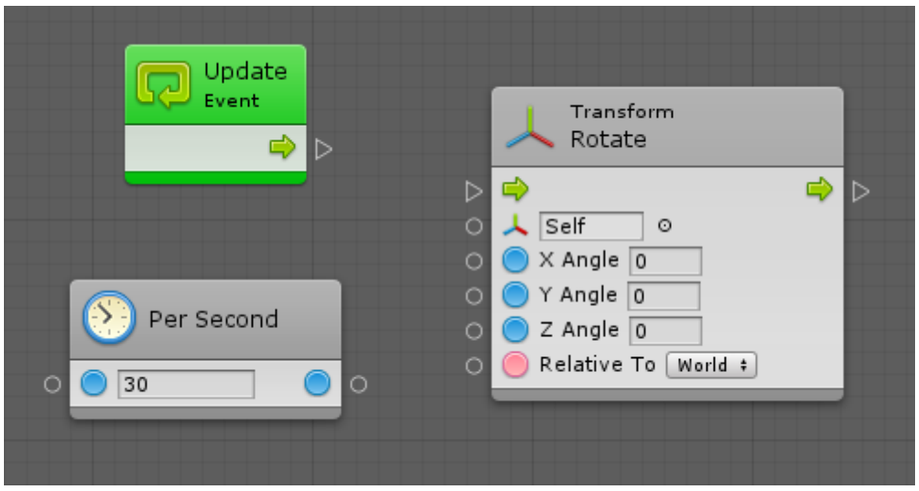
Connections & Relations (/topics/150-connections-relations/)

Let's start by creating a a simple graph with 3 units:

- Update (under Events > Lifetime): an event unit that fires at every frame.
- Per Second (under Math > Scalar): a math unit that returns the input value in a framerate-normalized way.
- Rotate (under Codebase > Unity Engine > Transform): a method unit that rotates the given transform by the specified angles. For this example, we'll use the (X, Y, Z, Relative To) overload.

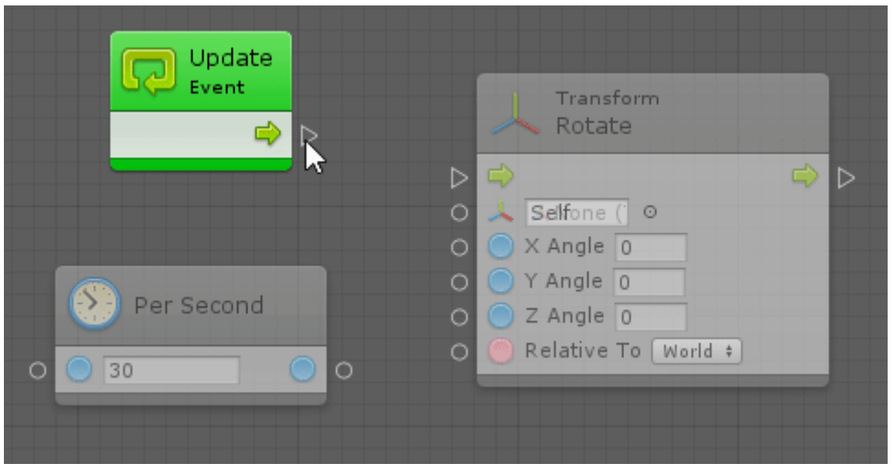


If you disabled the dimming from the toolbar in the last article (Dim), now would be a good time to re-enable it (Dim).

Connections

To create a connection, you can either:

- Click on the first port, then click on the second port.
- Click on the first port and hold, then release while over the second port.



This graph reads as "*at every frame, rotate your own transform in the Y axis at a rate of 30 degrees per second*".

There are two connections:

- A **Control Connection** between the Update event and the Rotate invocation.
- A **Value Connection** between the result of the Per Second node and the Y Angle of the Rotate node.

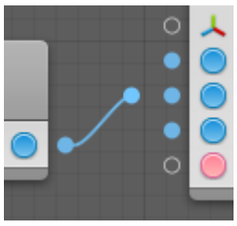
You'll notice that the nodes become fully visible as soon as they are in use.

