Conveyor System

Requirements

- representation of conveyor network
- edge/node addition in graph
- edge/node removal in graph
- graph path finding
- future extensibility for use of sorters
- detection of conveyor line made of blocks
- intergrid connections
- inventories of multiple systems connected together

Conveyors

ConveyorSystem

Conveyor system represents a network of ConveyorElements within one grid. These ConveyorElements form a graph structure and there can be any amount of completely separate graphs within one ConveyorSystem.

It also provides simple functions for testing whether 2 ConveyorNodes are connected, translating Inventories, block and conveyorPorts into ConveyorNodes they are associated with.

Graphs

There are multiple representations of the graphs made of ConveyorElements. **ConveyorGraph** and **CondensedGraph**.

ConveyorGraph consists of the **ConveyorElements** and contains all the elements as they were registered into ConveyorSystem by the outside forces.

CondensedGraph consists of simplified graph created from conveyorGraph by the process of Graph Condensation.

ConveyorElements

ConveyorLine

Element that creates an edge within the ConveyorGraph and functions as bidirectional. Always connect 2 ConveyorNodes together.

ConveyorNode

Element that creates a node within ConveyorGraph. Each Node can be a simple node with no function attached (eg. Conveyor junction block that connects multiple lines together) or can have multiple **Inventories** (eg. Refinery with input and output Inventory) or a **ConveyorFilter** (eg. Filter/Sorter block, that restricts the direction of item flow and what items are allowed).

FilterNode

A special decorated case of ConveyorNode with ConveyorFilter attached to it with Directional edge set. It alters pathfinding as it limits directionality of one edge to lead out of this node.

Sorter Node

A special decorated case of ConveyorNode with ConveyorFilter attached to it, with attached limit what items can pass through. It alters pathfinding as it limits what items can pass through the node.

Link Node

A special decorated case of ConveyorNode with ConveyorLink attached to it. It allows linking 2 separate ConveyorSystems together for them to work as one.

CondensedElements

Not implemented in first iteration

Strongly Connected Nodes (SCN)

Nodes of the CondensedGraph are created through the process of Graph Condensation out of ConveyorNodes and ConveyorLines.

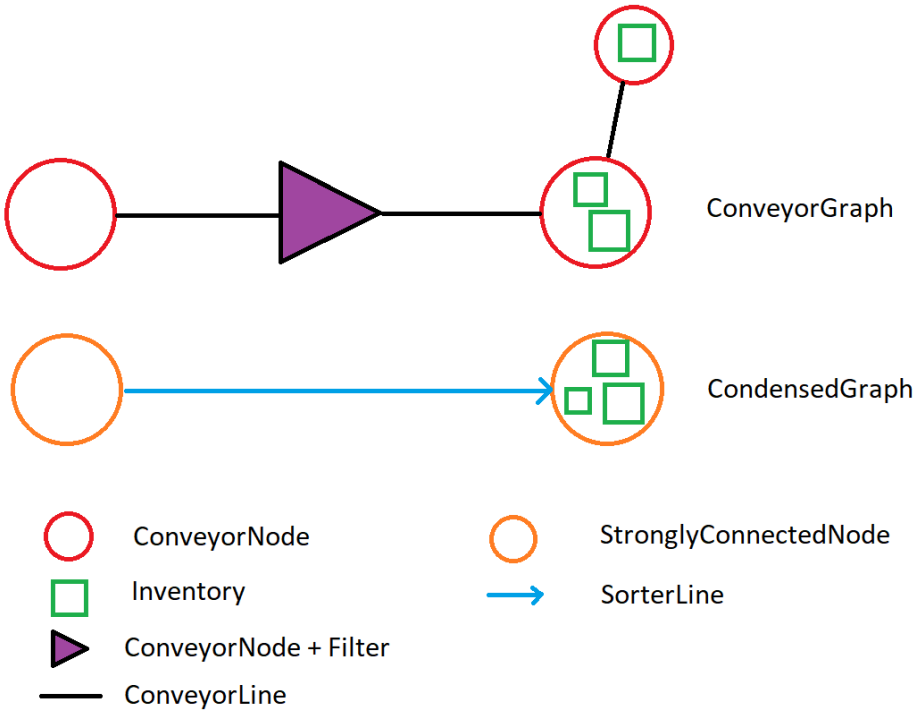
SorterLine

Edge in CondensedGraph created through the process of Graph Condensation out of **SorterNode** and **ConveyorLines** that are associated to it (that connect the **SorterNode** to

closest non-sorter **ConveyorNode**s).

All SorterLines must contain at least one SorterNode (but may contain multiple SorterNodes chained together, in that case **SorterLine** will contain multiple **ConveyorFilters**), but existence of SorterNode does not mean it will become SorterLine.

When both ConveyorNodes connected by a potential SorterLine are in the same SCN, the SorterLine creates a loop on the same node and can be dropped completely.



ConveyorElement registration

When a block with conveyor capabilities (has **ConveyorComponent**) is placed into the grid ConveyorSystems triggers on standard event of block added. Similar for unregistration.

ConveyorComponent (Component)

Component that determines that block is a part of ConveyorSystem. Addition of a block with this component triggers addition to the ConveyorSystem of the grid.

ConveyorCapableComponent also provides, from its definition, the ConveyorSubgraph and the ConveyorSubgraph must be editable through VRageEditor.

ConveyorFilter (Component)

Component that represents Filter/Sorter block and provides access to modify ConveyorFilter of specific ConveyorNode. It also performs necessary serialization and deserialization of data for the ConveyorFilter.

MechanicalConveyorComponent (Component)

Component that extends functionality of **MechanicalBlockComponent** by performing links between ConveyorSystem that are on both ends of MechanicalBlock pair.

When MechanicalBlocks connect, it listens to this happening and if both sides are capable of linking, a link is created.

ConveyorElement translation

When the block is added, it not only adds its own representation into the ConveyorSystem (such as adding CargoContainer adds ConveyorNode with Inventory), but also can finish surrounding structures, such as connecting open-ended ConveyorLines, thus technically also adding ConveyorLine to the ConveyorSystem. For some blocks (Sorter/Filter) it is also important what orientation there is and what line connects to a specific conveyor port.

For this reason, the block with **ConveyorComponent** will contain simple predefined **ConveyorSubgraph** in its definition with defined **ConveyorNodes**, **ConveyorLines** and also will contain pairings of what **ConveyorPort** translates into what edges of the ConveyorSubgraph this connects to (each ConveyorPort dummy will have reference what edge it represents, if no edge is defined, it connect directly to the first ConveyorNode).

Addition process

When a ConveyorCapableBlock is added, it will provide its **ConveyorSubgraph** into the ConveyorElementTranslator together with the ConveyorSystem it is registering into.

