

Transportation Library

The Transportation Library provides a collection of objects for modeling cargo transportation using trains, trucks, boats, pipes, and robots. The following is a summary of the objects in the library.

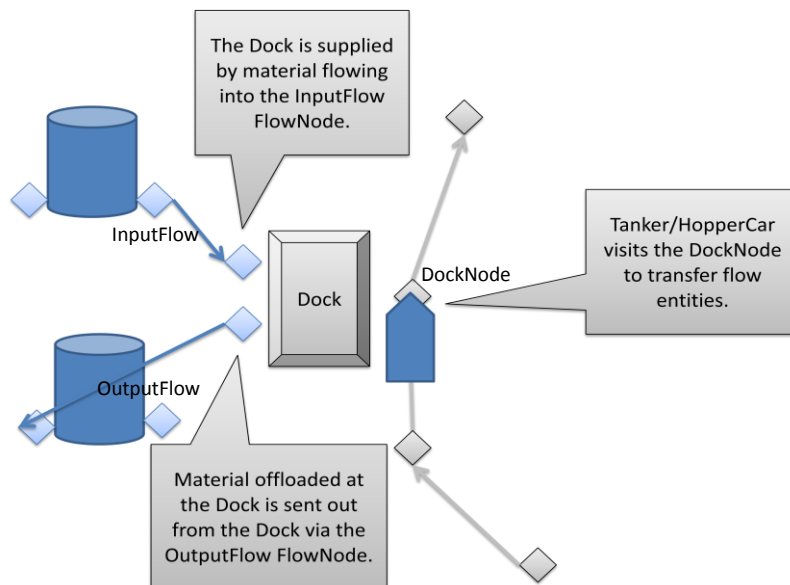
Object	Description
Tanker	The Tanker is sub-classed from Vehicle and used to transport discrete and flow entities between node locations. The Tanker has all the same functionality as Vehicle, except it has been extended to have weight and volume limits. In addition to standard discrete pickups and drop offs, the Tanker can also perform continuous flow pickups and drop-offs using a Dock object.
Dock	The Dock object provides support for Tankers and HopperCars to perform continuous flow pickups and drop offs. The Dock object has a ControlNode named <i>DockNode</i> where Tankers and HopperCars may visit a Dock object for loading and unloading continuous flow materials. It also contains FlowNodes named <i>InputFlow</i> and <i>OutputFlow</i> for sending flow materials into and out of the Dock. The Dock is seized/released by the Tanker/HopperCar during loading/unloading operations at the Dock. The Dock may also follow a Work Schedule.
Pipe	The Pipe object may be used to model continuous flow through a fully filled pipe. In contrast to the FlowConnector, the Pipe requires time for the flow material to travel.
RailCar	A RailCar is a Vehicle that can move as part of a train over a network of Tracks and ControlNodes and can pick up and drop off discrete entities at ControlNodes. RailCars can be dynamically added to and dropped from trains at ControlNodes. A RailCar can only move when it is attached to a Locomotive.
Locomotive	The Locomotive is a powered RailCar that can pull other RailCars over a network of ControlNodes and Tracks. The first RailCar in a train must be a Locomotive; however additional Locomotives may also be attached to a train.
HopperCar	A HopperCar combines Tanker and RailCar behavior into a single object. A HopperCar can be added to and dropped from trains, and may also perform continuous flow pickups and drop-offs using a Dock object. A HopperCar can only move when it is attached to a Locomotive.
ControlNode	The ControlNode is sub-classed from TransferNode and used to model the intersection of one or more inbound/outbound Tracks.
Track	A link segment between two ControlNodes that defines a pathway over which a train may move.
RobotBase	The RobotBase is placed at any node location in the model. The RobotBase remains fixed at this node location, however it can rotate clockwise/counter clockwise to move the connect robot arms through 360 degrees of rotation.
RobotLowerArm	The RobotLowerArm is connected to the top-center of the RobotBase and can move to any up/down pitch angle relative to the RobotBase.
RobotUpperArm	The RobotUpperArm is connected to the end of the RobotLowerArm and can move through any up/down pitch angle relative to the RobotLowerArm.
RobotHand	The RobotHand is sub-classed from Transporter and is connected to the end of the RobotUpperArm and can change its up/down pitch relative to the RobotUpperArm.

