

# Rail API Command Documentation

---

**addedge listener**(num edge, object sendmessage to, node msg from, num msgcode, num param2, num param3)

**Description:** Adds a deallocation listener to the edge. Once added, whenever a deallocation is made on the edge, a message will be sent to the listener object. The msgsendingobject for the message will be the listener, msgparam(1) for the message will be msgcode, and msgparam(2) for the message will be the edge number that had the deallocation call made to it. Added listeners will remain as listeners until they are removed with the removeedge listener() command. For more information, see allocateedge().

**Example:**     adddedge listener(2,current,1);

**addraildelay**(node sequence, num delaytime)

**Description:** Adds a rail delay operation to a rail sequence. The next operation in the rail sequence must wait until the specified delay time has transpired. For more information, see create rail sequence().

**Example:**     addraildelay(sequence,1200);

**addrailmove**(node sequence, node railpath, num speed, num acc, num dec, num length, num deallocateid [, num coupledecoupleoffset])

**Description:** Adds a rail move operation to a rail sequence. Specify the path that the railmove applies to. Distance is measured from the first node of the edge, where the first node is the node you dragged from when 'A' connecting. Length is the total amount of space that the set of travelers will take up, or the train size. The travelers will travel the total distance of the edge(s) minus the length, meaning the length will be completely inclusive within the distance of the edge(s). Specify viaedge if you want to travel through a particular edge on the way to the destination edge. This command returns a reference to a node representing the rail move operation. For more information, refer to create rail sequence(). If deallocateid is not 0, then as the train clears edges on the network, it will deallocate those edges, based on the given id. Refer to couplerailtraveler for more information on the coupledecoupleoffset parameter.

**Example:**     addrailmove(sequence, create rail path(5, 4.2, 7, 3.4,3), 1,0,0,10, 1);

**addrailpath listener**(node path, object sendmessage to, node msg from, num msgcode, num param2, num param3, num idmask)

**Description:** Adds a listener to the specified path. Once added, as soon as the entire path becomes available (!getpathallocated(path, idmask)), a message will be sent to the specified object with the given parameters. Remember to remove the listener after you are finished.

**Example:**

**addrailsendmessage**(node sequence, node toobj, node fromobj, num code, num delaytime)

**Description:** Adds a send message operation to a rail sequence. If delaytime is nonzero, it will send a delayed message instead of a direct message. The msgsendingobject for the message will be the fromobj. For more information, see create rail sequence().

**Example:**     `addrailmessage(sequence, current, fromobj, 1, 30);`

**addrailtraveler**(node railsequence, obj traveler, num offset)

**Description:** Adds a rail traveler to a rail sequence or a rail move operation. If no objects are added to an individual rail move, then the rail sequence's object list will be used for the move operation. The offset is the distance from a user chosen zero point for a given rail move. If you have built your network by dragging connections from left to right in your model, then the zero offset point is typically the right most point of a set of travelers on the network. The offset proceeds to the left from that point. Use only positive values for the offset. For more information, see `createrailsequence()`.

**Example:**     `addrailtraveler(sequence, current, 0)`

**addrailwait**(node sequence)

**Description:** Adds a rail wait operation to a rail sequence. When this operation is performed, the rail sequence will not proceed its next operation until `continuefromrailwait()` is called on the sequence. For more information, see `createrailsequence()`

**Example:**     `addrailwait(sequence);`

**allocateedge**(num edge, num startpoint, num dist, num id)

**Description:** Allocates a subsection of the specified edge as defined by a starting point and distance along the edge with the given id as a tag for that allocation. The starting point is defined as a distance from the first `NetworkNode` of the edge, meaning the `NetworkNode` you "A"-click-dragged from when creating the edge. Please note that allocations do not affect any functionality with respect to traveling on the network, or traveling using rail sequences. It is only a mechanism for storing network edge allocation/deallocation data. Each allocation represents a specific distance along a network edge. Each edge can have any number of allocations, but the allocations cannot overlap. If a call to `allocateedge` causes an overlap, then the overlapping section will be overwritten with the new allocation, and the old allocation will be truncated or removed. Use `getedgeminallocated()` and `getedgemaxallocated()` to query if and how much of an edge is allocated. As allocations are made for a given edge, they are sorted according to distance from the first `NetworkNode` of the edge.

