2.0.3

Contents

1 Getting started **12**

1.1 Quick start . . . 12

1.1.1 Creating a new composition . . . 13

1.1.2 Running the composition . . . 14

1.1.3 Adding a node . . . 15

1.1.4 Connecting nodes with cables . . . 16

1.1.5 Editing an input port value . . . 18

1.1.6 Adding another node . . . 19

1.1.7 Summary . . . 22

1.2 Tracing through a composition . . . 23

1.2.1 Port popovers . . . 23

1.2.2 Information flow . . . 23

1.2.3 Step 1: Fire on Display Refresh to Make Noise Image . . . 24

1.2.4 Step 2: Make Noise Image to Make Stained Glass Image . . . 25

1.2.5 Step 3: Make Stained Glass Image to Render Image to Window . . . 25

1.2.6 Summary . . . 26

1.3 Learning Vuo . . . 26

1.3.1 User manual . . . 27

1.3.2 Tutorials . . . 27

1.3.3 Example compositions . . . 27

1.3.4 Community support . . . 28

1.3.5 Node documentation . . . 28

1.3.6 SDK documentation . . . 28

1.3.7 Vuo in other applications . . . 29

1.4 Installing Vuo . . . 29

1.4.1 Activating Vuo Pro . . . 29

1.5 Changing language settings . . . 30

2 The basics **31**

2.1 A composition is what you create with Vuo . . . 31

2.2 Nodes are your building blocks . . . 31

2.3 Events are what cause nodes to execute . . . 33

2.4 Trigger ports fire events and sometimes data . . . 35

2.5 Events and data travel through cables . . . 36

2.6 Events and data enter and exit a node through ports . . . 37

2.7 Events and data enter and exit a composition through published ports . . . 40

3 How events and data travel through a composition **42**

3.1 Where events come from . . . 42

3.2 How events travel through a node . . . 43

3.2.1 Input ports . . . 43

3.2.2 Output ports . . . 45

3.3 How events travel through a composition . . . 46

3.3.1 The rules of events . . . 46

3.3.2 Straight lines . . . 47

3.3.3 Splits and joins . . . 47

3.3.4 Multiple triggers . . . 48

3.3.5 Feedback loops . . . 49

3.3.6 Summary . . . 50

3.4 How data travels through a composition . . . 51

3.4.1 Ignoring data . . . 51

3.4.2 Data flow without an event . . . 52

3.5 Solving problems with event flow . . . 53

3.5.1 “Infinite feedback loop” error . . . 53

3.5.2 “Deadlocked feedback loop” error . . . 55

3.5.3 Buildup of events . . . 55

4 How compositions process data 57

4.1 Data types . . . 57

4.1.1 Basic data types . . . 57

4.1.2 Type-converter nodes . . . 58

4.1.3 List data types . . . 59

4.1.4 Dictionary data types . . . 60

4.1.5 Ports with changeable data types . . . 60

4.2 Inputting data . . . 67

4.2.1 Editing data in a node's input port . . . 68

4.2.2 Editing data in a published input port . . . 70

4.2.3 Inputting lists . . . 70

4.2.4 Inputting dictionaries . . . 71

5 How nodes can be used as building blocks 72

5.1 Finding out what nodes are available . . . 72

5.2 Learning how to use a node . . . 72

5.3 Pro nodes . . . 73

5.4 Deprecated nodes . . . 73

5.5 The built-in nodes . . . 74

5.5.1 Graphics/video . . . 74

5.5.2 Sound/audio . . . 75

5.5.3 User input devices . . . 75

5.5.4 Music and stage equipment . . . 76

5.5.5 Applications . . . 76

5.5.6 Sensors, LEDs, and motors . . . 76
5.5.7 Displays . . . 77
5.5.8 Files . . . 77
5.5.9 Internet . . . 77
5.6 Adding nodes to the canvas by dropping files . . . 77
5.7 Creating a node . . . 78
5.8 Installing a node . . . 78
5.8.1 Installing a node the quick way . . . 78
5.8.2 Making a node available to all compositions . . . 78
5.8.3 Making a node available to one or a few compositions . . . 80
5.8.4 Uninstalling a node . . . 80

6 Using subcompositions inside of other compositions 81

6.1 Making a subcomposition available to other compositions . . . 84
6.2 Reasons to use subcompositions . . . 84
6.3 Creating a subcomposition . . . 85
6.3.1 Naming a subcomposition . . . 85
6.4 Editing a subcomposition . . . 86
6.5 Watching events and data inside a subcomposition . . . 86
6.6 How events travel through a subcomposition . . . 87
6.6.1 Events into a subcomposition . . . 87
6.6.2 Events out of a subcomposition . . . 89
6.6.3 Constant input port values . . . 91

7 Making compositions fit a mold with protocols 95

7.1 Image Filter protocol . . . 95

7.1.1 Published input ports . . . 95

7.1.2 Published output ports . . . 96

7.2 Image Generator protocol . . . 96

7.2.1 Published input ports . . . 96

7.2.2 Published output ports . . . 97

7.3 Image Transition protocol . . . 97

7.3.1 Published input ports . . . 97

7.3.2 Published output ports . . . 98

7.4 Time . . . 98

7.5 Quality . . . 99

7.6 Creating a protocol composition . . . 99

7.7 Editing a protocol composition . . . 99

7.8 Running a protocol composition . . . 100

7.9 How events travel through a protocol composition . . . 100

8 Exporting compositions 101

8.1 Exporting an image . . . 101

8.2 Exporting a movie . . . 101

8.2.1 Recording the graphics in a window . . . 102

8.2.2 Exporting a movie from an Image Generator composition . . . 103

8.3 Exporting a screen saver . . . 103

8.3.1 Sharing screen savers . . . 104

8.4 Exporting an FxPlug plugin . . . 104
8.4.1 Video effects . . . 104
8.4.2 Transitions . . . 104
8.4.3 Generators . . . 105
8.4.4 Category and name . . . 105
8.4.5 Parameters . . . 105
8.4.6 Image scaling . . . 106
8.4.7 Sharing plugins . . . 106
8.4.8 Uninstalling plugins . . . 107
8.4.9 Maintaining compatibility between plugin versions . . . 107
8.5 Exporting an FFGL plugin . . . 108
8.5.1 Sources . . . 108
8.5.2 Effects . . . 108
8.5.3 Blend modes . . . 108
8.5.4 Name . . . 109
8.5.5 Parameters . . . 109
8.5.6 Sharing plugins . . . 110
8.5.7 Uninstalling plugins . . . 110
8.6 Exporting an application . . . 110

9 Turning graphics shaders into nodes 111

9.1 Creating an ISF node . . . 111

9.2 Editing an ISF node . . . 112

9.3 Saving an ISF node . . . 112

9.4 How ISF source code translates to a Vuo node . . . 113

9.4.1 Node metadata . . . 113

9.4.2 Ports . . . 113

9.4.3 Data types . . . 114

9.4.4 Output image size and color depth . . . 115

9.4.5 Coordinates . . . 115

9.4.6 Examples . . . 115

9.5 Supported ISF features . . . 119

9.5.1 Functions . . . 119

9.5.2 Uniforms . . . 120

9.5.3 Unsupported . . . 121

10 The Vuo editor 122

10.1 The Node Library . . . 122

10.1.1 Docking and visibility . . . 122

10.1.2 Node names and node display . . . 122

10.1.3 Node Documentation Panel . . . 123

10.1.4 Searching for nodes . . . 123

10.2 Working on the canvas . . . 125

10.2.1 Putting a node on the canvas . . . 125

10.2.2 Drawing cables to create a composition . . . 125
10.2.3 Adding a comment . . . 126
10.2.4 Copying and pasting nodes, cables, and comments . . . 126
10.2.5 Deleting nodes, cables, and comments . . . 127
10.2.6 Rearranging nodes, cables, and comments . . . 127
10.2.7 Replacing nodes . . . 127
10.2.8 Editing node settings . . . 127
10.2.9 Editing port values and settings . . . 128
10.2.10 Finding nodes on the canvas . . . 128
10.2.11 Viewing a composition . . . 129
10.2.12 Publishing ports . . . 129
10.2.13 Using a protocol for published ports . . . 129
10.3 Running a composition . . . 130
10.3.1 Starting and stopping a composition . . . 130
10.3.2 Firing an event manually . . . 130
10.3.3 Understanding and troubleshooting a running composition . . . 130
10.4 Editing composition information . . . 131
10.5 Working with subcompositions . . . 131
10.5.1 Installing a subcomposition . . . 131
10.5.2 Editing a subcomposition . . . 132
10.5.3 Uninstalling a subcomposition . . . 132
10.6 Keyboard Shortcuts . . . 133
10.6.1 Working with composition files . . . 133
10.6.2 Controlling the composition canvas . . . 133
10.6.3 Creating and editing compositions . . . 134
10.6.4 Creating and editing shaders . . . 135
10.6.5 Running compositions (when the Vuo editor is active) . . . 135
10.6.6 Running compositions (when the composition is active) . . . 135
10.6.7 Application shortcuts . . . 136