Flow Transportation

Tanker, Dock, Pipe

We will begin by describing the Tanker, Dock, and Pipe objects. The Tanker object is used to model tanker trucks or ships that carry material modeled as flow; e.g. a coal truck, barge, or fuel truck. The Tanker is sub-classed from Vehicle and has all of the same behavior of the standard Vehicle, but is extended to transfer ride entities using flow transfers passing through regulators associated with the InputFlow and OutputFlow nodes that are attached to a Dock object. The Dock also contains a DockNode (type ControlNode) where a Tanker may stop and transfer flow into and out of the Dock object. The Tanker behavior is also extended to include a maximum Volume Capacity, Weight Capacity and Capacity Limit in addition to the standard limit on the number of entities; in this case flow segments. The Dock is a resource that is seized and released by the Tanker during fill/empty operations.

The following example shows a Tanker being filled or emptied at the DockNode of a Dock. When a Tanker enters the *DockNode* it executes the VisitNode Step that performs drop-offs and pickups at the *DockNode*. The rate of flow into and out of the Tanker is regulated by the *InputFlow* and *OutputFlow* FlowNodes that are associated with the Dock. The Dock is connected to two Tanks (one being filled, one being emptied) using a FlowConnector. Before flow into or out of the Dock can begin, the Tanker must first seize the Dock as a resource, and then hold the dock for the duration of the fill/empty operation.



The *Commonly Used Properties* for the Tanker are shown below:

Properties: Tanker1 (Tanker)	
<input checked="" type="checkbox"/> Show Commonly Used Properties Only	
Transport Logic	
Initial Ride Capacity	1
Task Selection Strategy	First In Queue
+ Load Time	0.0
+ Unload Time	0.0
Park to Load/Unload	False
Minimum Dwell Time Type	No Requirement
Capacity Limit	VolumeAndWeight
+ Volume Capacity	1
+ Weight Capacity	Infinity
Travel Logic	
+ Initial Desired Speed	2.0
Initial Network	Global
Network Turnaround Method	Exit & Re-enter
Routing Logic	
Initial Priority	1.0
Initial Node (Home)	
Routing Type	On Demand
Idle Action	Park At Node
Off Shift Action	Park At Node

Note that they are identical to Vehicle except for the addition of a *Capacity Limit*, *Volume Capacity* and *Weight Capacity*. The *Commonly Used Properties* for the Dock are shown below:

Properties: Dock1 (Dock)	
<input checked="" type="checkbox"/> Show Commonly Used Properties Only	
Process Logic	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
+ Financials	
+ Advanced Options	
Add-On Process Triggers	
OnShiftAddOnProcess	
OffShiftAddOnProcess	
Run Initialized	
Run Ending	
+ General	
+ Animation	

Note that in addition to the *Name* and *Description* the Dock properties include its *Capacity Type*, *Ranking Rule* and *Dynamic Selection Rule*. By default the Dock can only process one Tanker/HopperCar at a time and is available 24/7. The Dock may be connected to an input Tank and output Tank using a FlowConnector from the Flow Library.

With a FlowConnector the travel time for flow to move through the link is zero. The Transportation library also provides a Pipe to more accurately model the travel time of flow material from one location to another. This is particularly useful in applications where heavy liquids are transported over long distances through a pipe. The *Commonly Used Properties* for the Pipe are shown below:

Properties: Pipe1 (Pipe)	
<input type="checkbox"/> Show Commonly Used Properties Only	
Travel Logic	
Entry Ranking Rule	First In First Out
Fill Type	Full
Flow Type	Continuous
Routing Logic	
Selection Weight	1.0
Add-On Process Triggers	
Run Initialized	
Run Ending	
New Inflow Entering	

The *Fill Type* property can be specified as “Full” or “Partial”. In the case of “Partial” the *Initial Desired Flow Speed* for the pipe is also specified. The Desired Speed state variable on the Pipe can be reassigned to change the flow speed through the Pipe. The *Flow Type* property, which is available if the *Fill Type* is “Full”, controls how the material flows through the Pipe. The default is *Push from Start* and causes the Pipe to be fully filled as each flow segment is pushed along the Pipe by its trailing flow segment. The travel rate through the Pipe is adjusted to ensure that the Pipe is fully filled from the start based on the current flow rate into the Pipe. The “Continuous” option adjusts that travel rate of each flow segment so that it fully fills the Pipe on entering, however if the input flow is cut off, the segments inside the Pipe can continue moving without requiring a push from a trailing segment. Note: The current Pipe does not work when used to connect a Tank to a Dock for pickup by a Tanker. For this case, use a FlowConnector.

