

Vuo Manual

Vuo 0.8.0

Contents

1	Introduction	6
2	Quick Start	6
2.1	Install Vuo	6
2.2	Create a composition	7
2.3	Run the composition	7
3	The Basics	8
3.1	The ‘flow’ or execution of events	8
3.2	Nodes, ports, and cables	8
3.2.1	Sending data and events between nodes	9
3.3	Creating a composition	10
3.4	Live coding	11
3.5	Viewing example compositions	11

4	Controlling the Flow of Events	12
4.1	Where events come from	12
4.2	How events travel through a node	13
4.2.1	Input ports	13
4.2.2	Output ports	16
4.3	How events travel through a composition	17
4.3.1	The rules of events	17
4.3.2	Straight lines	17
4.3.3	Splits and joins	18
4.3.4	Multiple triggers	18
4.3.5	Feedback loops	19
4.3.6	Summary	21
4.4	Common patterns for event flow	21
4.4.1	Checking if a condition is met	21
4.4.2	Choosing which data and events to send through the composition	22
4.4.3	Choosing which parts of the composition to execute	23
4.4.4	Setting data with one trigger and using it with another	23
4.4.5	Merging events from multiple triggers	24
4.4.6	Executing slow nodes in the background	24
4.5	Controlling the buildup of events	25

5	Understanding Data	25
5.1	Setting a constant value for a port	25
5.2	Using the default value of a port	26
5.3	Sending the same data to multiple ports	26
5.4	Using drawers	27
5.5	Storing information	28
5.6	Types of information	28
5.6.1	Converting between types	29
5.6.2	Using generic types	29
5.7	Interacting with the environment	31
6	Interfacing with Applications	31
6.1	Published ports	31
6.2	Protocols	32
7	The Vuo Editor	33
7.1	The Node Library	33
7.1.1	Docking and visibility	33
7.1.2	Node names and node display	34
7.1.3	Node Documentation Panel	34
7.1.4	Finding Nodes	35
7.2	Working on the canvas	35
7.2.1	Putting a node on the canvas	35
7.2.2	Drawing cables to create a composition	36
7.2.3	Copying and pasting nodes and cables	36

7.2.4	Deleting nodes and cables	36
7.2.5	Modifying and rearranging nodes and cables	37
7.2.6	Viewing a composition	37
7.2.7	Publishing Ports	38
7.2.8	Using a protocol for published ports	38
7.3	Running a composition	38
7.3.1	Starting and stopping a composition	39
7.3.2	Firing an event manually	39
7.3.3	Troubleshooting a running composition	39
7.4	Exporting a composition to an application	39
8	The Built-in Nodes	40
8.1	Rendering graphics and video	40
8.1.1	Vuo Coordinates	40
8.2	Processing and playing audio	41
8.3	Communicating over a network	42
8.4	Controlling compositions with devices	42
8.5	Controlling window displays and accessing screen values	42
9	The Command-Line Tools	42
9.1	Installing the Vuo SDK	43
9.2	Getting help	43
9.3	Rendering a composition on the command line	43
9.4	Building a composition on the command line	44
9.5	Running a composition on the command line	44
9.6	Exporting a composition to an application on the command line	45

10 Adding Nodes to Your Node Library	45
10.1 Installing a node	45
10.2 Creating your own node	46
11 Troubleshooting	46
11.1 Common problems	46
11.1.1 My composition isn't working and I don't know why.	46
11.1.2 Some nodes aren't executing.	47
11.1.3 Some nodes are executing when I don't want them to.	48
11.1.4 Some nodes are outputting the wrong data.	48
11.1.5 The composition's output is slow or jerky.	48
11.2 General tips	49
12 Keyboard Shortcuts	50
12.1 Working with composition files	50
12.2 Controlling the composition canvas	51
12.3 Creating and editing compositions	51
12.4 Running compositions	52
12.5 Application shortcuts	52

1 Introduction

Vuo is a realtime visual programming environment designed to be both incredibly fast and easy to use. With Vuo, artists and creative people can create audio, visual, and mixed multimedia effects with ease.

This manual will guide you through the process of installing Vuo and creating your first composition. Then it will show you the details of all the pieces of a composition and how to put them together. It will explain how to create and run compositions with the Vuo Editor and, as an alternative, the command-line tools. It will show you how to make Vuo do even more by adding nodes to your Node Library.

Many of the compositions in this manual are available from the within the Vuo Editor (File->Open Example). It may help to open these example compositions in the Vuo Editor and run them as you work through the manual.

For more resources to help you learn Vuo, see our [support page](#) on vuo.org.

For more information on what you can do with Vuo, see our [features page](#) and [roadmap](#) on vuo.org.

If you have any feedback about this manual, please [let us know](#). We want to make sure you find the manual helpful.

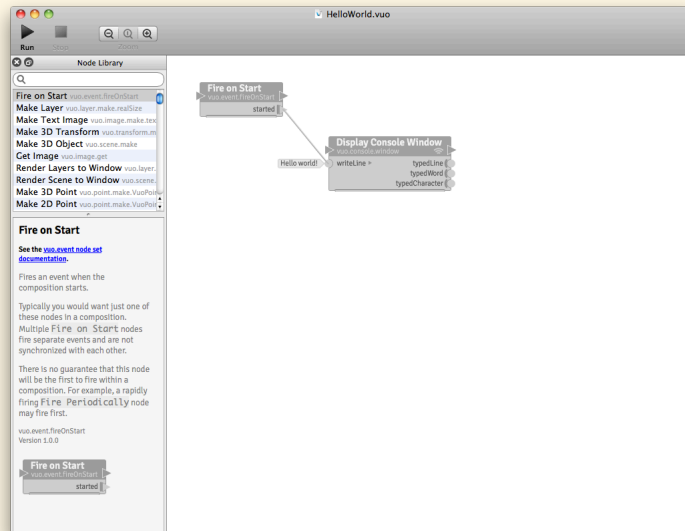
Welcome to Vuo. Get ready to create!

2 Quick Start

2.1 Install Vuo

- Go to <https://vuo.org/user> and log in to your account
- Click the “Download” tab
- Under the “Vuo Editor” section, download the Vuo Editor
- Uncompress the ZIP file (double-click on it in Finder)
- Move the Vuo Editor application to your Applications folder

2.2 Create a composition



Let's make a simple composition. It will just pop up a window with the text “Hello World!”

1. Open the Vuo Editor application in your Applications folder and the Vuo Editor will appear. The Node Library will be on the left, and a canvas with a **Fire on Start** node will be on the right.
2. In the Node Library, find the **Display Console Window** node. You can do this either by scrolling down the Node Library list or by typing part of the node title (such as “console”) into the search bar at the top of the Node Library. Drag the node onto the canvas.
3. Draw a cable from the **started** port of the **Fire on Start** node to the **writeLine** port of the **Display Console Window** node. You can do this by pressing the mouse button while over the **started** port, dragging the cable that appears, and releasing the mouse button while over the **writeLine** port.
4. Double-click on the **writeLine** port circle. This pops up a text box.
5. Type “Hello world!” in the text box and hit Return. This closes the text box.

2.3 Run the composition

Now let's run your composition.

1. Click the Run button (or go to Run > Run).
2. When you're finished admiring the “Hello world!” text, click the Stop button (or go to Run > Stop).

3 The Basics

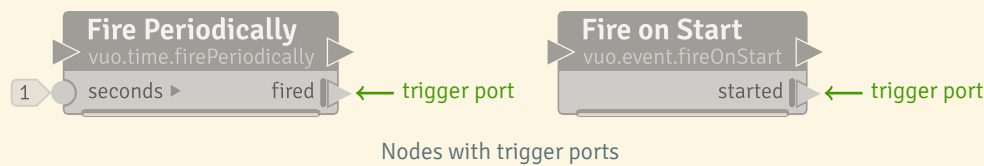
3.1 The ‘flow’ or execution of events

Vuo compositions are driven by **events**. Events are what cause nodes in your composition to **execute**. Without events, your composition won’t do anything!

Events come from **trigger ports**. Based on how you connect these trigger ports to other nodes, you have complete control over the timing of events in your composition.

The trigger port has a thick line on its left side. (We’ll explain that [later](#).)

One example of a trigger port is the **Fire Periodically** node’s **fired** port. Another is the **Fire on Start** node’s **started** port.



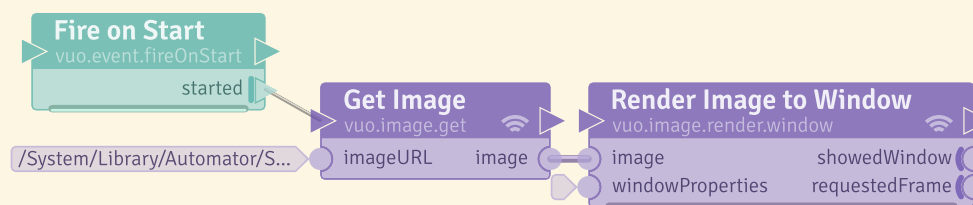
See the chapter [Controlling the Flow of Events](#) for more detail on event flow.

3.2 Nodes, ports, and cables

The rectangular boxes in your composition are **nodes**. Each **node** performs a task, like counting numbers or displaying a window or periodically firing events. Nodes are your tools for creating — they’re the building blocks of compositions.

When you look at a node, the title (large text along the top) tells you the task that the node performs. You can get more details by clicking on the node. This will bring up the node’s documentation in the Node Documentation Panel, which will show an in-depth description of the node.

Nodes talk to each other by sending data and events through **cables** plugged into **ports**. Data and events flow from the **output port** of one node, through a cable, to the **input port** of another node.



An example composition, DisplayImage.vuo (under vuo.image), with three **nodes** and two **cables**

In the composition above, an event from the **Fire on Start** node travels to the **Get Image** node, which fetches the image from a specific URL. The image data and event then travel to the input port **image** of the **Render Image to Window** node. This node will display the image in a window.

Note for
Quartz Composer users

A **node** in Vuo is analogous to a **patch** in Quartz Composer. Unlike Quartz Composer, which typically executes each patch once per video frame, nodes in Vuo can be executed whenever they receive an event — whether it’s 60 per second, 44,100 per second, or 1 per year.

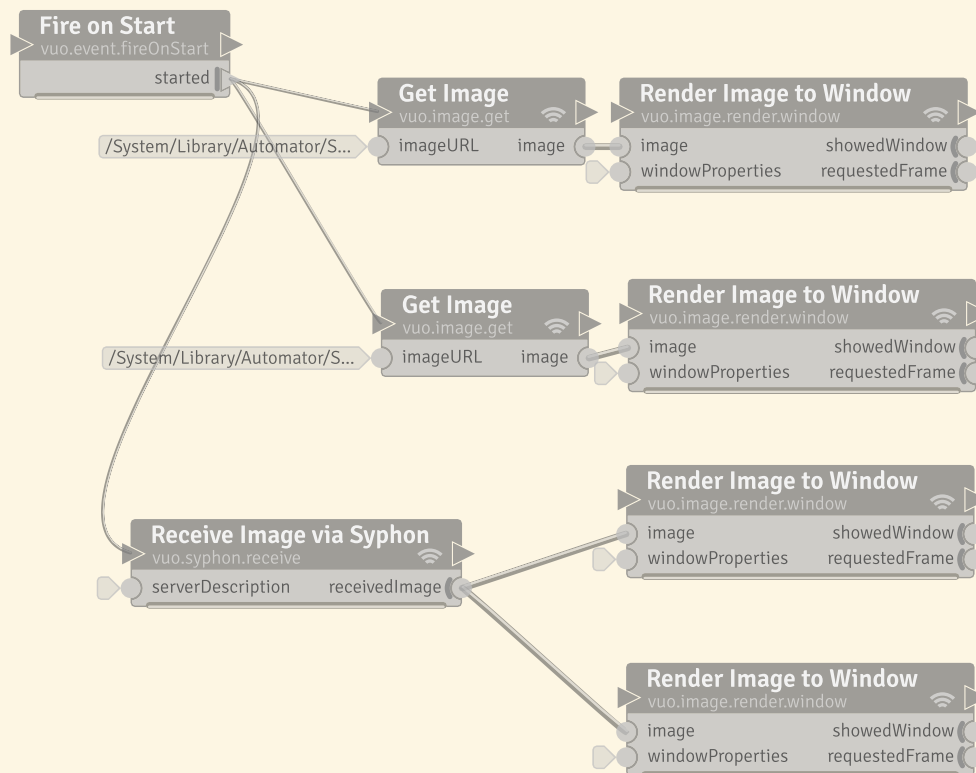
Note for
text programmers

A **node** in Vuo is analogous to a **class instance method** that takes a list of inputs and returns a list of outputs.