**Example:**     `int allocation_id = allocateedge(2,5.5,12.6);`

**allocaterailpath**(node path, num id)

**Description:** Allocates the entire rail path with the given id. This function encapsulates several `allocateedge()` calls.

**Example:**

**continuefromrailwait**(node sequence)

**Description:** Finishes a rail sequence's current wait operation. For more information, see `createrailsequence()`

**Example:**     `continuefromrailwait(sequence);`

**`couplerrailtraveler`**(node railmove, obj traveler, num offset)

**Description:** Couples a rail traveler to a given move of a sequence. The `addrailtraveler()` command is used to add the initial set of travelers to a rail sequence. However, at different times during the sequence, the train may couple/decouple certain members of the set of travelers. Use the `couplerrailtraveler()` and `decouplerrailtraveler()` commands to do this. While the `addrailtraveler` command is performed on a rail sequence, and sets the initial travelers for that sequence, `couplerrailtraveler` and `decouplerrailtraveler` should be performed on an individual rail move in that sequence. The `addrailmove` command returns a reference to that rail move, and this return value should be passed into the first parameter of `couplerrailtraveler` and `decouplerrailtraveler`. `couplerrailtraveler` will couple the traveler onto the train as part of that rail move (before the rail move starts as opposed to after the rail move finishes) with the associated offset from the head of the train. The traveler will remain a part of the train until the sequence finishes or the traveler is decoupled in a later move with the `decouplerrailtraveler` command. If you couple/decouple traveler(s) onto the "head" side of the train, then when you call `addrailmove`, you will need to pass in a non-zero value for the "coupledecoupleoffset" parameter of `addrailmove`. The reason for this is because, by the fact that you have added a new set of travelers onto the "head" side of the train, you have changed the location of the "head" of the train, and the offset locations for the other travelers on the train must be adjusted to take that into account. For example, if for a given move you add two cars onto then head of the train, and their cumulative sizes adds 30 meters to the train's size, then you should specify 30 as the `coupledecoupleoffset` parameter of `addrailmove()`. If you decouple two cars for a given move, and they decrease the train size by 30 meters then you should specify -30 as the `coupledecoupleoffset`. Once you've specified the proper `coupledecoupleoffset`, you should couple the travelers with the offset based on the new head of the train, i.e. the new front car should have an offset of 0. If you couple/decouple travelers from the tail end of the train, then you should specify 0 as the `coupledecoupleoffset` parameter. Note that the "head" of each train is based on the order in which you connect the network. For example, if you connect the rail network nodes from left to right, then the head of a train will always be on the right side.

**Example:**     `couplerrailtraveler(sequence, current, 0)`

**`createrailpath`**(num fromedge, num fromdist, num toedge, num todist, num via\_1, num via\_2, num via\_3)

**Description:** Creates a rail path. Use the return value of this command as the second parameter of the `addrailmove()` command. You can also create a rail path for explicit analysis using `getrailpath...` commands, instead of adding it to a railpath. Returns a reference to the railpath node. If you do not add the rail path to a rail move, you should destroy the rail path with the `destroyobject()` command once you are done with it. The path structure is always created in a "forward" direction based on the sequence that the network nodes were connected. This means if you create a path that travels backwards on the network, then if you want to traverse the path with `getpath...` commands in the direction of travel, you will need to traverse from the end of the path to the beginning. Via nodes are traveled to in sequence. The parameters are node indeces, as found in the `EDGETABLE_FROM` and `EDGETABLE_TO` columns of the Rail Manager's edgetable.