Train Transportation

RailCar, Locomotive, HopperCar, ControlNode

Let’s now turn our attention to the objects that are used for modeling trains. These objects are five sub-classed objects; a **RailCar** (Vehicle), **Locomotive** (RailCar), **HopperCar** (RailCar), **Track** (Path), and **Control Node** (Transfer Node). These objects add logic to their base class to add “train-like” behavior.

Train movements are modeled by defining a track network comprised of ControlNodes and Tracks, over which trains move. A train is a linked sequence of RailCars, where the first RailCar in the sequence is a Locomotive. The RailCar can represent a wide range of train cars including freight cars, cabooses, passenger cars, etc. In addition, a HopperCar is a special RailCar that is provided to support fill/empty

operations at a Dock. The HopperCar is a RailCar with the added functionality of a Tanker. The RailCars can be linked together in any order; for example a train might contain two or more Locomotives.

The first RailCar on the train is the lead Locomotive and controls the speed and direction of the train. Although a train may contain multiple Locomotives only the lead Locomotive can change the speed or direction of the train. All other RailCars in the train always follow the lead Locomotive. Whenever any RailCar in the train stops, it notifies the lead Locomotive which in turn stops the entire train. The *Commonly Used Properties* for the RailCar are shown below:

Properties: RailCar1 (RailCar)	
<input type="checkbox"/> Show Commonly Used Properties Only	
[-] Transport Logic	
Initial Ride Capacity	1
+ Load Time	0.0
+ Unload Time	0.0
Minimum Dwell Time...	No Requirement
+ Connect Time	0.0
+ Disconnect Time	0.0
[-] Travel Logic	
Initial Network	Global
Network Turnaround ...	Exit & Re-enter
[-] Routing Logic	
Initial Priority	1.0
Initial Node (Home)	
Initial Pickup Locomot...	null
Initial Drop Destination	null

A RailCar has reference pointers (object reference state variables) to its predecessor and successor RailCars, as well as a pointer to the lead Locomotive to which it is assigned (if any). Since Locomotives are sub-classed from RailCar they may also be added at other locations in the train.

When the last RailCar of a train departs from the Control Node it automatically looks to attach the next eligible RailCar that is parked at that Control Node. A RailCar is eligible for connecting to the train if its *PickupLocomotive* (an object reference state variable) is assigned to the lead Locomotive of the train. Hence, the first waiting RailCar in the parking area of the Control Node that references the lead Locomotive is removed from the parking area and is appended to the end of the train. The train is paused for a specified RailCar *Connect Time* (a property on the RailCar). Note that each newly appended RailCar that is added to the train will pull onto the outbound link and eventually depart the Control Node and may in turn pick up a new RailCar as it exits from that node. Also note that a RailCar that is assigned to the lead Locomotive will always wait to be picked up by the last RailCar in the train that is being pulled by its assigned Locomotive. The *InitialPickupLocomotive* property is used to assign the starting pickup Locomotive for a RailCar.

RailCars may also be dropped from a train into the parking area of a Control Node. Each RailCar has a *DropDestination* node that may be set to a Control Node where it is to be dropped. When the RailCar enters its *DropDestination* Control Node it is dropped from the train and parked in the parking area of

