing opportunities.

IN NO EVENT SHALL REGENTS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF REGENTS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. REGENTS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE AND ACCOMPANYING DOCUMENTATION, IF ANY, PROVIDED HEREUNDER IS PROVIDED "AS IS". REGENTS HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

SWEditor Version 1.0 User Reference

***University of California
Berkeley, PATH***



Table of Contents

SWEditor Version 1. 0.....	1
User Reference.....	1
University of California Berkeley, PATH.....	1
SWEditor license.....	6
Support	6
Outline of document	6
Chapter 2. System Requirements and Installation	7
System requirements.....	7
Install the SWEditor.....	7
Uninstall SWEditor.....	8
About SWEditor Format.....	8
Menus.....	9
Toolbars	11
File access toolbar.....	11
Viewing window toolbar.....	11
Grid toolbar.....	12
The Grid toolbar allows the user to display selected grid, sets the grid properties or clear all grids. One cannot clear just one grid. Please refer to section 4.1.3 Grid planes for details.....	12
Editing Mode Toolbar.....	13
The user can select from the following three editing modes. Please refer to section 4.1.2 Editing Modes in Chapter 4 for more details.....	13
Object property toolbar.....	13
Rendering toolbar.....	14
Connectivity toolbar	14
Animation toolbar.....	15
Object selection toolbar.....	15
Dialog boxes.....	15
General Properties of the objects.....	16
Coordinate Frame.....	16
Network Objects.....	18
Roadway.....	18
a. Engineering Design Parameters.....	18
b. Basic design parameters.....	19
c. Advanced design parameters.....	20
e. Roadside Attributes	23
Intersection	24
a. Components that can be changed.....	26
Due to the technical issues in synchronizing Paramics traffic control devices with SWEditor’s traffic control devices in animation run-time, traffic light phase is not animated in our Paramics off-line animations.	27
Supplemental Objects.....	27
Building.....	27
Barrier.....	29
Road Sign.....	29
Bus Stop and Bus stop sign	29
Trees.....	31

3DS Models.....	32
Camera.....	33
4.1 Manual Modeling Method.....	35
4.1.1 Viewing display setting.....	35
4.1.2 Editing Modes.....	35
4.1.3 Grid planes.....	36
a. Change Grid Plane properties.....	36
b. Move grid planes with mouse.....	36
4.1.4 Rendering Mode.....	36
4.1.5 Object Type.....	36
4.1.6 Create a new object.....	36
4.1.7 Select an existing object.....	37
4.1.8 Unselect the selected object.....	38
4.1.9 Delete an existing object	38
4.1.10 Modify an existing object.....	38
4.1.11 Move an existing object with mouse.....	38
4.1.12 Move the control points of a Roadway/Barrier object with mouse.....	38
4.1.13 Change the properties of a lane in roadway with mouse.....	38
Modify Intersection	38
Move junction branches with mouse.....	39
4.1.15 Move traffic control devices of a junction with mouse.....	39
4.1.16 Change the properties of a traffic control device.....	39
4.1.17 Connect two roadways with mouse.....	39
4.1.18 Connect roadway-junction with mouse.....	39
4.1.19 Connect Junction-Junction with mouse.....	40
4.1.20 Insect a roadway between roadway-roadway and roadway-junctions with mouse.....	40
4.1.21 Disconnect a connected object from the network	40
4.1.22 Create a new file.....	41
4.1.23 Open an existing file	41
4.1.24 Save the edited database.....	41
4.1.25 Merge two exiting files.....	41
4. 2 Create simulated world from script file.....	41
Script Modeling Method.....	41
a. Bus stop	42
b. Roadway.....	42
c. Intersection	42
Texture.....	43
3DS Model.....	43
6.1 Camera View Point.....	44

Table of Figures

Coordinate frame defined in the SWEditor.....	17
A straight two-way roadway created by default parameter values.	20
Roadway property dialog box with default numbers	20
Fig. 3.4 a. A straight two-way roadway with a merge	21
Advanced roadway property dialog box with default values.....	22
Lane property dialog box.....	23
A straight two-way roadway with gap (median) = 20 ft, grass and barriers by the sides	23
Attribute property dialog.....	24
Junction property dialog.....	25
Definition of l and s.....	26
A 4-way junction with default parameter number.....	26
Fig. 3.11. An example of overhead traffic light.....	27
Arco gas station.....	28
Building property dialog.....	28
Fig. 3.13.a . A speed limit sign.	30
Fig. 3.13.b. Road Sign property dialog	30
Bus stop property dialog.....	30
Bus Shelter Type 1 with length of 20ft.....	31
Bus Shelter Type 2 with length of 20ft.....	31
Tree property dialog.....	32
A building's 3DS model.....	33
Camera property toolbar.....	34
When the number of lanes of the roadway and the number of lanes on the corresponding branch of an intersection do not match the program connects them regardless of the mismatch. The user must be careful!	40

Chapter 1. Introduction

Simulated World Editor (SWEditor) is a graphical software developed to create a three-dimensional virtual world of urban traffic networks in close to photo-realistic quality in which vehicle movement can be animated. A specialty of SWEditor is its capability of visualizing transit infrastructure and animating transit operation. The greatest advantage of SWEditor is that it is easy and quick to learn and use.

SWEditor was developed to ease and speed up the process of building a three-dimensional virtual urban environment in which transit infrastructure and operation can be visually demonstrated. Users can create a virtual urban transportation environment in SWEditor by “building” roads and intersections complete with traffic lights and traffic signs. In addition, they can populate the surrounding of this transportation network with the usual urban street objects, such as buildings, sidewalks, or trees. This environment can be customized to resemble actual real world streets and buildings relatively inexpensively through using digital photographic images of real buildings as wallpapers on the virtual buildings. Once the virtual environment is built, using vehicle movement trajectory data from a separate file vehicle and bus movement can be animated within SWEditor’s virtual world.

Our goals were to develop a tool that

- requires no prior knowledge of other computer aided design (CAD) tools or any computer graphics skills,
- easy and quick to learn by anyone,
- provides an inexpensive, quick way to build 3D visual environment,
- provides an inexpensive, quick way to make this virtual environment resemble a real environment where the new transit system is planned to be deployed,
- utilizes vehicle trajectory data to animate vehicle movement in this 3D environment.

We developed SWEditor for quick and efficient demonstration of Bus Rapid Transit (BRT) infrastructure and operation concepts. BRT is still a relatively new concept. Our goal was to develop a tool that aids transit planners to communicate their design ideas to decision makers and to the public. Transit agencies would be greatly helped by such tool but they are unlikely to be able to afford an expensive CAD software and unlikely to have the personnel and time required by applying these tools. SWEditor is free, easy to learn and use. Transit planner need to communicate their ideas to the local community and local decision-makers, therefore it is very beneficial that SWEditor’s virtual environment can be built to resemble the actual streetscape where the project is proposed. Finally, unlike other CAD program, SWEditor give the special capability to transit planner to show not only the static transit environment, but also vehicle movement within this virtual environment.

SWEditor license

SWEditor is free software for research purposes. Applications for any commercial purposes need to obtain permissions from PATH, University of California at Berkeley.

Support

If you have any technical questions, please contact Swe-Kuang Tan at swekuang@path.berkeley.edu.

SWEditor is a stand alone visualization tool that runs in Windows operating systems (Windows 2000/XP). Unlike other commercial modeling tools, such as Multigen Creator and 3DS Max, that users have to build models at polygon level, SWEditor is designed for users to create the virtual world at model level. It means the users only need to define the parameters for predefined models.

SWEditor has used to animate the off-line Paramics simulation in its 3D virtual world for SmartBRT project. To do so, a translator program, PST (Paramics-SWEditor Translator) has implemented to translate the Paramics network to SWEditor's network script to regenerate the Paramics network in SWEditor automatically. In addition, vehicle trajectory files have to be generated by Paramics in order to replay the simulation in SWEditor. Please refer to Chapter 7 for details.

Outline of document

In this document we introduce SWEditor in detail in the order a user would proceed.

- Chapter 2 list the minimum system requirements, gives instruction as to how to install/uninstall SWEditor, and lists all file formats created or used by SWEditor.
- Chapter 3 introduces the graphical user interface through lists of menus, toolbars and dialog boxes. If needed commands are given short explanation. In addition, each command is linked to its detail explanation later in the document.
- Chapter 4 describes the coordinate system and gives detailed description of the design parameters of objects in SWEditor.
- Chapter 5 describe the process of manipulating individual objects to build the virtual world. It discussed both manual and script file based modeling methods.
- Chapter 6 instructs users about how to navigate within the virtual world to inspect it or to view their animation.
- Chapter 7 discussed animation in SWEditor's, such as what files are needed, how to use them, how to run the animation, and what file are generated as a results of the animation.

Chapter 2. System Requirements and Installation

In this section we specify minimum system requirements for the SWEditor software, as well as installation and uninstallation procedures.

System requirements

The following is the minimum system requirement for Windows operating system:

1. Intel Pentium III 800 MHz CPU or higher
2. 128 MB memory or higher
3. 150 MB or greater free hard drive space
4. OpenGL compatible 3D accelerator card with 32 MB texture RAM or higher
5. 1280x1024 or higher screen resolution
6. CD Rom
7. Windows 98, Windows 2000 Professional, Windows ME, or Windows XP operating systems.

Install the SWEditor

In the following we describe how to install SWEditor.

Step 1. Insert the SWEditor CD into the CD-ROM drive. The SWEditor InstallShield wizard will run automatically. If the wizard does not run, explore your CD-ROM drive and double click on Setup.exe to execute the “installshield wizard”.

Step 2. Click **Next** or press **Enter** on the *Welcome Window* to advance to *License Agreement Window*.

Step 3. Click **Yes** to agree to the licensing terms in order to advance to *User Information Window*. If terms are not accepted the setup will terminate.

Step 4. Enter information to **Name** and **Company** fields on the *User Information Window*. Key in anything on the **Serial Number** field. The software is not protected by serial number, but input is required to activate the **Next** button on the window. Click on **Next** or press **Enter** to continue the setup process.

Step 5. On the *Choose Destination Location Window*, choose your final destination for the software location. The default location is C:\Program Files\PATH\SWEditor\
After destination is selected, click on **Next** or press **Enter** to initiate the installation process.

Step 6. Click **Finish** in the *InstallShield Wizard Complete Window* to complete the setup process. An “explore” window will open to show the installed files.

Step 7. To startup the program, go to the **Bin** folder under your destination folder and double click on **Editor.exe**. We recommend creating a shortcut on the desktop for this executable.

Step 8. To create a shortcut, place your mouse cursor on the **Editor.exe** icon and press right mouse button, select Create Shortcut item in the pop up menu and left click on it. A shortcut icon of the executable will appear on the same window. Move the shortcut icon to your desktop.