**Example:**     `createrailpath(4, 2.1, 5,56.7, 0,NULL);`

**`createrailsequence()`**

**Description:** Creates and returns a reference to a rail sequence. A rail sequence is a sequence of move/message/delay/wait operations that are performed on a set of traveler objects. For instance, you can create a rail sequence to move a set of travelers from one point on your model's travel network to another point. The procedure for creating a rail sequence is as follows. First, your network (composed of standard NetworkNodes) must be built in such a way that it represents a directed graph with no cycles. What this means is, you must ♦ A?connect your network nodes uniformly in one direction. If you decide to connect your nodes from left to right, then there should be no instance where you ♦ A?connect a node back to a previous node, from right to left. To help you out, each network edge will display a red line with a dot on one end. The dots should always be on the same end of the line for all edges in the network. Once you've created a travel network and want to create a rail sequence for a set of travelers, call createrailsequence() and get the return value back as an fsnode\*. Then, using the return value pointer, add to the sequence the objects (travelers) that will be associated with this rail sequence by using the addrailtraveler() command. The travelers may be any object so long as they have spatial and visual attributes. With each traveler added, you will need to specify an offset distance. This is an offset distance from an arbitrary zero point chosen for the set of travelers. If you've connected your network from left to right, the zero point for a set of travelers will typically be chosen as the far right surface of the traveler that is furthest to the right along the rail network. When adding travelers, begin with the traveler furthest to the right and work left until all travelers for the sequence have been added. Each traveler should be given an offset value equal to the total length of previously added travelers plus half its own length. Once you've added all rail travelers for the sequence, you will add operations to the rail sequence. These operations will be executed in the order that you add them. There are four different types of operations you can add to a rail sequence: 1) Rail Move Operation, 2) Rail Send Message Operation, 3) Rail Delay Operation, 4) Rail Wait Operation. A rail sequence may be made up of any number and any order of these operations. 1) Rail Move Operation: a rail move operation is added with the addrailmove() command. You must first create a rail path that is associated with the move operation, using the createrailpath() command. In creating the path, you specify a travel operation from one point on a network edge to another point on a network edge. The origin and destination points may be on the same edge or on different edges of the network. You will declare both points by entering an edge number (or name) and a distance along the edge. Entered distances are measured from the first node of the edge, meaning the node you "A" connected from when creating the edge. When choosing the origin and destination points, keep in mind that it will be the outer most surfaces of the train that will travel between the two points. This means that for a train that will travel left to right, you will specify the origin point as the point associated with the far left surface of the train, and the destination point as the point where you want the far right surface of the train to travel to. The length of the train will be inclusive in the total distance between the origin and destination points. Shortest distance algorithms are employed to define the route between the two points; however you may also specify a "via edge" to have further control of the actual route taken. The addrailmove() command returns a reference to the move operation allowing you to add travelers to this move operation should the set of travelers for this move operation be different than the set of travelers specified for the whole sequence. 2) Rail Send Message Operation: add a rail send message operation with the addrailsendmessage() command. This causes the rail sequence to send a message to a specified object with a specified message code. The msgsendingobject of the message will be the sequence node. 3) Rail Delay Operation: add a rail delay operation with the addraildelay() command. This causes the rail sequence to pause for a specified length of time before continuing to the next operation in the sequence. 4) Rail Wait Operation: add a rail wait operation with the addrailwait() command. This causes the rail sequence to pause for an indefinite length of

time before continuing to the next operation. The rail sequence will not continue until the `continuefromrailwait()` command is executed independent of the rail sequence. 5) Once you have added all the elements that are needed for the rail sequence, call `startrailsequence()` to start the train moving. You can wait any amount of time before calling `startrailsequence()` and the train will not move along the rail sequence until `startrailsequence()` is called.

