

Simio Crane Library

Overview

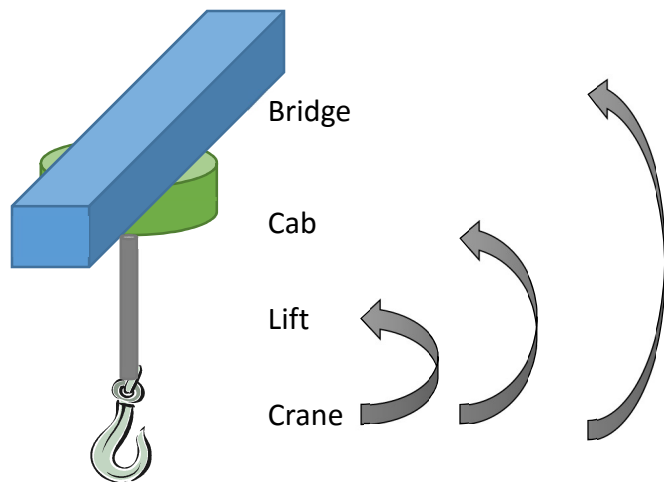
The Simio Crane Library is a collection of objects designed for modeling multiple cranes operating simultaneously in one or more bays. The library is provided as an example of complex material handling using the standard features of Simio. The library is currently not part of the standard installation or documentation of Simio, but may be downloaded for free from the Simio website. The library can be loaded and used with any edition of Simio with the exception of Simio Express. However models that are built with this library in other Simio editions can still be executed using Simio Express Edition.

The Crane library may be used in conjunction with the Simio Standard Library, and Crane pickups are done using the standard TransferNode (e.g. the output side of a Server). Crane drop-offs can be done at either a BasicNode or TransferNode. The Crane Library can also be used with custom libraries as long as they support rider pickups using the standard transporter ride features.

The Crane Library consists of objects representing the Bay, Bridge, Cab, Lift, and Crane (the end effector that actually picks up and drops off the item). These objects are combined together to model multiple cranes moving in a bay. A Crane movement occurs by first rising up from the pickup node to a specified travel height, traveling laterally at that height, and then lowering down to the specified drop-off node. All travel is done through free space without the need to explicitly draw a network. The Crane library also fully supports independent acceleration/deceleration and the ability for one crane to cause another blocking crane to move out of the way.

The key object in the library is the Crane which represents the carrier or end effector of the device and is the object that is specified as the Transporter for picking up and moving an item. The Crane in turn causes its associated Lift, Cab, and Bridge to move across the Bay.

Since the Crane, Lift, Cab, and Bridge are all separate objects their graphical appearance can be individually edited. For example the symbol for a Cab can be edited independently of the symbol for the Bridge. The Crane references its associated Lift, Cab, and Bridge as illustrated below:



Note that the Crane, Lift, Cab, and Bridge do not appear in each Bay until the model run state is initialized. At this point each Crane and Bridge is created and placed at the specified Home node, and each Crane in turn creates and positions the associated Lift and Cab Entities. Note that Bridges do not necessarily have an associated Crane, but Cranes always have an associated Bridge.

Let's now consider each of these objects in more detail.

Bay

A Bay is a fixed object that defines a rectangular region over which one or more Bridges may move. A Bay is placed in the model by dragging it from the Library and then setting its properties. A Bay has a *Traversal Direction* that can be specified as *Left and Right* or *Forward and Back* and defines the direction of movement of Bridges that are assigned to the Bay. The Bay also specifies the *Number of Zones* that are used to control Bridge movements to prevent collisions. Each Bridge in the Bay occupies its current zone, and only one Bridge may be in a zone at any one time. Before a Bridge can move to a new location it must seize its destination zone; however it is not permitted to seize its destination zone until all intermediate zones are free. During a Bridge move the Bridge holds both its starting zone and destination zone and only releases its starting zone once it has reached its destination. The busy/idle state of each zone is animated during the run.

The following depicts two Bridges moving in a Bay with a *Left and Right* movement direction and three Zones. The Bridges are currently located in Zones 1 and 3, causing each zone to be busy. Before either Bridge can move to the node in Zone 2, it must first seize Zone 2. Since only one Bridge can own a Zone at time, only one of the two Bridges may enter Zone 2 at any point in time. This automatically prevents collisions from occurring with the two Bridges.