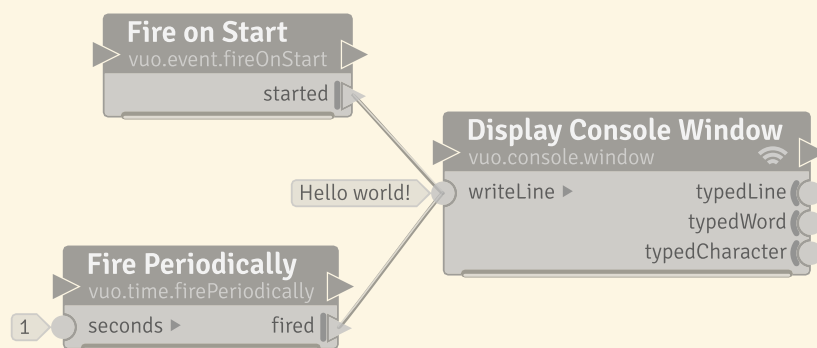
3.2.1 Sending data and events between nodes

All cables carry events. In the composition above, an example of a cable that carries an **event-only** is the cable into the **Get Image** node, while an example of the a **data-and event** cable is the cable from **Get Image** to **Render Image to Window**. We'll talk about differences in **input ports** in the section on [How events travel through a node](#).


Event-only cables are thin. **Data-and-event cables** are thicker than event-only cables.



Multiple data-and-event and event-only cables can be connected to an output port.



Multiple event-only cables, but only one data-and-event cable, can be connected to an input port.

 Note for
Quartz Composer users

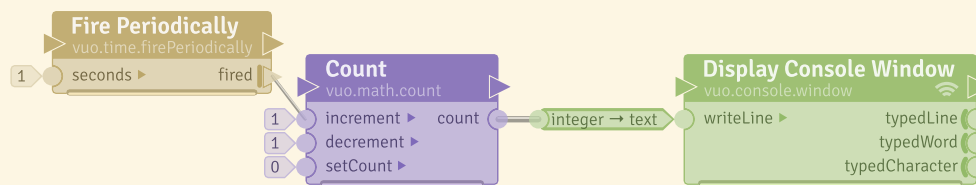
Unlike Quartz Composer, you may create as many or few windows as you like. This composition will create 4 separate windows.

You can connect a data-and-event output port to a data-and-event input port, but you can't directly connect a data-and-event output port to an event-only input port without a special type converter. See the section [Types of information](#). You *can* connect an event-only output port to any type of data-and-event input port.

3.3 Creating a composition

The easiest way to make a composition is in the Vuo Editor. Once you've installed Vuo (see the chapter [Quick Start](#)), you can find the Vuo Editor in your Applications folder.

See the chapter [The Vuo Editor](#) for more information on the Vuo Editor.



The example composition, Count.vuo, under the vuo.math examples

Above is a composition that counts. It displays a blank window, then, every one second, it writes a number upon the window: 1, 2, 3, 4, ... etc. Let's take a closer look.

The **Fire Periodically**, **Count**, and **Display Console Window** nodes each perform a separate task. Information always flows from the left side to the right side of a node. Inputs are on the left, outputs on the right. **integer** → **text** is a **type converter**; its job is to convert (or translate) information from one **type** to another. Types of data are covered in the chapter [Understanding Data](#).

This composition begins with the **Fire Periodically** node and ultimately flows to the **Display Console Window** node. Information travels between nodes by exiting output ports, flowing through cables, and entering input ports.

The **Fire Periodically** node's only task is to tell other nodes when it's time to perform their function. It does this by *firing* events out of its **fired** port. How often it fires an event is dictated by the value present at its **seconds** port. It sends events out its **fired** port, then along the cable to the **Count** node's **increment** port.

When an event from the **Fire Periodically** node flows through the cable and hits the **Count** node, it tells **Count** that it's time to execute. The **Count** node keeps track of a count. When an event hits its **increment** port, the node adds 1 to its count. The count starts out at 0, becomes 1 after the first event, 2 after the second event, and so on. The **Count** node sends the new count *and* the event out its **count** port, along the cable, to the **integer** port of the **integer** → **text** type converter.

When you take a cable from **count** you will see that the **writeLine** is highlighted. This indicates that although the two nodes process different types of data, the Editor will insert a type converter for you. The **integer** → **text** type converter takes the value in its **integer** port (a number) and converts it to text. That's because the next node, **Display Console Window**, only works with text, not numbers.

The final node, **Display Console Window**, creates a blank white window and writes the count upon it.

In summary:

- **Fire Periodically** node fires events every 1 second, which flow along the cable into **Count**.
- **Count** receives the events and outputs these events plus their corresponding numbers.
- **integer** → **text** converts the numbers to text and sends the events and numbers into **Display Console Window**
- **Display Console Window** creates a blank window and displays the numbers upon it, every one second.

3.4 Live coding

In Vuo you can change your composition while it's running. We'll use the example composition, **Count.vuo**, explained above. Start the composition again. While it's running, click on the cable from the **fired** port to the **Count** node's **increment** port. This will highlight it.

Either right click, hit **Delete** or use the Editor's menu **Edit > Cut** or **Edit > Delete** to delete the cable. Notice how the composition stops outputting the count, since there are no events going into the **Count** node.

Now draw a new cable from the **Fired** port to the **decrement** port. Notice how the composition resumes and starts decrementing the count, all while the composition is running.

You can change data, rearrange cables, and even add nodes while a composition is running.

3.5 Viewing example compositions

One way to learn how nodes work is to view Vuo's example compositions. To find the example compositions, use **File > Open Example** to see a list of node sets. You can then hover over a node set to see relevant examples. Hovering over **vu.console**, for example, will display the example compositions **CalculateTip.vuo** and **HelloWorld.vuo**. Clicking on an example composition will open it in the Vuo Editor.

To learn more about an example composition, open the composition and go to **Edit > Composition Information**. This displays the composition’s description, which includes instructions on how you can interact with it.

To see a list of descriptions for all example compositions for a node set, look in the node set documentation. In the Editor’s Node Library (**Window > Show Node Library**), find a node that belongs to that node set. For example, to learn more about the `vu.math` example compositions, use the keyword “math” in the Node Library search window to find the nodes in that node set, or click on a node in the library that contains “math” in its class name. Clicking on any node in the node set will show that node’s description in the Node Documentation Panel. The description will contain a link to the node set documentation. Using that link will display some general information about that node set, as well as descriptions of the associated example compositions.

4 Controlling the Flow of Events

Events are what make things happen in a composition. As you get to know Vuo, you’ll be able to look at a composition and imagine how an event comes out of a trigger port, flows through a cable into a node’s input port, and either gets blocked or flows through the node’s output ports and into the next cables. The previous section gave an overview of how that works. This section describes the process in detail.

4.1 Where events come from

Each event is fired from a **trigger port**, a special kind of output port on a node.



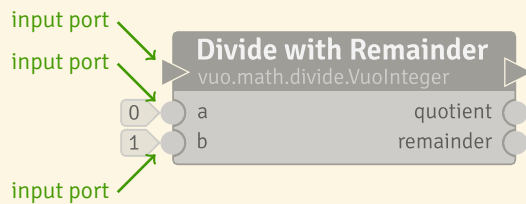
Some trigger ports fire events in response to things happening in the world outside your composition. For example, the **Receive Mouse Moves** node’s trigger port fires an event each time the mouse is moved. The **Play Movie** node’s trigger port fires a rapid series of events (for example, 30 per second), so that you can display images in rapid sequence as a movie. Similarly, the **Render Scene to Window** node’s `requestedFrame` trigger port fires a rapid series of events, so that you can use these events to display scenes in rapid sequence as an animation.

Other trigger ports fire events in response to things happening within the composition. For example, the **Fire on Start** node’s trigger port fires an event when the composition starts. The **Fire Periodically** node’s trigger port fires events at a steady rate while the composition is running. A node’s trigger port can even fire in response to an event received by the node, as happens with the **Spin Off Event** node. (However, this is a *different* event than the one that was received by the node. For more information, see the section on [Executing slow nodes in the background](#).)

4.2 How events travel through a node

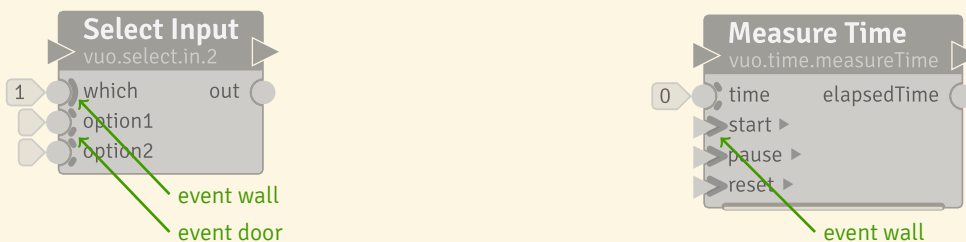
An event can come into a node through cables connected to one or more of its input ports. When an event reaches the node's input ports, the node executes, and it may or may not send the event through its output ports.

4.2.1 Input ports



An **input port** is the location on the left side of the node where you can enter data directly, connect a data-and-event cable, or connect an event-only cable. When an event arrives at an input port, it causes the node to execute and perform its function based on the data present at the node's input ports.

4.2.1.1 Event walls and doors Some nodes, like the two nodes shown below, have input ports that block an event. This means the node will execute, but the event associated with that data won't travel through any output ports, with the exception of the **done port**. Done ports are explained in the [Output ports](#) section. Event blocking is useful when you want part of your composition to execute in response to events from one trigger port but not events from another trigger port, or when you're creating a feedback loop. (For more information, see the sections [How events travel through a composition](#) and [Common patterns for event flow](#).)

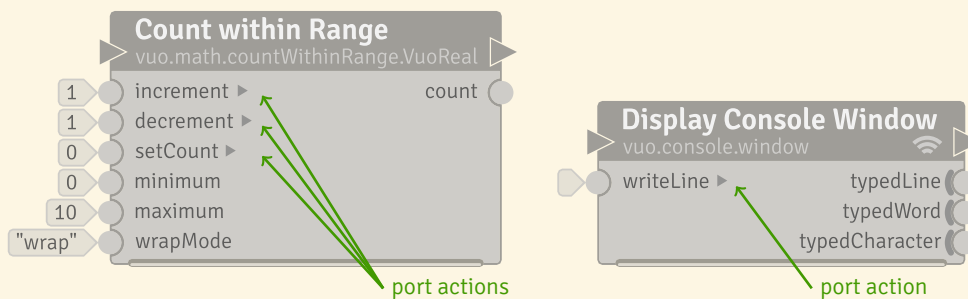


Ports that always block events have a solid semi-circle (like the **which** port above) or a solid chevron (like the **start** port above). This is called an **event wall**. The node must receive an event from another port without an event wall for the results of the node's execution to be available to other nodes. The event itself, even it arrives via an input port with an event wall, is available via the **done port**.

Tip: The event wall is visually placed inside the node to indicate that the event gets blocked inside the node (as it executes) — rather than getting blocked before it reaches the node.

Ports that sometimes block events have a broken semi-circle (like the **option1** port above) or chevron. This is called an **event door**. Event doors are useful when you want to take events from a trigger port and filter some of them out or route them to different parts of the composition. For example, in the **Select Input** node, the value at the **which** port will determine whether the data or data-and-event at **option1** or **option2** will be transmitted to the **out** port.

4.2.1.2 Port actions Some input ports cause the node to do something special when they receive an event. In the **Count within Range** node shown below, the **increment**, **decrement**, and **setCount** ports each uniquely affect the count stored by the node — upon receiving an event, they increment the count, decrement the count, or change the count to a specific number. Likewise, in the **Display Console Window** node, the **writeLine** input port does something special when it receives an event — it writes a line of text to the console window. Each of these ports has a *port action*.



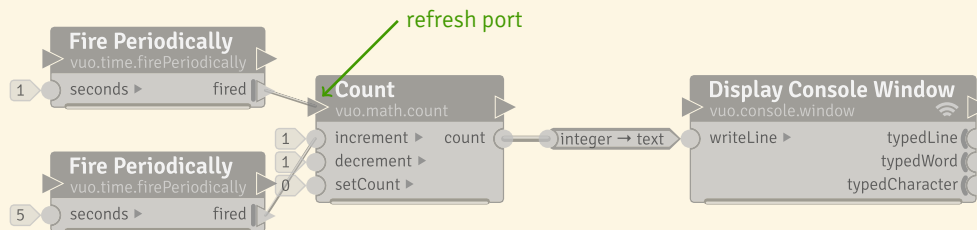
If an input port has a **port action**, then the node does something different when that input port receives an event than it does when any other input port receives an event. What counts as “something different”? Either the node outputs different data (immediately or later) or the node affects the world outside the composition differently.