Highlight

When creating a connection, you'll notice that the compatible ports are highlighted .

You cannot create a connection if the target port is not highlighted.

For example, here, you cannot connect the result of the math operation (a number) to the target (a transform component), because there is no way to convert between a number and a transform component.

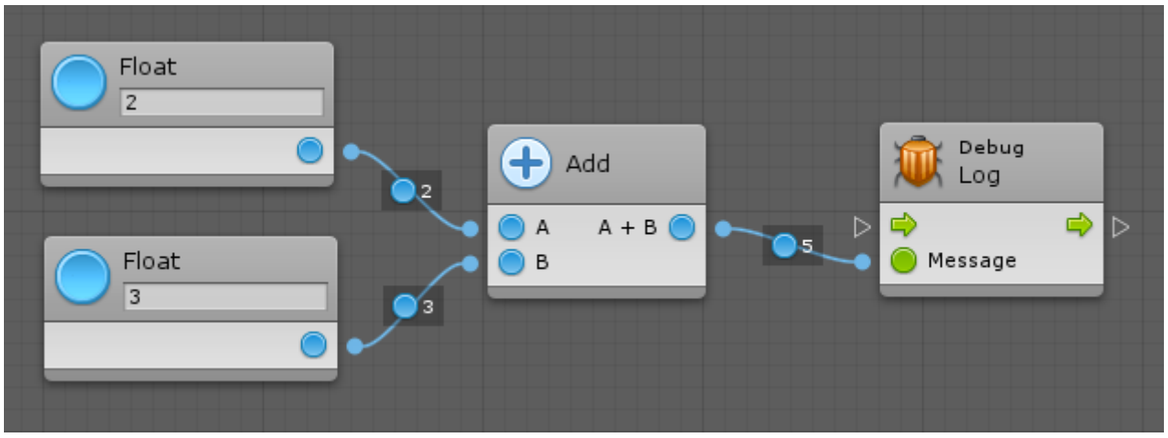


Color Coding

Control connections are always white, whereas value connections are color-coded based on their type. You can find the color of each type in the Types (http://support.ludiq.io/topics/132-types/) article. Any type that doesn't have a color pellet icon will have a **green** connection.

Values

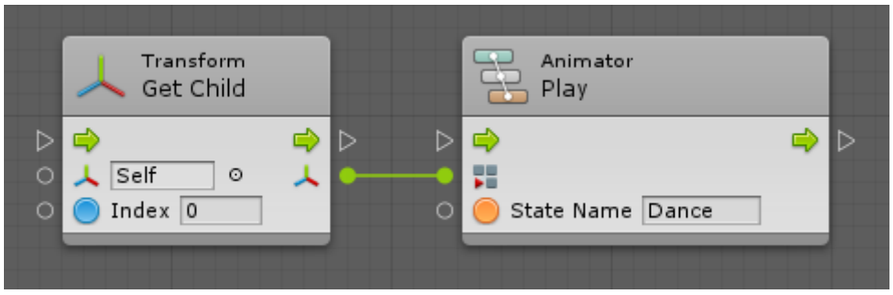
When the `Values` toggle is enabled in the toolbar, Bolt will show the predicted values of the connections whenever possible. At runtime, it will display the last value that traversed this connection. For example, here, because we're using literals, Bolt can anticipate the result of the addition:



Automatic Conversion

Bolt can automatically convert many types to keep your graphs tidy. This happens in the background, so you don't even have to think about it.

For example, here, you don't need to use Get Component between the Transform output and the Animator input, because Bolt will do it automatically.