Note that a Zone appears in the Crane Library as an object, but it is not placed by the user. Zones are automatically created for each Bay based on the *Number of Zones* specified by the user.

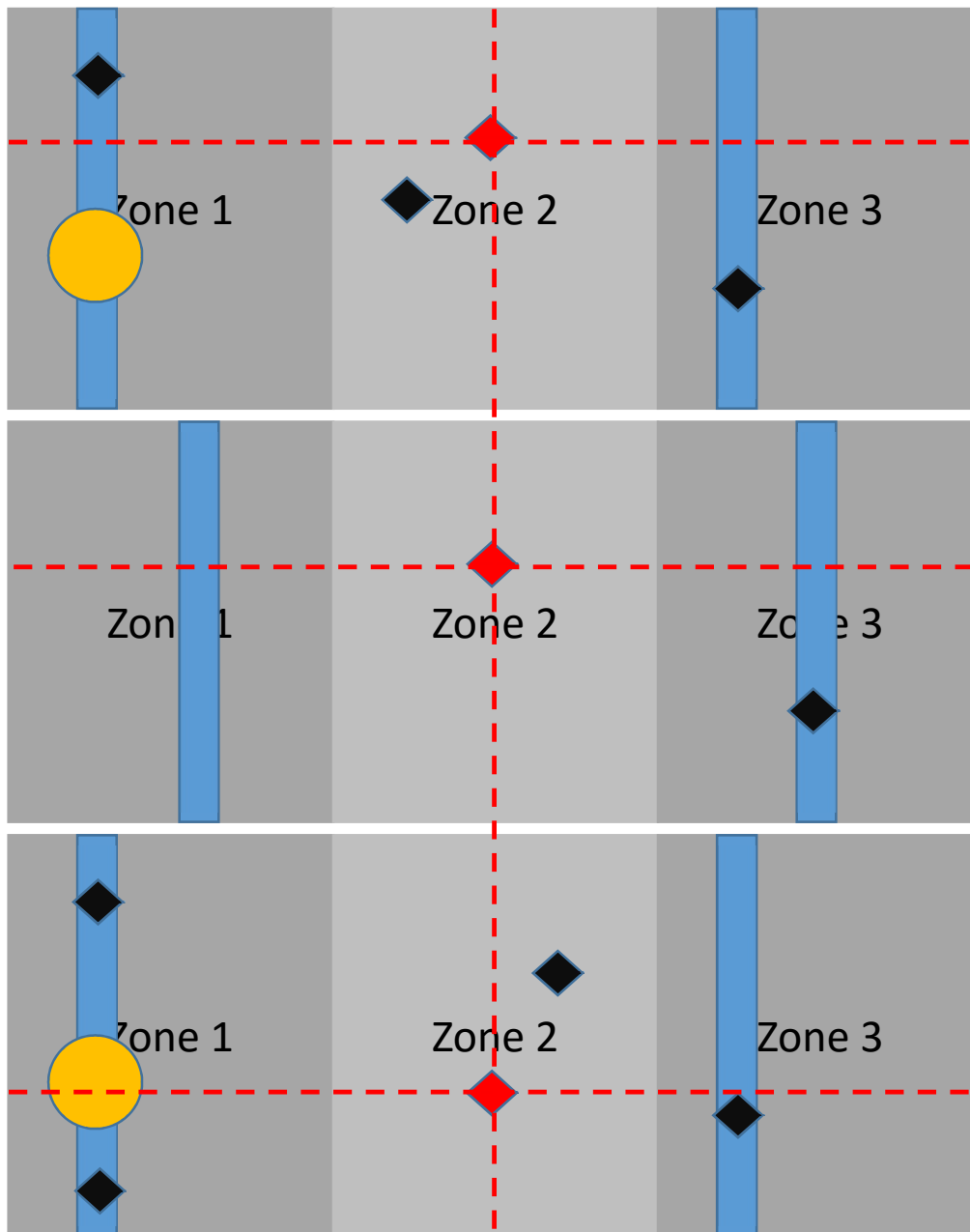
A Bay may also specify an optional *Transportation Node* which can be any node located within the boundary of the Bay. This node defines the coordinates of two perpendicular transportation aisles that run through this node; an *Internal Transportation Aisle* that runs along the length of the Bay, and a *Cross Bay Transportation Aisle* that runs across the Bay to support Crane movements between Bays. The *Use Internal Aisle* property on the Bay specifies if travel within the Bay requires use of the Internal Bay Transportation Aisle or can be done as a direct location-to-location move within the Bay. If this property is *True* then a within Bay movement takes place by first moving laterally to the Internal Bay Transportation Aisle, then moving along the Internal Bay Transportation Aisle, and then finally moving laterally to the final destination within the Bay. If this value is *False* then the Crane makes a direct movement between the two locations without first moving to and along the Internal Bay Transportation Aisle.