Looking again at the **Count within Range** node, you can see that the node has some input ports with port actions and some without. For the ports without port actions — **minimum**, **maximum**, and **wrapMode** — the node will output the same number regardless of whether the event causing the node to execute has hit one of these ports. The node uses the data from these ports and doesn’t care if they received an event. For each of the ports with port actions, however, it makes a difference whether the event has hit the port. The **increment** port, for example, only affects the count if the event came in through that input port.

Rather than affecting the node’s output data, as in the **Count within Range** node, the **Display Console Window** node’s port action affects the world outside the composition. (For more about nodes that affect the world outside the composition, see the section [Interacting with the environment](#).) When the **writeLine** input port receives an event, it doesn’t affect the data coming out of the node’s output ports. Rather, it affects what you see in the console window.

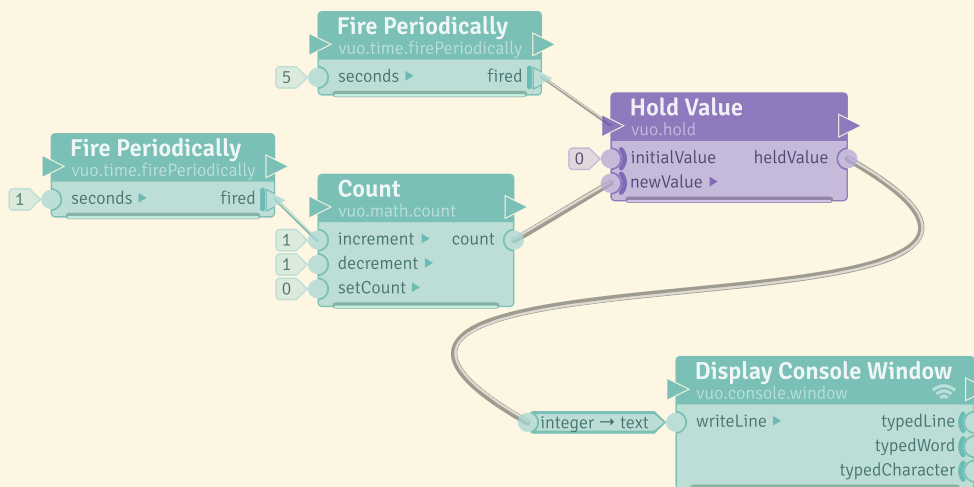
You can recognize an input port with a port action by the little triangle to the right of the port name. In Vuo, the triangle shape symbolizes events. The little triangle for the port action reminds you that this port does something unique when it receives an event.

4.2.1.3 Refresh ports Every node has a built-in event-only input port called the **refresh port**. The purpose of this port is to execute the node without performing any port actions.



A composition, CountSometimes.vuo, using a **refresh port**

For example, this composition above shows how you can use the refresh port to get the **Count** node's current count without incrementing or decrementing it. When the lower **Fire Periodically** node fires an event, **Display Console Window** writes the incremented count. When the upper **Fire Periodically** node — which is connected to the **Count** node's refresh port — fires an event, the count stays the same. **Display Console Window** writes the same count as before.



An example composition, CountandHold.vuo (under vuo.hold), using a **Hold Value** node

On other nodes, like **Hold Value**, the refresh port is the only input port that transmits events to any output ports. The **Hold Value** node lets you store a value during one event and use it during later

events. This composition shows how you can use a **Hold Value** node to update a value every 1 second and write it every 5 seconds.

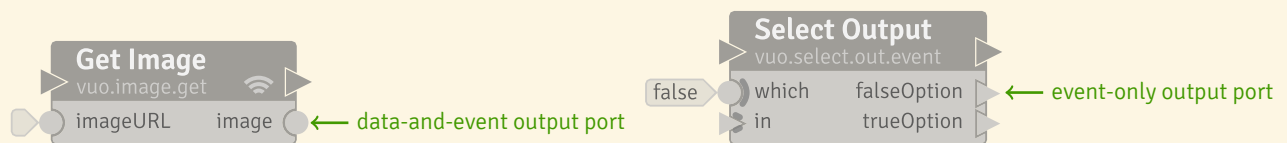
When the left **Fire Periodically** node executes, the count, as a data-and-event, is transmitted to the **Hold Value** node. The **Display Console Window** node doesn't execute because, when the event arrives at the **Hold Value** node, it is blocked.

When the upper **Fire Periodically** node executes, the count stored in the **Hold Value** node travels to the **Display Console Window** node and gets written.

Tip: Refresh ports are useful for allowing event flow out of nodes that have event walls. The event entering the **refresh port** can travel to downstream nodes, while an event encountering an event wall cannot.

4.2.2 Output ports

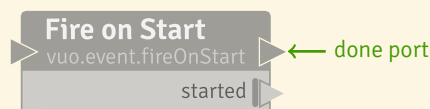
When an event executes a node, the event can travel to downstream nodes using the **output ports**. Like input ports, output ports can be data-and-event or event-only.



4.2.2.1 Trigger ports Although trigger ports can *create* events, they never *transmit* events that came into the node through an input port (hence the thick line to the left of each trigger port — an event wall), nor do they cause any other output ports to emit events. When a trigger port fires, no event comes out the done port.

Nodes that contain trigger ports will execute as described in the **Input ports** section when they receive an event, including transmitting events through the **done port**.

4.2.2.2 Done ports Every node has a built-in event-only output port called the **done port**. The done port outputs an event every time the node executes — even if the incoming event was blocked by an event wall or event door.



Tip: Done ports are useful for allowing event flow out of nodes that have no other output ports.

4.3 How events travel through a composition

Now that you’ve seen how events travel through individual nodes, let’s look at the bigger picture: how they travel through a composition.

4.3.1 The rules of events

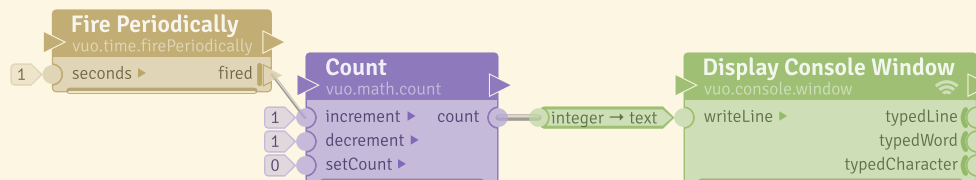
Each event travels through a composition following a simple set of rules:

1. **An event travels forward through cables and nodes.** Along each cable, it travels from the output port to the input port. Within each node, it travels from the input ports to the output ports (unless it’s blocked). An event never travels backward or skips around.
2. **One event can’t overtake another.** If multiple events are traveling through the same cables and nodes, they stay in order.
3. **An event can split.** If there are multiple cables coming out of a trigger port or other output ports, then the event travels through each cable simultaneously.
4. **An event can rejoin.** If the event has previously split and gone down multiple paths of nodes and cables, and those paths meet with multiple cables going into one node, then the split event rejoins at that node. The node waits for all pieces of the split event to arrive before it executes.
5. **An event can be blocked.** If the event hits an event wall or door on an input port, then although it will cause the node to execute, it may not transmit through the node.
6. **An event can travel through each cable at most once.** If a composition could allow an event to travel through the same cable more than once, then the composition is not allowed to run. It has an infinite feedback loop error.

Let’s look at how those rules apply to some actual compositions.

4.3.2 Straight lines

The simplest event flow in a composition is through a straight line of nodes, like the composition below.



In this composition, the **fired** trigger port fires an event every 1 second. Each event travels along cables and through the **Count** node, then the integer-to-text type converter node, then **Display Console Window** node. The event is never split or blocked.

Note for
Quartz Composer users

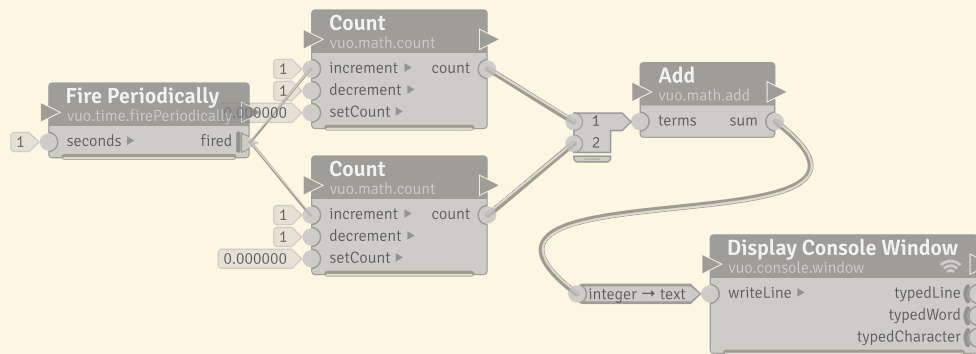
Quartz Composer provides less control than Vuo does over when patches execute. Patches typically execute in sync with a framerate, not in response to events. Patches typically execute one at a time, unless a patch has been specially programmed to do extra work in the background.

Note for
text programmers

This section is about Vuo’s mechanisms for control flow and concurrency.

4.3.3 Splits and joins

When you run a composition in Vuo, multiple nodes can execute at the same time. This takes advantage of your multicore processor to make your composition run faster.

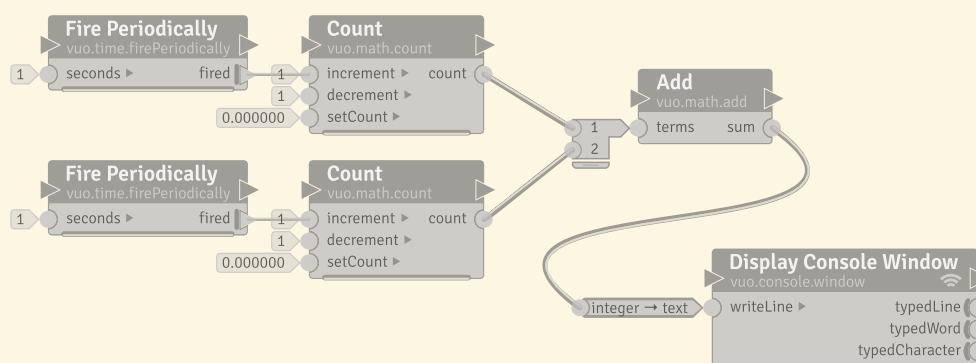


In this composition, the two **Count** nodes are independent of each other, so it's OK for them to execute at the same time. When the **Fire Periodically** node fires an event, the upper **Count** node might execute before the lower one, or the lower one might execute before the upper one, or they might execute at the same time. It doesn't matter! What matters is that the **Add** node waits for input from both of the **Count** nodes before it executes.

The **Add** node executes just once each time **Fire Periodically** fires an event. The event branches off to the **Count** nodes and joins up again at **Add**.

If the **Fire Periodically** node fires an event, and then fires a second event before the first has made it through the composition, then the second event waits. Only when the **Display Console Window** node has finished executing for the first event do the **Count** nodes begin executing for the second event.

4.3.4 Multiple triggers

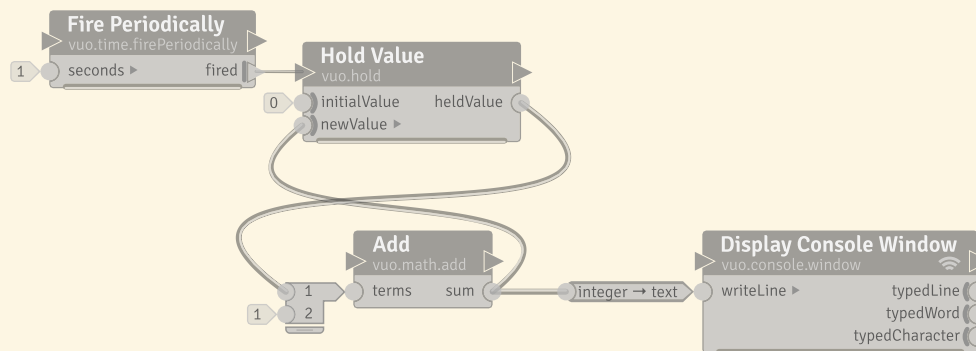


In this composition, the **Add** node executes each time either **Fire Periodically** node fires an event. If one of the **Add** node's inputs receives an event, it doesn't wait for the other input. It goes ahead and executes.