11 The command-line tools **136**

11.1 Installing the Vuo SDK . . . 136

11.2 Getting help . . . 137

11.3 Rendering a composition on the command line . . . 137

11.4 Building a composition on the command line . . . 138

11.5 Running a composition on the command line . . . 138

11.6 Exporting a composition on the command line . . . 139

12 Common patterns - "How do I..." **140**

12.1 Do something in response to user input . . . 140

12.2 Do something after something else is done . . . 140

12.3 Do something if one or more conditions are met . . . 141

12.4 Do something if an event is blocked . . . 142

12.5 Do something if data has changed . . . 143

12.6 Do something after an amount of time has elapsed . . . 144

12.7 Do something repeatedly over time . . . 145

12.8 Do something to each item in a list . . . 146

12.9 Create a list of things . . . 147

12.10 Maintain a list of things . . . 148

12.11 Gradually change from one number/point to another . . . 148

12.12 Set up a port's data when the composition starts . . . 149

12.13 Send the same data to multiple input ports . . . 150

12.14 Merge data/events from multiple triggers . . . 151

12.15 Route data/events through the composition . . . 152

12.16 Reuse the output of a node without re-executing the node . . . 153

12.17 Run slow parts of the composition in the background . . . 154

13 Troubleshooting **156**

13.1 Helpful tools . . . 156

13.2 Common problems . . . 157

13.2.1 My composition isn't working and I don't know why. . . . 157

13.2.2 Some nodes aren't executing. . . . 157

13.2.3 Some nodes are executing when I don't want them to. . . . 158

13.2.4 Some nodes are outputting the wrong data. . . . 158

13.2.5 The composition's output is slow or jerky. . . . 158

13.2.6 Vuo slows down when my computer heats up. . . . 159

13.2.7 Various compositions won't run . . . 160

13.3 General tips . . . 160

14 Contributors **162**

14.1 Contributors . . . 162

14.2 Software Vuo uses . . . 166

14.3 Resources Vuo uses . . . 167

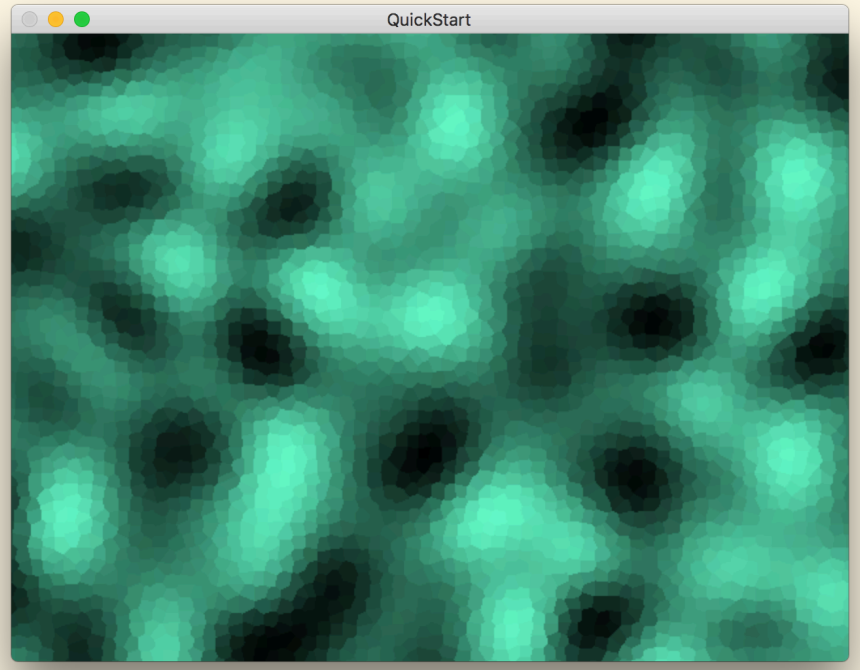
Glossary **167**

1 Getting started

Welcome to the Vuo community! So you want to learn how to use Vuo in your creative work. Let's jump right in with an example.

1.1 Quick start

This example will walk you through the process of creating an animated pattern like this:

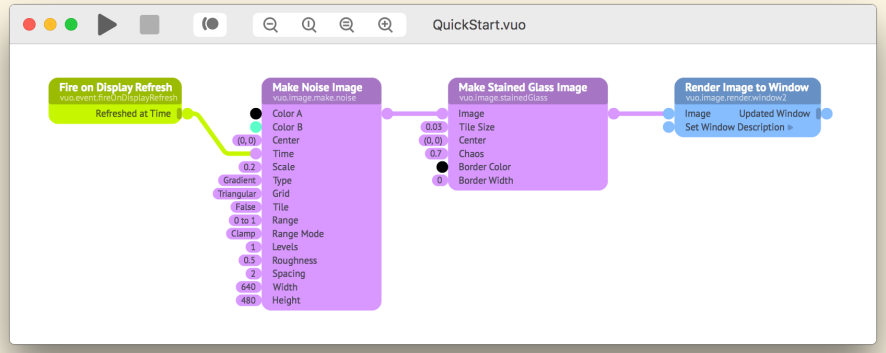


Tip

You can watch a video of this example at https://vuo.org/quickstart

Tip

You can find the completed example composition in File > Open Example > Quick Start.

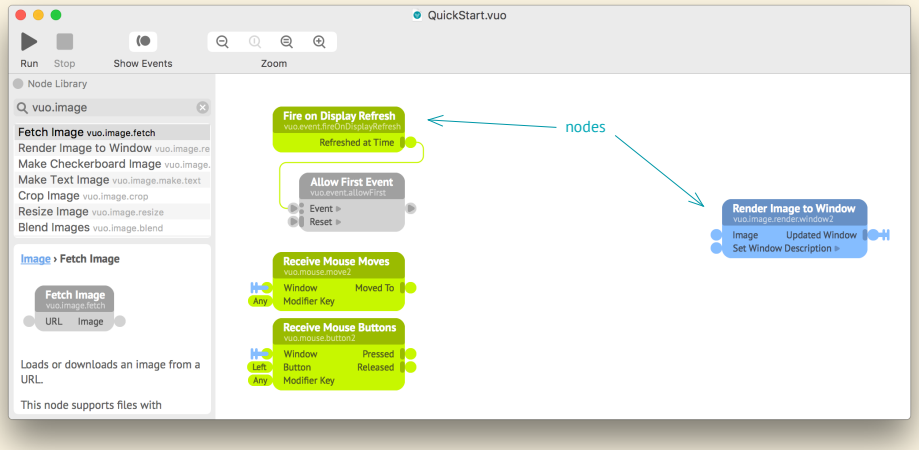


If you haven't already, download and install Vuo, as described in the Installing Vuo section.

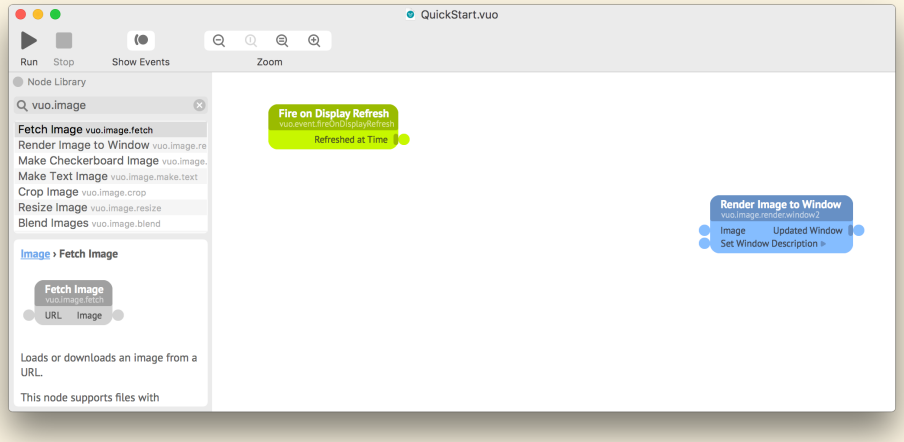
1.1.1 Creating a new composition

In Vuo, the documents that you work with are called *compositions*. Start a new composition by going to File › New Composition from Template › Window › Image.