In most cases the Cranes are confined to a single Bay. However any number of Bays may be combined together into a facility and a Crane (and associated Lift/Cab) may move from Bridge to Bridge across multiple Bays. If Bays A and B support direct transfers then a Crane may move from A to B (or B to A) by lining up the Bridges at any location along the boundary of both Bays and then directly transferring from one Bridge to the next Bridge. If Bays do not support direct transfers then Cranes may only move between Bays using the *Cross Bay Transportation Aisle*.

Multi-Bay Crane transfers are enabled by specifying an *Associated Bay List*, which is a list of all associated Bays, including this Bay. In addition a *Crane Specific Transfer Bridge List* must be specified for each associated Bay to specify the Bridges each Crane to evaluate when selecting an empty Bridge for transferring into the Bay. The Boolean property *Direct Transfers* specifies if movements can occur anywhere along the Bay, or be confined to movements along the Cross Bay Transportation Aisle. This property can be specified as *None*, *RightAndLeftBay*, *RightBayOnly*, or *LeftBayOnly*. For example *RightBayOnly* specifies that direct transfers can occur between this Bay and the Bay to the Right, however transfers to the left Bay require use of the Cross Bay Transportation Aisle.

A Bay can be involved in only one Cross Bay transfer at a time. At the start of each transfer the Crane seizes all the Bays between the starting and ending nodes. If any Bay is busy it waits until all Bays are available to seize. As the Crane moves from Bay to Bay it releases the Bay it is departing from, and then releases the last Bay once the move is complete. The busy/idle status for each Bay is animated as status label displaying 1/0 for busy/idle.

The following figure depicts three connected Bays each having a Transportation Node shown in red. The dashed red lines depict both the Internal Bay Transportation Aisle, and the Cross Bay Transportation Aisle. Although the Transportation Node can be placed in any location in the Bay, they are typically positioned in a straight line across the Bays.



A system comprised of multiple Bays and multiple Cranes can easily deadlock. The Bay object has a set of three properties that can help prevent deadlock from occurring. The first of these is a property named *Blocking Action* that controls the interaction between Bridges. If specified as *Push Idle Bridge* then a busy Bridge will automatically push an idle Bridge out of the way to prevent a blocking situation. A second property named *Pre-check Zone Availability* can be used to force Crane movements to wait until the necessary zones are available before starting a move. The third property named *Pre-check Cross Bay Availability* can be used to force Crane movements to wait until all cross bays are free before starting a move. When used in conjunction these features prevent many (but not all) deadlocks.

Bridge

A Bridge is a dynamic Entity that is created at runtime and moves along the Bay in the direction of movement specified by the Bay. A Bridge may sometimes have a Cab, Lift, and Crane that moves along the Bridge and is carried with the Bridge as it moves across the Bay.

A Bridge instance is placed off to the side of the Bay and references the *Associated Bay* to which it is assigned. The Bridge entity is created during model initialization and is placed above its *Initial Node* at its *Travel Height*. The *Initial Node* can be any node object that is placed within the boundaries of the Bay. The graphical appearance of each Bridge can be changed by editing the symbol for the Bridge instance. The *Travel Speed* specifies the lateral speed of an empty Bridge when moving without a Cab and Crane. When occupied by a Cab and Crane the Bridge speed is controlled by the Crane speed and direction.

Cab

A Cab represents the trolley that moves the Lift and Crane. The Cab moves with and across its Bridge. Each Cab is owned by and controlled by a Crane and is created during initialization of the Crane. A Cab instance is placed off to the side of the Bay and has a single property that specifies its *Travel Height*. Multiple Cab entities can be created from a single Cab instance.

Lift

A Lift represents the cable lifting mechanism that raises and lowers the Crane before and after a lateral movement. A Lift is a dynamic Entity that is owned by the Crane and is created at runtime. Both the Lift and the Cab move along the Bay with the Crane. The Lift also changes length as it moves the Crane.