Uninstall SWEditor

To uninstall SWEditor, go to **Control Panel** and double click on **Add or Remove Programs**. The *Add or Remove Program Window* will pop up and display a list of the program names the Windows uninstaller can uninstall. Double clicks or press the **Remove** button on SWEditor. The uninstaller will uninstall the program.

About SWEditor Format

SWEditor uses its own format, road database (rdb), to display the virtual world database. This format is designed in the form to minimize the memory space required for the SWEditor graphical objects. In general, the object properties and minimum set of data are stored to the database. This format can be converted to other formats, such as 3DS if proper tools are available.

Chapter 3. Graphical User Interface

SWEditor is a graphical editor used to construct three-dimensional virtual world with transit elements. Like other Computer Aided Design (CAD) tools, SWEditor has a set of modeling features and viewing windows the editing purposes. All these features are accessed through the Graphical User Interface (GUI).

The GUI consists of menu bar, toolbars and dialog boxes. Menu bar is mainly for file access and some display options. Toolbars are mainly used for editing settings on functionalities and dialogs show the properties of the settings. This chapter shows the functionalities embedded on toolbars. Items listed in this chapter are linked to their detailed explanations that are given in later chapters.

Menus

SWEditor has limited functions embedded into menus. There are only two menus: File and Options.

The **File** Menu consists of the functionalities for opening and saving files. The ‘Merge Current File With ...’ option is implemented so that two SWEditor database files (in rdb format) can be combined. The ‘File Merge Property’ is an option that allows users to define what types of objects will be copied from the selected merged file to the current file.

The **Options** menu provide the flexibilities of object type drawings and object merge type while combining two files. The object type drawing option is an option that allows users to define what types of objects to be drawn on the screen. For example, if camera objects are not desired to be seen in animations, the users can select ‘Object Drawing Options’ and a dialog with a list of object names will pop up. Simply unchecked the camera object, and camera objects will be disappeared from the screen. ‘File Merge Object Type Options’ is an option that allows users to define what types of objects will be copied from the selected merged file to the current file.

Chapter 4 will explains more about these two functions.

File and Options Manu bar

New	Ctrl+N
Open...	Ctrl+O
Close	
Save	Ctrl+S
Save As...	
Merge Current File With ... File Merge Property	
Recent File	
Exit	







Object Drawing Options
File Merge Object Type Options

Toolbars

File access toolbar








This toolbar includes functionalities related to manipulating files, such as creating new files and working with existing files:

- a.  Create a new file.
To create a new rdb file, click on this icon and a dialog will pop up and ask whether you want to save the current rdb file before this new rdb file is created. Click **Yes** if you want to save the current file, otherwise click **No**.
- b.  Open an existing rdb file.
To open an existing rdb file, click on this icon and a file browser will pop up to allow the users to select the desired file and click on **Open** to open the file.
- c.  Save current rdb file.
To save current rdb file, click on this icon and the current rdb file will be saved. If no file name has been assigned, a file browser will pop up to let the users to browse and enter the file name. Click on **Save** to save the file. If a file name has been assigned, the current file will be saved to the assigned file name.
- d.  Merge an existing rdb file with current opened rdb file
To merge current opened rdb file with an existing rdb file, click on this icon and a file browser will pop up and users can select the desired existing file and click on **Open** for merging.
- e.  Open a SWEditor network script file
To generate a transit network from a script file, click on this icon and a file browser will pop up. Select the desired network script file and click on **Open**. SWEditor will then create a network based on the descriptions in the selected script file. The **Data** folder in the SWEditor program directory is the default folder for network script files.
- f.  Open an animation trajectory file
To run an animation, click on this icon to load the vehicle trajectory file. A file browser will pop up to allow users to select the desired trajectory file. Click on **Open** to load the selected trajectory file. The **Trajectory** folder in the SWEditor program directory is the default folder for network script files.

Viewing window toolbar



This toolbar allows the user to select a viewpoint to view the whole scene. For the definition of front, side, top, and perspective cameras, please refer to section **Coordinate Frame** in **Chapter 4**.







- a.  Front view window
A view captured by the default front view camera.
- b.  Side view window
A view captured by the default side view camera.
- c.  Top view window
A view captured by the default top view camera.
- d.  Perspective window
A view captured by the default perspective view camera.
- e.  Camera selection
A list of user-created-camera allows user to select a desired camera from the list and change the current view point to the view point of the selected camera.

Grid toolbar



SWEditor defines an absolute three-dimensional coordinate system that applies to any objects within the animation. Axes X and Z form the horizontal plain with while the vertical Y axis measure elevation. The grid system visualizes the scale of this coordinate system, helping the user in sizing up objects and seeing dimensions.

The Grid toolbar allows the user to display selected grid, sets the grid properties or clear all grids. One cannot clear just one grid. Please refer to section **4.1.3 Grid planes** for details.

- a.  Show grid on YZ plane
- b.  Show grid on XZ plane
- c.  Show grid on XY plane
- d.  Show grid on all planes
- e.  Clear all grids
- f.  Show current grid properties

Editing Mode Toolbar

The Editing toolbar allows the user to select among editing modes.



The user can select from the following three editing modes. Please refer to section **4.1.2 Editing Modes** in Chapter 4 for more details.

a.  Surf mode

In surf mode, the users can navigate around the ‘world’ with mouse in addition to arrow keys. Press and hold on the left mouse button, the ‘world’ will be moved along with the mouse movement. No object can be created in this mode.

b.  Create mode

Objects can only be created in this mode. Once the left mouse button is clicked, a new object will be created at where the mouse is pointing on the screen. The created object type is defined by the selected object type in object selection tool bar.


c.  Modify mode

In this mode, the properties of existing objects can be modified. Use left mouse click to select the desired object, once the object is selected, a red bounding box will appear on the object as indication. A green ball will appear at the center of the object once an object is selected.

Object property toolbar



This toolbar handles object editing functionalities. Please refer to Chapter 4 for more details.

a.  Show new object properties

To create a new object (after selecting Edit mode and selecting the object type that you want to create) press this button. The object’s dialog box will pop up.

b.  Delete an object

When selected, the selected object will be deleted.

c.  Retrieves a selected object’s properties.

When selected, this button will open a dialog box containing the properties of the selected object allowing the user to view/review and change properties of already existing objects.

Rendering toolbar

This toolbar allows the user to select how the objects are drawn. The selection applies to all views.



This toolbar sets object rendering details

- a. Simple wire-frame drawing (see through)
When selected, the objects are drawn by the simplest wire-frame representation to reduce CPU (Central Processing Unit) load.
- b. Wire-frame drawing (see through)
When selected, the objects are drawn in wire-frame. This rendering mode provides more details in wire-frame than **simple wire-frame drawing mode** does.
- c. Polygon drawing (not see through, simple textures or representations of objects)
Objects are drawn in polygons. This rendering mode helps the users to visualize the virtual world without introduce much load onto CPU.
- d. Texture drawing (not see through, real textures)
This mode provides photo-realistic view of virtual world. All textures on the objects will be loaded and display on the screen.

Connectivity toolbar

This tool bar provides function to connect and disconnect object. Please refer to Chapter 4 for more details.








This toolbar executes

- a. Connect two selected network objects
When selected, the two selected network objects will be connected to form a network, if conditions are met.
- b. Disconnect selected objects from connected objects
If a connected network object is no longer desired, it can be removed by this function. Simply select the object and click on this icon, the selected objected will be removed from the database.
- c. Insert a roadway between two selected objects. This function is implemented to ease the editing effort if one wants to connect two network objects with an additional roadway object without defining explicit roadway parameters.

Animation toolbar












This toolbar sets animation functionalities

- a.  Play an off-line animation
- b.  Pause the running animation
- c.  Stop the running animation
- d.  Record the running animation shown on the screen to a movie file
- e.  Show the animation data

Object selection toolbar



This toolbar presents the user the available object types, such as:

- a.  Roadway
- b.  Intersection
- c.  Bus stop
- d.  Road/Traffic sign
- e.  Building
- f.  Barrier
- g.  Tree
- h.  3DS model
- i.  Camera

Dialog boxes

Dialog boxes display and allow modification of the properties of the SWEditor objects. In addition to the properties of the selected objects, most of the dialogs have a preview window to display the objects with updated properties. However, the objects will get updated only if the Preview button is pressed. To confirm the modifications, press the OK button. If Cancel button is pressed, the modifications are ignored. The details of dialogs for each object will be discussed in Chapter 3 where the objects are discussed in detail.

Chapter 4. Design Parameters of the SWEditor Objects

In SWEditor, an object is a 3D model with predefined parameters. There are two types of object in SWEditor: network objects and supplemental objects. Network objects are used to create a traffic network while supplemental objects are used to make the simulated world look more realistic. Network objects consist of Roadways and Junctions while supplemental objects are Buildings, Barriers, Trees, Road/Street Signs, Bus Stops, and other 3DS models.

Objects are created in a three-dimensional coordinate system. Their location and orientation are determined within this coordinate system. Furthermore, position and angle of the users' view point for inspecting the animated world is defined within this coordinate system.

In the first sections of this chapter, we introduce the coordinate system and view points, define object location and orientation. In the second section we introduce all available objects and discuss their design parameters. Later, in Chapter 4 we present how to manipulate these objects to create the animated world.

General Properties of the objects

All objects are created within SWEditor three-dimensional coordinate frame. This coordinate system also determines the viewpoint of the user, such as top, side or front view, as if looking through a binocular positioned along the axes of this coordinate system. With relationship to this coordinate system, all objects are characterized by their location and orientation. In this section first we describe the coordinate system and explain the available viewpoints, then, second, we present how location and orientation are defined. The following larger section of this chapter discusses each object in detail.

Coordinate Frame

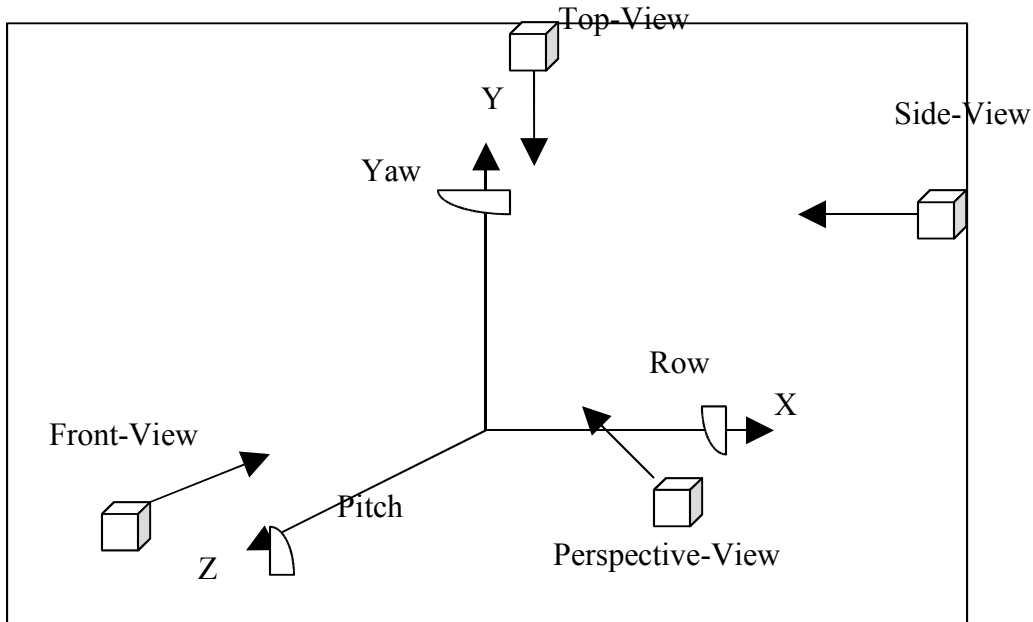
SWEditor defines an absolute three-dimensional coordinate system that applies to any objects within the animation. **Figure **** shows the coordinate frame defined in SWEditor. Axes X and Z form the horizontal plain with while the vertical Y axis measure elevation.