The following conversions are supported:

- Number to Number (e.g. integer to float and vice versa)
- Base Class to Child Class
- Child Class to Base Class
- Custom Operators (e.g. from Vector 2 to Vector 3)
- Game Object to Child Component
- Component to Parent Game Object
- Component to Sibling Component

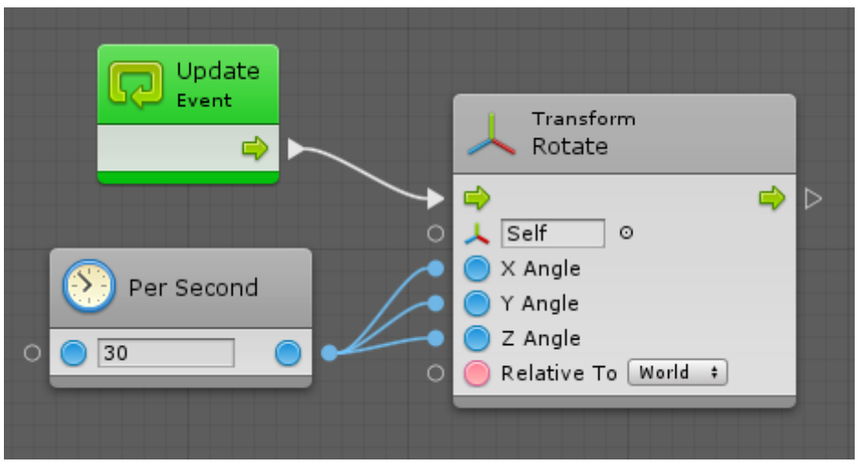
- Enumerable to Array
- Enumerable to List

Bolt also supports boxing and unboxing. That means it will allow you to connect any object type port to any other value port. However, it is your responsibility to ensure that the types are compatible, otherwise you will get an error in play mode.

Multiple Connections

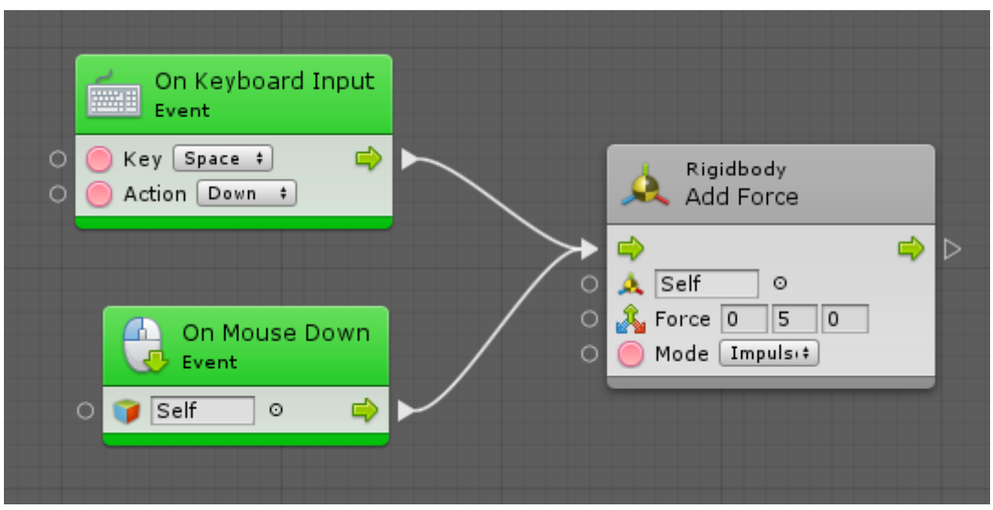
A single port can be connected multiple times for convenience.

For example, you can connect a single value output port to multiple value input ports :