A Lift instance is placed off to the side of the Bay and has properties that specify the *Transport Height* for lateral movements, as well as the *Initial Desired Vertical Speed* for raising and lowering items. Note that a single Lift instance can be shared by multiple Cranes within a Bay.

Crane

A Crane physically represents the tool or device (e.g. a hook or clamp) on the end of the Lift that is used to pick up the load. Locally, a Crane represents the entire device (including the Cab and Lift) and is what is specified to make any move. A Crane is a dynamic Transporter that is created at runtime and may pickup, move, and drop-off entities at Nodes. When using the Standard Library the Crane is specified in the same way as any other Transporter when traveling between Nodes.

The Crane object is sub-classed from the Vehicle and has most of the same properties. However the Crane has been “locked down” to run in free space only and has a fixed dynamic population of 1. The Crane location is initialized to its *Initial (Home) Node*, and can return this node after completing all moves if its *Idle Action* is set to *Go to Home*. The Crane has additional properties that specify it’s *Associated Lift*, *Associated Cab*, and *Associated Bridge* as well as its *Initial Desired Lateral Speed* and *Lateral Acceleration/Deceleration*. Note that the *Initial (Home) Node* for the Crane must match the *Initial Node* specified by its *Associated Bridge*.

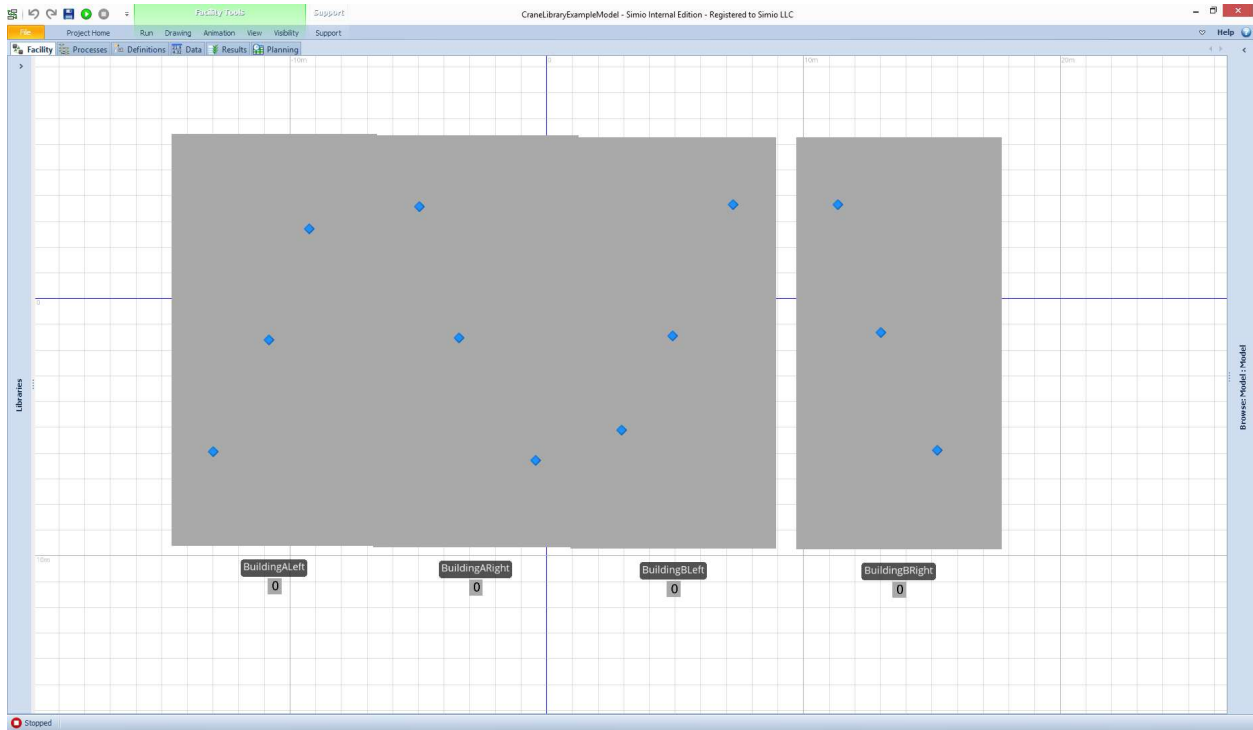
Example

In the following example we have two adjacent buildings (A and B) each with a left and right bay and cranes that can move between all four bays. We have a set of crane moves specified in a data table.