The rounded rectangles in the composition window are called *nodes*. The area that you place nodes on is called the *canvas*.



For this example, you won't need all of these nodes, just **Fire on Display Refresh** and **Render Image to Window**. So you can click on each of the rest of the nodes and hit ⌫ (Delete).



As you may have guessed from their titles, each node has a job or responsibility. The two nodes on the canvas will each perform a task that contributes to the animated pattern that you're creating. The **Render Image to Window** node will be in charge of displaying the pattern in a window. The **Fire on Display Refresh** node will be in charge of the timing of the animation.

1.1.2 Running the composition

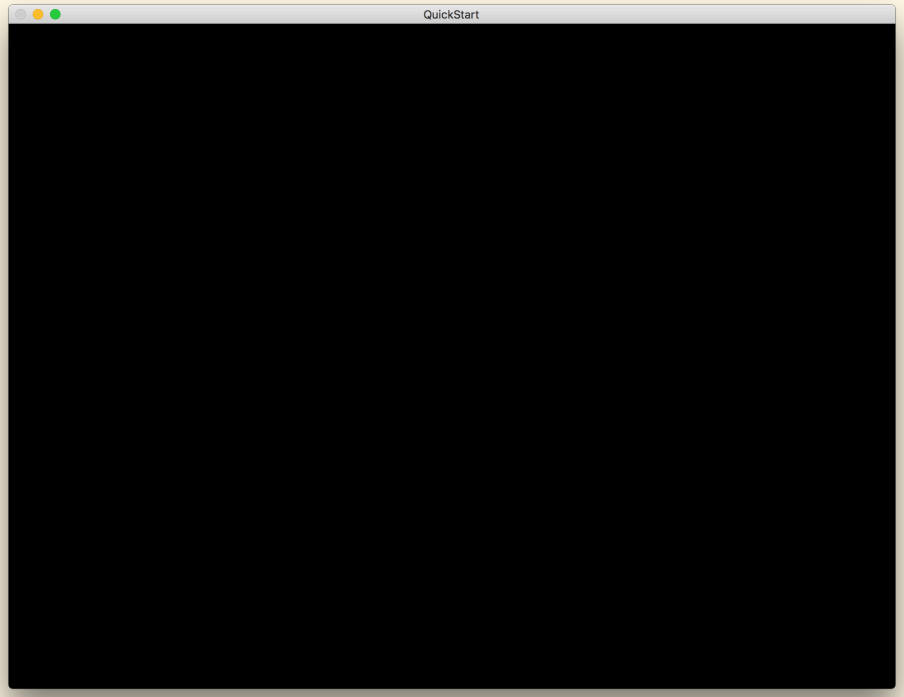
Let's see what you've built so far. Click the Run button. This launches the composition.

Before, on the canvas, you were looking at a blueprint or instructions for what the composition is supposed to do. Now, you're seeing the composition in action.

The window that pops up comes from the **Render Image to Window** node.

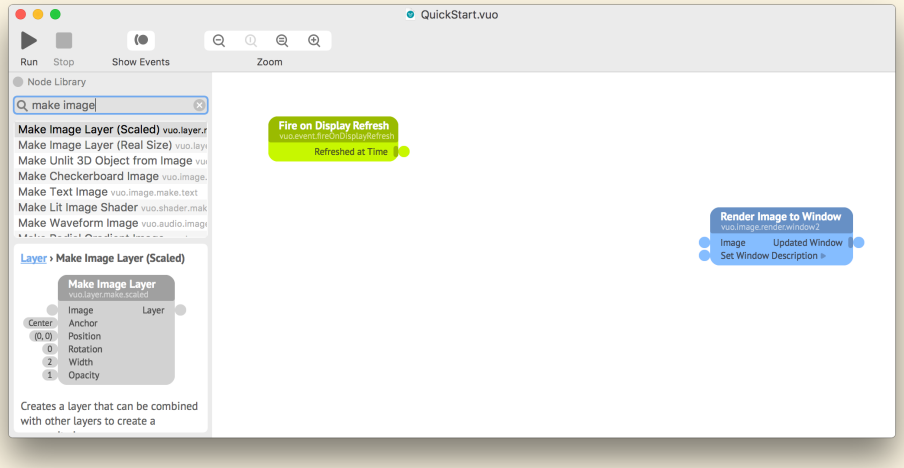
Tip

An animation is a series of images displayed in rapid succession. The fastest rate at which your computer monitor can display a series of images is called the *display refresh rate*. On many monitors, the display refresh rate is 60 times per second.



1.1.3 Adding a node

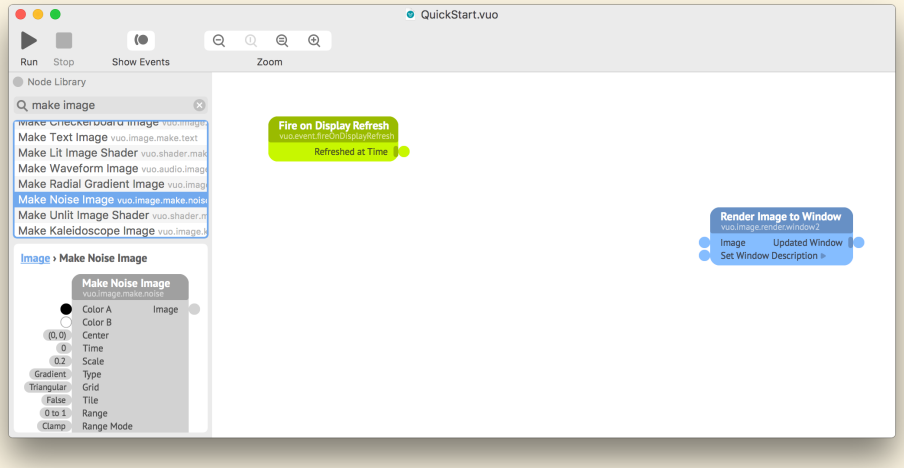
So far, you have a solid black graphics window. To show an image in the window, you'll need a node whose job it is to make an image. To find such a node, search the Node Library for "make image", like this:



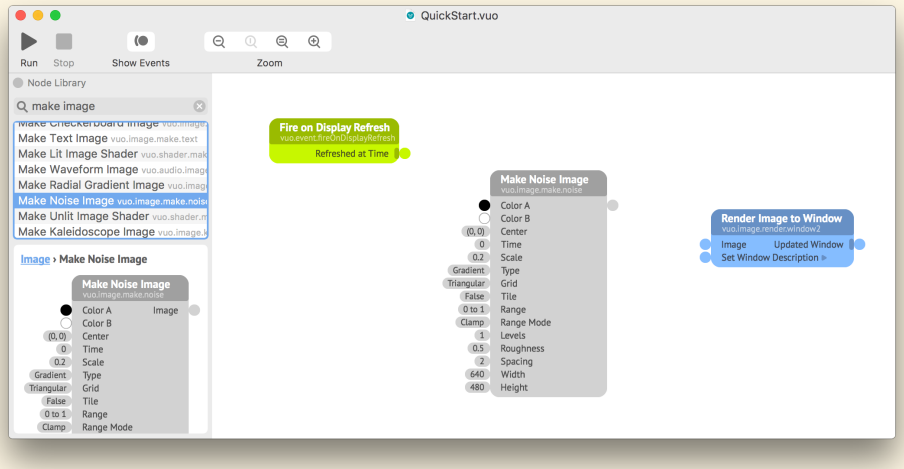
Tip

The Node Library is a directory of all available nodes. If you don't see it, go to View › Node Library › Show Node Library.

In the search results, locate the **Make Noise Image** node.



Drag the **Make Noise Image** node from the Node Library onto the canvas.

