# Vuo Manual

Vuo 0.6.0

# Contents

# 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.

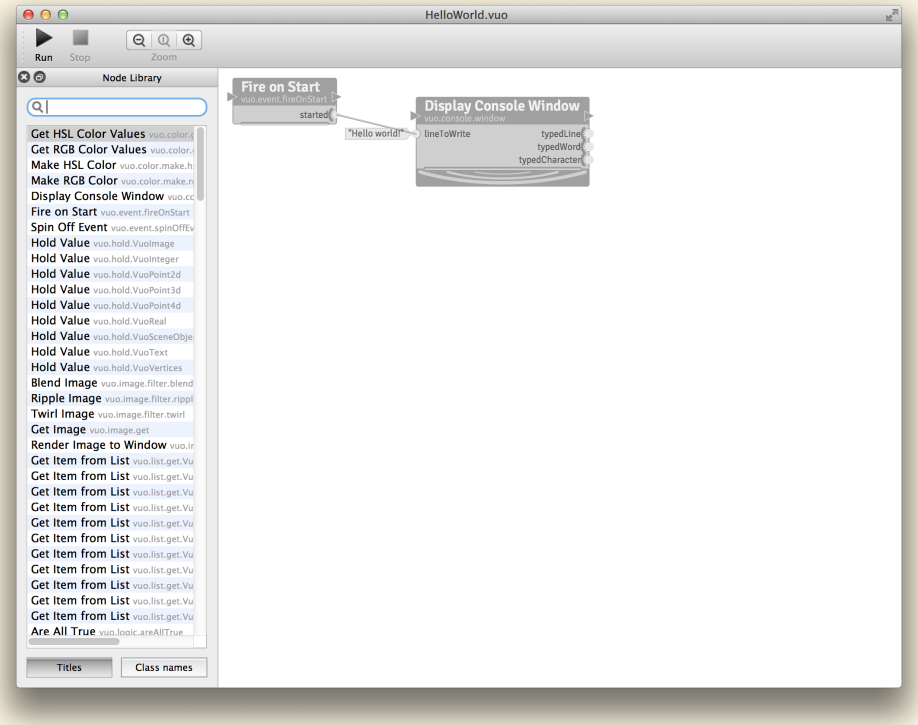
Welcome to Vuo. Get ready to create!

# 2 Quick Start

## 2.1 Install Vuo

- Go to https://vuo.org/user and log in to your account
- Click the “Subscriber” 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 blank canvas will be on the right.
2. In the Node Library, find the **Fire on Start** node. You can do this by scrolling down the Node Library list or typing the word, "fire" into the search bar on the top of the Node Library list. Drag it onto the canvas. (Note: If your Node Library lists your nodes as beginning with "vuo.", go to the bottom of the Node Library and click on the `Titles` button. This will now show you the nodes listed by title.)
3. In the Node Library, find the **Display Console Window** node. If you clear out the search bar, you can scroll down the Node Library to find it or use the search bar again. Drag it onto the canvas.
4. 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 clicking and holding over the port you'd like to start from, then dragging the cable to the port you'd like to connect to and letting go.
5. Double-click on the **writeLine** port of the **Display Console Window** node. This pops up a text box.
6. 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

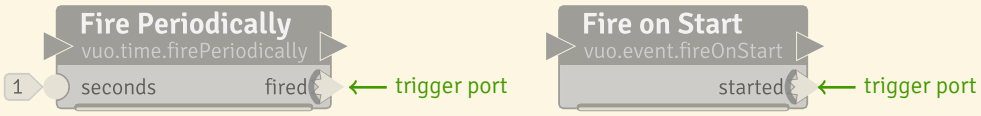
## 3.1 The 'flow' or execution of events

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 spikes on its left side, to indicate that it *fires* events. Ka-pow!

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




Nodes with trigger ports

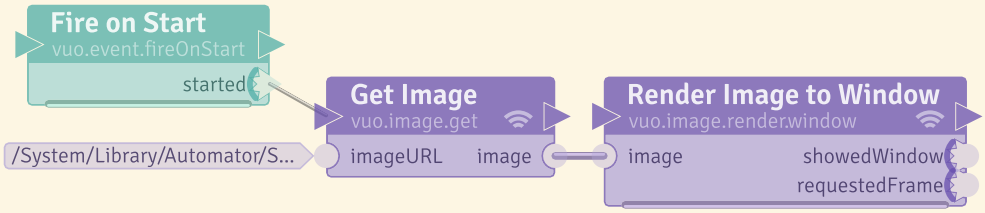
See the chapter Controlling the Flow of Events for more detail on event flow.