If the two **Fire Periodically** nodes fire an event at nearly the same time, then the **Count** nodes can execute in either order or at the same time. But once the first event reaches the **Add** node, the second event is not allowed to overtake it. (Otherwise, the second event could overwrite the data on the cable from **Add** to **Display Console Window** before the first event has a chance to reach **Display Console Window**.) The second event can't execute **Add** or **Display Console Window** until the first event is finished.

4.3.5 Feedback loops

You can use a **feedback loop** to do something repeatedly or iteratively. An iteration happens each time a new event travels around the feedback loop.



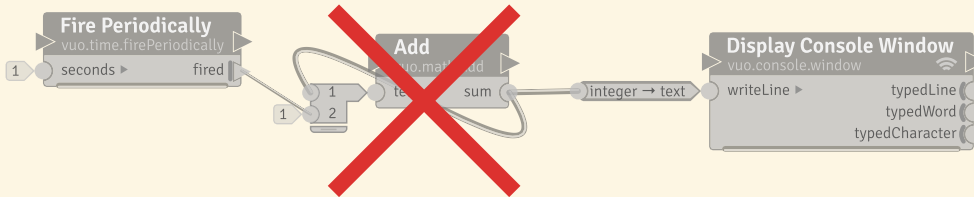
A composition, CountWithFeedback.vuo, showing a **feedback loop**.

The above composition prints a count upon a console window: 1, 2, 3, 4, . . .

The first time the **Fire Periodically** node fires an event, the inputs of **Add** are 0 and 1, and the output is 1. The sum, as a data-and-event, travels along the cable to the **Hold Value** node. The new value is held at the **newValue** port, and the event is blocked, as you can see from its event wall; **Hold Value** doesn't transmit events from its **newValue** port to any output ports.

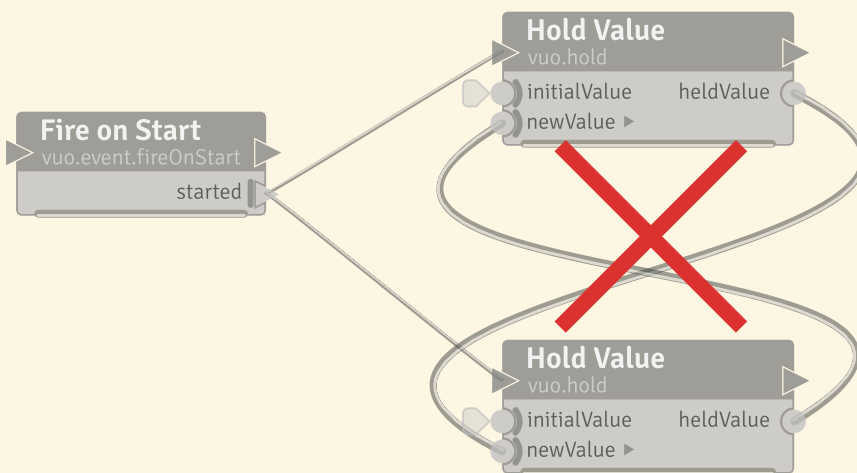
The second time the **Fire Periodically** node fires an event, the inputs of **Add** are 1 (from the **Hold Value** node) and 1. The third time, the inputs are 2 and 1. And so on.

4.3.5.1 Infinite feedback loops Each event is only allowed to travel once through a feedback loop. When it comes back to the first node of the feedback loop, it needs to be blocked by a walled input port. If your composition isn't set up like this, then Vuo will tell you there's an **infinite feedback loop** and won't allow your composition to run.



The above composition is an example of an infinite feedback loop. Any event from the **Fire Periodically** node would get stuck forever traveling in the feedback loop from the **Add** node's **sum** port back to the **item1** port. Because there's no event wall in the feedback loop, there's nothing to stop the event. Every feedback loop needs a node like **Hold Value** to block events from looping infinitely.

4.3.5.2 Deadlocked feedback loops In most cases, an event needs to travel through all of the cables leading up to a node before it can reach the node itself. (The one exception is the node that starts and ends a feedback loop, since it has some cables leading into the feedback loop and some coming back around the loop.) A problem can arise if the nodes and cables in a composition are connected in a way that makes it impossible for an event to travel through all the cables leading up to a node before reaching the node itself. This problem is called a **deadlocked feedback loop**. If your composition has one, Vuo will tell you so and won't allow your composition to run.



This composition is an example of a deadlocked feedback loop. Because the top **Hold Value** node could receive an event from the **Fire on Start** node through the cable from the bottom **Hold Value** node, the top **Hold Value** node needs to execute after the bottom one. But because the bottom **Hold Value** node could receive an event from the **Fire on Start** node through the cable from the top **Hold Value**

node, the bottom **Hold Value** node needs to execute after the top one. Since each **Hold Value** node need to execute after the other one, it's impossible for an event to travel through the composition. To fix a deadlocked feedback loop, you need to remove some of the nodes or cables involved.

4.3.6 Summary

You can control how your composition executes by controlling the flow of events. The way that you connect nodes with cables — whether in a straight line, a feedback loop, or branching off in different directions — controls the order in which nodes execute. The way that you fire and block events — with trigger ports and with event walls and doors — controls when different parts of your composition will execute.

Each event that's fired from a trigger port has its own unique identity. The event can branch off along different paths, and those paths can join up again at a node downstream. When the *same* event joins up, the joining node will wait for the event to travel along all incoming paths and then execute just once. But if two *different* events come into a node, the node will execute twice. So if you want to make sure that different parts of your composition are exactly in sync, make sure they're using the same event.

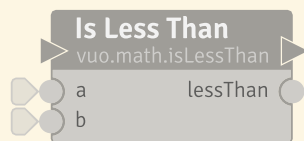
4.4 Common patterns for event flow

This section describes a few approaches to controlling event flow that you're likely to use in many different compositions.

4.4.1 Checking if a condition is met

Vuo has nodes that can evaluate whether a statement is **True** or **False**. True and false are called **Boolean values**.

The following is an example of a node that outputs a Boolean value, by determining if “a” is less than “b.”



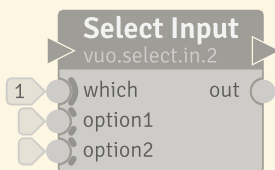
You can execute downstream nodes based on whether an output is true or false. Logic nodes are often used with nodes that route data, where the logic node acts as an “if” and the node that routes data as a “then do this, or if not, do that.”

Tip: With a type converter node, True can be converted to “1” and False can be converted to “0.”

4.4.2 Choosing which data and events to send through the composition

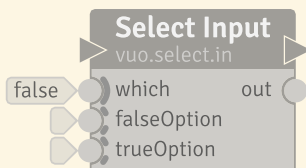
A **Select Input** node lets you select a value from different input ports and route it to the output. A **Select Output** node lets you select an input value to route to different output ports. There are two different classes of Select nodes. The first uses the values 1 and 2 to control how the node functions, while the second uses Boolean true or false values. The first set uses a class name beginning with “vuo.select.in.2.Vuo” or “vuo.select.out.2.Vuo” while the second set uses a class name beginning with “vuo.select.in.Vuo” or vuo.select.out.Vuo.” With these two different classes, you can control your composition based on what makes sense with respect to the rest of your composition.

When the **Select Input** node executes, it looks at what is present at the **which** port and outputs the value found at the corresponding port below it. For example, if the value “1” is present at the **which** port, whatever value is present at the **option1** port will be output.

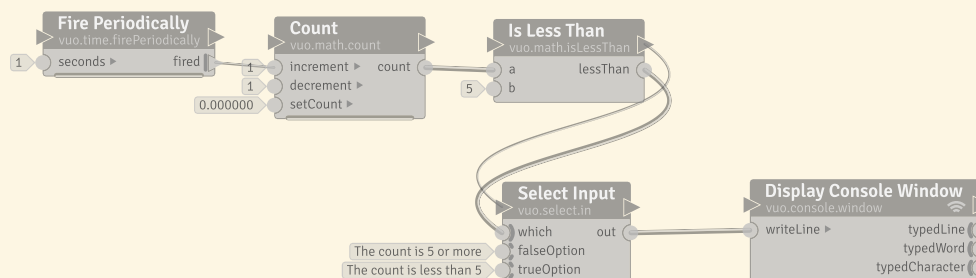


Notice that the **which** port has an event wall, as shown by the solid semi-circle. In order to get the node to output an event through the **out** port, you need to send an event to the selected input port or the refresh port. It won't work if you just send an event to the **which** port.

Here's an example of the node using Boolean values to control how it functions:



Here's a composition using both a logic node and a **Select Input** node. The composition will write “The count is less than five” four times, before “The count is 5 or more” begins to be written.



Note for
Quartz Composer users

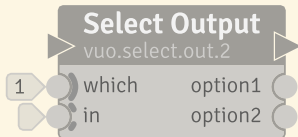
Vuo's **Select Input** node is similar to Quartz Composer's Multiplexer patch. Vuo's **Select Output** node is similar to Quartz Composer's Demultiplexer patch.

Note for
text programmers

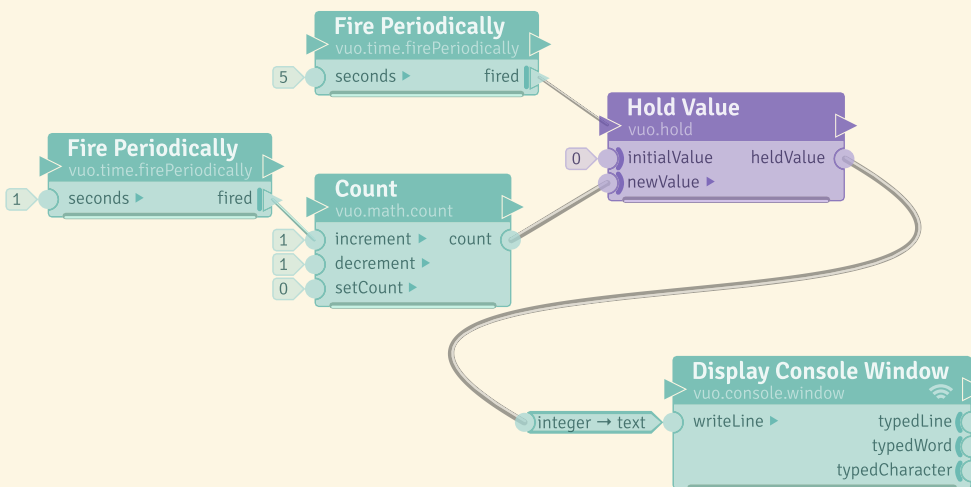
Vuo's **Select Input** and **Select Output** are similar to if/else or switch/case statements.

4.4.3 Choosing which parts of the composition to execute

A **Select Output** node lets you route an input value to one of several outputs. Similar to the **Select Input**, you pick an output port using the **which** input port. Here “1” will send the information from the **in** to **option1**, while an input of “2” will send the information to **option2**. You can see here that it is important to notice if the **which** is expecting a Boolean “True” or “False,” or if it is expecting a number “1” or “2.” Vuo provides switches that use both numbers and Boolean values.



4.4.4 Setting data with one trigger and using it with another



An example composition, **Count and Hold** (under **vuo.hold**), showing nodes executing at different rates

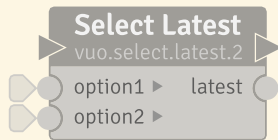
This composition writes a count upon the console window every 5 seconds. The count updates every 1 second.

The **Hold Value** node prevents the count from being written each time it’s updated by the 1-second **Fire Periodically** node.

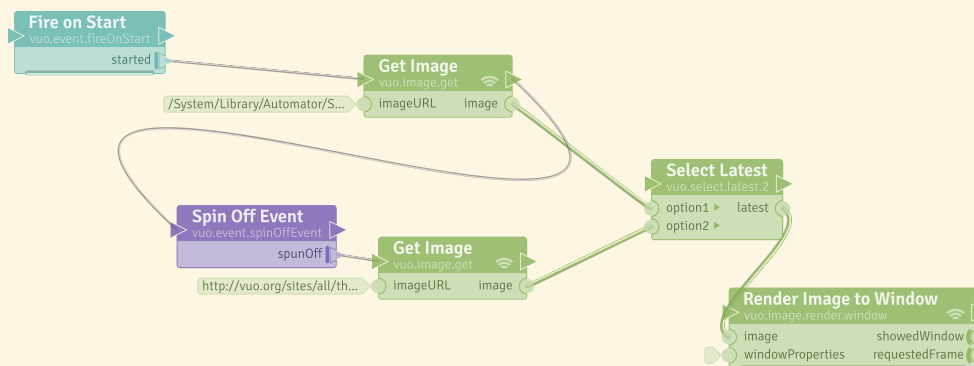
(Note: Every 5 seconds, when the two **Fire Periodically** nodes fire at nearly the same time, it’s unpredictable whether the count will be written before or after it’s incremented.)