Also, point of view of the user is defined with reference to this coordinate system. Figure 3.1 shows the default view point positions:

The front-view point is placed somewhere at +Z axis and is looking at -Z direction. The side-view point is placed somewhere at +X axis and is looking at -X direction. The top-view point is placed somewhere at +Y axis and is looking at -Y direction.

The perspective-view point is placed somewhere in the three-dimensional space, on X-Z plane with some elevation and is looking toward the Y axis.

Note that it is significant that not only the viewpoint's position is predefined within this coordinate system, but also the direction each viewpoint is looking.



Coordinate frame defined in the SWEEditor

These view points have an additional significance: objects can be edited in top, side and front view, but not in perspective view. Perspective position only allows viewing.

Location and orientation of any object is determined within this coordinate system as follows:

Location: consists of x, y, z coordinates in feet.

Orientation: measured by yaw, pitch, and roll, angles in degrees with relationship to the respective axes of the coordinate system. Roll is the angle associates with X axis, Pitch angle associates with Z, and Yaw angle associates with Y axis.

In practice, the view point's position and its viewing direction together with the orientation determine left and right of an object. Given such definition, left and right remains the same, since objects are always viewed from a predetermined view point that cannot be changed in top, side and front view. (Note, that the perspective camera may change its position within the three-dimensional space, but it is always looking toward the Y axes. Therefore, even looking through that camera, left and right remains the same.)

The following second section of this chapter describes all objects available within SWEditor, first network objects then supplemental objects.

Network Objects

Network objects are used to create the transportation infrastructure, such as roads and intersections. Therefore, there are two network objects: roadway and intersection. These are created individually and then connected together to form a network. Here we discuss the properties/parameters of these objects. The methods of manipulating these objects, connect them, to create a infrastructure network will be discussed later in Chapter 4.

Roadway

the first point on the dialog box is “method”. This determines how the roadway object is created either with engineering parameters or with control points. Engineering parameters are numerical input from the user. Control points are used to graphically create a roadway. These two modes have other repercussions. Roads’ shape can be changed by dragging with the mouse only in control points mode. In engineering parameter mode the road size and shape can only be changed through entering different values.

Roadway is defined by a group of parameters, such as:

- engineering design parameters,
- basic design parameters, and
- advanced design parameters.

a. Engineering Design Parameters

These parameters create the object Roadway in its basic outline: length and whether it is straight or curved. These parameters are displayed in the Roadway Properties dialog box. They are used to design a segment of a roadway in basic engineering terms that include:

Roadway Physical Type: describes whether this piece of roadway is straight or curved.

The default type is straight.

Length: defines length of the roadway at its center. The default length is 100ft.

Radius¹: defines the radius of a curve roadway

Side, such as left or right: describes the side where the center of the radii is located.

Start side and left side are defined depending on the way the roadway was generated. If it is generated by engineering parameters, then start side is on the left side of the screen.

Left side of the road is the bottom edge, right side is the top edge.

If the roadway is generated using control points, start side is where the first control point was put down.

Left and right side of the roadway is defined by looking out from the END side onto the roadway. Forward direction leaves the end side to arrive to the start side, backward direction arrives at the end side form the start side.

¹ We understand that in reality a curved road is rarely designed with one constant radius. However, Paramics makes this assumption. Given that trajectory data for car movement will come from Paramics it seemed necessary to match this assumption.

Merge lanes are added to directions, not to the roadway. Therefore, when the user decides which side to put the merge lane left and right are in relationship to the direction, not to the roadway. Left and right side of a direction is defined by looking in the direction you are going. For merge lanes, end and start side are still defined by the roadway.

Just by looking at a created roadway object one cannot determine any of these references. Since roads can be created in any direction and since after creation one cannot determine which was the first control point, the best way to orient oneself on a roadway object is to put grass or sidewalk on one side of the roadway, to mark it. (see two curved roads drawings)

Note that since objects can be created and changed in only the front, side and top view, that these views are tied to the coordinate system and their point of view is always the same, therefore left and right side of the objects remain always the same.

b. Basic design parameters

These parameters determine the details of the roadway through direction, number of lanes and width of lanes. These parameters are displayed in the Roadway Properties dialog box.

Roadway Type: determines whether a road is One Way and Two Way. If a piece of roadway is defined as one way, then only “forward” lane properties can be defined. Otherwise, “forward” and “backward” lane properties are used. The default type is Two Way.

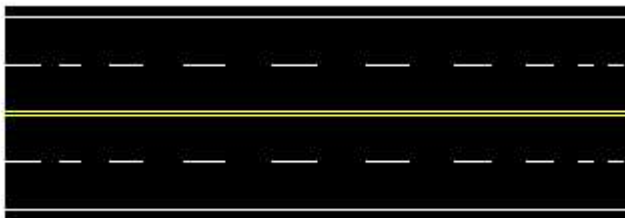
Roadway Name: Name of the roadway.

Number of Lane per direction: defines the number of lane on a roadway segment for each direction. The default number is 2 per direction.

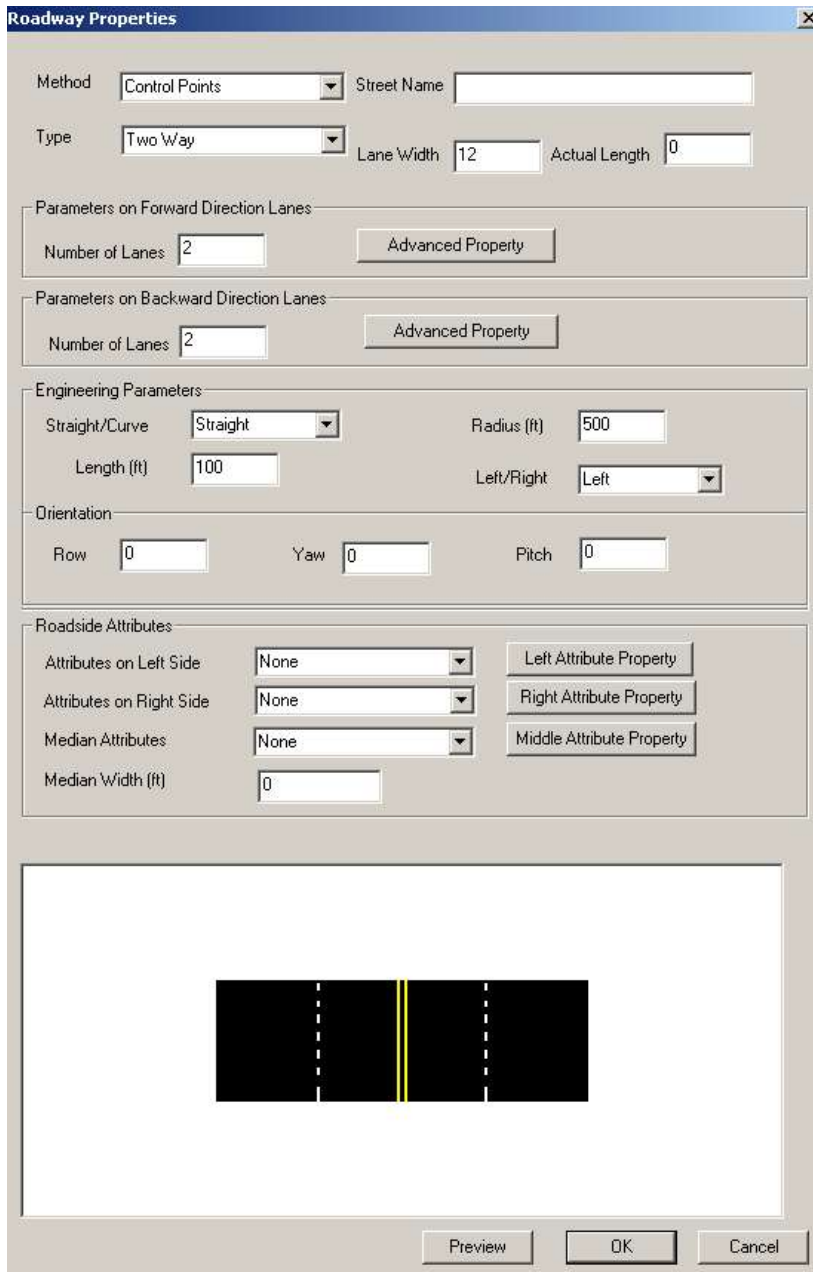
Lane Width: defines the width of all lanes in one direction. The default number is 12ft. This means that all lanes have the same width. Individual lane width cannot be defined.

Median: defines the distance between the two directions of a roadway. This parameter is used when the roadway is two way. The default is 0.

If the type of a roadway is two way, then Number of Lane and Lane Width are defined for forward and backward lanes separately.



A straight two-way roadway created by default parameter values.



Roadway property dialog box with default numbers

c. Advanced design parameters

An optional advanced design features that the Roadway object has is Merge. This feature allows the user to create an additional lane whose length is less than that of the roadway segment's total length. This feature can be used, for example, to create a turning lane. A merge lane can be added to both directions. That is, there are forward merge and backward merge properties on a segment of roadway if the roadway is two way. These parameters are displayed in the Advanced Roadway Properties dialog box that can be

pulled up by clicking on the Advanced Property button, next to the lane numbers in the Roadway Properties dialog box.

Merge: This optional property allows the roadway to have lane addition/reduction features.

The parameters that define a merge are:

Location: A merge can be placed at Start or End section of the roadway. If a merge is defined at the beginning of a roadway, it's used as lane reduction. Otherwise, it's used as lane addition.

Side: A merge can be located at Left or Right side of each direction of roadway. A merge can be located on either side of the through lanes of either direction.

Number of Merge Lane: the number of merge lane is 1. Only one merge can be added to a direction of a road section. (The total number of lane after merge is the sum of number of lane (+/-) number of merge.)

Merge Lane Width: The width of the merging lane. The default number is 12ft. This is a user input that can be changed. Also, it can be different from the width of other (through) lanes' width.