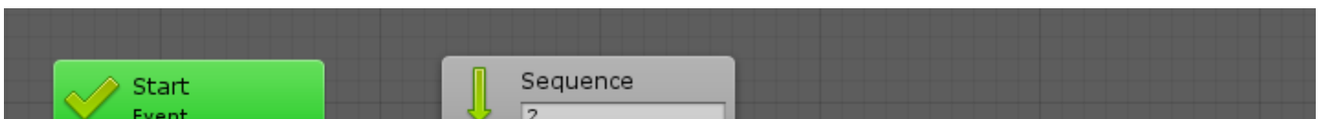


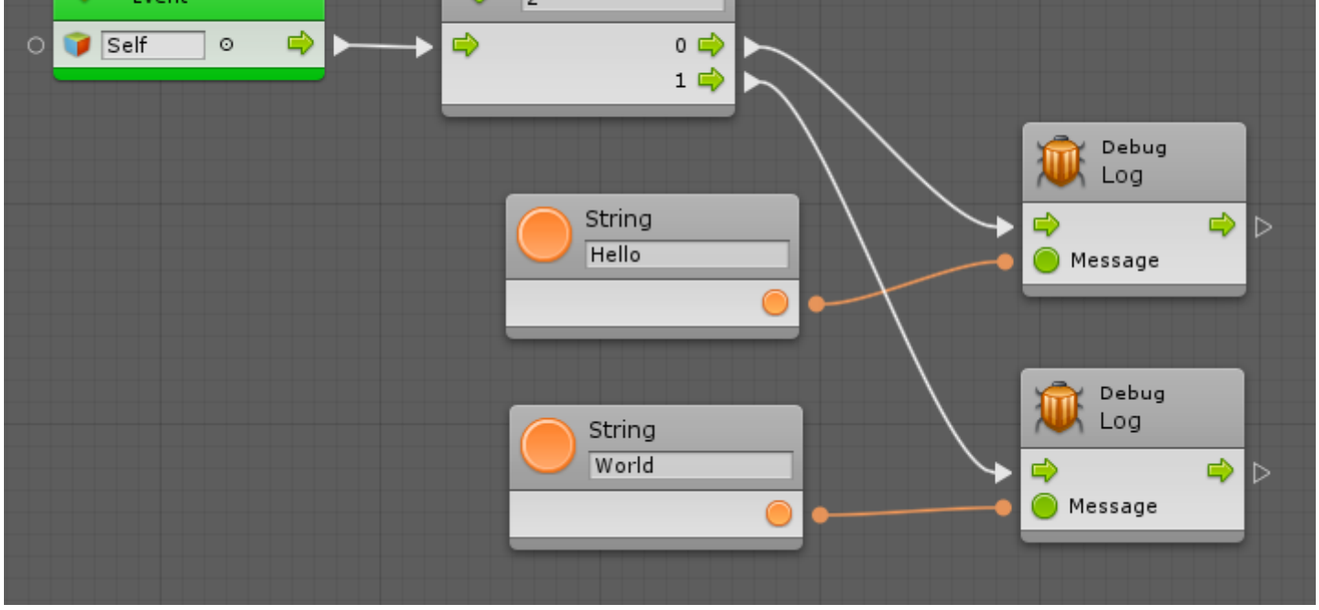
However, you **cannot** connect multiple value output ports to a single value input port , because then it wouldn't be clear which value should be used.

You can also connect multiple control output ports to a single control input port . For example, here, a jump force would be applied either when the player presses space or clicks on the object:



However, you **cannot** directly connect a single control output port to multiple control input ports , because then the order in which the exit units would be executed wouldn't be clear. However, you can use the special **Sequence** node under Control for this purpose:



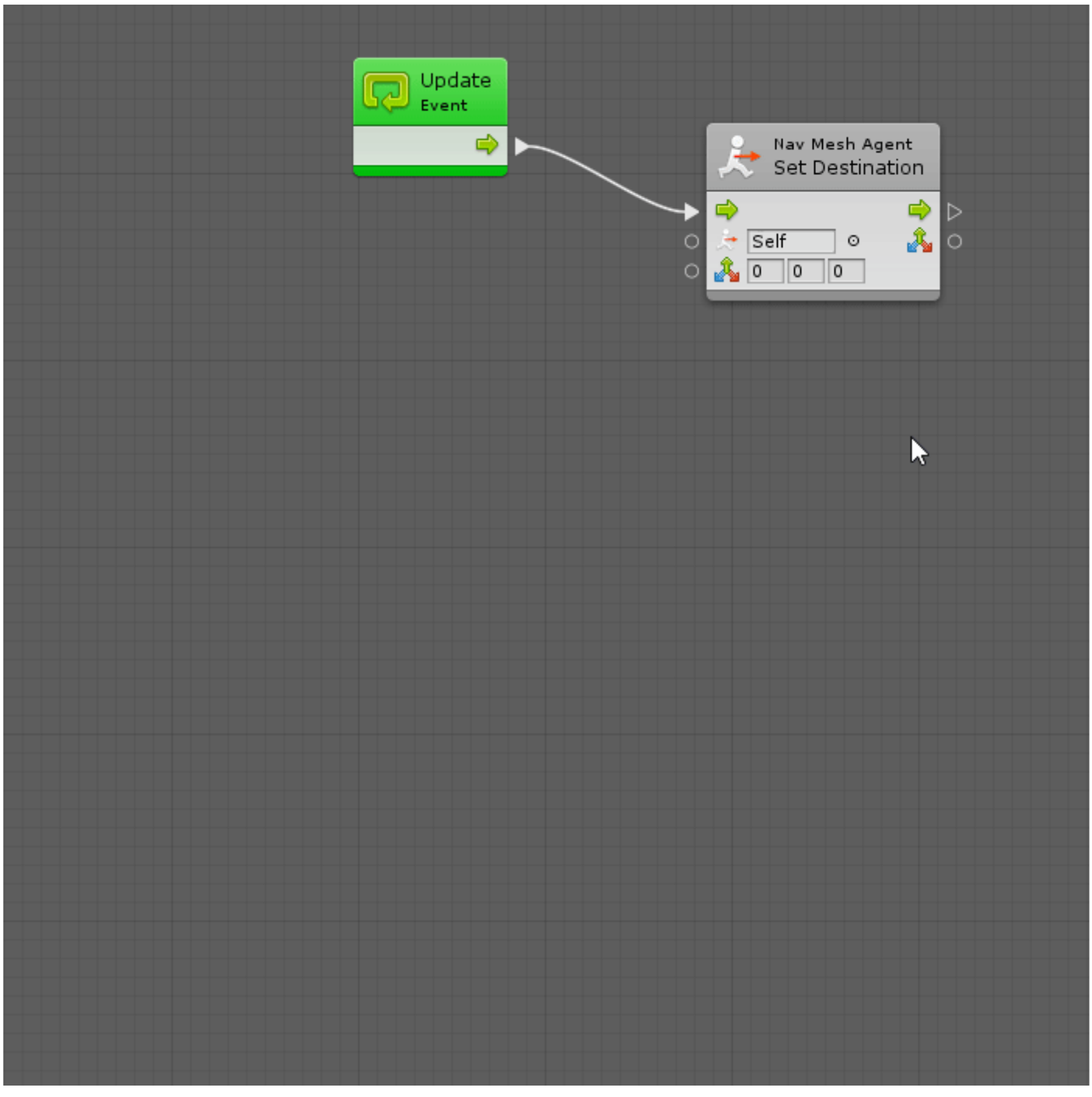


If you want to chain more than two consecutive actions, just use a higher number in the field in the sequence node header.

Contextual Options

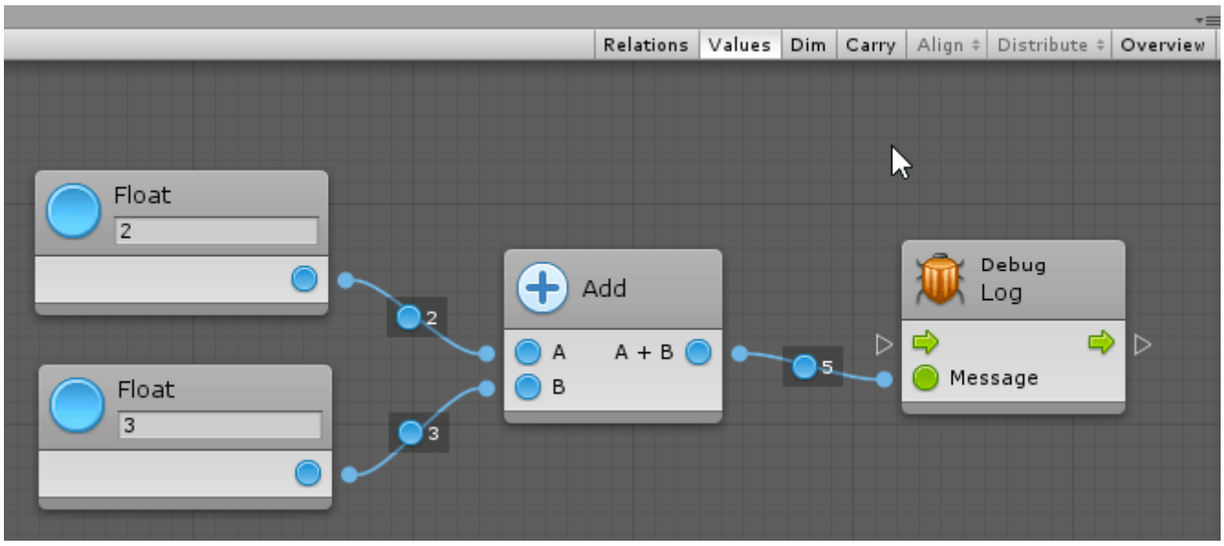
Instead of creating units first then connecting them after, you can start a connection, click on an empty space in the graph, and the fuzzy finder will display new unit options that are compatible with the selected source port. Then, when you choose the new unit, Bolt will automatically connect it to the matching port.