ConveyorElementTranslator will traverse the Grid searching from block's conveyor ports trying to finish conveyor lines. When lines are found, the translator can add ConveyorNodes from the ConveyorSubgraph into the ConveyorGraph and, based on the traversed conveyor lines, it will know what ConveyorLines will connect to what ConveyorNodes.

ConveyorSubgraph

Subgraph is defined in the definition of ConveyorCapableBlock and represents what will be inserted into ConveyorSystem's ConveyorGraph. It consists of the Nodes and Edges.

Nodes contain information similar to the ConveyorGraph nodes but in a form suitable for the editor and more human readable.

Edges of the ConveyorSubgraph are contained within the Nodes and they are split to Internal and External ones. Internal edges lead between 2 nodes, External edges lead between one Node and a dummy. For this purpose they hold the names of the dummies.

Nodes:

- **Node** - ConveyorNode with nothing special attached. Can have lines defined
- **Inventory** - ConveyorNode with inventory. Must have reference to its inventories. Can have lines defined
- **Sorter** - ConveyorNode with ConveyorFilter defined. Must have Input and Output line defined.

ConveyorPort Dummies - define a line they attach to

```
public partial class ConveyorSubgraph
{
      /// <summary>
      /// Nodes of the graph
      /// </summary>
      public List<ConveyorSubgraphNode> Nodes;
)

// Represents single node of ConveyorSubgraph
public class ConveyorSubgraphNode
{
      /// <summary>
      /// Ids of edges connected to a node
      /// </summary>
    public List<Edge> Edges;

      /// <summary>
      /// Tags of invnetory components related to this node
```

```
        /// </summary>
        public List<StringId> InventoryTags;

        /// <summary>
        /// Tag of filter component related to this node. (Optional)
        /// </summary>
        public StringId? FilterTag;

        /// <summary>
        /// Id of edge that turned to one-directional (Optional)
        /// </summary>
        public int? FilterOutputEdge;
}

// Base for the edges
public abstract class Edge
{
        /// <summary>
        /// Unique identifier of the edge in this subgraph
        /// </summary>
        public int Id;
}

// Inner edge between 2 nodes
public class InnerEdge : Edge {}

// Edge between node and dummy
public class ExternalEdge : Edge
{
        /// <summary>
        /// Name of dummy the edge leads to
        /// </summary>
        public string DummyName { get; set; }
}
```

Autogeneration

For the purpose of easier setup, ConveyorSubgraph can also be generated.

If ConveyorSubgraph is not specified and the block is to be a non-ConveyorLine, then a ConveyorSubgraph will be auto-generated.

One single ConveyorNode will be created, aggregating all identifiable inventories (inventories with at least one Tag for identification) and connecting all ConveyorPorts to itself (Both Pipable and Interactible dummies are connected. Both of them require to be recognized in the conveyor system). If any identifiable FilterComponent is present, it will be included in the auto-generated graph. As it is not possible to identify in which direction a filter should direct the flow of items, it will be directionless.

Conveyor Graph

Conveyor graph consists of Nodes, Edges and Decorations and represents the whole conveyor system at the most basic level with the lowest level operations..

Nodes

Nodes are a collection of all nodes within one ConveyorSystem. Nodes are connected with edges and provide structure for the traversing.

Each node is represented by their unique index within the system, dictionary of other nodes it connects to (there is no need to represent edges separately) and list of optional decorators. Nodes do not need to be iterated, but require quick searching by their index.

Nodes can be enabled or disabled. This is necessary for example when blocks are below functional level or are turned off, or there is insufficient power on the grid. In any case it disables one or more nodes. For this reason it requires to have information whether it **IsEnabled** and whether it can be used for its purpose.

Edges

Edges do not need separate representation. But there is some additional information needed.

Directionality

Edge can be directional because of filters, but that can be handled by the deformation.

Edge disabling

Edges can be enabled or disabled. Disabled edge behaves as if it was not present. This behavior is needed as the block can be broken to pieces and they may lose dummies, thus edges should be disabled.

This information is added as an **EdgeInfo** for holding this information.

Multiplicity

Furthermore 2 blocks, each represented by one node, can have multiple ports facing each other. This means that 2 nodes can be connected more than once. One node can even connect to itself by looping a conveyor line. This is crucial as one of the edges may be directed because of the filter.

If the multiplicity was not preserved, when one edge would be removed, further traversal would be needed to find out whether the two nodes are truly disconnected or not.

This requires the **EdgeInfo** to be extended to hold information of how many times is the edge present and also how many of them are enabled. If at least one edge is enabled, traversal is possible.

Connection with ConveyorSubgraph

For most of the common operations, such as searching for the items in the system, finding whether conveyor blocks are connected and such, there needs to be a connection between this low level representation and block level.

Once the ConveyorSubgraph of the block is processed by the ConveyorElementTranslator, the output is inserted into ConveyorGraph, but the connection is lost. What elements belong to the block? How many of them? If I want to check whether inventories are connected, what node to even start from?

For this reason there needs to be information about the **block's DEntity and all conveyor nodes that belong to it**. This can be achieved through a simple Dictionary of lists. But if a block contains multiple nodes, it would not be possible to discern between them. Thus each node in ConveyorSubgraph needs a pairing to ConveyorNode. This is solved by simple addition of SubgraphId to each Node.

```
public class Node
{
    /// unique id within this ConveyorGraph
    int NodeId;
    /// id from ConveyorSubtraph that was used to add this node
```

```
    int SubgraphId;
    /// whether node is enabled
    bool IsEnabled;
    /// nodes this node connects to
    Dictionary<int nodeId, NodeInfo info> Edges;
    /// one of various decorations
    int? XyzDecorator;
}

public class EdgeInfo
{
    /// how many times the edge is present
    int Multiplicity;
    /// how many of the edges is enabled
    int IsEnabledMultiplicity;
}
```

Decorations

Decorations are extensions of the nodes that allow further behavior. Most of the nodes will not have any decorations so it is pointless to keep the information on every node. Instead the nodes will keep id referencing a decoration.

InventoryDecoration
Simple list of inventories that are attached to the node. Each node can have any amount of inventories attached.

FilterDecoration
Represents filtering capabilities. Can modify directionality of an edge (holds information of what edge is limited in direction) whether directionality is active, whether backflow of items is allowed* and also what items are allowed to flow through the directed edge.

* Backflow feature allows either to be **Free** (all items may flow against the directed edge, regardless of white/blacklist) or **Blocked** (no item may flow against direction). It does not affect what items may flow with the direction of the edge (that is decided by black/whitelist of items).