## 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 hovering the mouse over the node. After a few seconds, this pops up a tooltip that shows 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.

### 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.

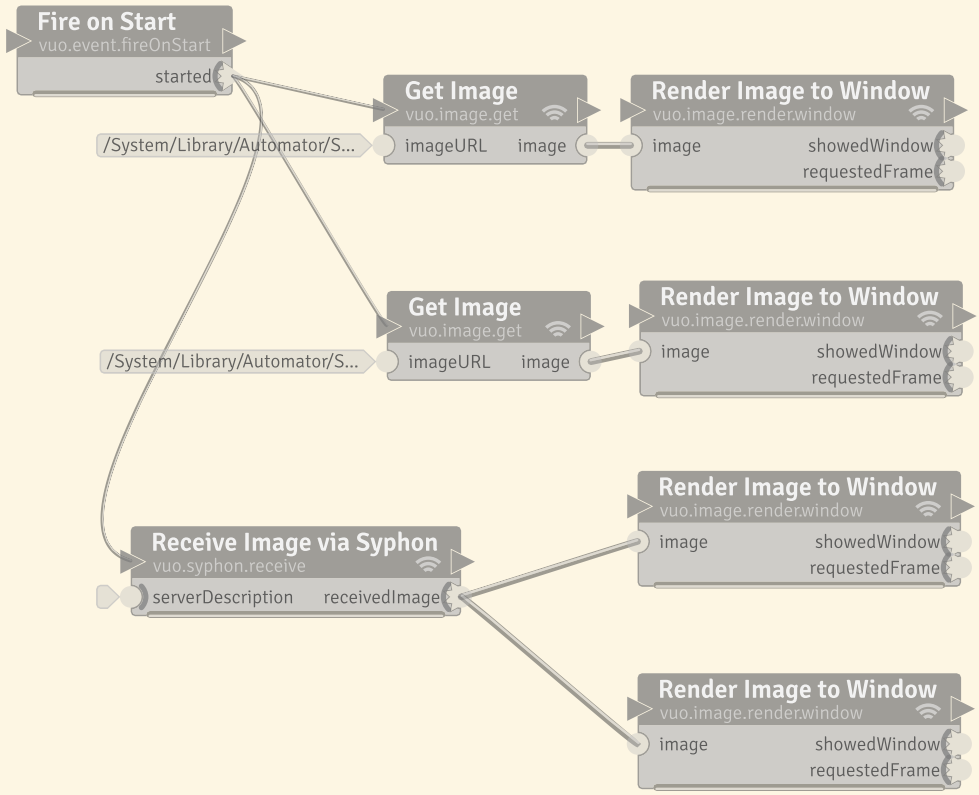
**Event-only cables** are thin. **Data-and-event cables** are thicker than event-only cables.

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

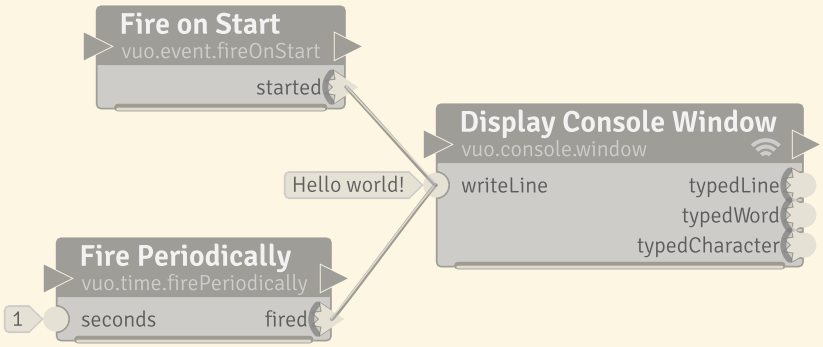
A **node** in Vuo is analogous to a **class instance method** that takes a list of inputs and returns a list of outputs.




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



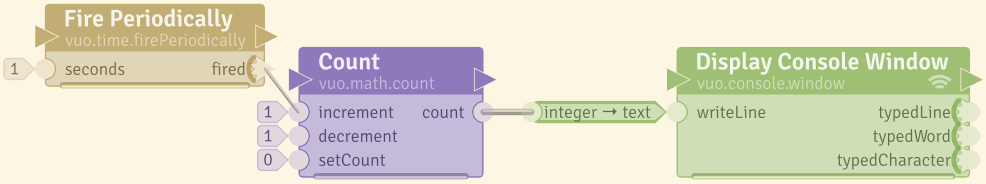

Multiple event-only cables, but only one data-and-event cable, can be connected to an input port.

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 selected cable` or `Edit > Delete selected cable` 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 vuo.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.