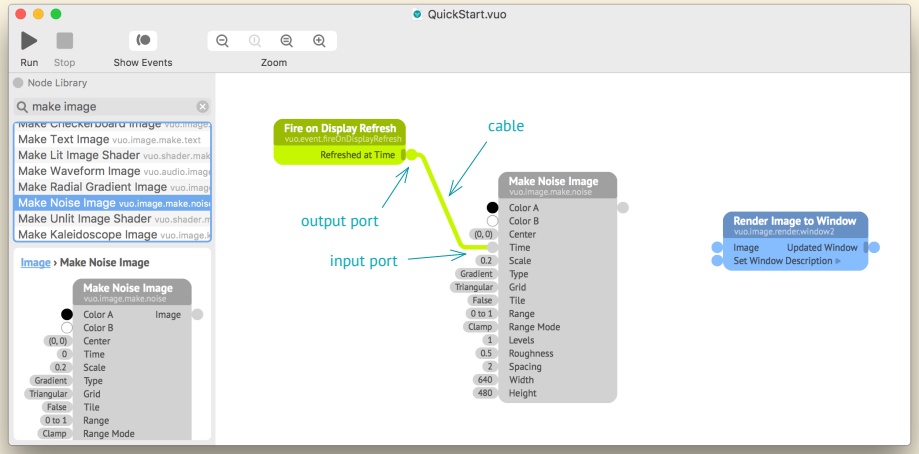


1.1.4 Connecting nodes with cables

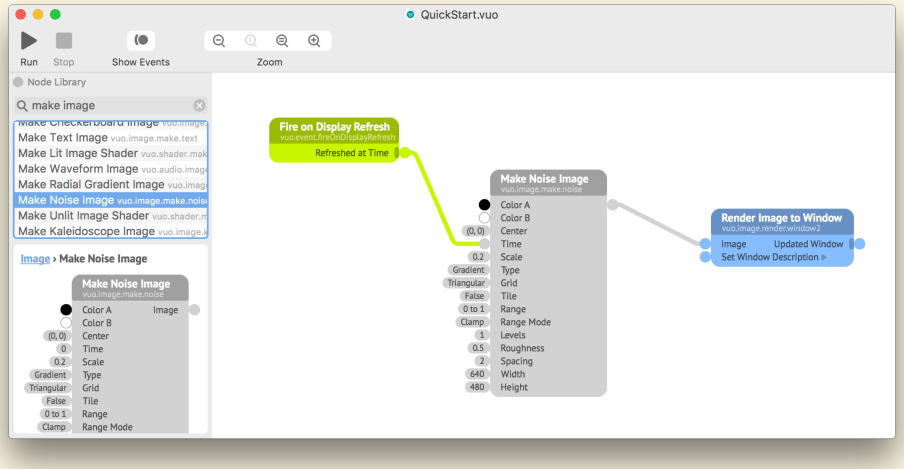
Now you have three nodes on the canvas. Individually, each node does a simple job. How do you make them work together to accomplish something bigger? You connect them with *cables*.

Start dragging from the circle on the right of the **Fire on Display Refresh** node, which is called an *output port*. The line that emerges from the port as you drag is called a *cable*. Drop the end of the

cable onto the circle on the left side of the **Make Noise Image** node labeled **Time**, which is called an *input port*.



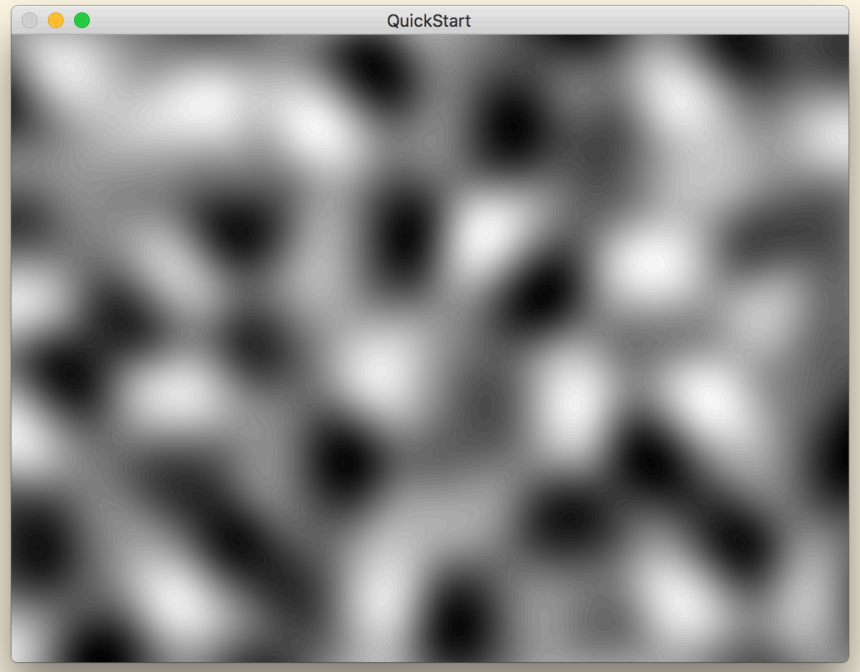
Next, drag a cable from the output port of **Make Noise Image** and drop it on the **Render Image to Window** node's **Image** input port.



Back in the graphics window, you can see that the three nodes are now working together to make a rapid succession of noise images and display them in a window.

Tip

To learn about noise images, read the **Make Noise Image** node's documentation in the lower panel of the Node Library.

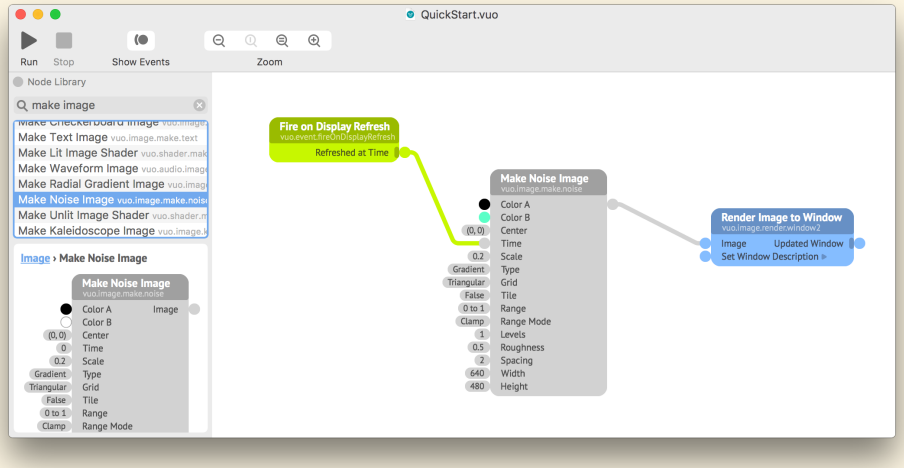


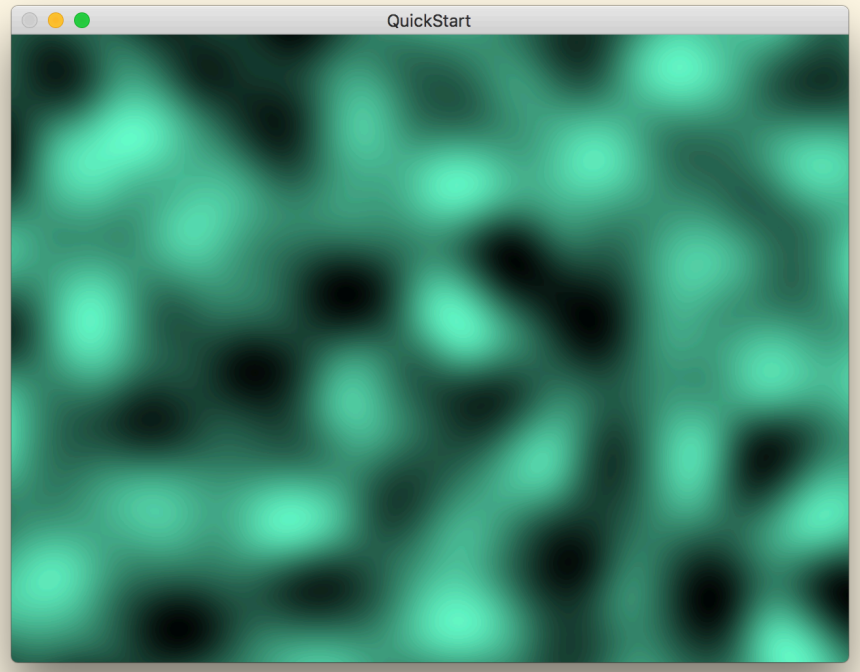
Tip

The ability to modify a composition while it's running and see the results immediately, as in this example, is called *live editing*.