After loading the Crane Library we place four instances of Bay into our model and name them *BuildingALeft*, *BuildingARight*, *BuildingBLeft*, and *BuildingBRight*. We will then use Ctrl-click to group select each of the four Bays, and right click any of the Bays and select *Add to Object List* to add the four Bays to a list named *BayList*. We then set the *Associated Bay List* to *Bay List* for the four selected Bays. We also set the *Travel Direction* to *ForwardAndBack*, the *Number of Zones* to *15*, the *Blocking Action* to *PushIdleBridge* and both the *Pre-check Zone Availability* and *Pre-check Cross Bay Availability* to *True*. We also set the *Length* to *8* and *Width* to *16*. We will allow direct transfers between the bays within building B by setting the property *Direct Transfers* for *BuildingBLeft* to *RightBayOnly*; and for *BuildingBRight* to *LeftBayOnly*.

We will next place three TransferNodes in each of the four Bays named *BuildingALeftNode1*, *BuildingALeftNode2*, *BuildingALeftNode3*, *BuildingARightNode1*, *BuildingARightNode2*, *BuildingARightNode3*, *BuildingBLeftNode1*, *BuildingBLeftNode2*, *BuildingBLeftNode3*, *BuildingBRightNode1*, *BuildingBRightNode2*, and *BuildingBRightNode3*. Make sure the *Ride On Transporter* flag is set to *True* on each transfer node and the *Transporter Name* is populated. In this example, the *Transporter Name* references a column in the *CraneMoves* table (e.g. *CraneMoves.Crane*).

Next, we will set the *Transportation Node* in each Bay to the second node in each of the Bays, and set *Use Internal Aisle* to *True* for Bays *BuildingALeft* and *BuildingBRight*. The layout of the four Bays and TransferNodes is depicted below, where nodes are numbered from the bottom to the top:



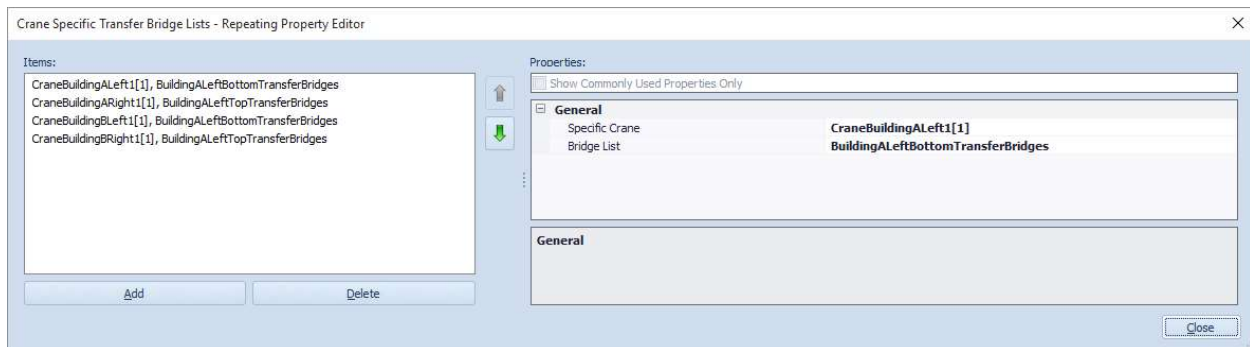
Next we place two Bridge instances for each of the four Bays, named *BridgeBuildingALeft1*, *BridgeBuildingALeft2*, *BridgeBuildingARight1*, *BridgeBuildingARight2*, *BridgeBuildingBLeft1*, *BridgeBuildingBLeft2*, *BridgeBuildingBRight1*, and *BridgeBuildingBRight2*. We will group select each pair of Bridges in each Bay and right click and select *Add to Object List* to place them in object list named *BuildingALeftTransferBridges*, *BuildingARightTransferBridges*, *BuildingBLeftTransferBridges*, and *BuildingBRightTransferBridges*. We then specify these lists for the *Transfer Bridge List* for each of the four Bays. We also set the *Associated Bay* property for each Bridge, and specify their *Initial Node*. For Bay *BuildingALeft* we assign the Bridges to nodes 1 and 2, and in the other three Bays we assign them to nodes 1 and 3.