Merge Distance: The actual length for the merging lane. The default number is 100ft.

Merge Transition Distance: This parameter describes the length needed for the lane deduction/addition. A user input

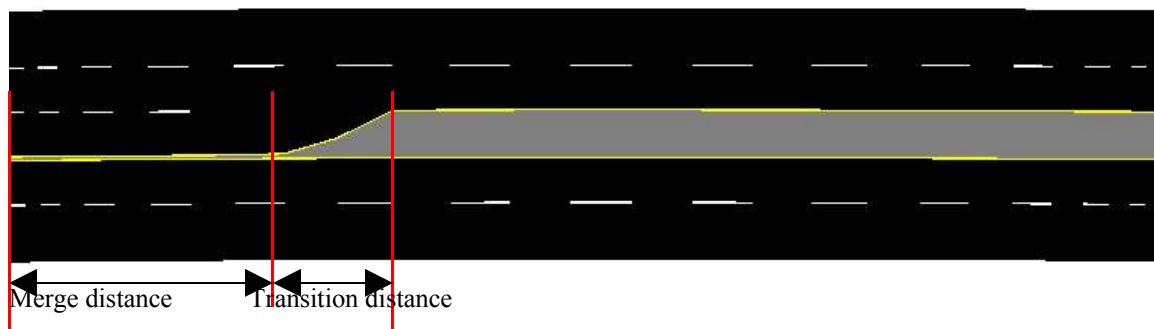
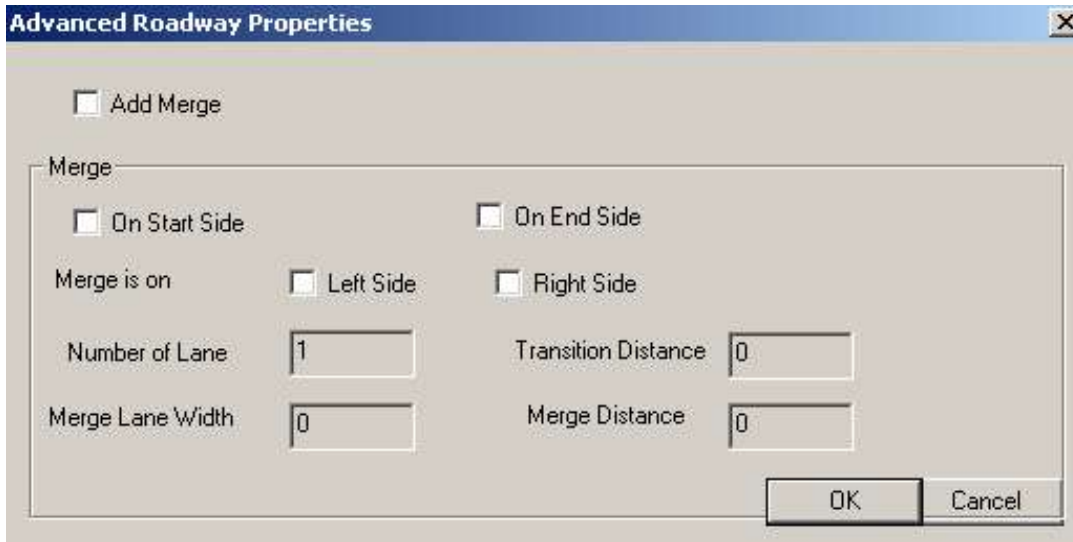


Fig. 3.4 a. A straight two-way roadway with a merge



Advanced roadway property dialog box with default values

d. Components: Lane:

Cannot get to this dialog box from the RoadWay properties dialog. Only through selecting the lane on the picture of the roadway.

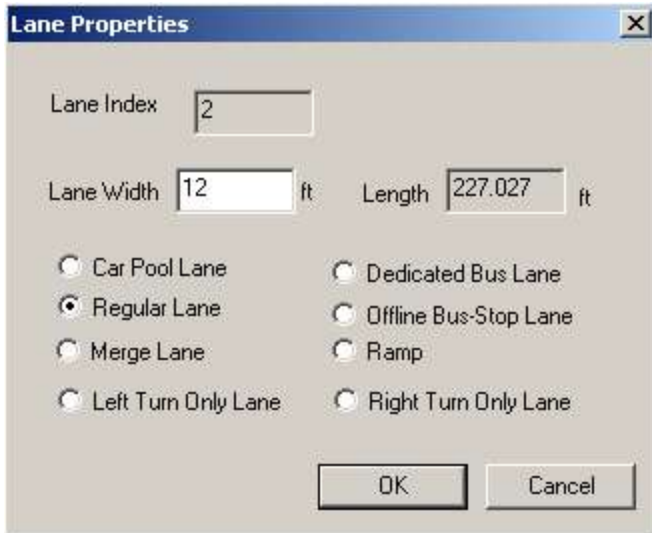
The properties of a lane include index, length, width, and type. Length and width are calculated and defined internally by the program when the segment of the roadway is generated. The only user accessible parameter is Type.

Index: use to correlate the lanes in a roadway. Given, user cannot change this parameter. Lanes are counted from the median out in each direction, starting from 0.

Length: length of the center line of the lane.

Width: width of the lane, cannot be changed.

Type: describes the use of the lane. A lane can be defined as a regular lane, car pool lane, dedicated bus lane, offline bus stop lane, merge lane, ramp way, left turn only lane, and right turn only lane. Choice of type effects the visualization of the lane.



Lane property dialog box

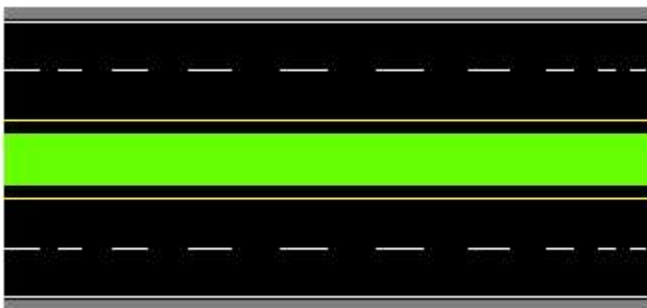
e. Roadside Attributes

These specify attributes of the roadsides as either “grass”, “barrier”, or “sidewalk”. In this case, “grass”, “barrier”, or “sidewalk” are not objects. Here they are attributes of the Roadway object and thus cannot be separated from the roadway. User can specify attributes of the left and right side of the roadway separately, as well as the attributes of the median.

The width of the median is defined on the Roadway Properties dialog box in feet. The Middle Attribute Property button brings up the dialog box called Barrier. In there user can select texture as either “grass”, “barrier”, or “sidewalk”.

Grass: Width and texture are the properties of grass. If these properties are not specified, the program will use the default settings. The default width is 12ft.

Barrier and Sidewalk: The size and dimensions of the barrier can be defined by the user by determining the top and bottom width and the height of the barrier. This way, the user can create a tall wall or can use this object to create a sidewalk.



A straight two-way roadway with gap (median) = 20 ft, grass and barriers by the sides.

Texture: the selected texture is pasted on the surfaces to show the material properties.

Top Width: Top width of barrier. The default number is 2.

Bottom Width: Bottom width of barrier. The default number is 4.

Height: Height of barrier. The default number is 3.5.



Attribute property dialog

Intersection

When an intersection object is created, it is first generated in its default setting. All the branches are assumed to have same number of lanes and all the lanes have the same width. The number of lane and lane width for all branches can be modified afterward. There are parameters used to define an intersection:

Number of branches: defines the number of branches the intersection has. The default number is 4.

Number of Lanes: defines the number of lanes that each branch has. The default number is 2 in each direction. Each branch can have different number of lanes as long as the configuration makes sense in term of traffic flow.

Lane Width: defines the lane width for lanes on all the branches. The default number is 12ft.

Traffic Control: defines the traffic control devices used at the intersection. Choices include: "overhead traffic light", "traffic light post", "stop sign", or "no control device". The default setting is "overhead traffic light".

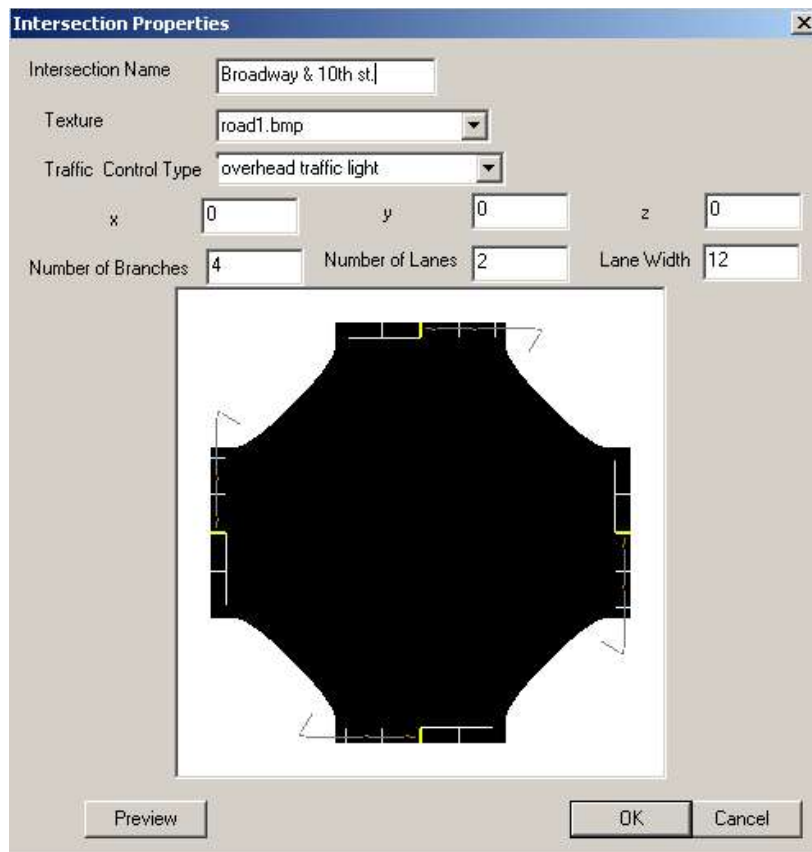
Graphically the intersection is characterized by the following

Center of the intersection: this point is geometric center of the default intersection. It will remain the same no matter how the default intersection is changed. This is the reference point when moving the image of intersections.