1.1.5 Editing an input port value

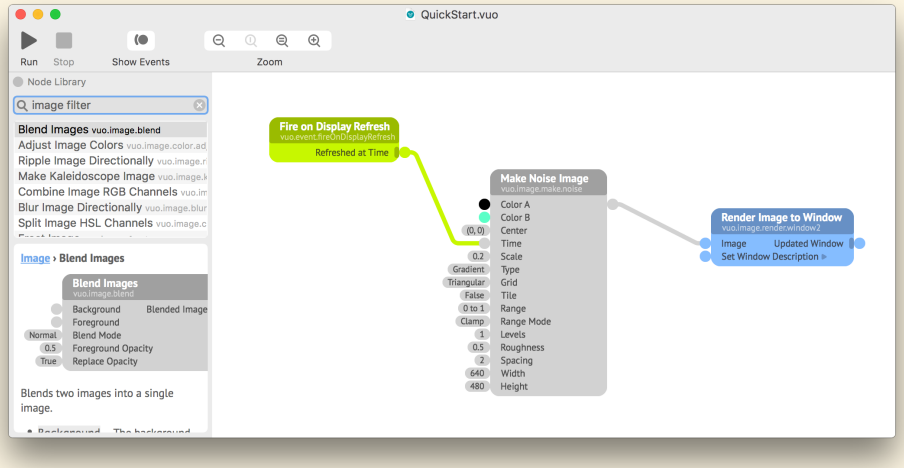
You can alter how a node does its job by editing its input port values. For example, double-click on the **Make Noise Image** node's **Color B** input port and choose a different color.



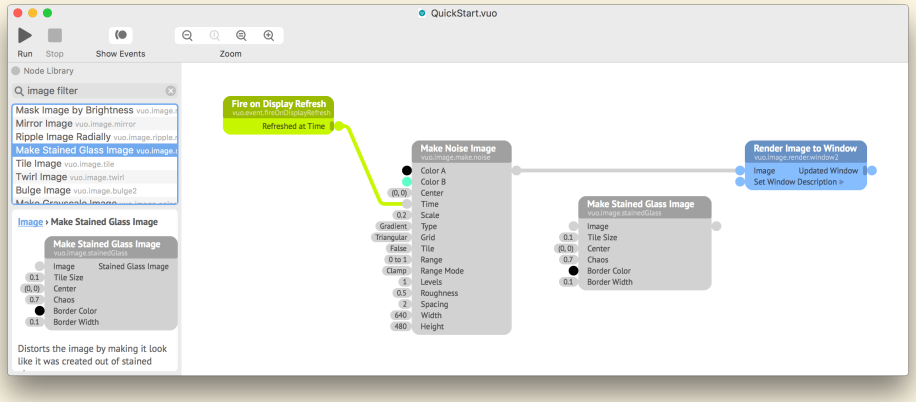


1.1.6 Adding another node

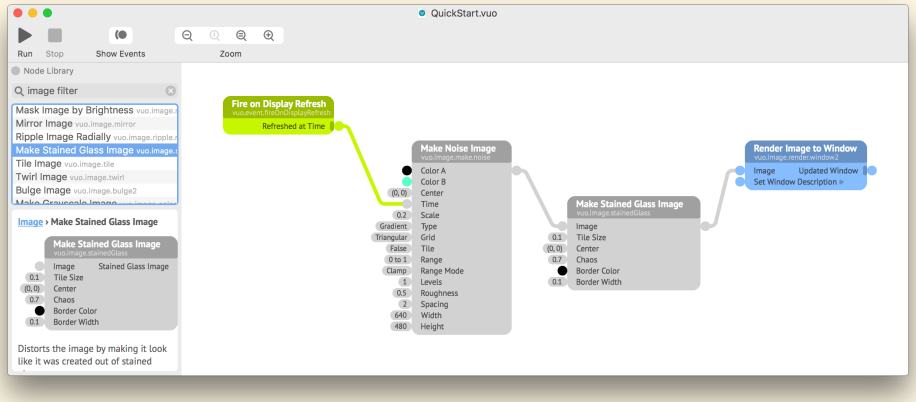
You can add more nodes to your composition to make more interesting effects. Search the Node Library for “image filter” to see the built-in image effects.

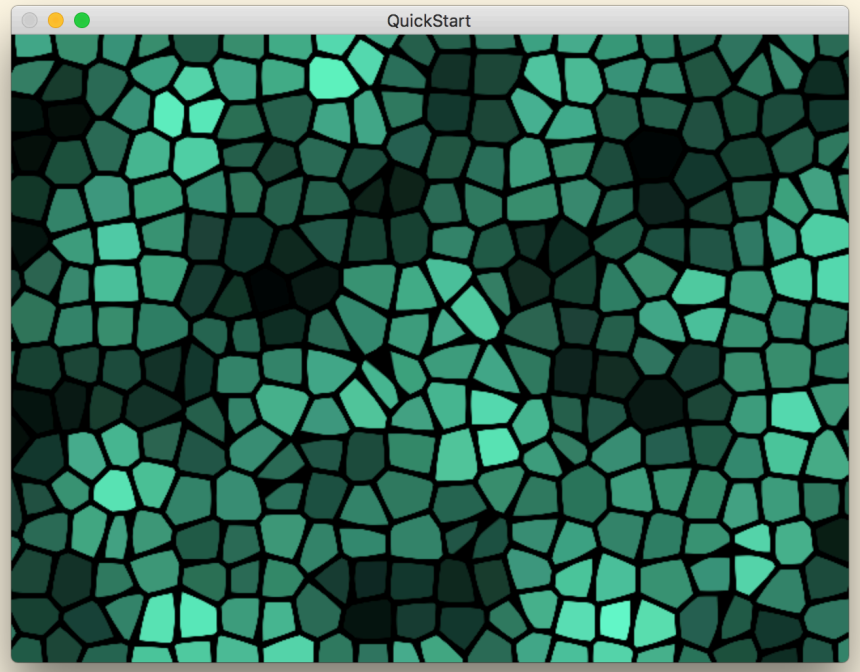


Drag the **Make Stained Glass Image** node from the Node Library onto the canvas.

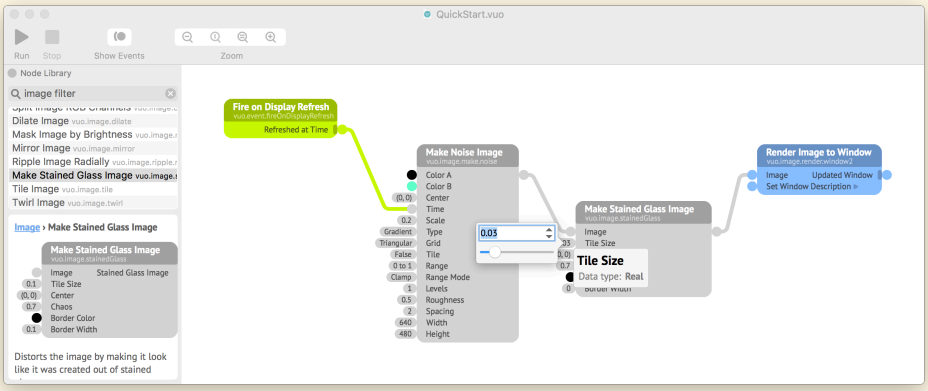


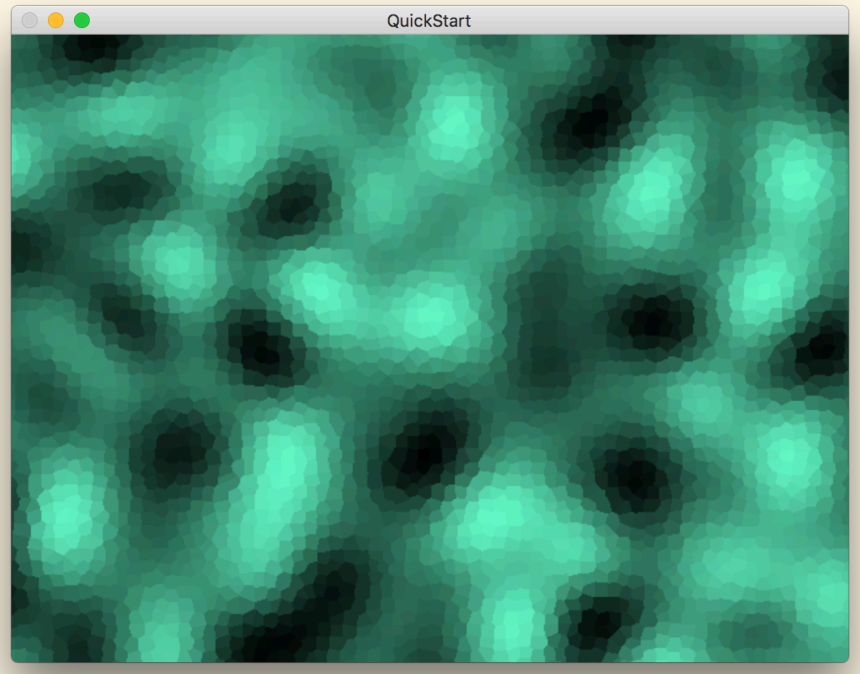
Draw a cable from the **Make Noise Image** node's output port to the **Make Stained Glass Image** node's **Image** input port, then another cable from the **Make Stained Glass Image** node's output port to the **Render Image to Window** node's input port.