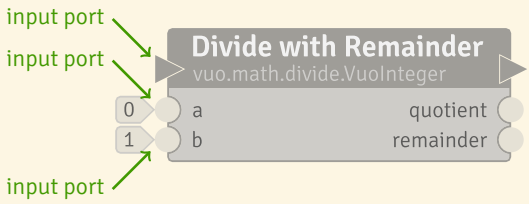
To see a list of descriptions for all example compositions for a node set, look in the node set documentation. In the Editor's Node Library (`Window > Show Node Library`), find a node that belongs to that node set. For example, to learn more about the vuo.math example compositions, use the keyword "math" 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 bring up a node popover that 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

## 4.1 How events travel through a node

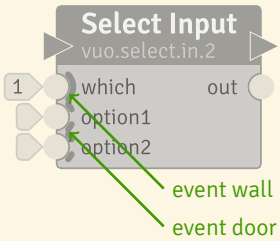
To understand how to control what happens in your composition, you first need to understand what happens to events when they arrive at a node, and how events travel to other nodes. This means understanding **input ports**, **refresh ports**, **output ports** and **done ports**.

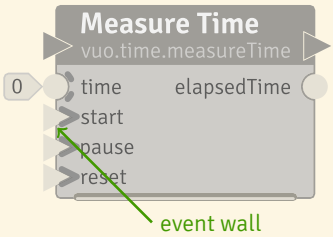
### 4.1.1 Input ports




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.

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. (See later in this chapter, Controlling when nodes execute, for more information.)




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**.

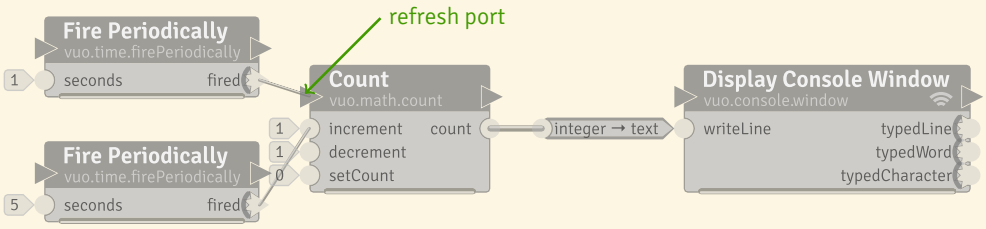
Tip: The event wall is visually placed inside the node to indicate that the event gets blocked inside the node (as it executes) — rather than getting blocked before it reaches the node.

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.

#### 4.1.1.1 Refresh ports

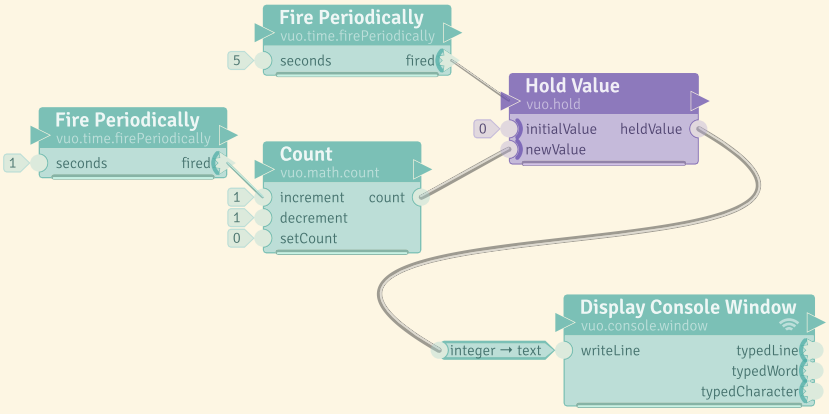
Every node has a built-in event-only input port called the **refresh port**. The purpose of this port is to execute the node without performing any **port-specific function**.

Tip: To determine what is a port-specific function, look at the input port. If the input port uses a present-tense verb phrase, such as **increment** or **writeLine**, then the port has a port-specific function, and the refresh node will not execute that function. If the input is a noun, such as **value**, **channel**, **x**, **y**, etc., then the node will execute using those values when an event arrives via the refresh port.




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.

### 4.1.2 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.


**4.1.2.1 Trigger ports** A **trigger port** is a special kind of output port. Trigger ports can fire (originate) events. However, trigger ports never transmit events that came into the node through an input port, nor do they cause any other output ports to emit events. When a trigger port fires, no event comes out the done port.


(Note: A trigger port can fire an event *in response to* an event that came into the node, but this is a *different* event. For more information, see the section on Executing nodes in the background.)

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**.