LinkDecoration
Identifies a node that is connected to other ConveyorSystem's node thus allowing traversal in one ConveyorSystem to search through the other one. Requires ConveyorSystem

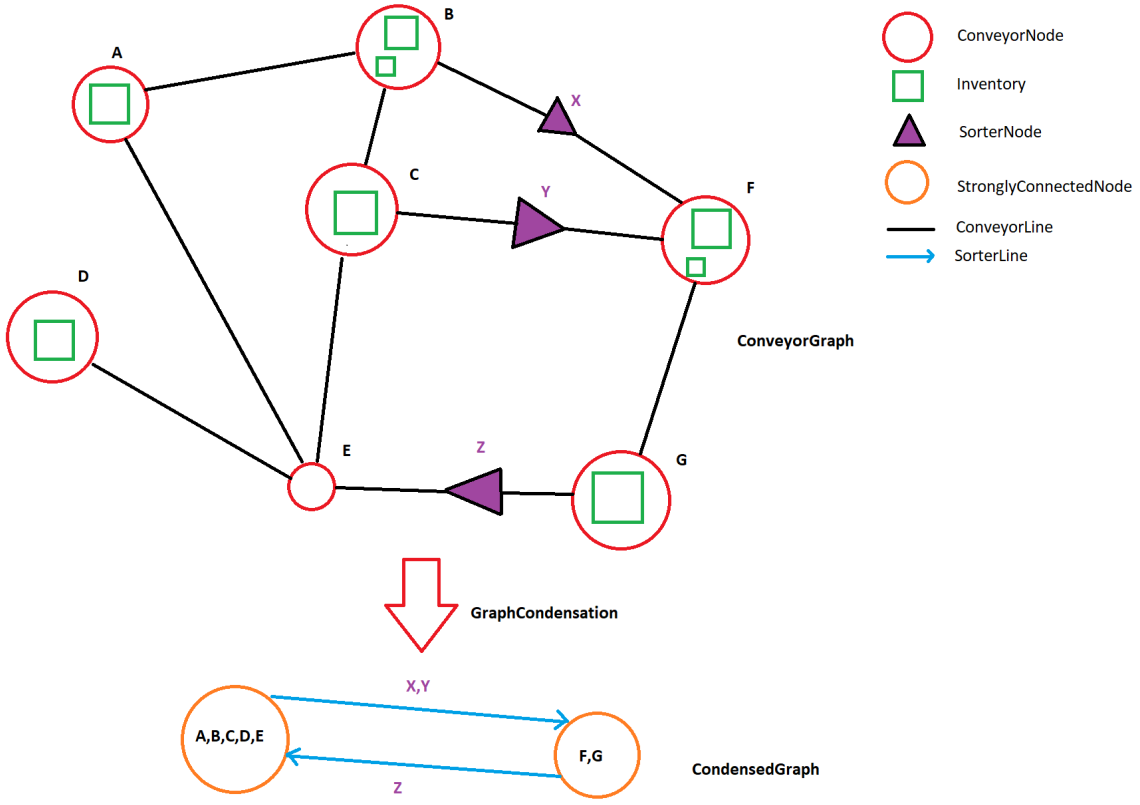
reference and also id of a node in the other system it connects to, to know where to start queries.

Graph Condensation

Not implemented in first iteration

Graph condensation is a process of converting ConveyorGraph into a CondensedGraph. All ConveyorNodes in ConveyorGraph that are freely accessible from one another without any restriction are turned into a StronglyConnectedNode(SCN) of a CondensedGraph. All ConveyorLines that are inside the SCN are not needed in CondensedGraph.

SorterNodes are excluded from the SCNs and are processed at the end. When SorterNode is not connected by the ConveyorLines to at least 2 ConveyorNodes, it can be dropped completely (it will have no effect). When both ConveyorNodes the sorter connects to are within the same SCN, it can be dropped as well as there is no reason to run it inside the SCN.



Construction

Condensed graphs will be built incrementally in most cases, but in certain cases, there may be a need to build it completely from scratch (breaking of SCN into smaller pieces on line removal, loading of the grid (if it was not saved or grid was modified outside of game)).

When a ConveyorLine is added into the system, it can be inside SCN, which changes nothing. Or between two SCN, which will merge them into one SCN. All SorterLines between those two SCN will be removed, and SorterLines leading to them will be connected to the new merged SCN. It can also fulfill the condition of SorterLine and thus become SorterLine

When the ConveyorLine is removed, a test must be performed to determine whether both sides of the conveyor line are still in SCN. If not, a graph condensation will have to be performed on the subset of the graph that consists of the nodes of said SCN that is being split.

When a Sorter line is being added (something that fulfills the SorterLine condition was added), it will be either inside one SCN, which can be ignored. It can be between 2 SCNs which will just add a line to the condensed graph (or modify the constraint of the line if it already exists).

Removal of the Sorter line will have no effect if inside SCN. But if between two SCNs it will remove a line (or modify the ConveyorFilter of the SorterLine, if multiple lines exist).

Addition of inventory is a bit more complicated, as it also serves as a conveyor line. Addition of an inventory block within the ConveyorSystem can either extend existing SCN (if placed next to PotentialConveyorLine or block with ConveyorPort) or may create new SCC when placed next to Sorter line or when not connected to any ConveyorLine at all. It may also create new SorterLine when it fulfills previously unfulfilled condition for a SorterLine (node-line-sorter-line-node)

Removal of inventory removes the node from SCC and also functions as removal of conveyor lines.

Determining in what SCN Inventory is

There is a need for a quick way to determine what SCN the inventory belongs to. This way in a normal ConveyorSystem, most queries will be quickly solved as most commonly there

will be one massive SNC.

A dictionary of (Inventory (Component) - SCN index) pairs should be sufficient for pur purposes.

Recomputation

When a whole conveyor line needs to be recomputed (possibly when grid is loaded, unless the information is saved, but due to modifications on the grids outside the game, recomputation may be needed.) There are readily available and well known algorithms to detect strongly connected components within the graph (Tarjan or Kosaraju). Same algorithm may be needed for when an SCN is being tested, whether it is split or not after conveyor removal.

Connectivity search

Not implemented in first iteration - connectivity and searches are done on ConveyorNetwork instead

When a Conveyor/Sorter line is destroyed, there is a need to find whether the end points are still connected or not. If it was within one SCN, we don't care about the Sorter line(it was not important there, there must be a Conveyor line, if it is SCN) but for conveyor lines we do. SCN could be split into 2. And if it is, the whole Conveyor System may be split in two. If the Sorter line between 2 SCNs was removed, we only need to remove the line from the graph and test whether the condensed graph is still connected at all.

This means we need a way to test connectivity inside SCN's subgraph and a similar connectivity test for the condensed graph (when there are changes in a Condensed graph such as removal of Sorter Line or split of SCN).

Due to operations on the ConveyorSystem being async, recomputation of the system can be performed in a lazy fashion and delayed until someone really wants to access the ConveyorSystem.

Conveyor Network construction

Conveyor Lines are not made of single blocks and a placement of a single conveyor block will not immediately add a Conveyor Line into the graph.

For a Conveyor Line to be created, a series of Conveyor blocks need to be placed in a connected fashion (correct orientations) and both ends must be connected to some ConveyorNode.