As you did with the **Make Noise Image** node, you can adjust how the **Make Stained Glass Image** node affects the image by editing the node's input port values.





1.1.7 Summary

This example covered many of the basics of using Vuo.

- You learned that each node is in charge of one job.
- You learned that nodes work together by communicating through cables connected at ports.
- You launched your composition into action with the Run button.
- You searched the Node Library for a node that does a specific job.
- You added nodes to the canvas and connected them with cables.
- You changed settings such as colors by editing input ports.
- You used specimens of some common varieties of nodes: nodes that are in charge of timing, nodes that generate and filter images, and nodes that render graphics to a window.

Next, we'll take a closer look at what exactly happens while a composition is running.

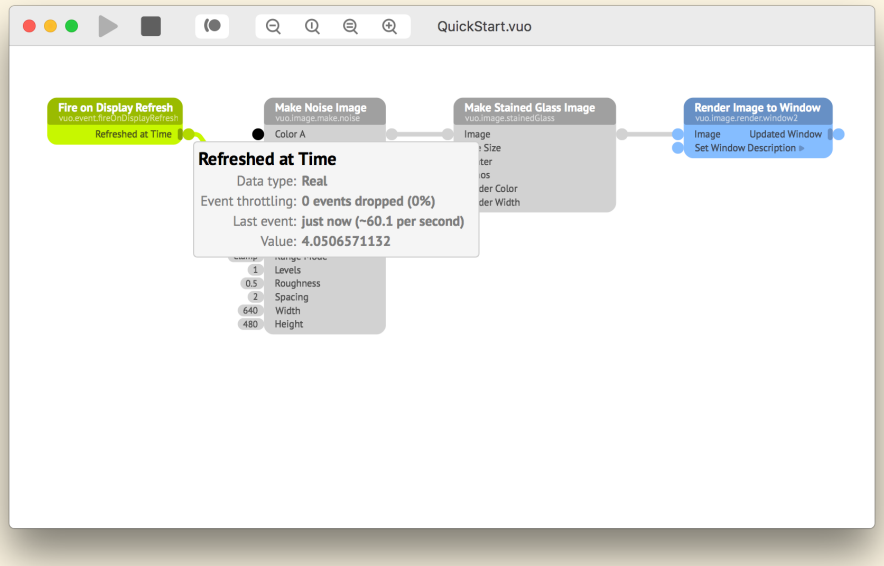
1.2 Tracing through a composition

When you ran the Quick Start composition, the four nodes in the composition worked together to create an end result: a window displaying an animated pattern. As you added each node to the composition, you saw how it contributed to the end result. Now let's look at each node's contribution in more detail.

1.2.1 Port popovers

If you want to understand the inner workings of a composition, *port popovers* are an extremely useful tool. You can visualize step by step how each node contributes to the end result.

Open the port popover for the **Fire on Display Refresh** node's **Refreshed at Time** output port by clicking on the port. The small window that appears is the port popover. As the composition runs, the port popover shows the information flowing through the port in real time.



1.2.2 Information flow

Two kinds of information can flow through ports: *data* and *events*.

For the **Refreshed at Time** port, the data is the time — the number of seconds since the composition started running. At the moment the screenshot above was taken, the port popover showed that the data was about 4.05.

The port popover also shows that the **Refreshed at Time** port is outputting information at about 60 times per second (the monitor's refresh rate). Every 1/60 second, the port outputs a slightly greater number of seconds — accompanied by a second piece of information called an event.

Events control the timing of your composition. An event is an impetus or signal that tells a node that it's time to do its job.

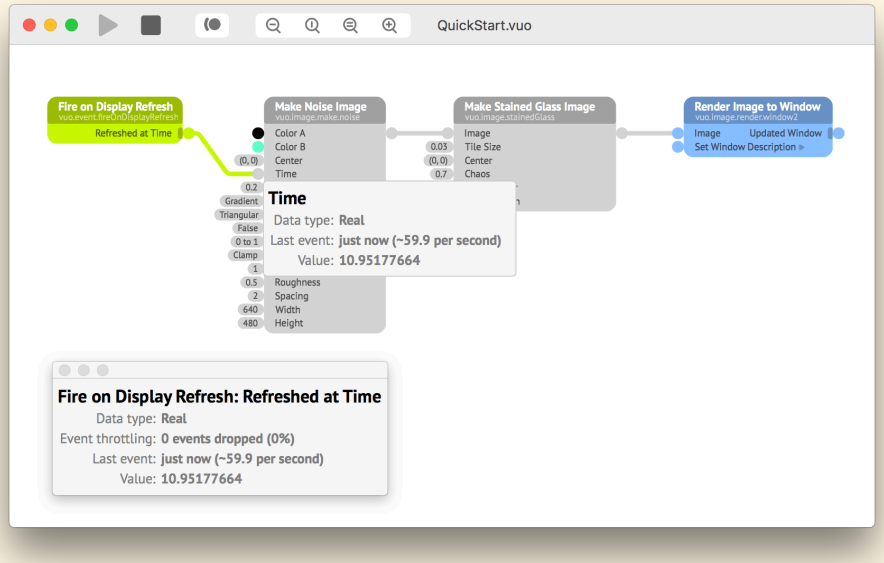
When you run the Quick Start composition, the nodes do their jobs one at a time, left to right. That's not because they happen to be placed left to right on the canvas, but because the events and data flow through nodes and cables in a methodical way — which we'll trace through now using port popovers.

Tip

Why are events and data separate things? This will become clear later when you learn about event-only ports and event-only cables, in which events travel without data.

1.2.3 Step 1: Fire on Display Refresh to Make Noise Image

Click on the port popover for **Refreshed at Time** so it will stay open on the canvas. Then click on the **Make Noise Image** node's **Time** input port to open its popover.

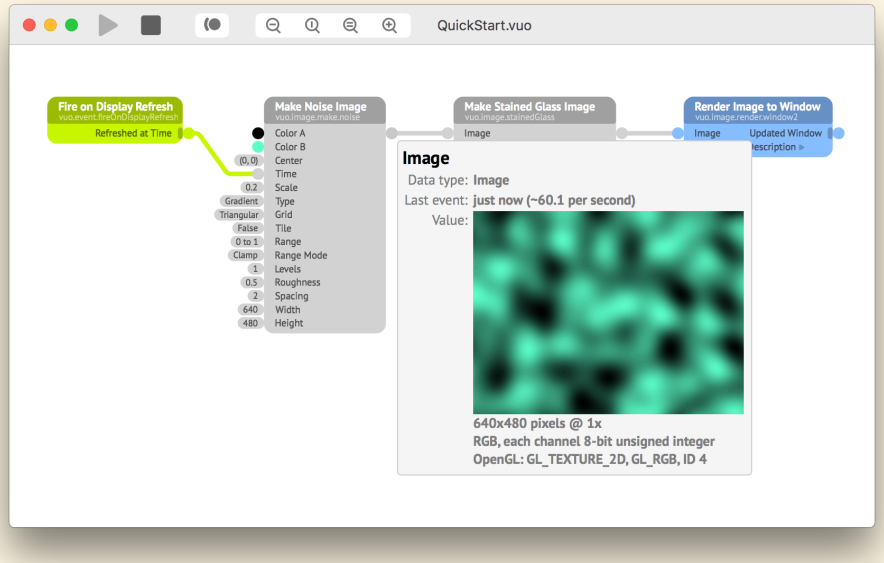


As you can see, the events and data shown in the two port popovers are the same. A stream of information is flowing out of the **Refreshed at Time** output port, along the cable, and into the **Time** input port.