**4.1.2.2 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.

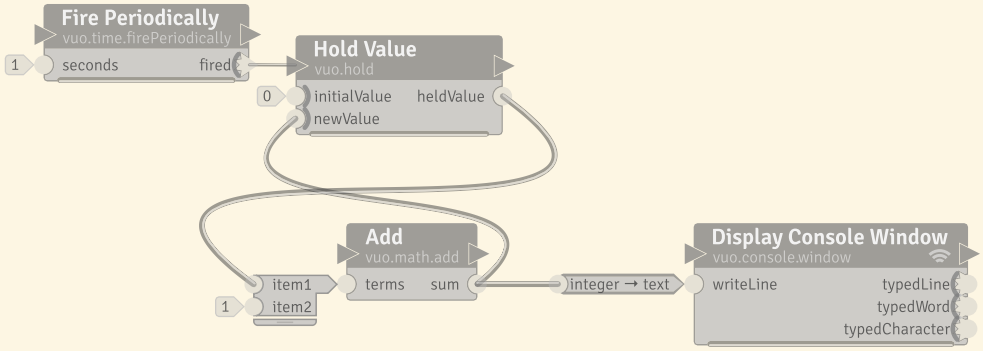

Tip: Done ports are useful for allowing event flow out of nodes that have no other output ports.

## 4.2 Controlling when nodes execute

A composition can be as simple as a straight line of nodes, one executing after the other. But it doesn't have to be. A composition can have cables branching off in different directions, allowing multiple nodes to execute simultaneously. It can decide to execute some nodes and not others. It can have feedback loops.

### 4.2.1 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.

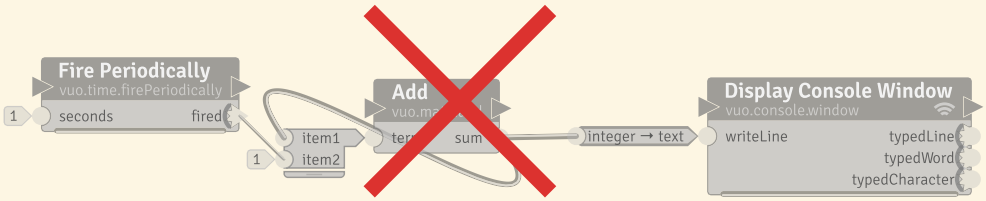
The second time the **Fire Periodically** node fires an event, the inputs of **Add** are 1 (from the **Hold Value** node) and 1. The third time, the inputs are 2 and 1. And so on.

**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.

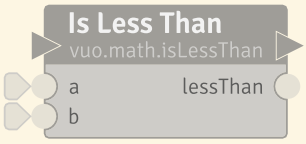



The above composition is illegal in Vuo — don't do this! The reason is that any event from the **Fire Periodically** node would get stuck forever traveling in the feedback loop from the **Add** node's **sum** port back to the **item1** port, and the composition would come to a standstill. Because there's 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.

### 4.2.2 Logic Nodes

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."



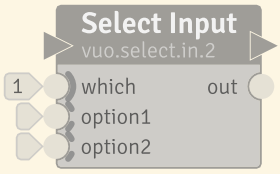

You can execute downstream nodes based on whether an output is true or false. Logic nodes are often used with nodes that route data, where the logic node acts as an "if" and the node that routes data as a "then do this, or if not, do that."

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

### 4.2.3 Routing data with select nodes

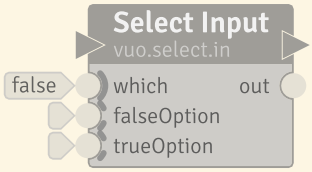
A **Select Input** node lets you select a value from different input ports and route it to the output. A **Select Output** node lets you select an input value to route to different output ports. There are two different classes of Select nodes. The first uses the values 1 and 2 to control how the node 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.

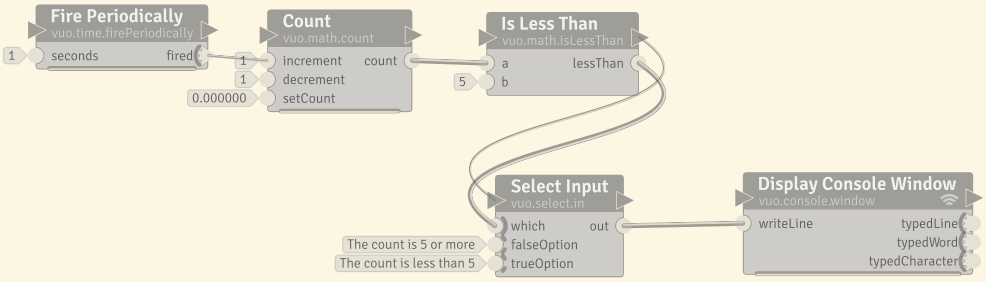



Notice that the **which** port has an event wall, as shown by the solid semi-circle. In order to get the node to output an event through the **out** port, you need to send an event to the selected input port or the refresh port. It won't work if you just send an event to the **which** port.