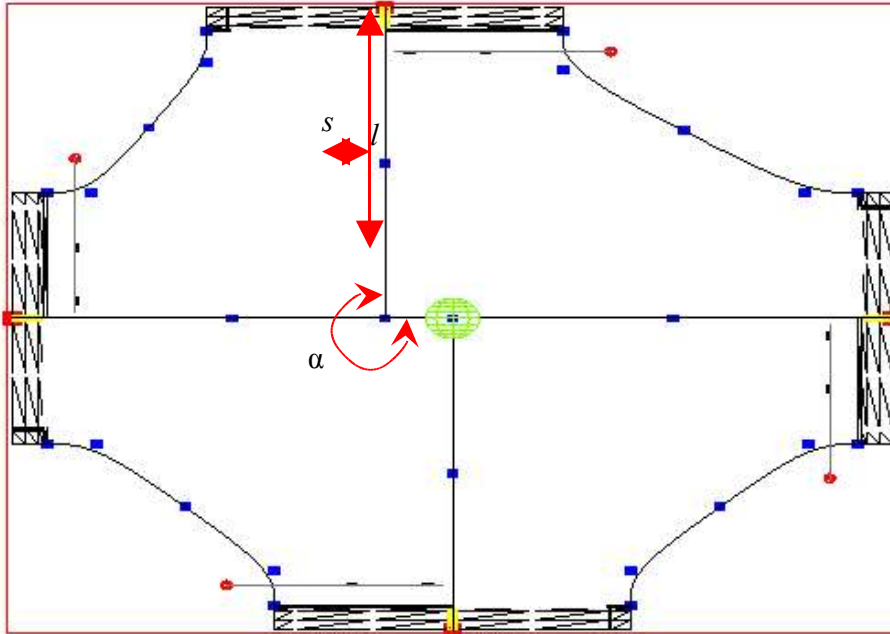
Length of Branch (l): defines the distance from the center point of the junction to the end of each branch. The default number for this length depends on the number of lanes on the adjoining branch. Cannot be changed by the user. See Figure 3.9.

Branch offset from the center of junction (s): defines the branch offset from the center of the junction. See Figure 3.9. The distance between two parallel lines that are both perpendicular to the edge of the branch, one drawn from the center of the intersection to the edge of the branch, the other drawn from the axes of the median of the branch (not from the axis of the branch since a branch can have different numbers of lanes in each direction). It is not an input.

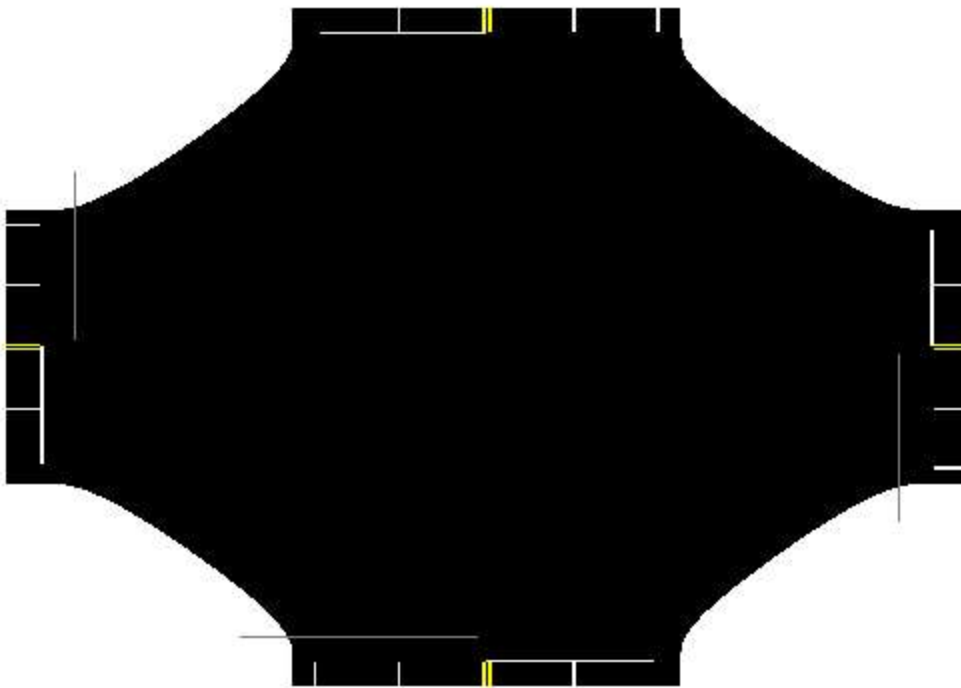
Orientation of the branch (α): The orientation angle of a branch is referenced to +x axis and clockwise positive. Not an input, although can be changed by the user. See in



Junction property dialog



Definition of l and s



A 4-way junction with default parameter number

a. Components that can be changed

Roadway on the branches: please refer to Roadway section for properties and descriptions. When the roadway is assigned, the branches will show the street name sign at the traffic lights.

Traffic Control Device: Three types of traffic control devices are available. There are “traffic light post”, “overhead traffic light”, and “stop sign”.

A Traffic light has following design parameters:

Type: defines the type of the traffic light, Post / Overhead. The default type is “Overhead”.

Height: defines the height of the traffic light. The default height is 15ft.

Number of Panels: defines the number of panels the traffic light has, only works with overhead traffic light. This number is calculated internally by the program based upon the number of lanes on the associated roadway. Therefore the user cannot change it.

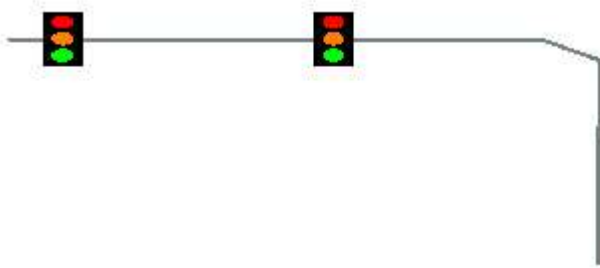


Fig. 3.11. An example of overhead traffic light

Due to the technical issues in synchronizing Paramics traffic control devices with SWEditor’s traffic control devices in animation run-time, traffic light phase is not animated in our Paramics off-line animations.

Supplemental Objects

Supplemental objects make the simulated world look more realistic by populating it with objects on the roadside. These objects include: buildings, barrier, road sign, bus sign

Building

Rectangles are used to represent buildings in the SWEditor. Parameters used to define a building are: Type, Front Texture, Side Texture, Length, Width, and Height. There are only two types of building: Building and Gas Station. Texture is a picture that makes the simple rectangles look like buildings or gas stations. Building type has front and side textures. Gas Station has only front textures. If Gas Station is chosen as Type, Front Texture field will display a list of gas stations.

Type: two types of buildings, one is Building and another one is Gas Station.

Front Texture: the selected texture is pasted on the front surface of the rectangle.

Side Texture: the selected texture is pasted on the side and back surfaces of the rectangle.

Length: defines the length of the rectangles. The default Length is 30 ft.

Width: defines the width of the rectangles. The default Width is 20 ft.

Height: defines the height of the rectangles. The default Height is 40 ft.



Arco gas station



Building property dialog

Barrier

Barrier can be created separately from roadway. In this case, Barrier is not a property of the roadway, but a separate object. It can be placed anywhere and in any direction, for example, even across a roadway to close that roadway. The size and dimensions of the barrier can be defined by the user by determining the top and bottom width and the height of the barrier. This way, the user can create a tall wall or can use this object to create a sidewalk. This object's parameters are: texture, top width, bottom width, and height. See **Figure **** for the property dialog box.

Texture: the selected texture is pasted on the surfaces to show the material properties.

Top Width: Top width of barrier. The default number is 2.

Bottom Width: Bottom width of barrier. The default number is 4.

Height: Height of barrier. The default number is 3.5.

Road Sign

There are two kinds of sign in Road Sign object: Traffic Sign and Street Sign. Traffic sign works with textures that the user can select from image library (however, user cannot change the text on the image found in the library). Street sign works with user-input text. The only common parameter is height.

Type: two types, traffic sign and street sign.

Street Sign Name: If Type is street sign, then the user can type in the name of the street and that will be shown on the sign.

Traffic Sign: if Type is Traffic sign, the user can select from texture to be displayed.

Height: the height of the sign. The default number is 6 ft.

Location and orientation determines where the sign is.

Bus Stop and Bus stop sign

Bus stop is the object specially designed for BRT system. Bus stop can be a simple bus stop sign post or a bus shelter. There are two kinds of shelters, available in two sizes each. The sign post and the shelters can be used together or separately. The following properties define this object:

Type: Total of five choices available: bus pole and two types of bus shelter with two sizes each.

Sign and Schedule Texture: The user can select the texture to be shown on the selected object type.

Logo Texture: The selected texture only applies to shelters.

Height: This is the height of the pole.

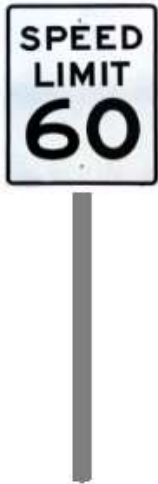


Fig. 3.13.a . A speed limit sign.

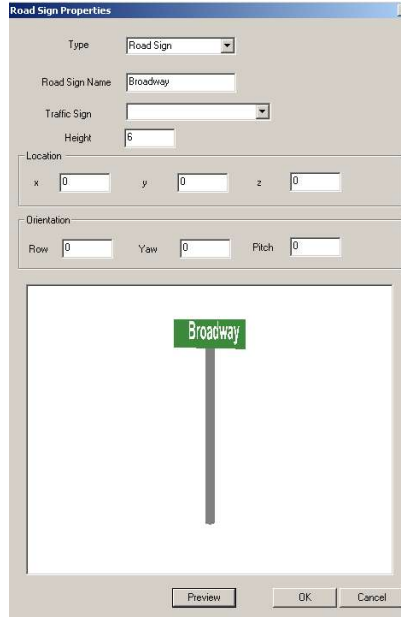
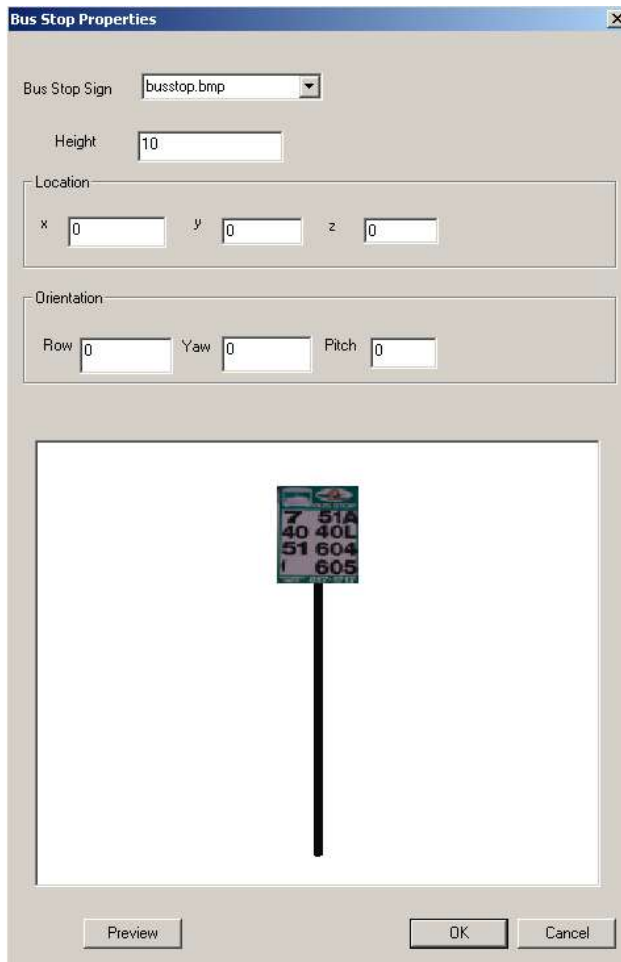


Fig. 3.13.b. Road Sign property dialog



Bus stop property dialog



Bus Shelter Type 1 with length of 20ft.