4.4.5 Merging events from multiple triggers

A **Select Latest** node lets you select the latest data and event to arrive at the node. If an event comes into both **option1** and **option2**, then **option1** is used.



4.4.6 Executing slow nodes in the background



An example composition, `LoadImagesAsynchronously.vuo` (under `vuo.event`), has nodes execute in the background

This example shows how a composition can do some work in the background (asynchronously). It displays one image while downloading another image from the internet, then displays the second image.

The **Spin Off Event** node is what allows the image to download in the background. When an event reaches the **Spin Off Event** node, the **Spin Off Event** node fires a new event. Because it's a new event instead of the same old event, other parts of the composition can go on executing without having to wait on the event.

Let's take a more detailed look. When the **Fire on Start** node fires an event, the event travels to the **Spin Off Event** node (where it stops) and through the upper **Get Image** node, **Select Latest**, **Place Image in Scene**, and **Render Scene to Window**. All of these nodes execute without waiting for the lower **Get Image** node. Meanwhile, the **Spin Off Event** node fires a new event, the lower **Get Image** node downloads the image, and eventually the new event travels onward through **Select Latest** and **Render Image to Window**.

4.5 Controlling the buildup of events

What if a trigger port is firing events faster than the downstream nodes can process them? Will the events get queued up and wait until the downstream nodes are ready (causing the composition to lag), or will the composition skip some events so that it can keep up? That depends on the trigger port's **event throttling** setting.

Each trigger port has two options for event throttling: **enqueue events** or **drop events**. If enqueueing events, the trigger port will keep firing events regardless of whether the downstream nodes can keep up. If dropping events, the trigger port won't fire an event if the event would have to wait for the downstream nodes to finish processing a previous event (from this or another trigger port).

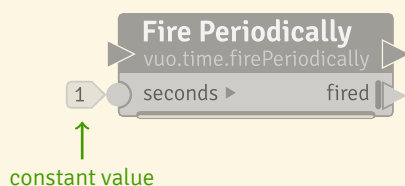
Each of these options is useful in different situations. For example, suppose you have a composition in which a `Play Movie` node fires events with image data and then applies a series of image filters. If you want the composition to display the resulting images in real-time, then you'd probably want the `Play Movie` node's trigger port to drop events to ensure that the movie plays at its original speed. On the other hand, if you're using the composition to apply a video post-processing effect and save the resulting images to file, then you'd probably want the trigger port to enqueue events.

When you add a node to a composition, each of its trigger ports may default to either enqueueing or dropping events. For example, the `Play Movie` node's trigger port defaults to dropping events, while each of the `Receive Mouse Clicks` node's trigger ports defaults to enqueueing events.

In the Vuo Editor, you can right-click on a trigger port and go to the "Set Event Throttling" menu to view or change whether the port enqueues or drops events.

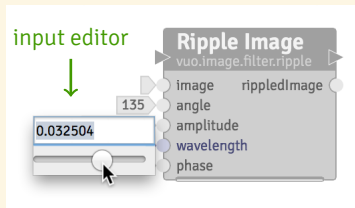
5 Understanding Data

5.1 Setting a constant value for a port



Instead of passing data through a cable, you can give a data-and-event input port a **constant value**. A constant value is "constant" because, unlike data coming in from a cable, which can change from time to time, a constant value remains the same unless you edit it.

For many types of data (such as integers and text), you can edit a constant value by double-clicking on the constant value attached to an input port. This will pop up an **input editor** that lets you change the constant value. (If nothing happens when you double-click on the constant value, then the Vuo Editor doesn't have an input editor for that data type. To change the data for the input port, you have to connect a cable.)



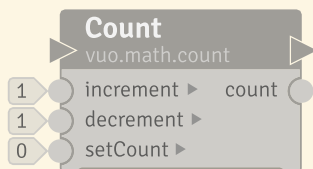
When using an input editor to edit text, you can enter text that contains multiple lines (line breaks) by composing your text in a basic text editor (such as TextEdit), then copying and pasting it into the input editor.

If you edit a constant value for a node's input port, the node will use the new port value the next time it executes. Setting a constant value won't cause the node to execute.

5.2 Using the default value of a port

When you add a node to a composition, each input port has a preset constant value called its **default value**. The default value for an input port is the same for all nodes of a given type. For example, the **increment** port of all **Count** nodes defaults to 1. The port stays at its default value until it receives an event.

If you disconnect a data-and-event cable to a port that previously had a constant value, then the port goes back to its previous value. If you did not set a constant value, it goes back to its default value.

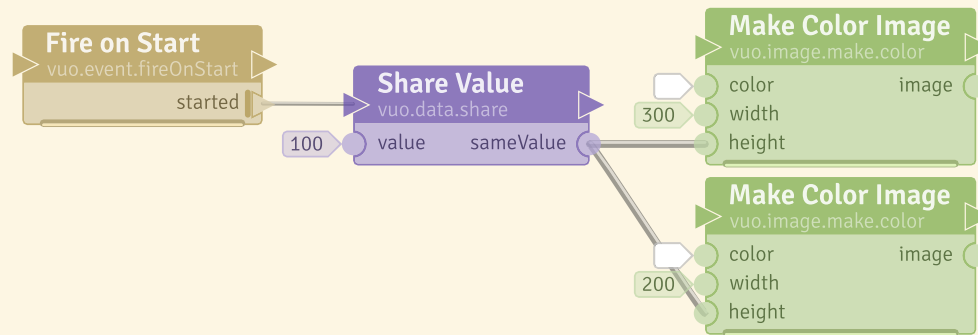


5.3 Sending the same data to multiple ports

What if you want to set the same value on multiple input ports? For example, suppose you want to draw several images, all having the same height. One way to accomplish this would be to go one by one to each node's height input port, open the input editor, and set it to the height you want. But

what if you later decide to change the height? It would be better to set the height in one place and have it affect all nodes.

You can do this with the **Share Value** node, as illustrated here. When the **Share Value** node receives an event, it sends the value in its input port (100) through its output port to all connected nodes.

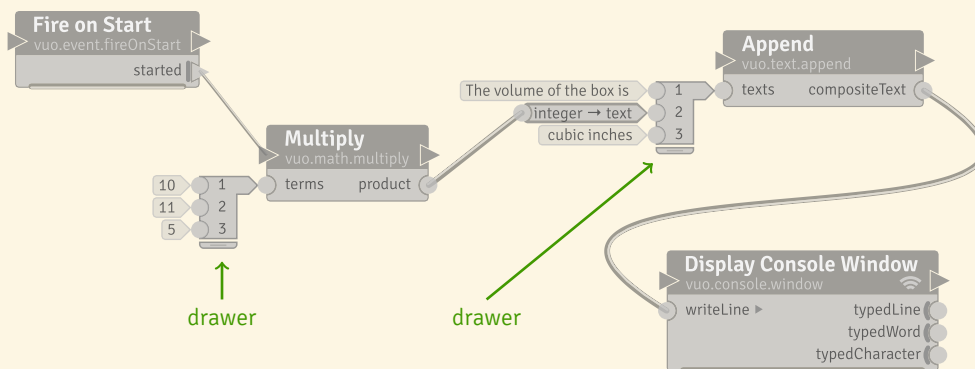


5.4 Using drawers

Some input ports accept a list of values rather than a single value. If no cable is connected, a list input port has a special attachment called a **drawer** that lets you input each item of the list separately. For each input port within the drawer, you can either connect a cable to it or set its constant value.

In most cases, you can change the number of items in the drawer. (The one exception is the drawer on the **Calculate** node's **values** input port, which changes automatically when you edit the **expression** input port's constant value.) To change the number of items in a drawer, you can either drag on its handle (the bar along the bottom of the drawer) or right-click on the drawer and select **Add Input Port** or **Remove Input Port**.

In the following composition, the **Multiply** node and the **Append** node have drawers.

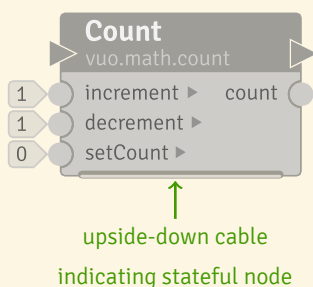


A composition that takes the height, length, and width of a box and calculates its volume

When you edit a drawer's constant values or change the number of drawer items, this affects the attached list input port in the same way as if you had edited a constant value on the list input port itself. For example, in the composition above, the **Multiply** node's **terms** input port contains the list 10, 11, 5. When the **Multiply** node receives an event from **Fire on Start**, it calculates the product of those numbers, 550, and sends that through its output cable.

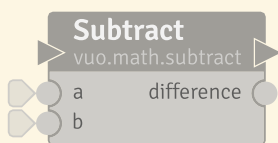
5.5 Storing information

Some nodes have **state**. They remember information from previous times they were executed. An example of a **stateful** node is **Count**. If a **Count** node is told to increment, its state (the count) changes.



Stateful nodes have a thick bar along the bottom that looks like an upside-down **data-and-event cable**. This symbolizes that the node's state data is kept after the node executes, and used next time it executes.

A **stateless** node doesn't remember anything about previous times it was executed. If you give it the same inputs, it'll always give you the same outputs.



Stateless nodes have a thin bottom border.

5.6 Types of information

Nodes are sensitive to **data type**. Vuo works with various types of data, such as integer (whole) numbers, real (decimal) numbers, Boolean (logical) values, text, 3D points, images, and more.

For example, the **Count Characters** node's **text** input port has data type text, and its **characterCount** output port has data type integer. If you see a **Count Characters** node that has a cable connected to its **characterCount** port, then you can know that the port at the other end of the cable must also have type integer.

5.6.1 Converting between types

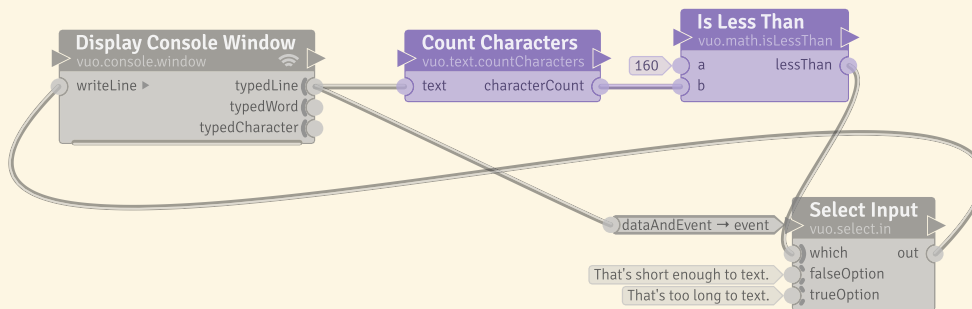
What if you want to connect the **Count Character** node's integer output port to a text input port? If you start to drag a cable from the **characterCount** port, any ports in the composition that the cable can connect to are highlighted. This includes not just ports of type integer, but also ports of type text. If you drag the cable to a text input port and release the mouse, then a type converter node will be automatically added to convert the integer data to text data.

A **type converter node** is just a node that happens to have a single input port of one data type and a single output port of another data type. Whenever you want to connect a port of one data type to a port of another data type, you can try drawing the cable to find out if any type converters can be inserted automatically. Or you can search the Node Library for a node that has an input port of the type that you want to convert from and an output port of the type that you want to convert to.

For some types, there's more than one way to do the conversion. For example, if you want to convert from a real number to an integer, you can round the real number to the nearest integer, round it down to the integer below, or round it up to the integer above. If you try to connect a cable from a real-number output port to an integer input port, a menu will pop up allowing you to pick the conversion you want to use.

Vuo has a special set of type converters for data-and-event cables to remove (discard) the data from an event. If you want to convert a data-and-event cable to an event-only cable, use a **Discard Data from Event** type converter.

You can see type converters used in previous compositions in this manual, such as in the section [Creating a composition](#) with the Count composition.



An example composition, Check Sms Length (under vuo.text), showing a type converter discarding data from an event

5.6.2 Using generic types

Some nodes can work with many different types of data. For example, a **Hold Value** node could hold an integer, an image, a 3D point... or really, any type of data. Rather than cluttering up the Node

Note for
Quartz Composer users

The Quartz Composer equivalent to a **type-converter** is represented as a darker red line in QC. The benefit of exposing these conversions is greater control over how your data is interpreted.