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




Here is 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.



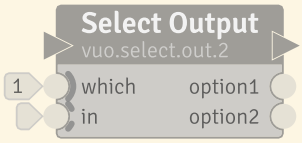

Note for Quartz Composer users

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.

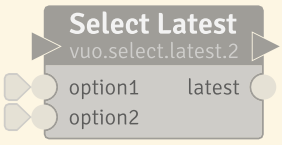
Note for text programmers

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

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.

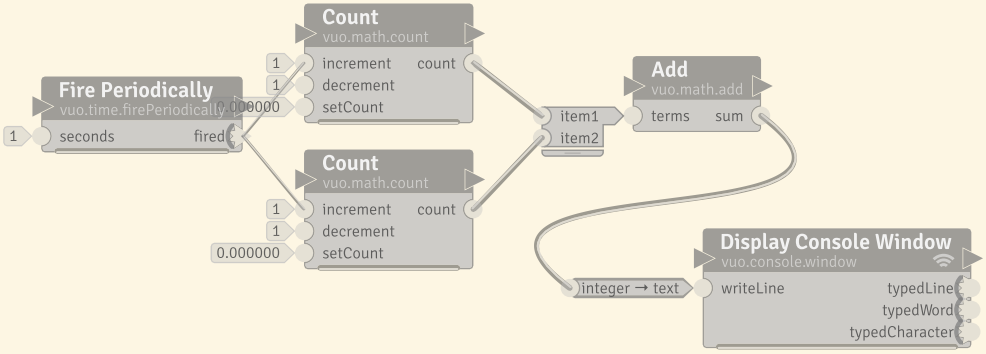



A **Select Latest** node lets you select the latest data and event to arrive at the node. If an event comes into both **option1** and **option2**, then **option1** is used.




### 4.2.4 Executing nodes in parallel

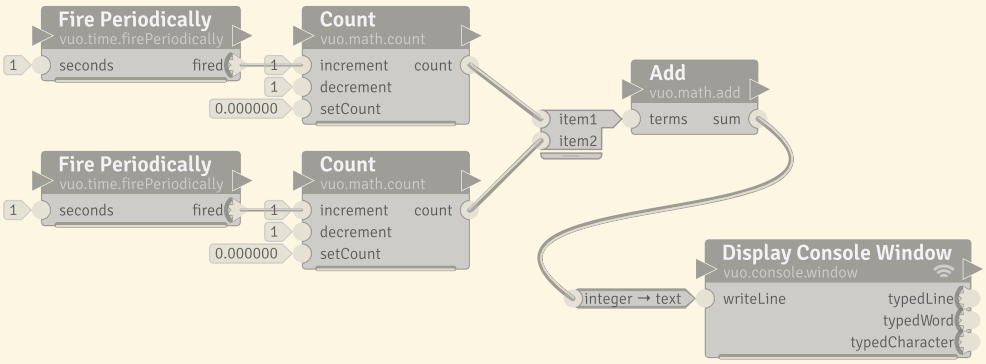
When you run a composition in Vuo, multiple nodes can execute at the same time. This takes advantage of your multicore processor to make your composition run faster.




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**.

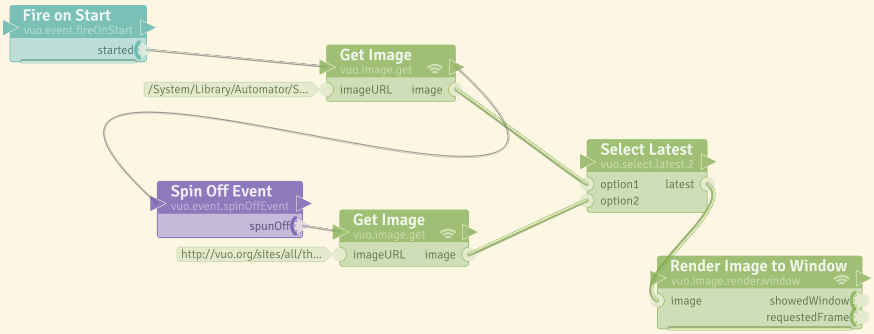
If the **Fire Periodically** node fires an event, and then fires a second event before the first has made it through the composition, then the second event waits. Only when the **Display Console Window** node has finished executing for the first event do the **Count** nodes begin executing for the second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




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.