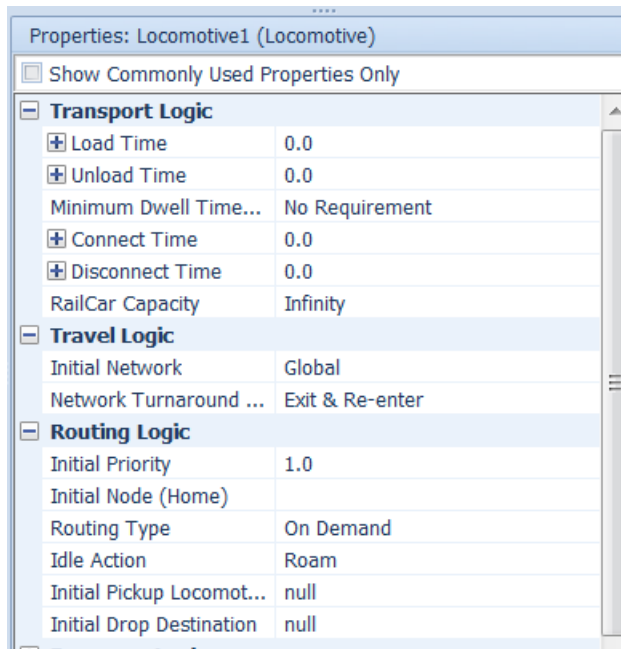
the Control Node. The train is stopped during the specified *Disconnect Time*, which is a property that may be specified on the RailCar. In many cases the dropped RailCar will be the last RailCar and no additional action is required. However, if the dropped RailCar is not the last RailCar, the trailing RailCars move forward and reconnect to the forward RailCar before the lead RailCar continues moving. If a RailCar has no specified *DropDestination* it will continue to follow its lead Locomotive through the network and will never be dropped from the train. Note that *InitialDropDestination* property on the RailCar may be specified to assign the initial drop destination for the RailCar.

The RailCar is sub-classed from Vehicle and has a default (but changeable) initial population of 1. Note that by assigning the initial population a single RailCar instance can be used to create any number of RailCars assigned to a lead Locomotive and parked at a specified home Control Node.

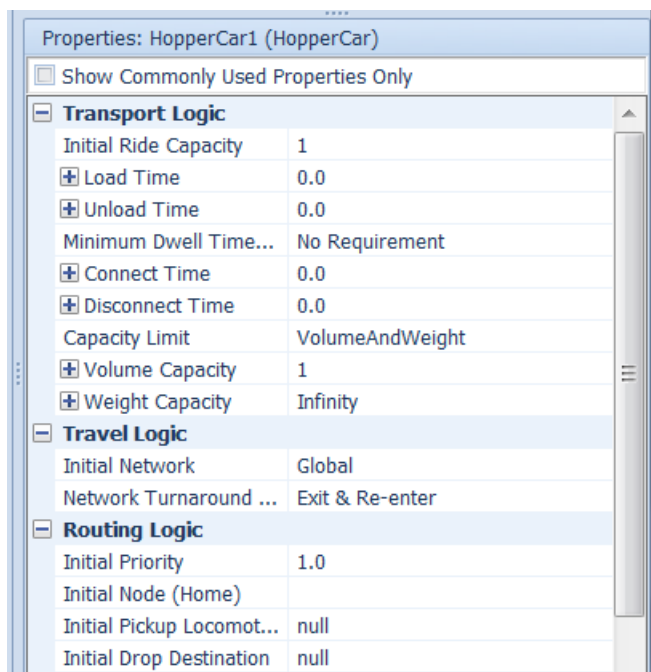
In addition to connecting and disconnecting to a train, a RailCar may also pickup and drop-off entities like any other transporter; however a Locomotive rejects all pickup/reservation requests. If a RailCar accepts a reservation using the PlanVisit step, it never follows up executing the SelectVisit step to set its destination to that pickup node location. RailCars have no control over their travel through the network; they always follow their lead Locomotive. Note that a pickup request is made for a specific RailCar – and not for a train as a whole. The *Task Selection Strategy* and *Routing Type* properties that are inherited from Vehicle are hidden and are never used by the RailCar. A reserved pickup will eventually occur once the requested RailCar arrives to the pickup node.

Also, a RailCar may request a Locomotive to come pick it up. It does so by executing the RequestTransport process on the desired Locomotive. The Locomotive can use the “remain in place” option, and then respond to the request to go pick up and move a train car – and sit idle when it’s not being used.

The *Commonly Used Properties* for the Locomotive are similar to RailCar, except that the Initial Ride Capacity is hidden, and the *Routing Type* and *Idle Action* properties are exposed. This allows the Locomotive to freely route over the network. Note that RailCars do not have this flexibility since they must always follow their lead Locomotive.