Bus Shelter Type 2 with length of 20ft.

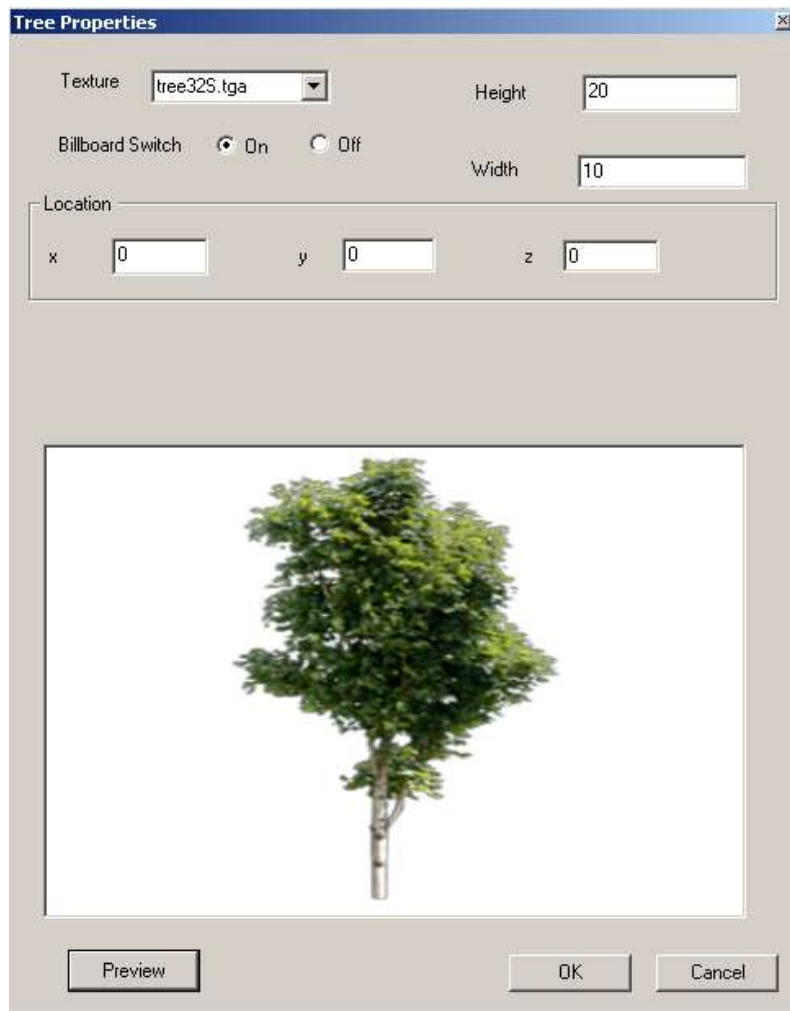
Trees

Tree is another supplemental object that makes the simulated world more realistic. The current properties of a tree object are texture, height, and width.

Texture: use to create different types of trees.

Height: defines the height of the tree. The default number is 20 feet.

Width: defines the width of the tree. The default number is 10 feet.



Tree property dialog

Billboard switch on/off plays a role in how the tree object is visualized in side and front view. To see the best picture of the tree in both views, click on. If Off is set, the image can be seen only in the front view. In side view only the surrounding rectangle will show. This switch does not effect the image in perspective view.

3DS Models

In addition to the already available library of images, 3DS models can be used to enrich the simulated world. The 3DS format is a popular format used for 3d modeling. Many models are freely others commercially available in the market. SWEditor can only load the 3DS models that are version 3.0 or later

Once downloaded, these 3DS models can be resized to desirable size. The parameters used for resizing can be height, width, or length. Users can input a desirable number to

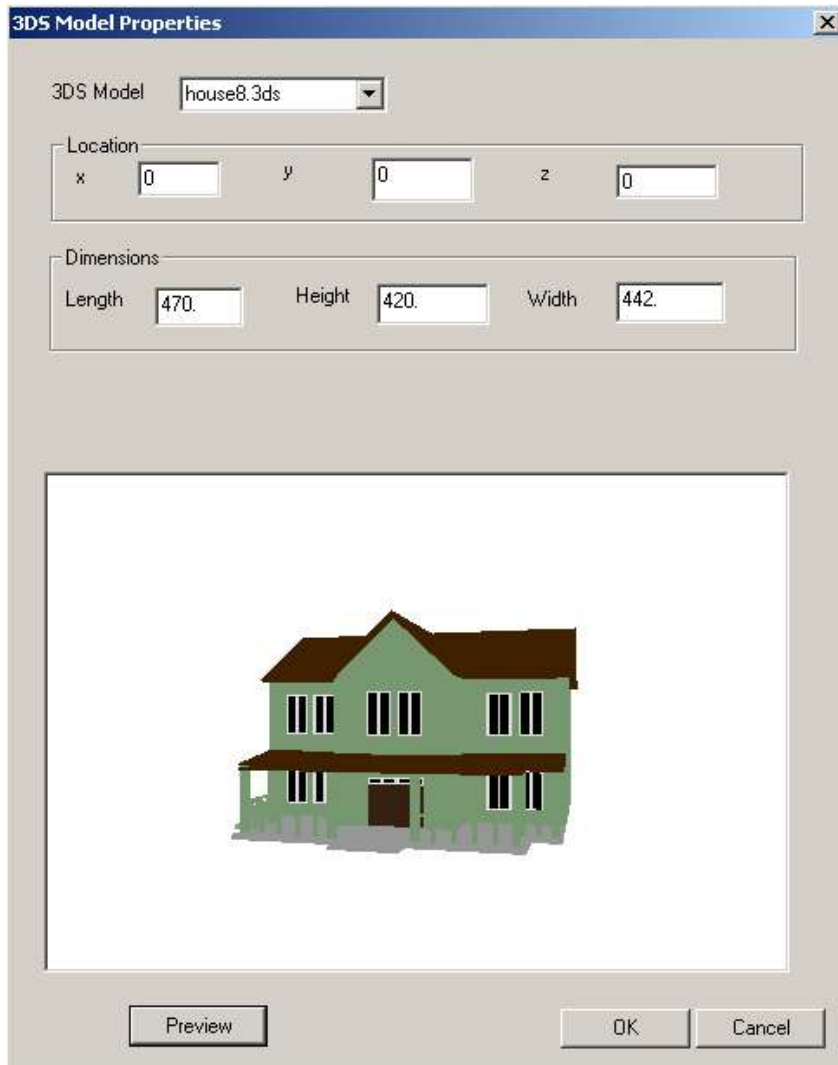
one of this parameters and the corresponding scale value to the original size will be calculated and other two parameters will be changed accordingly

Height: Desirable height of the model.

Width: Desirable width of the model.

Length: Desirable Length of the model.

Note: only one of these three can be an input by the user.



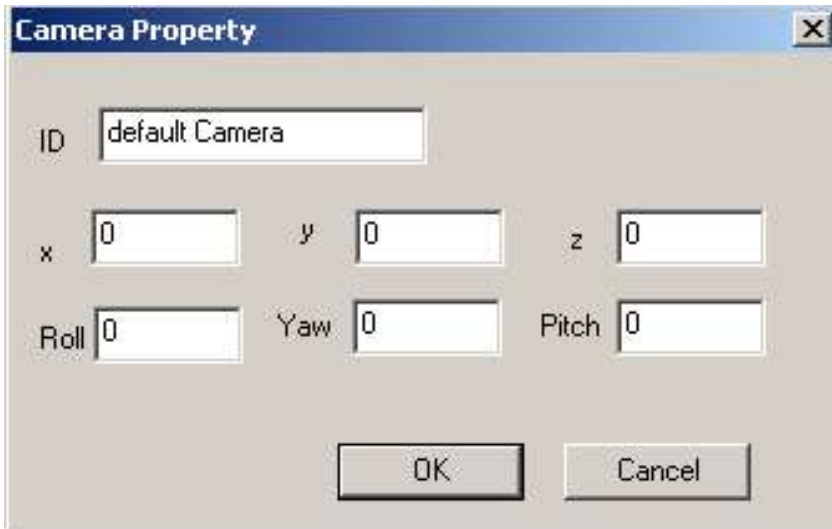
A building's 3DS model.

Camera

Camera is a special supplemental object. Its purpose is not to beautify the scene; instead, it provides additional view point in addition to the default scenes during simulations/animations. User can specify and place numerous cameras at any location

with any orientation to view the animation. Cameras can be moved, but only before starting the animation run. These cameras provide opportunity to view the animation from different locations. Also, the user can jump from one camera location to another, easily and quickly changing view points. A camera object has the following properties:

ID: Identity of the camera. This helps the users to identify/choose the desired camera
Location and orientation determine the position of the camera.



Camera property toolbar

Chapter 5. Modeling Methods

In this chapter, we will discuss the basic modeling methods used in the SWEditor. To accomplish the modeling tasks, the SWEditor has included viewing windows, mouse interactions and a number of features. A simulated world can be created by two ways. One method is the manual method that users edit the networks and scenes manually. Another method is to write down the network descriptions in certain formats and load the descriptions into the SWEditor to generate the scene automatically.

4.1 Manual Modeling Method

4.1.1 Viewing display setting

To begin your editing, the first thing is to set the desired viewing window. The SWEditor provides front, side, top, and perspective viewing windows. Front, side, and top windows are editing windows that handle process the editing work. The perspective view is only used to show the edited database at first person view. To choose a viewing window, simply press on the corresponding icon on viewing windows toolbar and the windows will be changed immediately.

The camera icon contains the list of camera objects created in the editing database. The change on camera selection does not change the viewing windows but the location of the view point.

4.1.2 Editing Modes

There are three editing modes, Surf, Edit, and Modify mode.

Surf Mode: Allows the users to move around the edited database (the animated environment) with the mouse. However, it only works for top, side, and front windows. To move the animated environment, press the left mouse button and hold it, then move the mouse around. The view-point will be updated according to the mouse movement. If the objects on the plain cannot be seen from the perspective view point, in this mode the animated world can be moved so that the objects become visible from this view point. This mode also allows the change on grid planes and will be discussed in section 4.1.3.

Edit Mode: Allows the users to create new objects.

Modify Mode: Allows the users to modify and move existing objects.