Next we place down a Cab named *StandardCab*, and a Lift named *StandardLift*. A single Cab and Lift instance will be shared by the four Cranes in the model. We then place four Cranes named *CraneBuildingALeft*, *CraneBuildingARight*, *CraneBuildingBLeft*, and *CraneBuildingBRight*. We multi-select the four Cranes and set the *Associated Lift* to *Standard Lift*, the *Associated Cab* to *StandardCab*, and the *Idle Action* to *Go To Home*. We also set the load and unload times to .2 minutes. We then individually select each Crane and set the *Associated Bridge* to the first Bridge in each Bay and the *Initial Node* to the same initial node assigned to the associated Bridge.

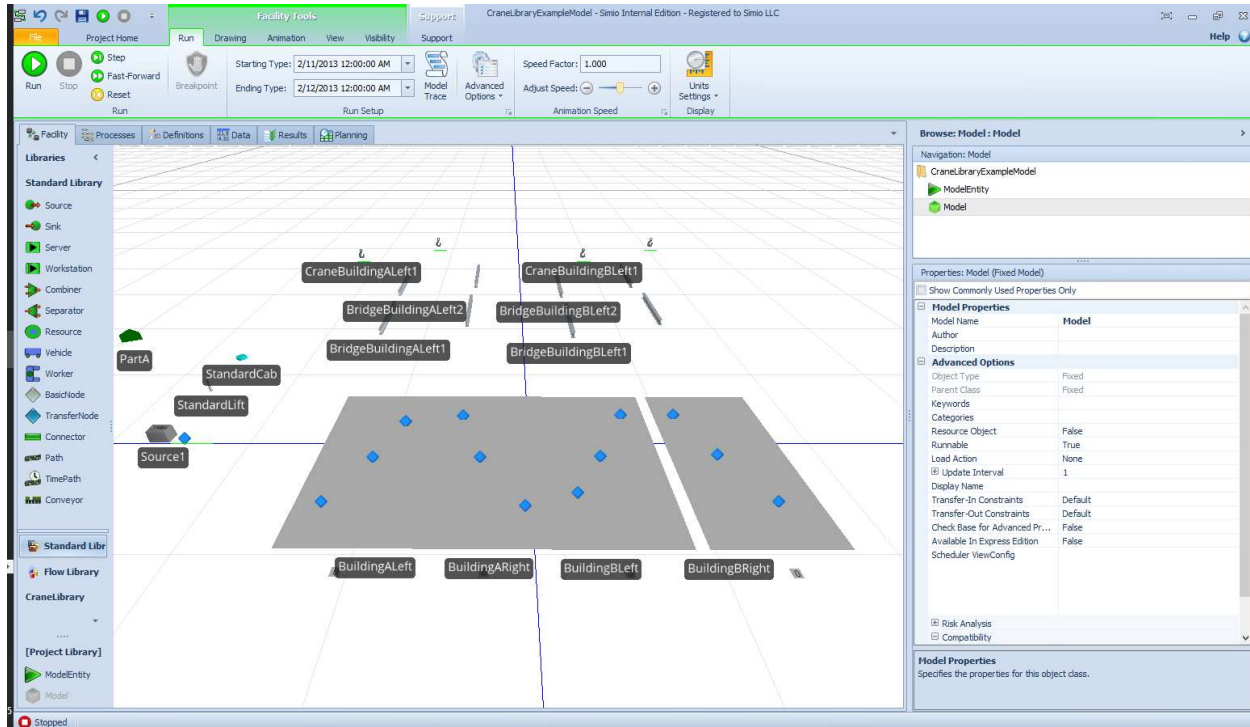
Set the *Crane Specific Transfer Bridge Lists* on each bay. This repeating property is used to determine the order to evaluate that bridges the crane should transfer to. This is used to manage some of the

congestion in the facility. If some cranes tend to work on one side of the bay compared to other cranes.

For example, *CraneBuildingALeft* and *CraneBuildingBLeft* tend to work at the top of the bay and *CraneBuildingARight* and *CraneBuildingBRight* tend to work at the bottom of the bay. You would set *CraneBuildingALeft* and *CraneBuildingBLeft* to use the bridges at the top of the bay and *CraneBuildingARight* and *CraneBuildingBRight* to use the bridges at the bottom of the bay.