Library with a separate **Hold Value** node for each data type, Vuo lists a single **Hold Value** node that can be made to work with any type.

When you drag a **Hold Value** node from the Node Library onto the canvas, each of the node's ports has a **generic data type**. This means that the data type for these ports hasn't been decided yet. Right now, each of the **Hold Value** node's ports has the potential to connect to any type of port in the composition. You can see that a port is generic by hovering over it with the mouse and reading the port popover that pops up.

When you connect one of the **Hold Value** node's ports to a non-generic port, then all of the **Hold Value** node's generic ports change to non-generic ports with the same type as the connected port. For example, if you connect the **Hold Value** node's **heldValue** port to a **Count Character** node's **text** port, then all of the **Hold Value** node's generic ports change to text ports, matching the **text** port.

Another way to change a generic port to a non-generic port is to right-click on the port, go to the **Set Data Type** menu that pops up, and choose a data type.

If you want to change the port back to generic, you can right-click on the port and select the **Revert to Generic Data Type** menu item. (This will delete any cables between non-generic ports and ports changed back to generic.)

Some generic nodes can work with several different types of data, but not all types. For example, the **Add Points** node can work with 2d points, 3d points, and 4d points, but not text or images. When you drag an **Add Points** node from the Node Library onto the canvas, its **sum** port is generic, but it can only connect to a port of type 2d point, 3d point, or 4d point. You can see the list of compatible types for a generic port by looking at the port popover.

Some generic nodes are automatically turned into non-generic nodes when first created. For example, when you drag an **Add** node from the Node Library onto the canvas, its ports are automatically changed from generic to real numbers, because real numbers are usually a suitable choice for the **Add** node. But if you want to work with integers instead, you can use the **Revert to Generic Data Type** menu and then the **Set Data Type** to change the ports to integers.

You can connect a generic port to another generic port, as long as they work with compatible types of data. For example, you can connect a **Hold Value** node to an **Add** node, since the **Hold Value** node is compatible with any type. However, you can't connect an **Add** node to an **Add Points** node, since the **Add** node is only compatible with integers and reals while the **Add Points** node is only compatible with 2d points, 3d points, and 4d points.

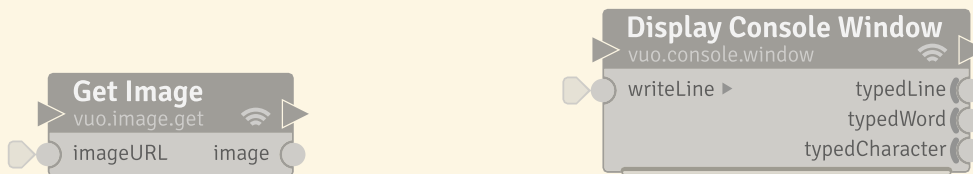
A generic node can have ports that each work with a different generic type. For example, the **Get Message Values** node in the **vuio.osc** node set has several output ports with independent generic types. You can change one output port's generic type to integer and another output port's generic type to text, for example. You can see if a node's generic ports use the same generic type or different generic

types by looking at the port popover. The port popover displays a name for the port’s generic type, such as “generic #1” or “generic #2”. If two generic ports have the same name for their type (including if they’re on separate nodes connected by cables), then changing the generic type of one port will also change the other port. If two generic ports have different names for their type, then changing one will not affect the other.

5.7 Interacting with the environment

Some nodes interact with the world outside the composition — such as windows, files, networks, and devices.

Nodes that bring information into the composition from the outside world and/or affect the outside world are called **interface** nodes. An example of an interface node is the **Get Image** node, which can download an image from the Internet. Another example is the **Display Console Window** node, which reads text typed in a console window and writes text to that window.



Interface nodes have three radio arcs in the top-right corner, symbolizing that they send or receive data with the world outside your composition.

Note for Quartz Composer users

Instead of Vuo’s interface and non-interface nodes, Quartz Composer has an execution mode for each patch: provider, consumer, or processor. A patch’s execution mode not only indicates how it interacts with the outside world, but also controls when it executes and whether it can be embedded in macro patches.

6 Interfacing with Applications

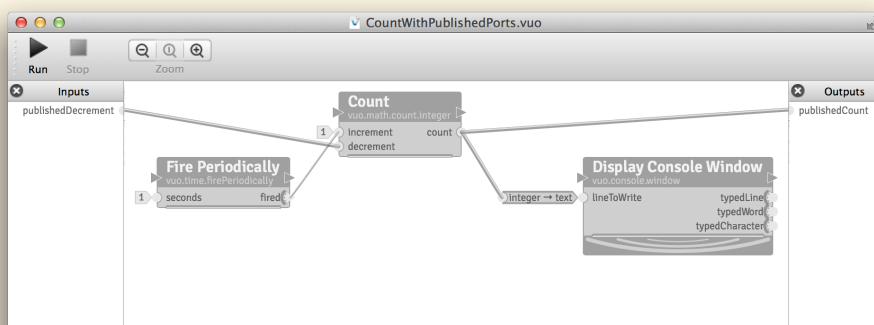
Some applications allow you to use Vuo compositions as plugins — making it possible to customize or add to the application’s behavior. For example, an application for video production may let you define a video effect by creating a Vuo composition that alters an image. The way that the application communicates with the composition is through published ports.

6.1 Published ports

A **published port** is an input port or output port for the composition as a whole. A published input port receives events and possibly data from somewhere outside the composition and passes it along to input ports on nodes inside the composition. A published output port takes events and possibly data from output ports inside the composition and sends it to somewhere outside the composition.

In the Vuo Editor, published ports appear in sidebars on the left and right of the composition canvas. (These sidebars are hidden by default if a composition has no published ports. To display them, go to Window > Show Published Ports.)

Here's a composition with a published input port called **publishedIncrement** and a published output port called **publishedCount**:



6.2 Protocols

If you're going to use a composition as a plugin in some application, usually the application expects the composition to have certain published input and output ports. For example, an application that supports plugins for video effects would expect the composition to have a published input port that receives an image and a published output port that sends an altered image. To guarantee that your composition's published ports match what the application is expecting, you can use a protocol.

A **protocol** is a predetermined set of published ports with certain names and data types. Vuo supports the following protocols:

- **Image Filter** — Alters an image.
 - Published input ports:
 - * **image** (Image) — The original image.
 - * **time** (Real) — A number that changes over time, used to control animations or other changing effects. When previewing the composition in the Vuo Editor, this number is the time, in seconds, since the composition started running. In other applications, this number should be described in their instructions for creating plugins.
 - Published output ports:
 - * **outputImage** — The altered image.

- **Image Generator** — Creates an image.
 - Published input ports:
 - * **width** (Integer) — The requested width of the image, in pixels.
 - * **height** (Integer) — The requested height of the image, in pixels.
 - * **time** (Real) — A number that changes over time, used to control animations or other changing effects. When previewing the composition in the Vuo Editor, this number is the time, in seconds, since the composition started running. In other applications, this number should be described in their instructions for creating plugins.
 - Published output ports:
 - * **outputImage** — The created image. Its width and height should match the **width** and **height** published input ports.

7 The Vuo Editor

7.1 The Node Library

When you create a composition, your starting point is always the **Node Library** (Window > Show Node Library). The node library is a tool that will assist you in exploring and making use of the collection of Vuo building blocks (“nodes”) available to you as you create your artistic compositions.

Because you will be working extensively with the node library throughout your composition process, we have put a great deal of effort into maximizing its utility, flexibility, and ease of use. It has been designed to jump-start your Vuo experience — so that you may sit down and immediately begin exploring and composing, without having to take time out to study reams of documentation.

When you open a new composition, the Node Library is on the left. The Node Library shows all the nodes that are available to you. In the Node Library, you can search for a node by name or keyword. You can see details about a node, including its documentation and version number.

7.1.1 Docking and visibility

By default, the node library is docked within each open composition window. The node library may be undocked by dragging or double-clicking its title bar. While undocked, only a single node library will be displayed no matter how many composition windows are open.

The node library may be re-docked by double-clicking its title bar.

The node library may be hidden by clicking the X button within its title bar. Once hidden, it may be re-displayed by selecting Window > Show Node Library or using Command + Return. The same command or shortcut, Command + Return, will put your cursor in the node library’s search window.

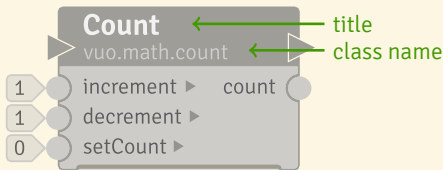
Whether you have left your library docked or undocked, visible or hidden, your preference will be remembered the next time you launch the Vuo Editor.

 Note for
Quartz Composer users

Many of the same shortcuts from Quartz Composer also work in Vuo. As an example, Command + Return opens the Node Library, and Command + R begins playback of your composition.

7.1.2 Node names and node display

Each node has two names: a title and a class name. The **title** is a quick description of a node's function; it's the most prominent name written on a node. The **class name** is a categorical name that reveals specific information about a node; it appears directly below the node's title.



Let's use the **Count** node as an example. "Count" is the node's title, which reveals that the node performs the function of counting. The class name is "vuo.math.count". The class name reveals the following: Team Vuo created it, "math" is the category, and "count" is the specific function (and title name).

Depending on your level of familiarity with Vuo's node sets and your personal preference, you might wish to browse nodes by their fully qualified family ("class") name (e.g., "vuo.math.add") or by their more natural human-readable names ("Add").

You may select whichever display mode you prefer, and switch between the modes at your convenience; the editor will remember your preference between sessions. You can toggle between node titles and node class names using the menu items View > Node Library > Display by class or Display by name.

7.1.3 Node Documentation Panel

The node library makes the complete set of Vuo nodes available for you to browse as you compose. By clicking on a node in the library, a description of the node will appear in the **Node Documentation Panel** below the node library. It describes the general purpose of the node as well as details that will help you make use of it.

If you are interested in exploring new opportunities, this is an ideal way to casually familiarize yourself with the building blocks available to you in Vuo.

7.1.4 Finding Nodes

In the top of the Node Library there is a search bar. You can type in part of a node name or a keyword and matching nodes will show up in the Library. Pressing Esc while in the search bar will clear out your selection and show the entire library, as will deleting your search term.

Your search terms will match not only against the names of relevant nodes, but also against keywords that we have specifically assigned to each node to help facilitate the transition for any of you who might have previous experience with other multimedia environments or programming languages.

For example, users familiar with multiplexers might type “multiplex” into the Vuo Node Library search field to discover Vuo’s “Select Input” family of nodes with the equivalent functionality; users with a background in textual programming might search for the term “string” and discover the Vuo “Text” node family. Users don’t have to know the exact node title or port name. To find a node with a trigger port, for example, go to the Node library and type in the keywords “events,” “trigger,” or “fire.”

If you do not see a node, particularly if you have received it from someone else, review the procedures under [Installing a node](#).

7.2 Working on the canvas

7.2.1 Putting a node on the canvas

The node library isn’t just for reading about nodes, but for incorporating them into your compositions. Once you have found a node of interest, you may create your own copy by dragging it straight from the node library onto your canvas, or by double-clicking the node listing within the library.

Not a mouse person? Navigating the library by arrow key and pressing Return to copy the node to your canvas works just as well.

You may copy nodes from the library individually, or select any number or combination of nodes from the library and add them all to your canvas simultaneously with a single keypress or mouse drag — whatever best suits your work style.

7.2.2 Drawing cables to create a composition

You can create cables by dragging from an output port to a compatible input port or by dragging backwards from an event-only input port to a compatible output port.

Compatible ports are those that output and accept matching or convertible types of data. Compatible ports are highlighted as you drag your cable, so you know where it's possible to complete the connection.

If you complete your cable connection between two ports whose data types are not identical, but that are convertible using an available type converter (e.g., `vu.math.round` for rounding real numbers to integers), that type converter will be automatically inserted when you complete the connection.

Sometimes existing cables may also be re-routed by dragging (or “yanking”) them away from the input port to which they are currently connected. It is possible to yank the cable from anywhere within its **yank zone**. You can tell where a cable's yank zone begins by hovering your cursor near the cable. The yank zone is the section of the cable with the extra-bright highlighting. If no yank zone is highlighted, you will need to delete and add back the cable.

7.2.3 Copying and pasting nodes and cables

You can select one or more nodes and copy or cut them using the `Edit > Copy` and/or `Edit > Cut` menu options, or their associated keyboard shortcuts. Any cables or type converters connecting the copied nodes will automatically be copied along with them.

You can paste your copied components into the same composition, a different composition, or a text editor, using the `Edit > Paste` menu option or its keyboard shortcut.