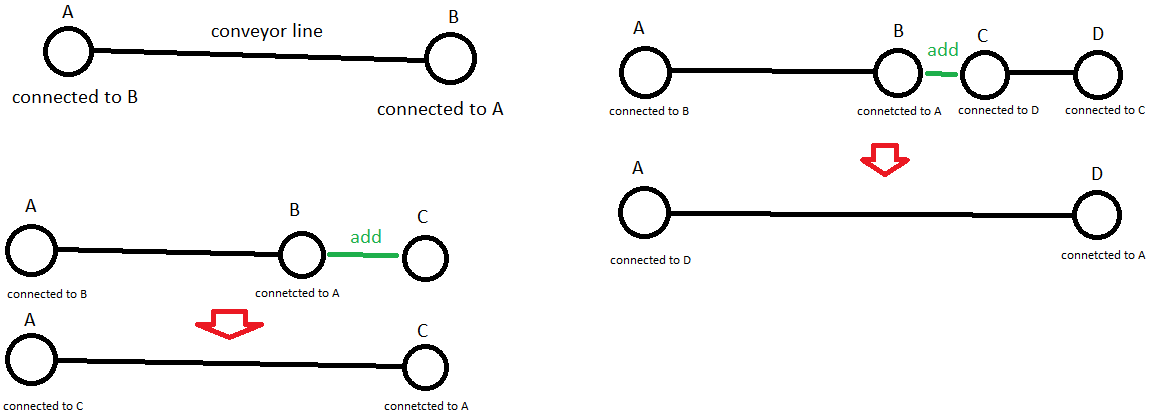
When a Conveyor Block is placed, the grid has to be traversed in both directions (the conveyor block has only 2 sides that connect, otherwise it would be Conveyor Node). Traversing a long line of blocks is not optimal and performing the test for every placed block would be a waste of performance.

Potential conveyor line

Not implemented in first iteration

When a new potential line is in the grid, both end blocks of the line will have Data attached that point to the other end. When a new block is added to the line, the no-more-end will have data removed, new block will receive its data and in them, it knows what is the other end of the potential line and its reference will be modified.

When two potential lines are being merged by placing a conveyor block in between, the other ends (that are not being connected will have their data modified, and will become the ends on the new line).



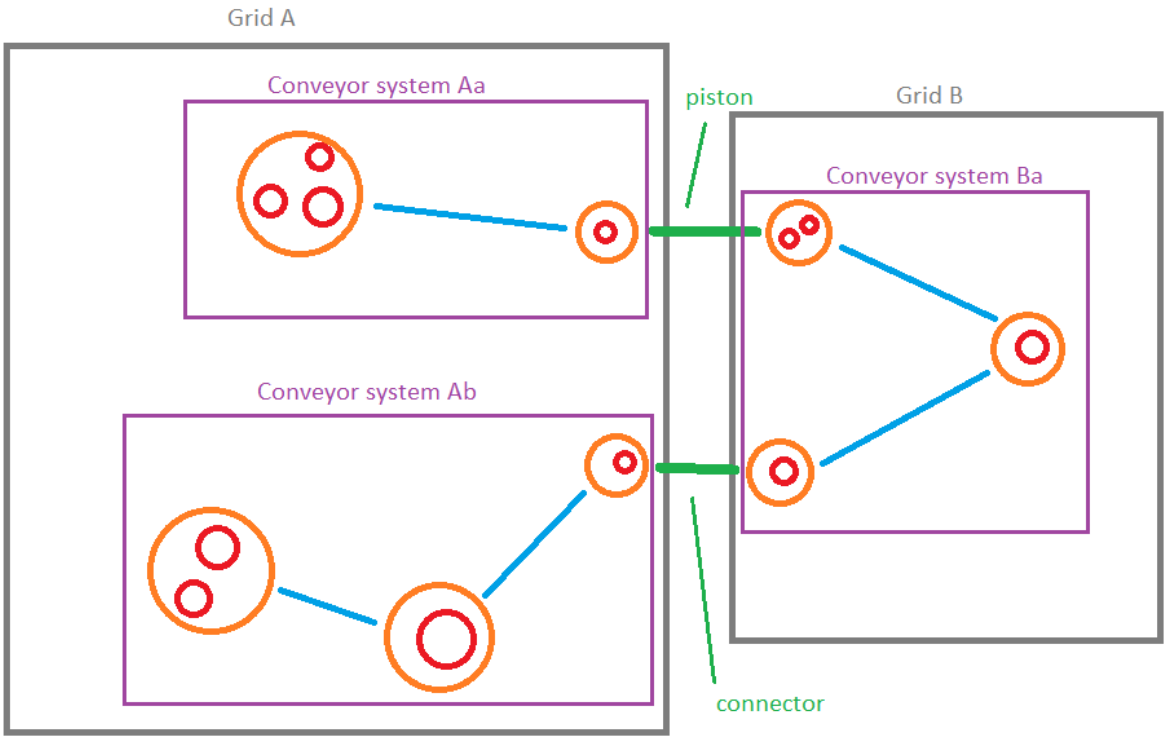
Potential conveyor line becoming real conveyor line

When a conveyor line changes, or when a new block with a conveyor node is placed, connected potential conveyor lines will perform a check on both ends to see whether it is attached to some ConveyorNode. If both ends are connected, a new Conveyor Line is added to the ConveyorSystem (we know which one by the Conveyor Nodes it connects to).

Breaking the line
When a Conveyor line block is removed and it is not an end piece, both sides will have to be traversed and turned into Potential conveyor lines if they weren't already or at least their end references will have to be modified.

Intergrid Connection

Due to the use of Rotors, Pistons and Connectors, it is possible to have two complete Conveyor Systems to be connected together into one (there can be any number linked together even in circular fashion, but convection are always between 2 points). At that moment, they are technically one system, but a System that can be spread over multiple grids, with each grid being able to have multiple ConveyorSystems at once.
Furthermore, one Conveyor system can be connected to another Conveyor System at multiple points and it would still make sense (one line only for Ingots, one line only for Components, etc.)



For this reason the Conveyor System graph will have to have a special node that references Node in a different Conveyor system. These nodes can be used in the level of Condensed graph.

Consequences

When ConveyorSystem A contains a Reference node to another ConveyorSystem B, it may access its Condensed graphs for query purposes without any issue. When one of the ConveyorSystems changes in any way, the change will be self-contained and will not affect connectivity of the other graph, but all the connected ConveyorSystems need to be notified of the change for caching purposes (Change in Aa will not only affect cached paths of Ba, but also Ab and possibly any other ConveyorSystems connected to them). Furthermore, if any ConveyorSystem is split, reference nodes need to be changed and ConveyorSystems notified.

Conveyor Linking

As a solution to connecting multiple ConveyorSystems together, ConveyorNodes in the ConveyorSystem can now be decorated with ConveyorLinks, which are essentially a reference to different ConveyorSystem and a node inside that system.