4.1.3 Grid planes

Grid plane provides reference location and scale in the animated world (database). To show grid planes, click on an icon associates with the desired grid plane from Grid toolbar. (Please refer to section 2.1.2) However, the grid planes do not show up on texture mode.

a. Change Grid Plane properties

The properties of grid planes can be changed through a dialog box. To bring up the dialog box, first set the editing mode to Surf, then double left clicks at the center (the little blue box) of the grid plane, the dialog will pop up. The properties of grid planes can be modified now. Click on OK button to confirm the changes.

Alternatively, one can click on the “Show current grid properties” icon to access this dialog box.

b. Move grid planes with mouse

The grid planes can be moved with the mouse. To do so, the editing mode must be Surf and press the left mouse button at the center of the grid planes (no need to hold it down). The grid planes now are following your mouse movement. To deactivate this feature, press the right mouse button once where you want to place the center of the grid.

Moving the grid can change what can be seen in the perspective view. While in top, side and front you can move the entire view screen to navigate around (in Surf mode), in perspective view that is not possible. But moving the grid will move the position of the perspective view point with it, thus one can place it so that the desired objects are within the view point’s vision.

4.1.4 Rendering Mode

The last thing to do before editing is to set the object-drawing mode. The rendering mode decides the presentation of the animated world. There are simplified wire-frame, wire-frame, polygon, and texture.

Simplified wire-frame mode: Render objects in simple wire-frame

Wire-frame mode: Render objects in wire-frame with more complexity (see through)

Polygon mode: Render objects in polygon (not see through)

Texture mode: Render objects with texture, if applies.

4.1.5 Object Type

Before creating or modifying or selecting an object, the desired object type has to be selected. Please refer to Chapter 3 for object types.

4.1.6 Create a new object

To create a new object, one of intersection, building, bus stop, sign post, or tree, the editing mode must be Edit and the desired object type has to be selected on the object toolbar. To define the object properties, press the new object property icon (2.1.4a), an object dialog will pop up and show the new object properties. The properties now can be changed.

After the desired properties have been set, click OK – the dialog box closes. If the object that you created is one of intersection, building, bus stop, sign post, or tree, move the mouse cursor to a desired location and press the left button to place the object. The clicked location will be the center location of the newly created object. In case you want to have two of the same object, left click again to create a second of the same object. Otherwise, either change to Modify mode, or select your next object type.

There are two ways to create a new roadway. Both start by selecting Roadway object, Edit mode, new object. The new object button will bring up the property box of the roadway. There are two methods to create a roadway: control point or engineering parameters.

If the desired edit method is set to control points, set the desired parameters, but not the engineering parameter, click OK, then left clicks on the view window to create control points for the new roadway object. At least three control points are needed to create a new roadway object but you can have as many as desired. Once done with control point creation, press the right mouse button once, a new roadway object will be created based upon the control points.

If the editing method is set to Engineering parameters, set all parameters, including the engineering parameters, and then just click the left mouse button on the view screen once to create/place the object. Instead of being the center location of the new created roadway, the clicked location is the left end of the roadway object.

If you have: engineering parameters – cannot modify the shape of the road by the mouse.

If you have: control point – you can change the shape by dragging the points

If you switch to control points and change the shape of the curve, the engineering parameters will not get updated. If you accidentally change mode from control points to engineering parameters after you manipulated the object, you lose all modifications you made while in control point mode.

To create a barrier object, select Edit mode, select Barrier object type, select new object, the property box pops up, select desired parameters, click OK. This object can be created only through using control point. So once the dialog box closes after hitting OK, click the left mouse button on the view screen to place the first control point. Minimum three control points are needed to create a barrier object. Click right mouse button once after you placed all desired control points.

For most of the object types, a green sphere is drawn to indicate the center location of the new created object.

4.1.7 Select an existing object

To select an existing object, first set the Edit mode to Modify and the object type selection to be the object that you want to select. Then, click the left mouse button within the bounding box of the particular object, the bounding box of the object will be drawn in red lines, which means your selection has been confirmed.

4.1.8 Unselect the selected object

There are two ways to unselect the selected object: left click to any other object of the selected object class (selecting another object), or right click anywhere.

4.1.9 Delete an existing object

To delete an object, select Modify mode, select the type of object that need to be deleted, then select the particular object to be deleted and press the “delete object button” (2.1.4b). An object can be deleted regardless of its correlation to the network. If the deleted object is connected to other objects, the end of the connected objects will be free.

4.1.10 Modify an existing object

To modify the properties of the selected object, select Modify mode, select the type of object you want to modify, select the particular object to be modified, press the object property icon, a property dialog will pop up allowing modification of the object. Once the modification is done, click on OK button to confirm the change. If Cancel button is hit, the properties will not get updated.

4.1.11 Move an existing object with mouse

Most of the object types have a green sphere to show its center point while the object is selected. To move the selected object, place mouse cursor inside the sphere and click on left mouse button. Hold on the left mouse button and move your mouse, the selected object will be moved to keep its center location where the mouse cursor is.

For the object types without sphere at its center location, just click within the bounding box.

When reaching the desired location, just release the mouse button to place the object.

4.1.12 Move the control points of a Roadway/Barrier object with mouse

The control points of a roadway/Barrier object can be moved to a new location by mouse. Place the mouse cursor on the control point's bounding box and press and hold on the left mouse button. If the control point is highlighted, then it is activated and it will follow the mouse movement to modify the roadway geometrical properties. Once the selected control point is at the desired location, release the left button.

4.1.13 Change the properties of a lane in roadway with mouse

Place the mouse cursor within the desired lane and double click the left mouse button. The lane property dialog will pop up. (This is the only way to get to this dialog box.) Select the desired description. Click on OK button to confirm the changes. A single left click will highlight the selected lane when the drawing mode is on wire-frame and polygon modes.

Note that while it looks as if the lane width were an input, it is actually not. Even if the number is changed, there will be no change in the visualization of the roadway. Lane width is defined on the Roadway dialog box. Furthermore, all lanes must have the same width.

Modify Intersection

First, you have to create a default intersection. Select Edit mode, object type intersection, new object properties, input desired parameters, click OK, left click on view screen where

you want the intersection. Note that at this time, the intersection can only be created so that it has the same number of lanes in all branches in both directions.

Move junction branches with mouse

To move a branch of a junction, place the mouse cursor at the end point of the desired branch (over the little red box at the middle). Press and hold on the left mouse button, a blue arrow will show at the end point and the branch location will follow the cursor's movement. Note that only the orientation of the branch can be changed this way, the length remains the same.

Changing the number of lanes on an intersection's branches

To change the number of lane on the branches of an intersection, you have to proceed branch by branch. Double left click on the end point of the desired branch (over the little red box at the middle). The Roadway properties dialog box will pop up. Changing the configuration of the branch is now the same procedure as it is to modify the configuration of a roadway.

4.1.15 Move traffic control devices of a junction with mouse

To move traffic control devices of a junction, place the mouse cursor at the bounding sphere of the particular traffic control device, press and hold on the left mouse button and move the mouse, the selected traffic control device will be relocated to where the mouse cursor is located. Let go of the button to place the traffic control device to its new location.

4.1.16 Change the properties of a traffic control device

To change the properties of a particular traffic control device double clicks on left mouse button within the device's bounding box; the traffic control device dialog will pop up.

4.1.17 Connect two roadways with mouse

To connect two roadways, select object type roadway, select one of the end control points (little red box) of roadway A and then select the end control points of roadway B that you wish to connect. Blue arrows will be shown on the selected end control points if the selections are detected. Press the Connect button (2.1.7a) to execute the connection.

If both of the roadways are not connected to other objects, the first selected roadway will be relocated to match the geometry of the second selected roadway while doing the connection. If one of the roadways is connected to other objects and another one is free of connection, then the free-connected roadway will be relocated. If both of the selected roadways are connected to other objects, then the connection is invalid.

In other words, in order for a roadway-roadway connection to be valid, one of the two selected roadways must not be connected to other objects.

4.1.18 Connect roadway-junction with mouse

Select the end point of a branch of a junction (when the object type selection is junction) and select an end point of a roadway (when the object type selection is Roadway). Press the Connection button for the connection.

When the roadway has not connection with other objects, the roadway will be relocated to match with the branch. Otherwise, if the roadway has connection with other object and the junction does not have connections, then the junction will be relocated. If both of the objects have connections, then the connection is invalid.

When the number of lanes of the roadway and the number of lanes on the corresponding branch of an intersection do not match the program connects them regardless of the mismatch. The user must be careful!

4.1.19 Connect Junction-Junction with mouse

Direct connection between two junctions is not allowed. To connect two junctions, a piece of roadway has to go in between.

4.1.20 Insect a roadway between roadway-roadway and roadway-junctions with mouse

This function is necessary because you cannot connect two objects that are already connected on their other sides. In this case, you need to insert a roadway in between. The software automatically matches the properties of the roadway by the inserted roadway.

To insert a roadway between two objects (roadway-roadway, roadway-junction), select the end points of the objects that the inserted roadway connects to, and press the Insert icon (2.1.7c), the inserted roadway will be created and connects to the selected end points.

Manipulating connected objects

Even though objects are connected, they can still be changed just like when they are not connected. For example, one can add a right turning lane to the roadway and to the connecting intersection even after the two object had been connected.

If one end of a roadway is connected to another object, the other still free control points can still be moved. Even if both ends are connected the control points in between the two ends can still be moved.

If a junction is connected by one branch to a roadway, that connected branch is fixed. The other free branches can still be moved.

When two objects are connected together they cannot be moved around as a unit, their location is fixed.

4.1.21 Disconnect a connected object from the network

To disconnect a roadway from the network, the other end of the roadway must be free. If both ends of the roadway are connected, this roadway cannot be disconnected unless one end of the roadway is free.

If intersection only has one branch connected to the network, then the junction can disconnect from the network, otherwise, the connected roadways to the branches of the junction must be disconnected first before the junction can be disconnected from the network.

To execute the disconnect feature, select the object and press the Disconnect icon (2.1.7b). The object will be disconnected from the network if the conditions mentioned above are fulfilled.

4.1.22 Create a new file

To create a new file, press the New (2.1.1a) icon. A pop up message will ask the users whether they want to save the current file before the new file is created.

4.1.23 Open an existing file

To open an existing file, press the Open (2.1.1b) icon and a file dialog will pop up. Then the users can browse the directories and select the files. Double (left mouse button) click on the filename and the selected file will be opened.

4.1.24 Save the edited database

To save the edited database, press the Save (2.1.1c) icon and a file dialog will pop up to let the users to key in the file name and the database will be saved under the selected directory with the key-in name. The SWEditor files are saved as rdb extension, which stands for road database.

4.1.25 Merge two exiting files

As copy and paste functions are not implemented in SWEditor, the merge function is an alternative to copy objects from one file to the other. This function is helpful, for example, if you want to copy only some particular objects (i.e. the whole transit network without any other surroundings) from an existing file to current one to change the surroundings.