Tip: Select one or more nodes and drag them while holding down `Option` to duplicate and drag your selection within the same composition. Press `Escape` during the drag to cancel the duplication.

7.2.4 Deleting nodes and cables

Delete one or more nodes and/or cables from your canvas by selecting them and either pressing `Delete` or right-clicking one of your selections and selecting the “Delete” option from its context menu.

When you delete a node, any cables connected to that node are also deleted. A cable with a yank zone may also be deleted by yanking it from its connected input port and releasing it.

Any type converters that were helping to bridge non-identical port types are automatically deleted when their incoming cables are deleted.

7.2.5 Modifying and rearranging nodes and cables

You can move nodes within your canvas by selecting one or more of them and either dragging them or pressing the arrow keys on your keyboard.

Tip: Hold down `Shift` while pressing an arrow key to move the nodes even faster.

You can change the constant value for an input port by double-clicking the port, then entering the new value into the input editor that pops up. (Or you can open the input editor by hovering the cursor over the port and hitting `Return`.) When the input editor is open, press `Return` to accept the new value or `Escape` to cancel.

Input editors take on various forms depending on the data type of the specific input being edited – they may present as a text field, a menu, or a widget such as color picker wheel, for example.

Some ports take lists as input. These ports have special attached “drawers” containing 0 or more input ports whose values will make up the contents of the list. Drawers contain two input ports by default, but may be resized to include more or fewer ports by dragging the “drag handle.”

You can change how a trigger port should behave when it’s firing events faster than downstream nodes can process them. Do this by right-clicking on the port, selecting “Set Event Throttling” from its context menu, and selecting either “Enqueue Events” or “Drop Events”.

You can change a node’s title (displayed at the top of the node) by double-clicking or hovering over the title and pressing `Return`, then entering the new title in the node title editor that pops up. You may save or dismiss your changes by pressing `Return` or `Escape`, respectively, just as you would using a port’s input editor. You can also select one or more nodes from your canvas and press `Return` to edit the node titles for each of the selected nodes in sequence.

You can change a node’s tint color by right-clicking on the node, selecting “Tint” from its context menu, and selecting your color of choice. Tint colors can be a useful tool in organizing your composition. For example, they can be used to visually associate nodes working together to perform a particular task.

7.2.6 Viewing a composition

If your composition is too large to be displayed within a single viewport, you can use the Zoom buttons within the composition window’s menubar, or the `View->Zoom In/Zoom Out/Actual Size` menu options, to adjust your view. You can use the scrollbars to scroll horizontally or vertically within the composition. Alternatively, if you have no nodes or cables selected, you can scroll by pressing the arrow keys on your keyboard.

Tip: Hold down `Shift` while pressing an arrow key to scroll even faster.

7.2.7 Publishing Ports

A composition's published ports are displayed in sidebars, which you can show and hide using the menu **Window > Show/Hide Published Ports**.

You can publish any input or output port in a composition. Do this by right-clicking on the port and selecting **Publish Port** from the context menu. Alternatively, drag a cable from the port to the **Publish** well that appears in the sidebar when you start dragging. You can unpublish the port by right-clicking on the port again and selecting **Unpublish Port**.

In the sidebars, you can rename a published port by double-clicking on the name or by right-clicking on the published port and selecting **Rename Published Port**.

Special copy/paste behavior to note: If you copy a node with a published port, that port will be published under the same name (if possible) in whatever composition you paste it into. The published port will be created if it does not already exist, merged if an existing published port of the same name and compatible type does exist, or renamed if an identically-named published port already exists but has an incompatible type.

7.2.8 Using a protocol for published ports

To create a composition with a predetermined set of published ports defined by a protocol, go to the **File** menu, select **New Composition with Protocol**, and select the protocol you want. Typically, a protocol is used to have a Vuo composition as a plugin inside another application. That application should instruct you about the protocol to select.

The published ports in a protocol appear in a tinted area of the published port sidebars, with the protocol name at the top. You can't rename or delete these published ports. However, you can add other published ports to the composition and rename or delete them as usual.

7.3 Running a composition

After you've built your composition (or while you're building it), you can run it to see it in action.

7.3.1 Starting and stopping a composition

You can run a composition by clicking the Run button. (Or go to Run > Run.)

You can stop a composition by clicking the Stop button. (Or go to Run > Stop.)

If you start a composition that was created using New Composition with Protocol, then extra functionality will be added to the composition to help you preview it. Its protocol published input ports will receive data and events, and its protocol published output ports will send their data and events to a preview window. For example, if you run a composition with the Image Filter protocol, then image and time data will be fed into the composition, and the composition's image output will be rendered to a window.

7.3.2 Firing an event manually

As you're editing your running composition, you may want to fire extra events so that your changes become immediately visible, rather than waiting for the next time a trigger port happens to fire. You can cause a trigger port to fire an event by right-clicking on the trigger port to pop up a menu, then choosing Fire Event. Or you can hold down Command while left-clicking on the trigger port. If the trigger port carries data, it outputs its most recent data along with the event.

7.3.3 Troubleshooting a running composition

In case your composition isn't working correctly, the Vuo Editor provides features that can help you see exactly what events and data are flowing through your composition. For more information, see the section on [Troubleshooting](#).

7.4 Exporting a composition to an application

Using the File > Export App... menu item, you can turn your composition into a Mac application (.app file) that you can distribute to others. You don't need to have Vuo installed to run the application.

When exporting a composition that refers to files on your computer (such as images, scenes, or movies), you need to make sure that those files also exist on the application user's computer. Typically, you'll want to do this by copying the files into the application package. For example, if your composition uses a **Get Image** node to load a file called `image.png`:

- Place `image.png` in the same folder as your composition (.vuo file).
- In the Vuo Editor, edit the **Get Image** node's **imageURL** input port value to `image.png`.
- In the Vuo Editor, go to File > Export App... and create `MyApp.app`.
- Right-click on `MyApp.app` and choose Show Package Contents.
- In the package contents, go to the Contents folder, then the Resources folder. Copy `image.png` into that folder.

8 The Built-in Nodes

Right out of the box, Vuo lets you create multimedia compositions that animate graphics, display video, communicate with network devices, provide user interaction, and more. Now that you know the basics of creating Vuo compositions and using the Vuo Editor, you're ready to explore what you can create.

This section gives an overview of some of Vuo's built-in nodes. Much more information about the built-in nodes is available through the Vuo Editor. Remember that you can search for a node by name or keyword in the Node Library, and you can see the details about any node by clicking on it.

8.1 Rendering graphics and video

For working with 3D graphics, models, and meshes, the `vu0.scene` node set is your starting point. It lets you put 3D objects into a scene, which you can render in a window or image. The `vu0.mesh` node set, with the `vu0.shader` node set and the `vu0.transform` node set allow you to build your own 3D objects in Vuo. Other nodes in the `vu0.scene` node set allow you to modify objects using object filters.

For working with 2D graphics, designs, and animations, the `vu0.image` and `vu0.layer` node sets are your starting point. These let you arrange and manipulate 2D images and render them in a window or composite image.

When animating 2D and 3D graphics, the `vu0.motion` node set lets you control the path and speed of a moving object.

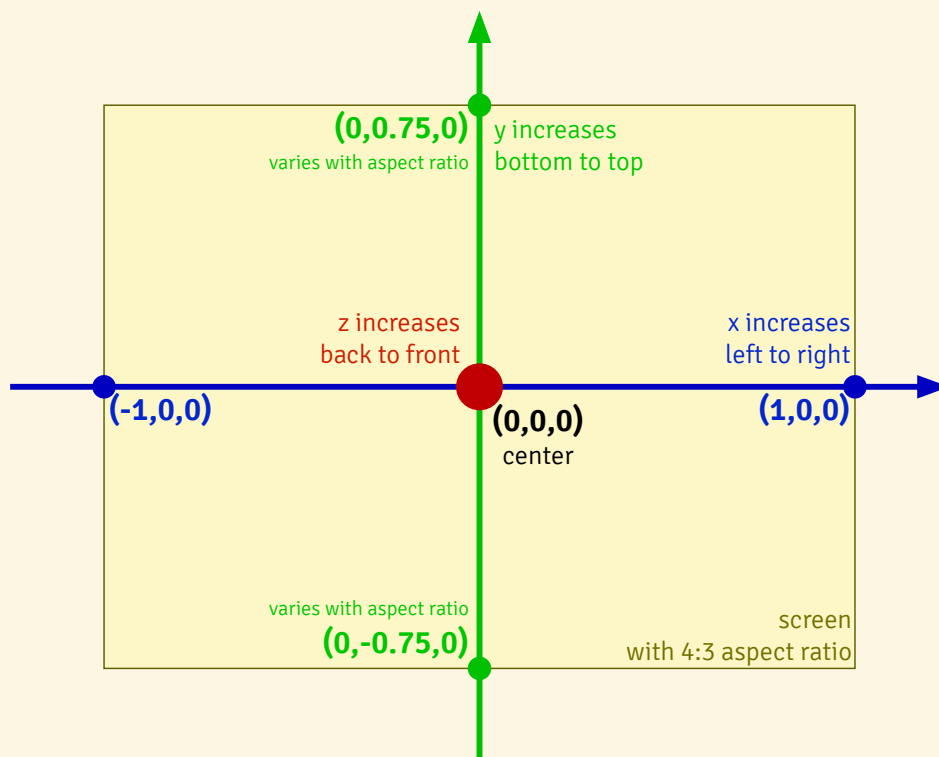
For doing calculations with 2D, 3D, and 4D points and vectors, use the `vu0.point` node set.

Several node sets let you work with video in different ways. For playing movie files, use the `vu0.movie` node set. For sending and receiving video between Vuo compositions and other applications, use the `vu0.syphon` node set. For importing video from a [Kinect](#) camera, use the `vu0.kinect` node set.

8.1.1 Vuo Coordinates

When drawing graphics to a window or image, you need to understand the **coordinate system** of the area you're drawing to. For example, when you use the `Render Scene to Window` node to display a 3D scene in a window, typically the point in your 3D scene with coordinates (0,0,0) will be drawn at the center of the window. (If you're not familiar with the concept of 2D and 3D coordinates, see http://simple.wikipedia.org/wiki/Cartesian_coordinate_system and other references to learn more.)

All of the built-in nodes that work with graphics use **Vuo Coordinates**:



Typically, as illustrated above, the position $(0,0)$ for 2D graphics or $(0,0,0)$ for 3D graphics is at the center of the rendering area. The x-coordinate -1 is along the left edge of the rendering area, and the x-coordinate 1 is along the right edge. The rendering area's height depends on the aspect ratio of the graphics being rendered, with the y-coordinate increasing from bottom to top. In 3D graphics, the z-coordinate increases from back to front.

When working with 3D graphics, you can change the center and bounds of the rendering area by using a **Make Perspective Camera** or **Make Orthogonal Camera** node. For example, you can use a camera to zoom out, so that the rendering area shows a larger range of x- and y-coordinates.

8.2 Processing and playing audio

The `vu.audio` node set lets you work with audio input and output. You can use audio input to create music visualizations or control a composition with sound. You can use audio output to synthesize sounds. Together, audio input and output can be used to receive a live audio feed, process the audio, and play the processed audio.

To interface with MIDI keyboards, synthesizers, and sequencers, you can use the `vu.midi` node set.

8.3 Communicating over a network

Several node sets let you connect with other devices such as iPhones and iPads, Android phones and tablets, musical synthesizers and sequencers, and stage lighting.

For working with MIDI devices, use the `vu0.midi` node set.

For working with OSC devices, use the `vu0.osc` node set.

8.4 Controlling compositions with devices

Several node sets let users interact with your compositions using an input device.

To receive input from the mouse or trackpad, use the `vu0.mouse` node set.

To receive keyboard buttons or typing, use the `vu0.keyboard` node set.

To receive hand and finger movements with a [Leap Motion](#) device, use the `vu0.leap` node set.

For controlling a composition with your body movements using a [Kinect](#), use the `vu0.kinect` node set.

8.5 Controlling window displays and accessing screen values

To control the display of a Vuo window, including aspect ratio and full-size option, use the `vu0.window` node set. The `vu0.screen` node set can provide information about available screens and their values.

9 The Command-Line Tools

As an alternative to using the Vuo Editor, you can use command-line tools to work with Vuo compositions. Although most Vuo users will only need the Vuo Editor, you might want to use the command-line tools if:

- You're using the free version of Vuo, which doesn't include the Vuo Editor.
- You're writing a program or script that works with Vuo compositions. (Another option is the [Vuo API](#).)
- You're working with Vuo compositions in a text-only environment, such as SSH.