As noted earlier the HopperCar is a RailCar with the addition of Tanker functionality to allow it to execute flow transfers at a Dock. The Hopper Car has a Capacity Limit, Volume Capacity and Weight Capacity, in addition to the standard Ride Capacity that is inherited from the Vehicle base class. The Commonly Used Properties for the HopperCar are shown below:



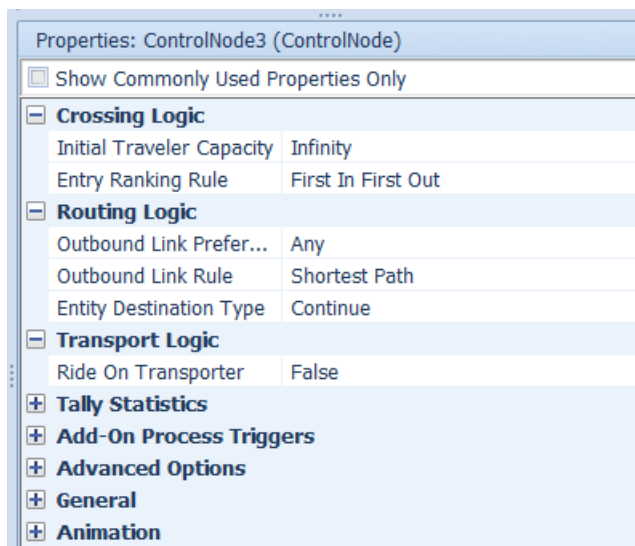
For the Tanker and HopperCar, the property *Capacity Limit* can be specified as “Volume”, “Weight”, or “VolumeAndWeight”. If specified as “Volume” then only the volume limit applies. If specified as

“Weight” only the weight limit applies. “VolumeAndWeight” cause both to be applied (which ever limit is hit first).

Note that if a HopperCar or RailCar is being detached from a Train and it has to undergo a loading/unloading process, the Car will detach from the Train and the Locomotive will restart while the Cars are unloading. If the Cars are not being dropped from the Train then loading/unloading will take place on the track while attached to the train, and the train will be stopped during this operation.

Also, if a detached Car is in the middle of the Train, the Car will be parked, allowing the trailing cars to be rejoined to the train.

Now let’s turn our attention to the track network over which the train travels. The ControlNode is sub classed from the TransferNode and is used to model track intersections where routing decisions may be made. The *Commonly Used Properties* for the ControlNode matches those for the TransferNode, however the internal process logic is changed to properly route RailCars to follow their leader. Note that the *Ride On Transporter* property is used by Entities requesting a pickup by a RailCar.



The Track is sub-classed from Path and is used to model individual track sections. The *Commonly Used Properties* for the Track matches those for the Path and are shown below:

Properties: Track1 (Track)

☐ Show Commonly Used Properties Only

Travel Logic

Type	Unidirectional
Initial Traveler Capacity	Infinity
Entry Ranking Rule	First In First Out
Drawn To Scale	True
<input checked="" type="checkbox"/> Speed Limit	Infinity