**Example:** `fsnode * sequence = createrailsequence(); addrailtraveler(sequence,trav1,xsize(trav1)/2);  
addrailtraveler(sequence,trav2,xsize(trav1) + xsize(trav2)/2);  
addrailtraveler(sequence,trav3,xsize(trav1) + xsize(trav2) + xsize(trav3)/2);  
addrailmove(sequence, 1,5, 4,7, 1,.5,.5, xsize(trav1) + xsize(trav2) + xsize(trav3));  
addraildelay(sequence,1200); fsnode *moveop2 = addrailmove(sequence, 4,7,6,8,  
1,.5,.5, xsize(trav1)); addrailtraveler(moveop2,trav1,xsize(trav1)/2);`

**deallocateedge**(num edge, num id)

**Description:** Deallocates the allocation on the specified edge with the specified id. For more information, see `allocateedge()`.

**Example:** `deallocateedge(2,allocation_id);`

**deallocateedgebypoint**(num edge, num point)

**Description:** Deallocates the edge's allocation that includes the point defined by the specified distance along the edge. For more information, see `allocateedge()`.

**Example:** `deallocateedgebypoint(2,26.4);`

**deallocateedgebyrank**(num edge, num rank)

**Description:** Deallocates the edge's allocation with the specified rank. For more information, see `allocateedge()`.

**Example:** `deallocateedgebyrank(2,1);`

**deallocateedgerange**(num edge, num startpoint, num dist, num id)

**Description:** Deallocates the range of the specified edge. If `id = 0`, then it will deallocate/truncate all allocations within that range, no matter what id they are allocated as. If `id > 0`, then it will only deallocate allocations that match the given id.

**Example:**

**deallocatetrailpath**(node path, num excludedist, num id)

**Description:** Deallocates all edges on the rail path. The `excludedist` will be excluded from the deallocation. This is so you can leave allocated the length of track that the train finally rests on after a rail move operation. If the path is a forward path, then the `excludedist` will be applied to the end of the path. If it is a backwards path, the `excludedist` will be applied to the start of the path. If `id` is 0, then it will deallocate the path no matter what ids the path was allocated with. If `id > 0`, then it will only deallocate allocations that match the given id.

**Example:**

**decouplerailtraveler**(node railmove, obj traveler)

**Description:** Deouples a rail traveler to a given move of a sequence. Refer to couplerailtraveler for more information.

**Example:** decouplerailtraveler(sequence, current, 0)

**delayrailmoveatswitches**(node railmove, num delaytime, num maxspeed, num acc, num dec, num shortdist)

**Description:** Searches through the rail moves entire path and changes the move so that the train will stop short of the switch and wait for the delaytime at the switch before resuming on.

**Example:** delayrailmoveatswitches(railmove, 5,1,0,0)

**getedgeallocateddist**(num edge, num rank)

**Description:** Returns the total length of the edge's nth ranked allocation. For more information, see allocateedge().

**Example:** getedgeallocateddist(2,1);

**getedgeallocatedid**(num edge, num rank)

**Description:** Returns the id of the edge's nth ranked allocation. For more information, see allocateedge().

**Example:** getedgeallocatedid(2,1);

**getedgeallocatedstartpoint**(num edge, num rank)

**Description:** Returns the start distance of the edge's nth ranked allocation. For more information, see allocateedge().

**Example:** getedgeallocatedstartpoint(2,1);

**getedgemaxallocated**(num edge)

**Description:** Returns the maximum distance point of the edge's last ranked allocation (allocations are sorted by their location on the edge). If the edge has no allocations, -1 is returned. For more information, see allocateedge().

**Example:** getedgemaxallocated(2);

**getedgeminallocated**(num edge)

**Description:** Returns the minimum distance point of the edge's first ranked allocation (allocations are automatically sorted by their location on the edge). If the edge has no allocations, -1 is returned. For more information, see allocateedge().

**Example:** getedgeminallocated(2);

**getedgenrallocated**(num edge)

**Description:** Returns the total number of current allocations for the edge. For more information, see `allocateedge()`.

**Example:** `getedgenrallocated(2);`

**getnetnodeport**(num edge, num startend)

**Description:** Returns the output port number of the node associated with the start or end of the given edge. Pass in either `EDGE_START` or `EDGE_END` as the second parameter.