A Vuo composition (`.vu0` file) is actually a text file based on the [Graphviz DOT format](#). You can go through the complete process of creating, compiling, linking, and running a Vuo composition entirely in a shell.

9.1 Installing the Vuo SDK

- Go to <https://vuo.org/user> and log in to your account
- Click the “Download” tab
- Under the “Vuo SDK” section, download the Vuo SDK
- Uncompress the ZIP file (double-click on it in Finder)
- Move the folder wherever you like

Do not separate the command-line binaries (`vuo-compile`, `vuo-debug`, `vuo-link`, `vuo-render`) from the Framework (`Vuo.framework`) — in order for the command-line binaries to work, they must be in the same folder as the Framework.

Next, add the command-line binaries to your PATH so you can easily run them from any folder.

- In Terminal, use `cd` to navigate to the folder containing the Vuo Framework and command-line binaries
- Run this command:

```
echo "export PATH=$PATH:$(pwd)" >> ~/.bash_profile
```

- Close and re-open the Terminal window

9.2 Getting help

To see the command-line options available, you can run each command-line tool with the `--help` flag.

9.3 Rendering a composition on the command line

Using the `vuo-render` command, you can render a picture of your composition:

Listing 1: Rendering a composition

```
1 vuo-render --output-format=pdf --output CheckSmsLength.pdf CheckSmsLength.vuo
```

`vuo-render` can output either PNG (raster) or PDF (vector) files. The command `vuo-render --help` provides a complete list of parameters.

Since composition files are in DOT format, you can also render them without Vuo styling using Graphviz:

Listing 2: Rendering a Vuo composition using Graphviz

```
1 dot -Grankdir=LR -Nshape=Mrecord -Nstyle=filled -Tpng -oSmsLength.png CheckSmsLength.vuo
```

9.4 Building a composition on the command line

You can turn a .vuo file into an executable in two steps.

First, compile the .vuo file to a .bc file (LLVM bytecode):

Listing 3: Compiling a Vuo composition

```
1 vuo-compile --output CheckSmsLength.bc CheckSmsLength.vuo
```

Then, turn the .bc file into an executable:

Listing 4: Linking a Vuo composition into an executable

```
1 vuo-link --output CheckSmsLength CheckSmsLength.bc
```

If you run into trouble building a composition, you can get more information by running the above commands with the `--verbose` flag.

If you're editing a composition in a text editor, the `--list-node-classes=dot` flag is useful. It outputs all available nodes in a format that you can copy and paste into your composition.

9.5 Running a composition on the command line

You can run the executable you created just like any other executable:

Listing 5: Running a Vuo composition

```
1 ./CheckSmsLength
```

Using the `vuo-debug` command, you can run the composition and get a printout of node executions and other debugging information:

Listing 6: Running a Vuo composition

```
1 vuo-debug ./CheckSmsLength
```

9.6 Exporting a composition to an application on the command line

Using the `vu-export` command, you can turn a composition into an application:

Listing 7: Exporting a Vuo composition to an application

```
1 vu-export --output CheckSmsLength.app CheckSmsLength.vuo
```

If you run into trouble exporting a composition, you can get more information by running `vu-export` with the `--verbose` flag.

This command is equivalent to the `File > Export App...` menu item in Vuo Editor. See the section [Exporting a composition to an application](#) for more information.

10 Adding Nodes to Your Node Library

You can expand the things that Vuo can do, or save yourself the work of creating the same composition pieces over and over, by adding nodes to the Vuo Editor's Node Library.

10.1 Installing a node

If you download a node (`.vuonode` file), you can install it with these steps:

First, place the node in one of these folders:

- In your home folder, go to `Library > Application Support > Vuo > Modules`.
 - On Mac OS X 10.7 and above, the `Library` folder is hidden by default. To find it, go to `Finder`, then hold down the `Option` key, go to the `Go` menu, and pick `Library`.
- In the top-level folder on your hard drive, go to `Library > Application Support > Vuo > Modules`.

You'll typically want to use the first option, since yours will be the only user account on your computer that should have access to the node class. Use the second option only if you have administrative access and you want all users on the computer to have access to the node class.

Second, restart the Vuo Editor. The node should now show up in your Node Library.

10.2 Creating your own node

Programmers can use Vuo's API to create new nodes for Vuo. See [Developing Node Classes and Types for Vuo](#).

11 Troubleshooting

What if you run into problems using Vuo? This section describes some common problems and how you can fix them. If you need help, you can always check out the additional resources on the [Vuo Support](#) page.

11.1 Common problems

11.1.1 My composition isn't working and I don't know why.

The first step is to take a deep breath and relax! OK, now the second step is to understand the problem. Here are some questions to ask yourself (or go through with a friend or collaborator):

- What do you expect the composition to do?
- What is the composition doing instead?
- Where in the composition does the problem begin?

To answer that last question, the Vuo Editor provides several helpful tools:

- **Show Events** mode lets you watch the events flow through your composition. You can turn it on and off with the Run > Show Events and Run > Hide Events menu items. In Show Events mode, trigger ports are animated as they fire events. Nodes turn opaque as they're executed and gradually become more transparent as time passes since their most recent execution. Using Show Events mode, you can see if certain parts of your composition are executing.
- **Port popovers** let you inspect the data and events flowing through individual ports. A port popover pops up when you click on a port. If you want to keep the port popover open for a while, for example to look at several port popovers at once, click on the popover. While the composition is running, the port popover shows several pieces of information that can help with debugging:

- Last event — The time of the most recent event through the port, and the average number of events per second.
 - Value — For data-and-event ports only, the most recent data through the port.
 - Event throttling — For trigger ports only, whether the port enqueues or drops events.
- **Node descriptions** tell you how the node is supposed to work. The node description appears in the lower panel of the Node Library whenever you select the node in the Node Library or on the canvas.

Using these tools, try to narrow down the problem. Figure out exactly which nodes aren't working as you expect. Then try some of the more specific troubleshooting steps in the rest of this section.

11.1.2 Some nodes aren't executing.

If a node doesn't become opaque in Show Events mode, or if its port popover says "Last Event: (none observed)", then the node might not be executing. (Currently, Show Events mode and port popovers miss events that happened within the first fraction of a second that the composition was running, such as events from a **Fire on Start** node. To see if a node executed during that first fraction of a second, connect one of its output ports to a **Count** node's **increment** input port, and check the **Count** node's output.)

If a node isn't executing, that means events aren't reaching it. Here are some things to check:

- Is there a trigger port connected to the node? Trace backward through your composition, starting at the node that isn't executing, and looking at the cables and nodes feeding into it. Do you find a trigger port? If not...
 - Add a node with a trigger port, such as **Fire on Start**, and connect the trigger port to the node that isn't executing.
- Is the trigger port firing? Check the trigger port's popover (or connect a **Count** node, as described above). If the trigger isn't firing...
 - Check the node description for the trigger port's node. Make sure you understand exactly when the trigger is supposed to fire.
 - Check the trigger port's event throttling, which is displayed in the port popover. If it says "drop events", try changing it to "enqueue events". (See the section [Controlling the buildup of events](#).)
- Are events from the trigger port reaching some nodes but not others? Trace forward through your composition, from the trigger port toward the node that isn't executing, and find the last node that's receiving events.

- Look at the input ports on that last node. Do they have walls or doors? (See the section [Event walls and doors](#).) Check the node's description to help you understand when and why the node blocks events. To send events through the node, you may need to connect a cable to a different input port.
- Look at the output ports on that last node. Are they trigger ports? Remember that events into input ports never travel out of trigger ports. To send events through the node, you may need to connect a cable to a different output port.

11.1.3 Some nodes are executing when I don't want them to.

A node executes every time an event reaches it. If you don't want the node to execute at certain times, then your composition needs to block events from reaching the node. For more information, see the section [Common patterns for event flow](#).

11.1.4 Some nodes are outputting the wrong data.

If your composition is outputting graphics, audio, or other information that's different from what you expected, then you should check the data flowing through your composition. Here are some things to check:

- Where exactly does the data go wrong?
 - Check each port popover along the way to see if it has the data you expected.
 - Add some nodes to the middle of the composition to help you check the data (for example, a **Render Image to Window** node to check the data in an image port).
- Is there a node whose output data is different than you expected, given the input data?
 - Read the node description carefully. The node might work differently than you expected.

11.1.5 The composition's output is slow or jerky.

This can happen if events are not flowing through your composition often enough or quickly enough. Here are some things to check:

- Is each trigger port firing events as often as you expected? Check its port popover to see the average number of events per second. If it's firing more slowly than you expected...
 - Check the node description for the trigger port's node. Make sure you understand exactly when the trigger is supposed to fire.

- Check for any nodes downstream of the trigger port that might take a long time to execute, for example a **Get Image** node that downloads an image from the internet. Change your composition so those nodes receive fewer events. (See the section [Common patterns for event flow](#).)
- Check the trigger port’s event throttling, which is displayed in the port popover. If it says “drop events”, try changing it to “enqueue events”. (See the section [Controlling the buildup of events](#).)
- Check the event throttling of each other trigger port that can fire events through the same nodes as this trigger port. If the other trigger port’s event throttling is “enqueue events”, try changing it to “drop events”.
- Is each node receiving events as often as you expected? If not...
 - Check if there are any event doors that might be blocking events between the trigger and the node. (See the section [Event walls and doors](#).)
- Is the composition using a lot of memory or CPU? You can check this in the Activity Monitor application. If so...
 - Check if any parts of the composition are executing more often than necessary, and try not to execute them as often. (See the section [Common patterns for event flow](#).)
 - Export the composition to an application. When run as an application instead of in Vuo Editor, compositions use less memory and CPU.
 - Quit other applications to make more memory and CPU available.
 - Run the composition on a computer with more memory and CPU.

11.2 General tips

Finally, here are a few more tips to help you troubleshoot compositions:

- If you’re having trouble with a large and complicated composition, try to simplify the problem. Create a new composition and copy a small piece of your original composition into it. It’s much easier to troubleshoot a small composition than a large one.
- If you’re having trouble with a composition that has rapidly firing trigger ports, try to slow things down. For example, in place of the **Render Scene to Window** node’s **requestedFrame** trigger port, use a **Fire Periodically** node’s **fired** trigger port connected to a **Count** node’s **increment** port.
- If your composition used to work but now it doesn’t, figure out exactly what changed. Did you add or remove some cables? Were you using a different version of Vuo? Knowing what changed will help you narrow down the problem.
- Check the Console application while running your composition. Some nodes send console messages when they have problems.

- Keep your composition neat. If your nodes and cables are nicely laid out instead of jumbled together, then it's easier to spot problems.
- Don't be afraid to experiment (but first save a copy of your composition). If you're not sure if a node is working as you expect, try it with various inputs.
- Don't be afraid to ask questions. For help, go to [Vuo Support](#).

12 Keyboard Shortcuts

Vuo has [keyboard shortcuts](#) for working with your composition and the Vuo Editor.

In the keyboard shortcuts below, these symbols represent keys in Mac OS X:

Symbol	Definition
⌘	Command key
⌥	Control key
⌥	Option key
⇧	Shift key
⌫	Delete key
↵	Return key
⌘	Escape key

12.1 Working with composition files

Shortcut	Definition
⌘N	New Composition
⌘O	Open Composition
⌘S	Save Composition
⇧⌘S	Save Composition As
⌘W	Close Composition

12.2 Controlling the composition canvas

Shortcut	Definition
⌘=	Zoom In
⌘-	Zoom Out
⌘9	Zoom to Fit
⌘0	Actual Size
⌘⇧	Show Node Library
Spacebar Drag	Move the canvas viewport.

12.3 Creating and editing compositions

Shortcut	Definition
⌘A	Select all
⇧⌘A	Select none
⌘C	Copy
⌘V	Paste
⌘X	Cut
⌘Z	Undo
⇧⌘Z	Redo
⌘X	Delete
⌘ Drag	Duplicate selected nodes and cables.
↑↓←→	Move nodes and cables around on the canvas. Hold ⇧ to move further.
⇧	Hover over a node title and press ⇧ to edit it.
⇧	Select one or more nodes and press ⇧ to edit their titles.
⇧	Hover the mouse over a constant value and press ⇧ to edit it. Press ⇧ to accept the new value, or ⌘ to go back to the old value.

12.4 Running compositions

Shortcut	Definition
⌘.	Stop
⌘R	Run
⌘ Click	Do this on a trigger port to manually fire an event.

12.5 Application shortcuts

Shortcut	Definition
⌘Q	Quit the Vuo Editor
⌘H	Hide the Vuo Editor