Routing Logic

Selection Weight	1.0
------------------	-----

☒ **State Assignments**

☒ **Add-On Process Triggers**

☒ **Advanced Options**

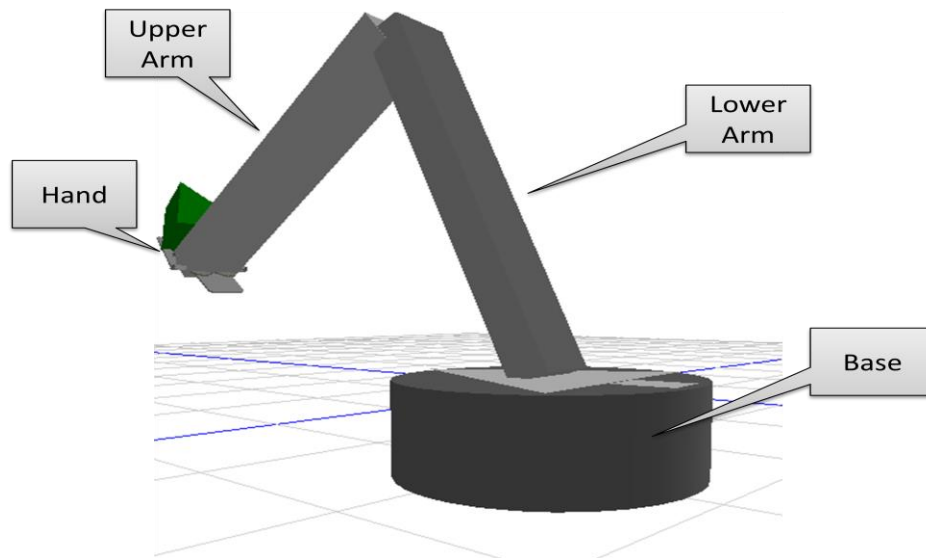
☒ **General**

Note that the track path decorator is typically assigned to each Track link, and Track links are typically assigned to a separate rail network. This network is then assigned as the *Initial Network* for all RailCars, Locomotives, and HopperCars that run on the tracks.

Robots

RobotBase, RobotLowerArm, RobotUpperArm, RobotHand

The last group of four objects in the Transportation library is used to model a robot. These objects are based on an earlier implementation by Mosimtec. The four objects are used to model the RobotBase, RobotLowerArm, RobotUpperArm and the RobotHand, as depicted below:



When using the robot library, the robot hand is the object that is the transporter that actually picks and drops off entities, and hence is specified as the Transporter at a pick node.

The RobotBase object (sub-classed from Entity) models the base of the robot that is located at a specified node and rotates to move the lower/upper arm of the robot in a clockwise/counter-clockwise direction. The base references its associated components that include the lower arm, upper arm, and hand. The base also specifies the servo motor speeds (in degrees per second) that control the base rotation rate and upper arm, lower arm, and hand pitch rates. The *Commonly Used Properties* for the RobotBase object are shown below:

Properties: RobotBase1 (RobotBase)	
<input checked="" type="checkbox"/> Show Commonly Used Properties Only	
Components	
Base Node Location	BasicNode1
Lower Arm	RobotLowerArm1
Upper Arm	RobotUpperArm1
Hand	RobotHand1
Servo Speeds (Degrees Per Second)	
Base Rotation Rate	1
Lower Arm Pitch Rate	.3
Upper Arm Pitch Rate	.5
Hand Pitch Rate	1
General	
Name	RobotBase1
Description	
Animation	
Degrees Rotation Per Update	10

The RobotLowerArm and RobotUpperArm objects are sub-classed from Entity and are used to model the upper/lower arms of the robot. The lower arm is connected to the top center of the base, and the upper arm is connected to the top of the lower arm. The arms rotate with the base, and change pitch angle to position the hand at the desired location. There are no properties required for these objects.

The RobotHand object is attached to the end of the upper arm, and is used to pickup and drop-off entities from node locations. The RobotHand is sub-classed from Transporter and adds additional properties to specify the loading/unloading/travel pitch angle for the hand. Before executing a pickup, drop-off, or base rotation the hand will first rotate to the specified pitch angle. The RobotHand also has a *Travel Mode* that can be specified as *Fixed Height* or *Direct*. In the case of *Fixed Height* the RobotHand first moves to a specified *Travel Height* before the base begins to rotate. All base rotation is done at this fixed height. In the case of *Direct*, all servo motors are allowed to move simultaneously and there is no restriction on the travel height. The Commonly Used Properties for the RobotHand are shown below:

Properties: RobotHand1 (RobotHand)	
<input checked="" type="checkbox"/> Show Commonly Used Properties Only	
Travel Logic	
Travel Mode	FixedHeight
Travel Height	1
Travel Pitch Angle	0.0
Routing Logic	
Initial Node (Home)	BasicNode1
General	
Name	RobotHand1
Description	

To create a Robot, simply place an instance of RobotBase, RobotHand, RobotUpperArm and RobotLowerArm into your model. You should have three nodes; one where the RobotBase will be initialized, another where the entity will be picked up from and the node where the entity will get dropped off by the RobotHand. The TransferNode where the entity will be picked up at should have its *RideOnTransporter* property set to 'True' and its *TransporterName* property set to the instance of the RobotHand. The *EntityDestinationType* property should be set to the name of the node where the entity should be dropped. The RobotBase and the RobotHand both have properties that must be set to control the angle of the movements and the speed of the movements.