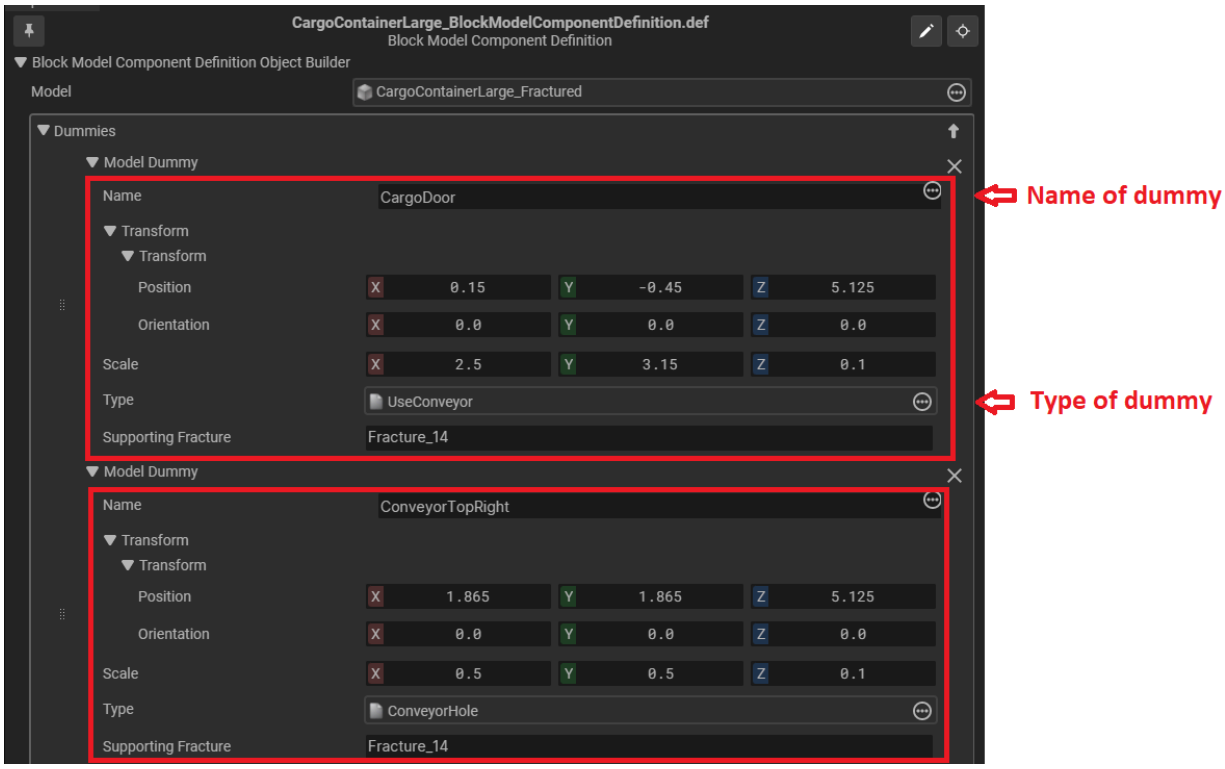
This allows the traversal algorithm to continue searching other systems as it has knowledge of what system to use and what node should be the starting point, as if it was a normal connection.

Furthermore ConveyorSystem now holds information about what systems it connects to so it is possible to quickly determine impossible queries.

Block setup guide

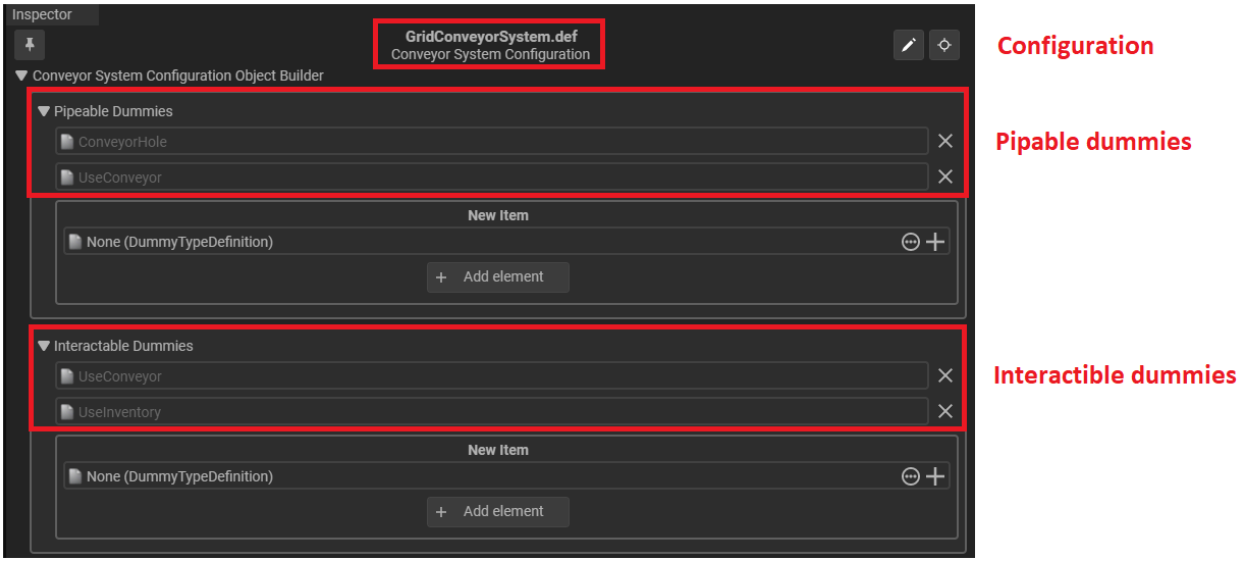
BlockModelComponent (Component)

In order to set up a conveyor-capable block, block must have dummies through which it can be interacted with or to which conveyor pipes can connect.



Dummy types

Type of dummy, that is required, vary depending on the purpose. They are all defined within **ConveyorSystemConfiguration**.



Pipable dummies are dummies that can connect conveyor pipes. If dummy of a type listed in the collection is used, ConveyorSystem will attempt to use dummies to connect conveyor pipes together using these dummies

Interactible dummies are dummies that allow interaction with the inventory itself. Interaction with them allows for manipulation with inventories, transfer of items, etc.

Default types are:
ConveyorHole - just pipable, not intractable. Usable when ports should not be interacted with (ends of pipes in SE).
UseConveyor - both pipable and intractable. Example - cargo crates
UseInventory - only intractable. When conveyor lines should not be created. Example is front loading port of the Rocketlauncher.