**Example:** `command36()`

**getrailedgedist**(num edge)

**Description:** Returns the total distance of the edge.

**Example:** `getrailedgedist(2);`

**getrailmovekinematics**(node railmove)

**Description:** Returns a reference to the kinematics node that drives the travel for the rail move. You can get access to this node, re-initialize the kinematics, and add your own kinematics to change the way the train moves along the rail path. Make sure you do any changes before the rail move starts execution. Any added kinematics should be applied to the x component of the kinematic, as all other components are ignored. The x location of the kinematic is associated with the 0 offset point of the train.

**Example:**

**getrailmovepath**(node railmove)

**Description:** Returns a reference to the path associated with the rail move.

**Example:**

**getrailmovetimetotravel**(node railmove, num dist)

**Description:** Returns the amount of time it will take to travel the given distance from the time the rail move operation starts.

**Example:**

**getrailmovetrainsize**(node railmove)

**Description:** Returns the stored train size of the rail move.

**Example:**

**getrailmovetraveldist**(node railmove)

**Description:** Returns the total travel distance for the rail move. If the rail move's kinematics have not

been manually changed, then this is the total distance of the path minus the size of the train.

**Example:**

**getrailpathallocated**(node railpath, num idmask)

**Description:** Searches edges along the path to find allocations that are NOT associated with the idmask. If it finds any allocation whose allocation id is not equal to idmask, it will return 1. Otherwise it will return 0.

**Example:**

**getrailpathdist**(node railpath)

**Description:** Returns the total distance of the rail path.

**Example:** double dist = getrailpathdist(railpath);

**getrailpathedge**(node railpath, int edgenum)

**Description:** Returns the nth edge of the rail path. The return value is the row number of the global edge table. You can use the return value with getedge... commands.

**Example:**

**getrailpathedgeenddist**(node railpath, num edgenum)

**Description:** Returns the distance from the start of the path to the end of the edge specified.

**Example:**

**getrailpathedgeendnode**(node railpath, num edgenum)

**Description:** Returns a reference to the node associated with the end of the path's given edge.

**Example:** command32()

**getrailpathedgefordistalong**(node railpath, num distance)

**Description:** Returns the path's edge number for the edge that contains the given distance from the start of the path.

**Example:** getrailpathedgefordistalong(my\_rail\_path, 133.23);

**getrailpathedgestartdist**(node railpath, num edgenum)

**Description:** Returns the distance from the start of the path to the start of the specified path edge.

**Example:**

**getrailpathedgestartnode**(node path, num edge)



**Description:** Returns a reference to the network node associated with the start of the path's given edge

**Example:** editExample

**getrailpathfirstdist**(node railpath)

**Description:** Returns the distance along the path's first edge that represents the start point of the path.

**Example:**

**getrailpathforward**(node railpath)

**Description:** Returns 1 if the rail path direction (from the fromedge to the toedge specified in createrailpath() or addrailmove()) is forward.

**Example:** getrailpathforward(railpath);

**getrailpathnrofedges**(node railpath)

**Description:** Returns the total number of network node edges that the rail path encompasses.

**Example:**

**removeedgelistener**(num edge, object listener, node frommsg, int msgcode, num param2, num param3)

**Description:** Removes a listener from the edge's listeners list by matching both the listener object and the msgcode. For more information, see allocateedge().

**Example:** removeedgelistener(2,current,1);

**removerailpathlistener**(node path, object sendmessaget, node msgfrom, num msgcode, num param2, num param3, num idmask)

**Description:** Removes a given rail path listener.

**Example:**

**startrailsequence**(node sequence)

**Description:** Starts the rail sequence. For more information refer to the createrailsequence command.

**Example:** startrailsequence(sequence)

**truncateedgeallocation**(num edge, num allocationrank, num newstartpoint, num newdist[, num nonotify])

**Description:** Truncates one or both sides of an allocation. This is mostly for lower level edge allocation management.

**Example:**