We will have two types of parts (A and B) moving through the system. We place two model entities and rename them PartA and PartB, change their network to free space, and set their initial desired speed to 9999999. We paint PartB red, and resize both parts. The following shows our facility model at this point.



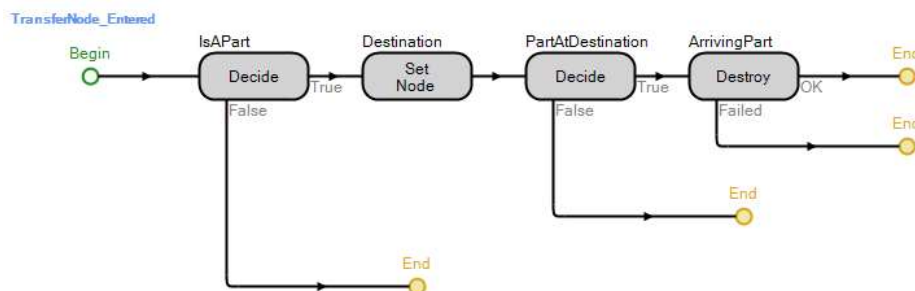
Our final step is to create arrivals to the system to be picked up and moved by the Cranes and dropped off at their destination. We will specify our arrivals by defining an arrival table named *Crane Moves* that

has columns named *Start Time* (Date-Time), *Part Type* (Entity), *Start Node* (Node), *End Node* (Node) and *Crane* (Transporter).

| | Start Time | Part Type | Start Node | End Node | Crane |
|----|-----------------------|-----------|---------------------|---------------------|----------------------|
| 1 | 2/11/2013 12:00:00 AM | PartA | BuildingALeftNode1 | BuildingALeftNode3 | CraneBuildingALeft1 |
| 2 | 2/11/2013 12:00:00 AM | PartB | BuildingALeftNode3 | BuildingALeftNode2 | CraneBuildingALeft1 |
| 3 | 2/11/2013 12:00:30 AM | PartA | BuildingALeftNode1 | BuildingBRightNode3 | CraneBuildingALeft1 |
| 4 | 2/11/2013 12:01:00 AM | PartB | BuildingBRightNode3 | BuildingALeftNode1 | CraneBuildingBRight1 |
| 5 | 2/11/2013 12:01:30 AM | PartA | BuildingBRightNode3 | BuildingBRightNode1 | CraneBuildingBRight1 |
| 6 | 2/11/2013 12:02:00 AM | PartB | BuildingBRightNode1 | BuildingBRightNode2 | CraneBuildingBRight1 |
| 7 | 2/11/2013 12:02:30 AM | PartA | BuildingARightNode1 | BuildingARightNode3 | CraneBuildingARight1 |
| 8 | 2/11/2013 12:03:00 AM | PartB | BuildingARightNode3 | BuildingARightNode1 | CraneBuildingARight1 |
| 9 | 2/11/2013 12:03:30 AM | PartA | BuildingBLeftNode2 | BuildingBLeftNode3 | CraneBuildingBLeft1 |
| 10 | 2/11/2013 12:04:00 AM | PartB | BuildingBLeftNode3 | BuildingBLeftNode2 | CraneBuildingBLeft1 |
| 11 | 4/11/2013 12:04:00 AM | PartA | BuildingBLeftNode2 | BuildingALeftNode1 | CraneBuildingBLeft1 |

To generate arrivals from this table we place a Source in the model and specify the *Entity Type* as *CraneMoves.PartType*, the *Arrival Mode* as *Arrival Table*, the *Arrival Time Property* as *CraneMoves.StartTime*, and *Node Name* for the entity destination as *CraneMoves.StartNode*. Note that these entities will exit the Source into free space and immediately enter their specified starting node.

When a part enters its starting node we want to set its destination. In addition when a part is dropped by a Crane at its destination we want it to be destroyed. To do so we will define the following *Entered* add-on process named *TransferNode_Entered* that is executed by all the TransferNodes in our model.



The first Decide step evaluates the condition *!Is.Crane* to determine if the entering entity is not a Crane. The parts (entities) exit the *True* branch and then set their destination to *CraneMoves.EndNode*. The next Decide step test if the part is at its destination (*Entity.CurrentNode == CraneMoves.DestinationNode*). If so the entity is sent to the Destroy step where it is destroyed. Note that this logic also supports entities with starting node and ending node being the same.