### 4.2.5 Executing nodes in the background



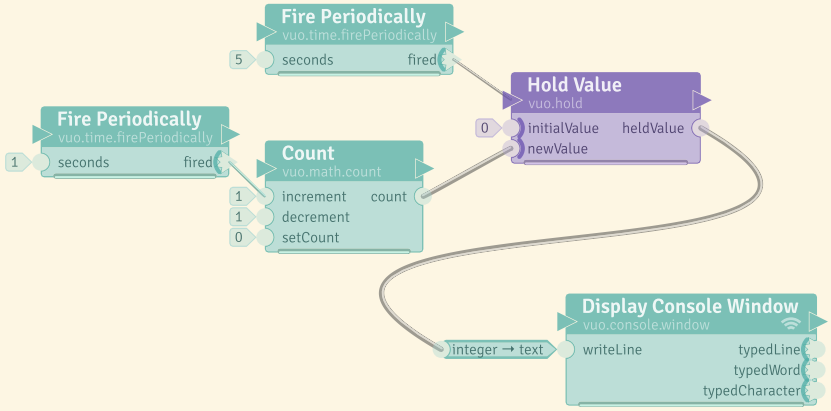

An example composition, LoadImagesAsynchronously.vuo (under vuo.event), has nodes execute in the background

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**.

### 4.2.6 Executing nodes at a steady rate




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.)

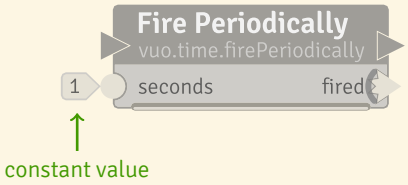
### 4.2.7 Summary

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, with **Select** nodes, and with other nodes 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.

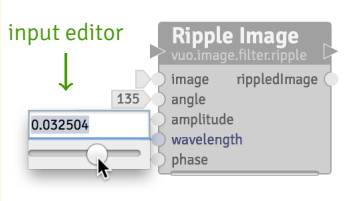
# 5 Understanding Data

## 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.)

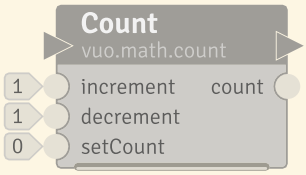



If you edit a constant value for a node's input port, the node will use the new port value the next time it executes. Setting a constant value won't cause the node to execute.

## 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.

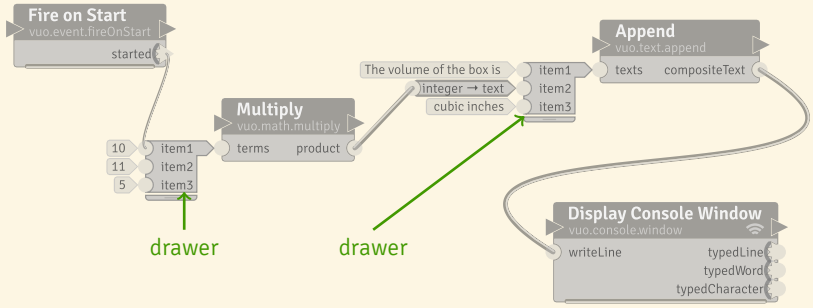
If you disconnect a data-and-event cable to a port that previously had a constant value, then the port goes back to its previous value. If you did not set a constant value, it goes back to its default value.




## 5.3 Using drawers

Some nodes are able to use a list of inputs. These nodes have **drawers**. The initial configuration of a drawer has two inputs, but you can change the number of inputs by dragging on the handle (the bar beneath the drawer), or by right-clicking and selecting `Add Input Port` or `Remove Input Port`.

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

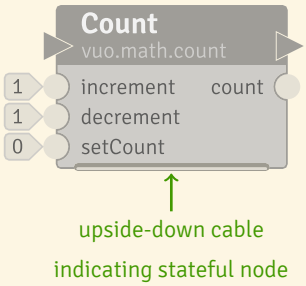



A composition that takes the height, length, and width of a box and calculates its volume

Like a node, a drawer will only execute if it has an event coming into one of its input ports. (To help you remember this, drawers have the same background color as nodes.) If a node isn't getting any input from its attached drawer, make sure there's a cable connected to at least one of the drawer's input ports.

## 5.4 Storing information

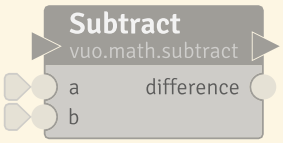
Some nodes have **state**. They remember information from previous times they were executed. An example of a **stateful** node is **Count**. If a **Count** node is told to increment, its state (the count) changes.