As each event hits the **Make Noise Image** node's **Time** input port, it prompts the **Make Noise Image** node to do its job. The node does so, using the data that accompanied the event as one of its parameters.

1.2.4 Step 2: Make Noise Image to Make Stained Glass Image

Open the port popover for the **Make Noise Image** node's output port. It shows a small version of the image created by the node.

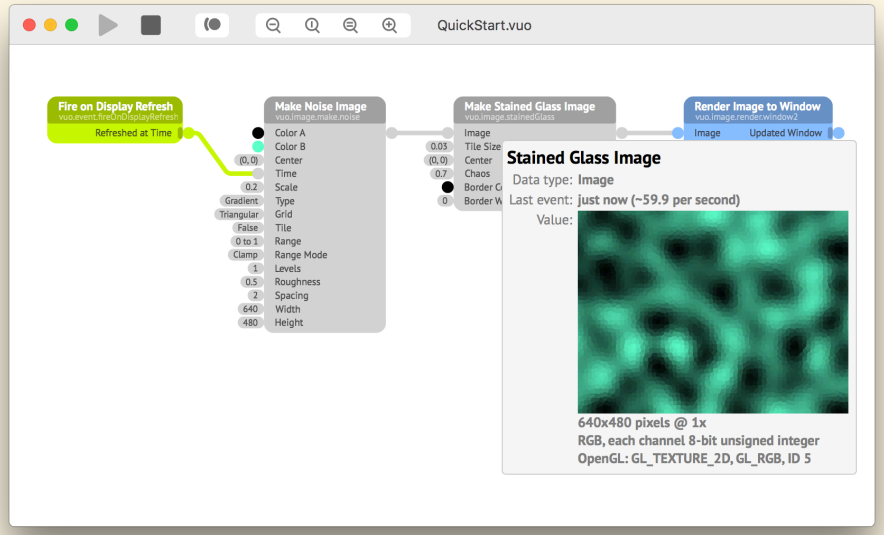


Each time the **Make Noise Image** node does its job, it sends two pieces of information through its output port — the same event that came in the **Time** input port, accompanied by new data: the image created by the node.

The event and data then flow along the cable to the **Make Stained Glass Image** node's **Image** input port.

1.2.5 Step 3: Make Stained Glass Image to Render Image to Window

Open the port popover for the **Make Stained Glass Image** node's output port.



Now that the **Fire on Display Refresh**, **Make Noise Image**, and **Make Stained Glass Image** nodes have worked together to produce the image shown in the port popover, the final step is for the **Render Image to Window** node to display the image in a window.

1.2.6 Summary

By tracing through the Quick Start composition, this section illustrated some skills and concepts for understanding how a composition works.

- You learned that Vuo has two basic kinds of information: data and events.
- You opened port popovers to reveal the data and events flowing through the composition.
- You watched two types of data, numbers and images, flowing through the composition.
- You observed the rate at which events were prompting nodes to do their jobs.

1.3 Learning Vuo

Now that you've perused the Quick Start and Tracing through a composition sections, you've been exposed to the key concepts underlying Vuo and are well on your way to creating your own custom compositions. Before digging deeper into those concepts, we'll mention some resources that teach Vuo in different ways. You can choose the path that best matches your learning style.

1.3.1 User manual

The user manual (this document) provides the most detailed documentation of the concepts underlying Vuo and of Vuo's user interface. You may choose to read it all the way through, or you may refer to it when you have questions about a specific topic.

Terms used in this manual are defined in the glossary at the end.

A high-contrast version of the manual is available on our release notes page.

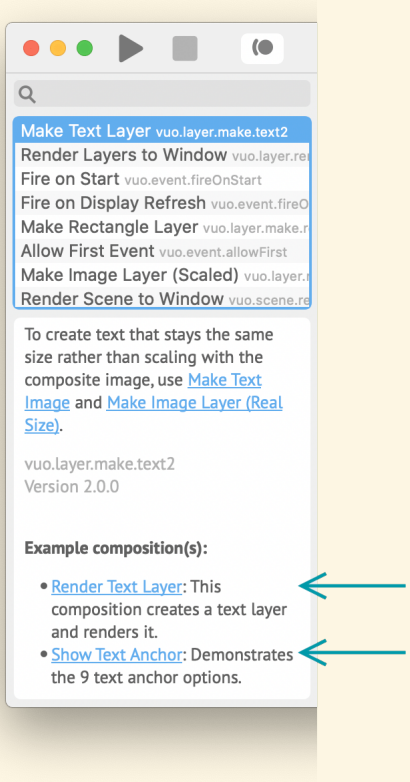
1.3.2 Tutorials

Video tutorials are available on our tutorials page.

1.3.3 Example compositions

Vuo comes with over 200 example compositions that demonstrate how to accomplish tasks in Vuo.

Example compositions can be quite helpful when learning how to use a node. Many nodes have relevant example compositions listed in their Node Documentation Panel.



Tip

You can search for an example composition by name in the Help menu's Search box.

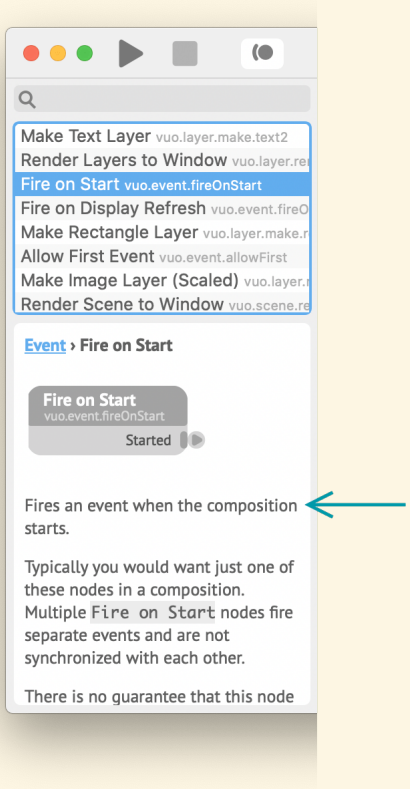
To browse the list of all example compositions, go to File > Open Example.

1.3.4 Community support

The community of people who use Vuo can be an incredibly helpful resource when you're learning Vuo. As part of that community, you can discuss questions and answers on how to use Vuo, share compositions, and suggest features to improve Vuo. To get started, visit our community page.

1.3.5 Node documentation

Every built-in node and node set in Vuo comes with documentation that explains how to use it.



Alternatively, you can browse the online node documentation.

1.3.6 SDK documentation

If you're a developer who would like to embed Vuo in an application or to implement custom nodes, you can explore the API documentation.

1.3.7 Vuo in other applications

If you use CoGe or VDMX (VJ applications that mix and composite media), you can install Vuo compositions to add to your available visual effects. This manual explains how to set up your compositions in Making compositions fit a mold with protocols. To learn how to install and use compositions in CoGe and VDMX, check out their documentation.

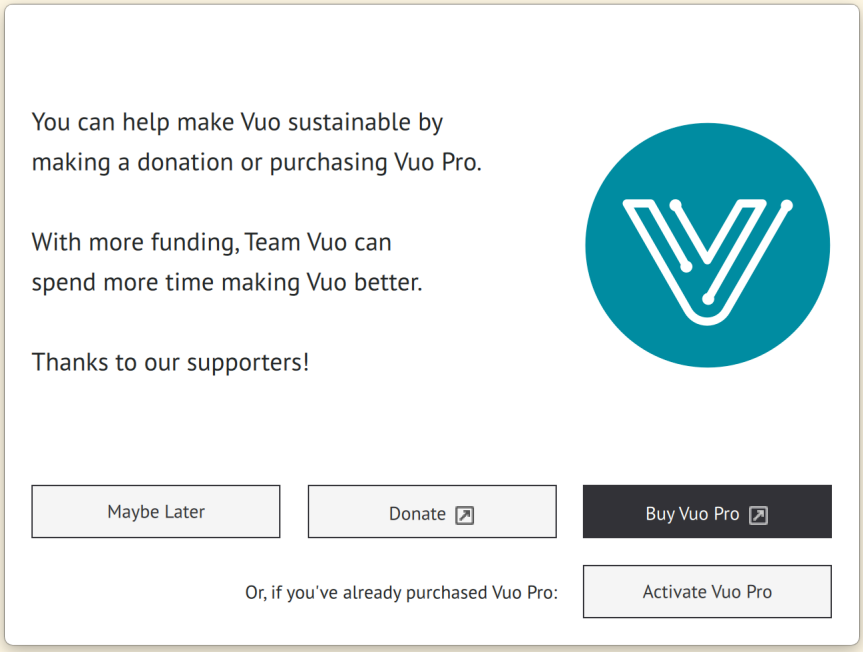
1.4 Installing Vuo

- Go to https://vuo.org/download.
- Click the "Download Vuo" button.
- Uncompress the ZIP file (double-click on it in Finder).
- Move the Vuo application to your 🗀`Applications` folder.
- Open the Vuo application.
- Follow the instructions in the dialogs.