For example, here, we want to get a 3D vector for the destination of our nav mesh agent. The fuzzy finder only shows options that have a Vector 3 output, and automatically creates and connects the node for us:



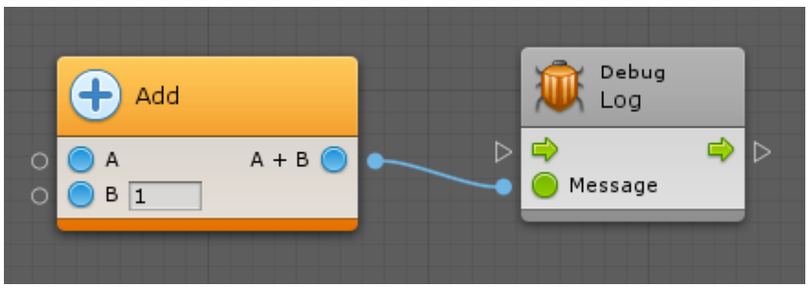
Relations

If you enable the `Relations` toggle in the toolbar, the inner connections of each node will be displayed:

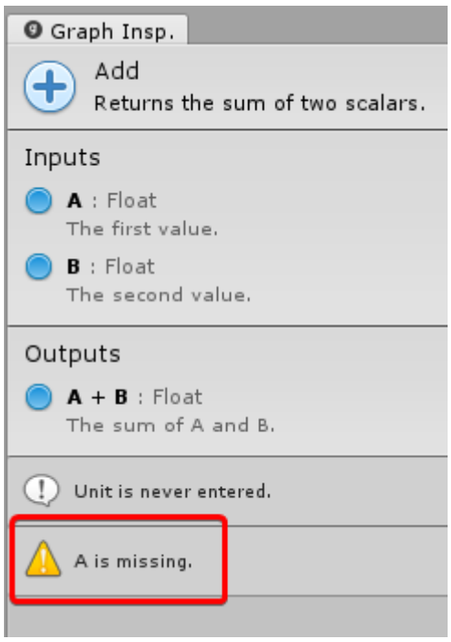


Relations are useful to understand what are the dependencies between each port of a unit. For example, in the above Add unit, you can see that if you want to get the result of A + B, you need to provide a value for A and B. Likewise, you can see that before invoking the Log unit, you should provide a value for its Message input port.

Bolt uses this information in the background for Predictive Debugging (http://support.ludiq.io/topics/153-predictive-live-debugging/). For example, if you tried to get the value of A + B without providing a value for A, the node would show up as orange to indicate that it will fail in play mode:



When that happens, you can use the warnings shown in the graph inspector to know exactly what is missing:



Relations can also help you figure out which ports are required and which ports are optional. For example, in the Get Child unit (under Codebase > Unity Engine > Transform), we can see that we don't actually need to connect the control ports if we just want to get the transform value output.

Note that you cannot edit relations. They are predefined for each type of unit.