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.

## 5.5 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.

### 5.5.1 Converting between types

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.

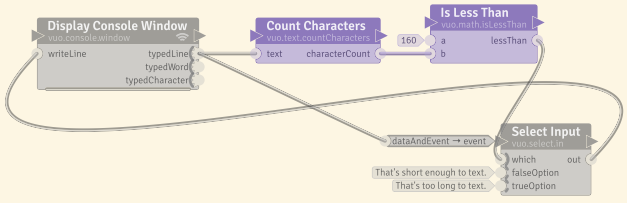
What if you want to connect the **Count Character** node's integer output port to a text input port? If you start to drag a cable from the **characterCount** port, any ports in the composition that the cable can connect to are highlighted. This includes not just ports of type integer, but also ports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

### 5.5.2 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 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 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** nodes 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 **vuo.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.

## 5.6 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 **Check Mouse Status** node, which outputs information about the mouse's position and buttons. 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 along the bottom edge, 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 The Vuo Editor

## 6.1 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.

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.

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.

### 6.1.1 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.

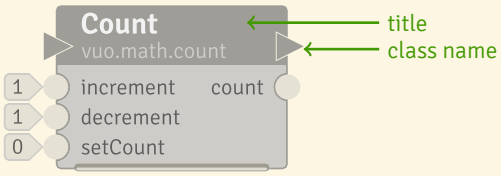
The node library may be re-docked by double-clicking its title bar.

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.

### 6.1.2 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 by using the buttons at the bottom of the Node Library or by `View>Node Library> Display by class` or `Display by name`.

### 6.1.3 Node Tooltips

The node library makes the complete set of Vuo nodes available for you to browse as you compose. By hovering the mouse cursor over a node in the library, you will be presented with a tooltip describing 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.

### 6.1.4 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.

## 6.2 Working on the canvas

### 6.2.1 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.

### 6.2.2 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., vuo.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.

### 6.2.3 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.

### 6.2.4 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.

### 6.2.5 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 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.

### 6.2.6 Viewing a composition

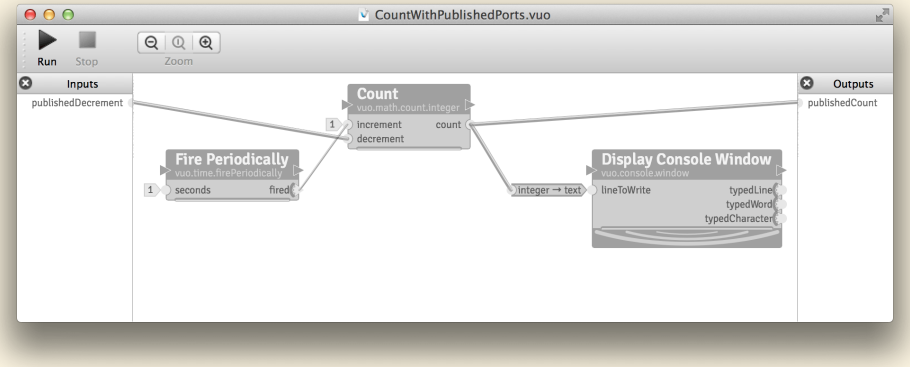
If your composition is too large to be displayed within a single viewport, you can use the Zoom buttons within the composition window's menubar, or the `View->Zoom In/Zoom Out/Actual Size` menu options, to adjust your view. You can use the scrollbars to scroll horizontally or vertically within the composition. 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.

### 6.2.7 Publishing Ports

Publishing a port creates an interface for your Vuo composition to interact with other applications. An application may let you customize or add on to its behavior by creating Vuo plugins. For example, an application for video production may let you define a video effect by creating a Vuo composition with certain published ports. An application can send data into a composition through published input ports and get data out of it through published output 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.

## 6.3 Running a composition

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

### 6.3.1 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.)

### 6.3.2 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.

### 6.3.3 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.

## 6.4 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.

# 7 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.

## 7.1 Rendering graphics and video

For working with 3D graphics, models, and meshes, the `vuo.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.