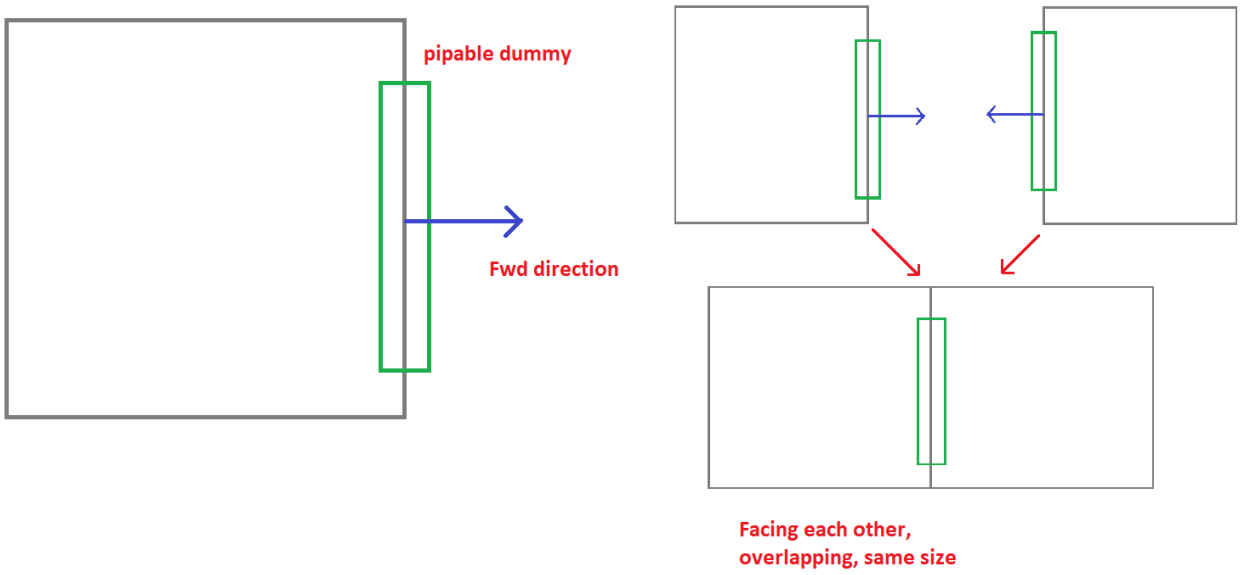
Dummy transformation

Dummies just for the purpose of interaction do not have any restrictions on the transformation.

Dummies for the purpose of conveyor pipes have specific constraints.
If two blocks should connect using pipable dummies, the dummies must have the same scale. Scale is used to determine the size of the conveyor pipe and all 3 values must match.

They must be oriented towards each other. Forward direction (-Z axis) is used for determining where the dummy is facing and dummies should point outward of the block.

Dummies of the neighboring blocks must overlap. The best practice is to place pipable dummies by its center on the edge of the block/cell so when two blocks are placed near each other, their dummies will overlap.



ConveyorComponent (Component)

Obviously the most important component that must be present for all conveyor capable blocks.

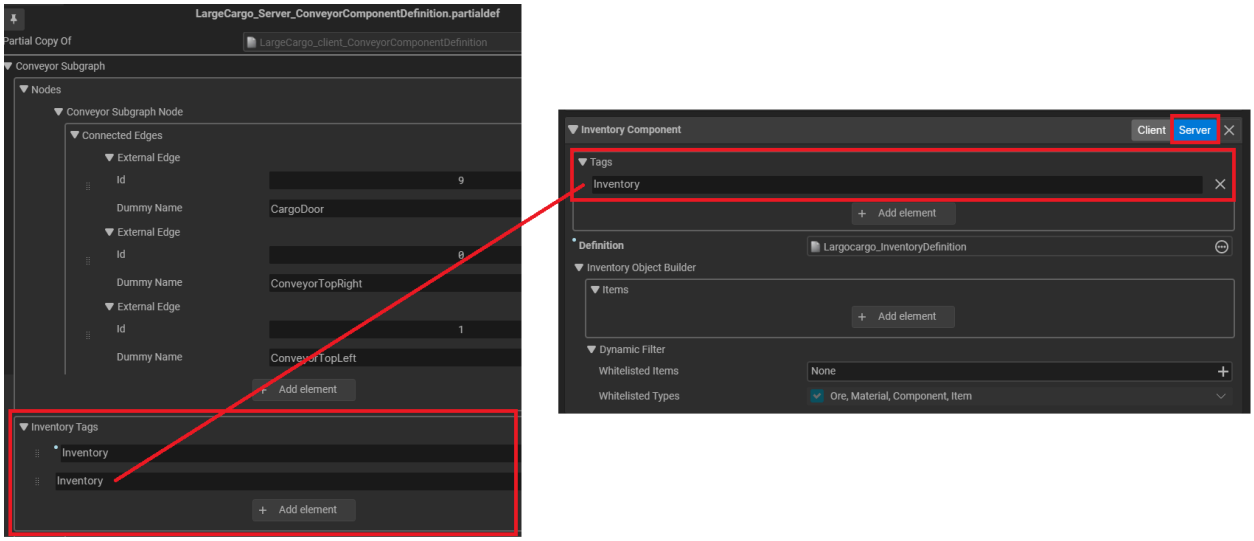
It contains a ConveyorSubgraph which needs to be set up according to the needs of the block.

InventoryComponent (Component)

Optional component that signifies that the block has inventory and allows for such inventory to be used with the ConveyorSystem, be interacted with and so on.

There can be any amount of inventories on a block.

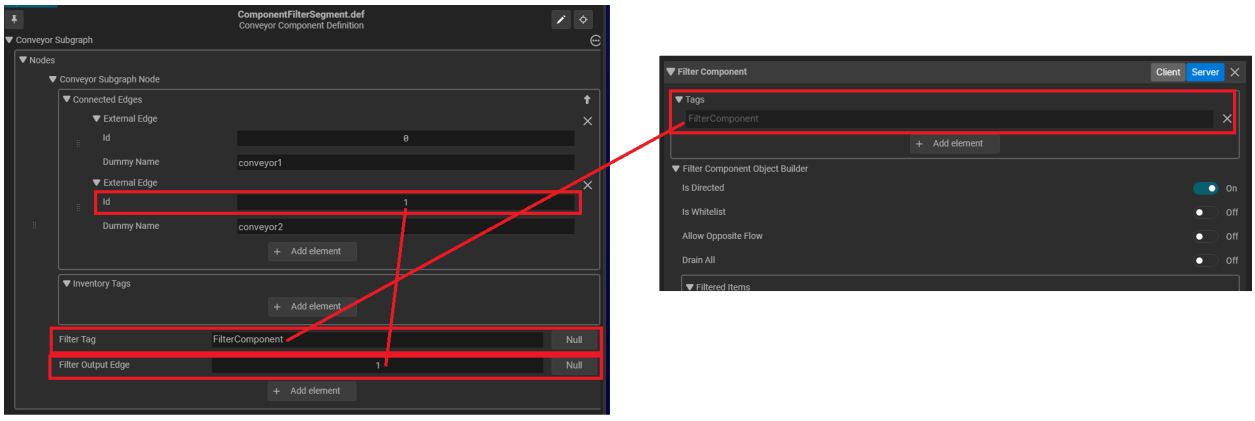
When setting up inventory, it must be present only on the server. Client inventories will be ignored. Tags, unique identifiers of the component, will be used by the ConveyorSystem to identify and search for the inventory. If the inventory is to be used by ConveyorSystem, a tag must be assigned. Tag is to be provided to a specific ConveyorNode within ConveyorSubgraph.