To merge two existing files, one of the file has to be opened. Then press the Merge (2.1.1d) and a file dialog will pop up. Select the desired file and click on OK. The selected file will be merged with the current opened file.

However, there is an option to define what kind of object types are desired to merge from the files. The default sets to all objects. To set the merging object types, click on Options from menu bar and select “File merge object type option”. The checked object types are the object types that will be merged.

4. 2 Create simulated world from script file

In the previous section we discussed how to create the animated environment manually. The second way of creating this animated world is loading existing script files. To load an existing script files, press the Data (2.1.1e) icon. A file dialog will pop up. Select the desired script file and click on OK. The scripts will be loaded and the objects will be created automatically. Script files are normal text files in dat extension.

Script Modeling Method

One can write a new script file. In version 1.0, only roadway and junction can be generated this way.

To define an object, parse keywords are defined for each objects and the design parameters.

The object keywords are busstop, roadway, and junction. The property keywords for objects are the names of its design parameters. Note that the keyword input is case insensitive.

To describe an object, the object keyword has to be defined. Then define the properties at the following lines with property keywords. The value of the property should be at least a space gap after the keywords. Leave a line between the descriptions of two objects to separate them.

The following is the object and property keywords and values.

a. Bus stop

Object keyword: busstop

Property Keyword	Value
texture	Texture name that apply on bus stop
height	Height of the pole
x, y, z	Location
roll, pitch, yaw	Orientation

b. Roadway

Object keyword: roadway

Property Keyword	Value
id	Roadway id
physical-type	straight/ curve
road-type	one-way/ two-way
left-attr	sidewalk/barrier/grass
right-attr	sidewalk/barrier/grass
lanes-forward	Number of forward lanes
lanes-backward	Number of backward lanes
lane-width	Lane width
x,y,z	Location
roll,pitch, yaw	Orientation

c. Intersection

Object keyword: junction

Property Keyword	Value
id	Intersection id
lanes	Number of lanes on each branch

Control	Stop-sign/traffic-light-long/traffic-light-stand
x,y,z	Location
roll,pitch, yaw	Orientation
branch	Branch id Under branch keyword, the branch properties can be redefined. The definitions are exactly the same as roadway, except location and orientation could not be defined here

Please refer to Index A for an example of script file.

Texture

SWEditor uses two types of image formats, BMP and TGA. Most of the objects use BMP images as texture. Only trees and some of traffic signs use TGA. The reason for TGA images is that it provides transparency to the images.

To add a new texture for an object, you need to modify the image size to 2x, i.e 256 or 512 pixels. To change the image size and the format image editing software, such as Adobe Photoshop or Windows Paint, is required. Then, save the image as BMP format to the appropriate folders. Save the image in TGA format if there's need for transparency. Once the image is in the appropriate folder, it is ready to be used in SWEditor!

3DS Model

To add a new 3DS model to the SWEditor's image library, move the 3DS models to the Models folder and SWEditor will load the models automatically. The 3DS models must have the version greater than 3.0 in order to work in SWEditor.

Chapter 6. Navigation

Navigation is a handy feature in SWEEditor to view and inspect the simulated world. To provide a perspective view of the simulated world, a view point is predefined at (0,0,0) when a new database is created or an existing file is loaded.

There are four view points: front, side, top and perspective. Each can be moved around to navigate in the simulated world. The following table lists the controls for each view point.

Controls	Front View	Side View	Top View	Perspective View
Arrow Up	Up	Up	Forward	Forward
Arrow Down	Down	Down	Backward	Backward
Arrow Left	Left	Left	Left	Left
Arrow Right	Right	Right	Right	Right
Shift+Arrow Up	Forward	Forward	Zoom In	Up
Shift+Arrow Down	Backward	Backward	Zoom Out	Down
Control+Arrow Up	*	*	*	Turn Up
Control+Arrow Down	*	*	*	Turn Down
Control+Arrow Left	*	*	*	Turn Left
Control+Arrow Right	*	*	*	Turn Right

* not applicable

6.1 Camera View Point

As mentioned in Chapter 4, camera is the object that allows users to customize their viewing points in the virtual scenes. Once the user created the virtual world he can “install” these cameras at any desired locations with any desired orientations. Each camera object gets an identification name which can be changed by the users. Once the cameras are “installed” the user can select through which camera he views the virtual world and the animation running in it.

To view the virtual scenes from a particular camera, press Camera icon (2e) and a dialog box with camera list will pop up. Choose the desired camera and click OK. The perspective scene will be updated to show the virtual world from the selected camera’s view point. The chosen camera will not be drawn on the screen.

Chapter 7 Animation

SWEditor is not only a graphical modeling tool that lets the user create a simulated world. It is capable of visualizing vehicle movement in this simulated world.

In addition to SWEditor's modeling capabilities, discussed in previous chapters, it is capable of Off-line² animation of vehicle movement. To run an off-line animation, the vehicle trajectories from a previous and separate simulations have to be recorded in a predefined format (, which will be shown later in this section). SWEditor virtual world has to be built to match with this simulation's transportation environment. Once the trajectory data – is available, load the file to SWEditor. SWEditor automatically creates vehicles to match with the trajectories for the animation run.

The animation toolbar (discussed in Chapter 3) consists of the buttons that controls the animations. However, one feature worth mentioning here: the movie recording feature. There is a list of video format available to choose from before starting recording the animation. Press on OK to confirm the selected format and the animation will be recorded. To stop the recording, simply press the STOP button.



Video format dialog box

Before looking at the steps of running an animation, the format of trajectory data needs to explain. The trajectory files can be generated by any simulation programs as long as the format is correct. SWEditor only recognize the data format, it does not look for any software generation headers. The following is the format of trajectory file. A sample trajectory file is included in Appendix A.

Time	ID	Type	X	Y	Z	Yaw	Pitch
------	----	------	---	---	---	-----	-------

Time : in second with at least one significant digit , i.e. 4.2 sec

² It is called off-line because the animation uses prerecorded trajectory data that was recorded as an output file of a previous, separate simulation run by an appropriate software, such as Paramics. On-line animation would mean that the simulation and the animation runs simultaneously.

- ID : Identity number of the vehicle, it helps to identify the vehicle and assign vehicle types during the animations.
- Type : Vehicles are all Type 1, buses are Type 100.
- X,Y,Z : Coordinate of the vehicle at the particular time stamp. The unit is in feet.
- Yaw : Heading of the vehicle. The unit is in degrees.
- Pitch : Rotation about Z axis. Use when the vehicle is going up/down on a hill. The unit is in degrees.

The following is the steps for starting an animation.

1. Open a SWEditor database (the rdb format file)
2. Load a Trajectory file. Please make sure that the rdb file that opened in step 1 synchronize with trajectory file.
3. Press the Play button and the animation will start playing. The users can navigate around while the animation is running.
4. If video recording is desired, press the Record button and select the video format to record the animation.
5. Press the Stop button to stop the simulation. It stops the video recording as well. The Stop button is pressed automatically when the animation ends.
6. The movie file is saved in the Trajectory folder or the folder that last accessed by the users before the recording. The filename is Movie.avi.

Chapter 8 Future Work

SWEditor version 1.0 provides a set of models to create photo-realistic virtual world for animation. We are planning to enhance the features of this software in the following areas in version 2.0

- a. include more editing features, such as copy/paste, undo, etc.
- b. include more object types, such as more complex buildings
- c. include traffic phase into animation
- d. include more commercial 3D models, such as Multigen Creator's models.
- e. improve rendering algorithm
- f. improve software resource efficiency

Index A Network script file

```
roadway
  id 0
  control-point begin
    x 25178.089844
    y 0.000000
    z -14937.303711
  control-point end
    x 24376.501953
    y 0.000000
    z -14937.839844
  physical-type two-way
  lanes-forward 2
  lanes-backward 2
  midwidth 8.000000
  lane-width 3.700000
  connection start
    roadway 1
    point begin
  connection end
    roadway 3
    point begin
```

```
junction
  id 0
  branches 4
  x 20687.501953
  y 0.000000
  z -12521.927734
  branch 0
    x 20688.496738
    y 0.000000
    z -12547.308246
    connector end
      roadway 30
      point begin
  branch 1
    x 20697.808106
    y 0.000000
    z -12535.947121
    connector end
      roadway 29
      point begin
  branch 2
    x 20688.232891
    y 0.000000
    z -12496.538253
    connector end
      roadway 28
      point begin
  branch 3
    x 20675.504458
    y 0.000000
    z -12499.539788
    connector end
      roadway 27
      point begin
```

			-2543.105713	1695.835449	0.000000	-356.000000	0.000000
Index B. Trajectory file			-2516.322266	1341.743530	0.000000	-356.000000	0.000000
25548.500000	46413a8	1	-2552.691406	1870.955322	0.000000	-356.000000	0.000000
25548.500000	4642a68	1	-2547.874512	1807.271362	0.000000	-356.000000	0.000000
25548.500000	46425a8	1	-2543.695801	1752.028442	0.000000	-356.000000	0.000000
25548.500000	4642ba8	1	-2520.384277	1443.837158	0.000000	-356.000000	0.000000
25548.500000	463d468	1	-3274.500977	2005.880249	0.000000	-185.457733	0.000000
25548.500000	4642c68	1	-3296.204102	1854.296021	0.000000	-354.000000	0.000000
25548.500000	46423a8	1	-3272.696533	1865.987183	0.000000	-268.000000	0.000000
25548.500000	4642a28	1	-3266.303223	1866.280029	0.000000	-268.000000	0.000000
25548.500000	4640328	1	-3259.911377	1866.572876	0.000000	-268.000000	0.000000
25548.500000	4640728	1	-3253.512939	1866.710815	0.000000	90.285492	0.000000
25548.500000	4640da8	1	-3272.864502	1869.649780	0.000000	-268.000000	0.000000
25548.500000	4640c28	1	-3266.630127	1869.935303	0.000000	-268.000000	0.000000
25548.500000	46404a8	1	-3261.640137	1870.163940	0.000000	-268.000000	0.000000
25548.500000	46405a8	1	-3255.244629	1870.358154	0.000000	90.849907	0.000000
25548.500000	4640768	1	-3282.659912	1891.024902	0.000000	-185.000000	0.000000
25548.500000	46409a8	1	-3285.468994	1897.810913	0.000000	-153.584076	0.000000
25548.500000	46419a8	1	-3286.316162	1891.293823	0.000000	-185.000000	0.000000
25548.500000	4641b68	1	-3289.972900	1891.562744	0.000000	-185.000000	0.000000
25548.500000	463ec68	1	-3247.515137	1866.871826	0.000000	-268.000000	0.000000
25548.500000	463eb28	1	-3241.125244	1867.192383	0.000000	-268.000000	0.000000
25548.500000	46411a8	1	-3249.245361	1870.438843	0.000000	-268.000000	0.000000
25548.500000	46414a8	1	-3242.854248	1870.759521	0.000000	-268.000000	0.000000
25548.500000	4640a28	1					
	4641268	1					