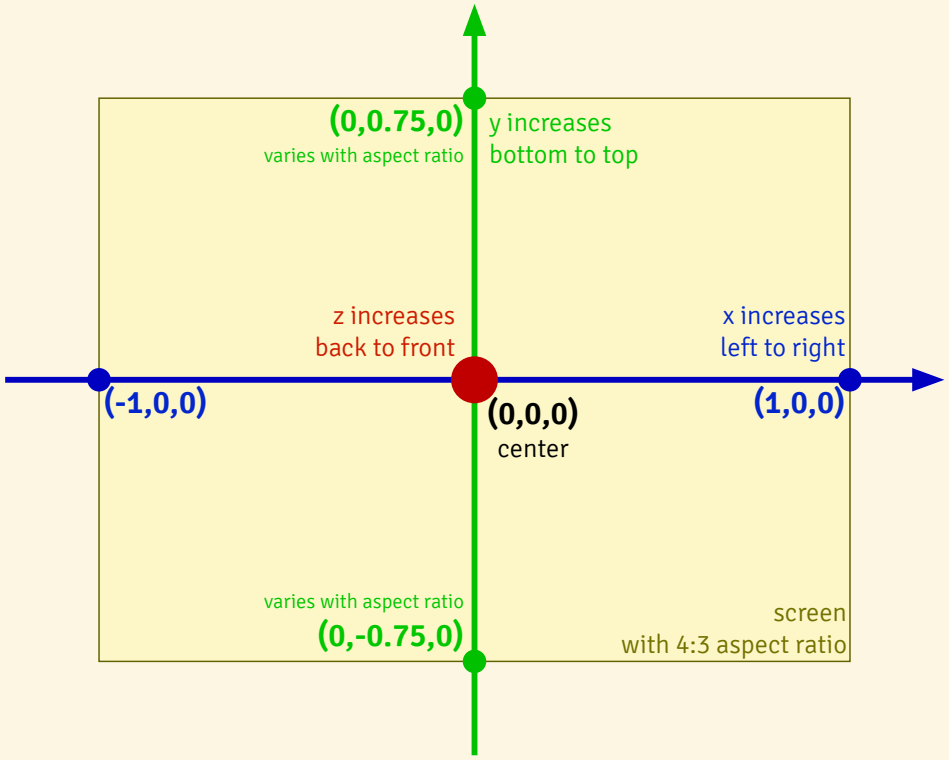
For working with 2D graphics, designs, and animations, the `vuo.image` and `vuo.layer` node sets are your starting point. These let you arrange and manipulate 2D images and render them in a window or composite image.

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

### 7.1.1 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.

### 7.2 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 `vuo.midi` node set.

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

### 7.3 Controlling compositions with devices

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

For controlling a composition with a mouse or trackpad, use the `vuo.mouse` node set.

For controlling a composition with your hand and finger movements using a Leap Motion device, use the `vuo.leap` node set.

For controlling a composition with your body movements using a Kinect, use the `vuo.kinect` node set.

## 8 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 (.vuo 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.

## 8.1 Installing the Vuo SDK

- Go to https://vuo.org/user and log in to your account
- Click the “Subscriber” 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

## 8.2 Getting help

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

## 8.3 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
```

## 8.4 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 bitcode):

**Listing 3: Compiling a Vuo composition**

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

Then, turn the .bc file into an executable:

**Listing 4: Linking a Vuo composition into an executable**

```
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.

## 8.5 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**

```
./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**

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

## 8.6 Exporting a composition to an application on the command line

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

**Listing 7: Exporting a Vuo composition to an application**

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

If you run into trouble exporting a composition, you can get more information by running `vuo-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.

# 9 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.

## 9.1 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.

### 9.2 Creating your own node

Programmers can use Vuo's API to create new nodes for Vuo. See Developing Node Classes and Types for Vuo.

# 10 Troubleshooting

Why isn't my composition executing? How do I make things happen in my composition? Both questions have to do with first checking to see if you are generating events and if those events are moving through the composition as you expect. The Vuo Editor provides three ways to review the event flow in your composition.

You can use the Vuo Editor's **Show Events** mode, under the "Run" menu to debug your composition or help better understand its behavior. In Show Events mode, nodes turn opaque as they are executed and gradually become more transparent as time passes since their most recent execution. In Show Events mode, trigger ports are animated as they fire events. You may have to slow down the rate at which events execute in order to see them flowing through your composition. This will show you if certain parts of your composition are not executing as expected.

When you have a composition running, you can hover the mouse over any port in the Vuo Editor and a **port popover** will appear that reveals real-time information about that port. Next, you can drag the popover and drop it wherever you'd like and it will become its own independent window, continuously displaying information about the selected port. Some of the key information displayed is: the current value present at the port, the amount of time since the last event occurred, and the type of port selected. You can create many popovers, showing all the information you'd like to keep track of. This feature will allow you to easily monitor compositions and serve as a useful tool for problem-solving and creating.

Don't forget **node tooltips**. They can be displayed when mousing over the title of a node on the canvas, not just when browsing node listings within the node library. By hovering the mouse cursor over a node, you will be presented with a tooltip describing the general purpose of the node as well as details that will help you make use of it.