FilterComponent (Component)

Component that extends the behavior of the block by allowing it to have a directed edge and providing white/blacklist of items that are allowed/forbidden by this edge.

Only one FilterComponent can be present on one block. ConveyorSubgraph must have the tag of the component attached to one Node and FilterOutputEdge must be specified. FilterOutputEdge corresponds to an ID of one of the Node's Edges. This edge will be directed and will have the Filter applied to it.

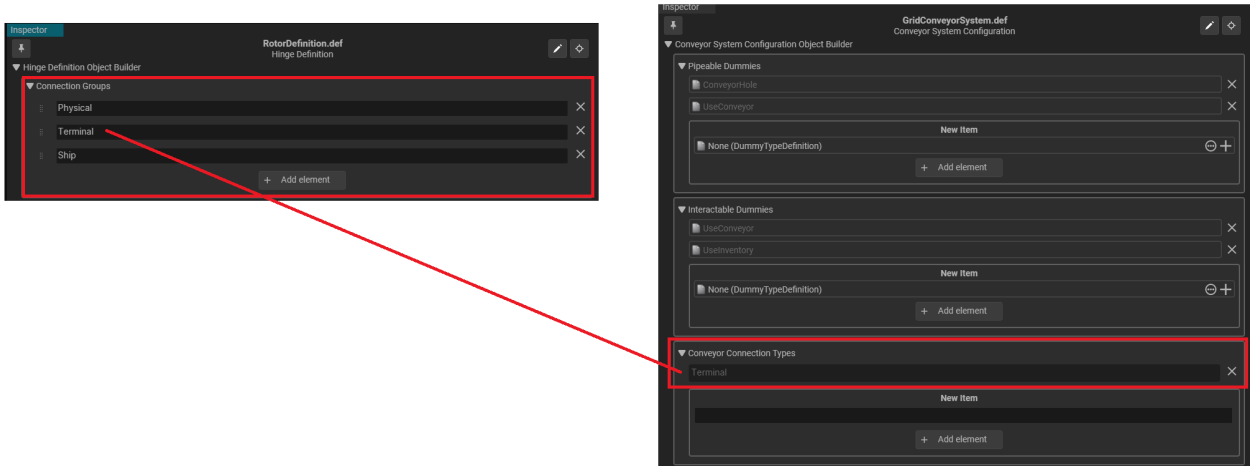


MechanicalConveyorComponent (Component)

Component that extends behavior of a MechanicalBlockComponent to allow for the connections of the ConveyorSystems.

When this component is present on the block (it requires MechanicalBlockComponent), whenever the block connects to another MechanicalBlockComponent, an attempt to link the two ConveyorSystem is made.

Both blocks must have a MechanicalConveyorComponent for this to function and both MechanicalBlockComponents must support the connection type that allows for linking. At least one of the Connection types that are listed in ConveyorSystemConfiguration must be present in the MechanicalBlockComponent, to allow linking.



Simple conveyor setup

Conveyor pipes (conveyor line segment)

Conveyor pipes are the simplest conveyor block. ConveyorLineSegments are parts of the conveyor pipes and allow connection of 2 points only. They are optimized to not increase the performance load with their length.

Requirements:

- No inventories
- No filters
- No MechanicalConveyorComponents
- Exactly 2 pipable dummies
- No ConveyorSubgraph (must be set to null)

ConveyorLineSegments are automatically recognized and there is no need for any additional setup.

Autogenerated conveyor block

For trivial blocks that do not have any complex insides (that require just one node), autogeneration of ConveyorSubgraph may be used.

If ConveyorSubgraph is set to null, it will be autogenerated.
One ConveyorSubgraph node will be created. All inventories that have any tags will be attached to it. All Pipable and Interactable dummies will be linked to it.

This is the most common case for the most blocks. If the block is not ConveyorLineSegment and does not have any special needs, it is probably this case.

Important: Autogeneration of conveyor nodes is performed both on server and client compositions separately. Blocks that do have only 2 dummies and inventory may be mistaken for the ConveyorLine on the client. Client does not have InventoryComponent thus it appears as if it was just 2 conveyor dummies. If such an issue could arise, always define your own ConveyorSubgraph.

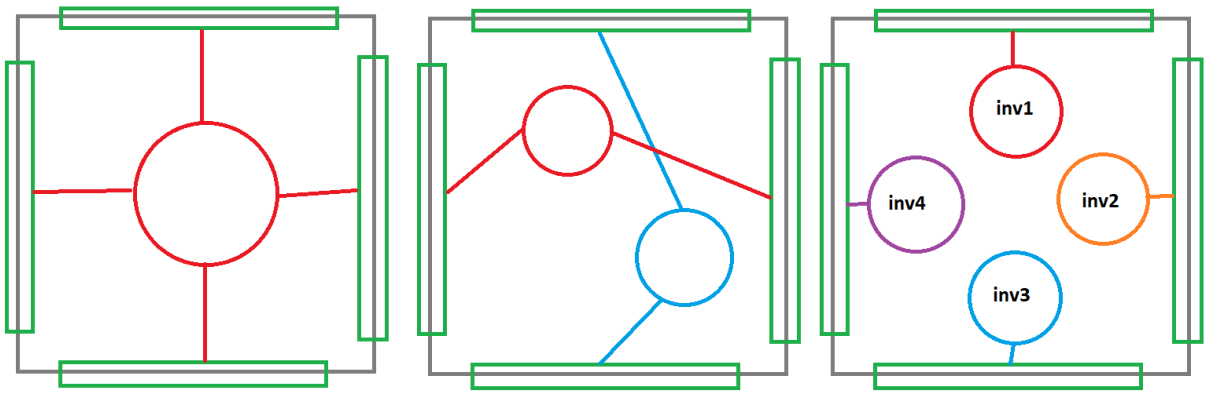
Advanced conveyor block

Users may need a more complex ConveyorSubgraph.
Use cases are for example **2 or 3 way bridges**, blocks that allow for crossing of conveyor pipes without mixing them, **separate inventories** where different conveyor ports of the block could be used to access different inventories but not mixing them.

Nodes

To set up ConveyorSubgraph, nodes must be created, the amount depends on the specific needs. Nodes represent one subunit of the block.

For a trivial conveyor block, one node is enough to represent the whole block. If there is a need to have multiple separate sub-networks within the block, for each of them a node is needed at minimum



Simple ConveyorLineNode | 2 way bridge | separate inventory for each port

In the image above 3 examples are provided. First, the simplest **ConveyorLineNode** with 4 ports is present. It requires only one node and connects all 4 dummies. It could be auto generated. The second case of **2-way bridge** contains 2 nodes and each connects to 2 dummies. Pipes from left and right do not mix with pipes from top and bottom, but top and bottom are connected and so are left and right pipes. **Third example** shows 4 nodes each with their own inventory and connected to their own dummy. The nodes are not mixed in any way and from each port only one inventory can be accessed.

Each node can have inventories assigned using the tags of the components. Filter can be assigned to it.

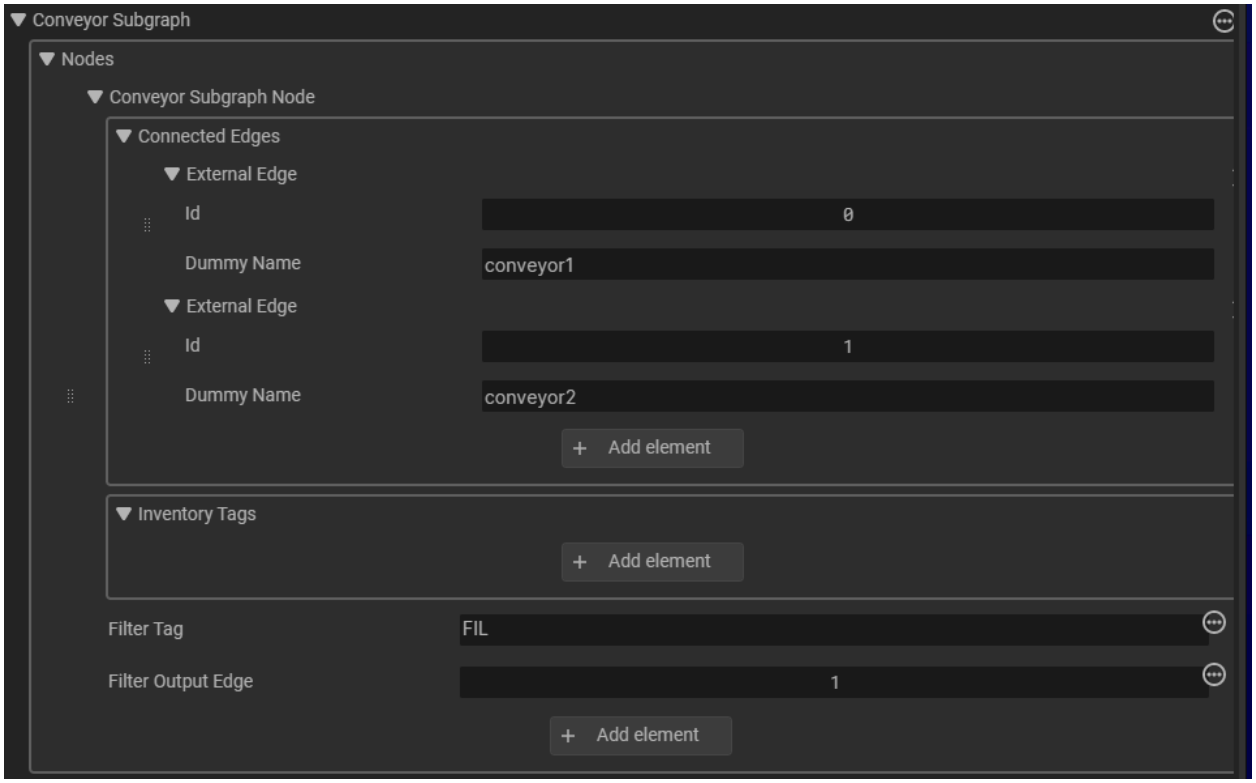
Edges

Each node may have internal and external edges assigned. Internal edges lead from one node to another and connect them together. External edges lead from a node to a Dummy (of the model).

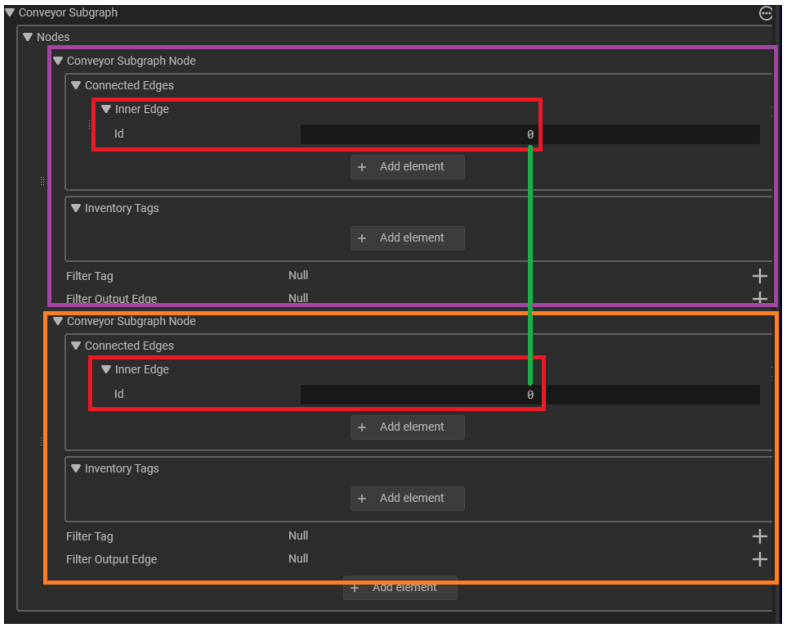
Edges are distinguished by their ID. If two edges within the ConveyorSubgraph have the same ID, they are considered the same edge.

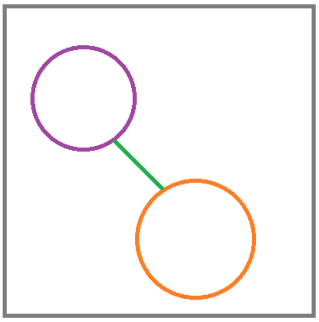
Internal edges must lead between exactly 2 Nodes in the ConveyorSubgraph
External Edges must be present in exactly one Node and must have a dummy of the block assigned to it. The dummy is assigned by its name..

Examples



Example above is of the Filter block that has one node, that connects to dummies **conveyor1** and **conveyor2** with an external edge, has filter with tag **FIL** attached and the edge to dummy **conveyor2** is directed and filtered (in direction from the node outwards)





Example above is of the ConveyorSubgraph with 2 nodes that are connected together using InnerEdge. Because the edges have the same Id, they are the same edge. This will make the nodes connected in the ConveyorSystem.