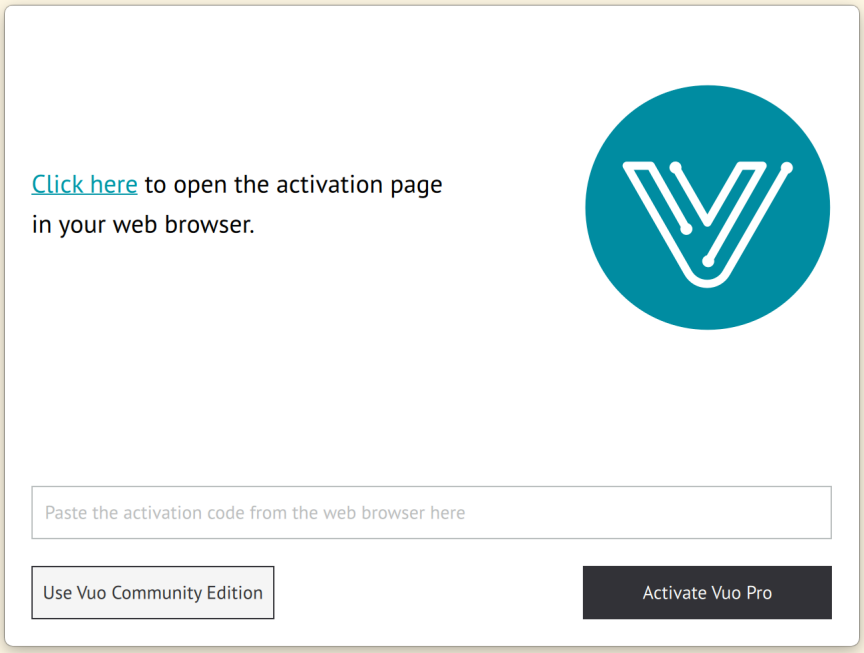
1.4.1 Activating Vuo Pro

If you've purchased Vuo Pro, you'll need to activate your license in the application.

After launching Vuo, when you reach the dialog below, click Activate Vuo Pro.



In the next dialog, follow the instructions to activate Vuo Pro.



1.5 Changing language settings

New in Vuo 2.0

Vuo has been partially translated to *español* (Spanish), *français* (French), and *Deutsch* (German). If your computer is already using one of those languages, then the Vuo editor will appear in that language.

To change the language of Vuo and other applications:

- In macOS System Preferences, go to Language & Region > General (or just Language & Region on earlier macOS versions).
- Under "Preferred languages", move the desired language to the top of the list.
- Relaunch Vuo.

To change the language of Vuo only (macOS 10.15+):

- In macOS System Preferences, go to Language & Region > Apps.
- Add Vuo and select the desired language.
- Relaunch Vuo.

If you can help correct translations or translate Vuo to another language, we'd appreciate it! Please contact us.

2 The basics

The previous section walked you through the steps of creating a simple composition. By now, you may know a bit about the process of composing with Vuo, but you may not understand exactly how compositions work or how make your own from scratch. This section introduces the major concepts you need to understand when working with Vuo.

If you prefer to learn by doing, we recommend that you read this section and then experiment with Vuo's example compositions to learn how to create your own. If you prefer to have a deeper understanding of the concepts underlying Vuo, we recommend that after this section you continue to the next sections, which cover the concepts in more detail — How events and data travel through a composition, How compositions process data, and How nodes can be used as building blocks.

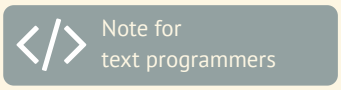
2.1 A composition is what you create with Vuo

When musicians create a piece of music, they call it a composition. When you create something in Vuo, that's also called a **composition**.

In the Quick Start section, you saw how to create a composition that displays a moving twirly stripy design. That's one type of Vuo composition — an animation that displays in a window. Vuo can be used to create much more complex and interesting animations. It can also be used to create many other types of compositions. A composition could be a game. It could be an art installation. It could be a controller for stage lighting. It could be digital signage. It could be a plug-in for other software. Those are just some examples of what a composition could be.

One thing that all compositions have in common is the process of creating them in Vuo. You start with either a new canvas or an existing composition, and you pick out building blocks and connect them to make many smaller pieces work together as a larger whole.

Another thing that all compositions have in common is the way that they run. When you click the Run button, all of those building blocks and connections that you laid out as a blueprint get turned into a running application.



A composition is a program whose source code is a visual representation of the program's data flow. It's compiled and linked to create an application or library.

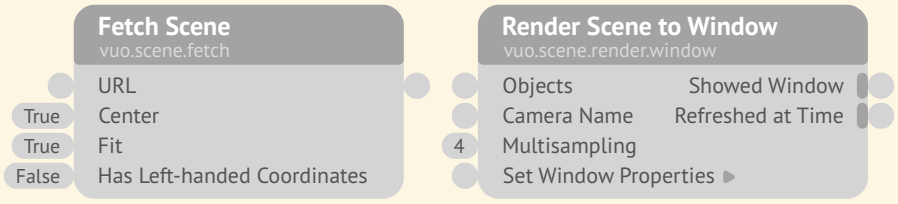
2.2 Nodes are your building blocks

Each composition does something unique, and the way that you build up that something is by putting together **node**s. These are your building blocks.

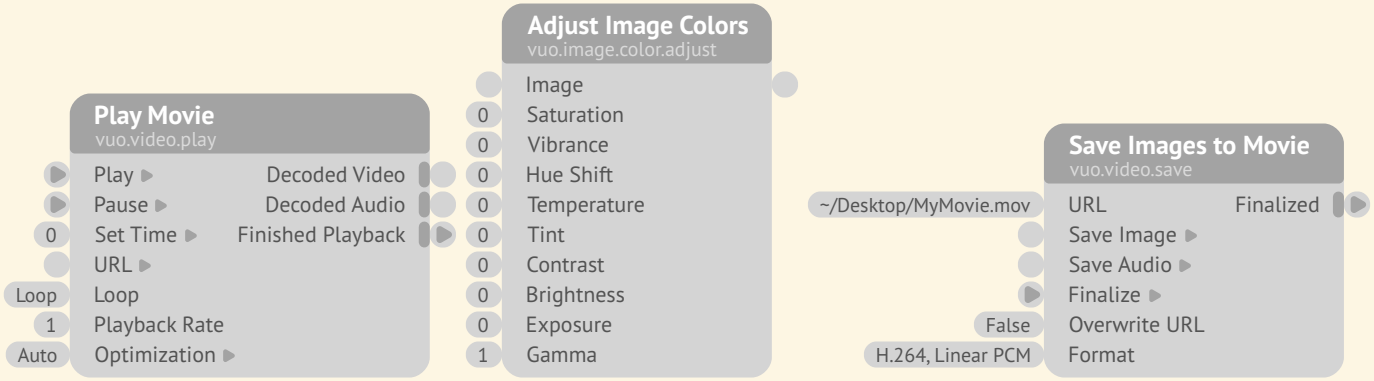
Note for text programmers

A node is like a function. It encapsulates a task. It takes inputs and produces outputs. More precisely, nodes are like class instance methods, since they can also maintain a state.

Let's say you're creating a composition that displays a 3D model. You might use the **Fetch Scene** node to load the 3D model from a file and the **Render Scene to Window** node to render the model in a window.



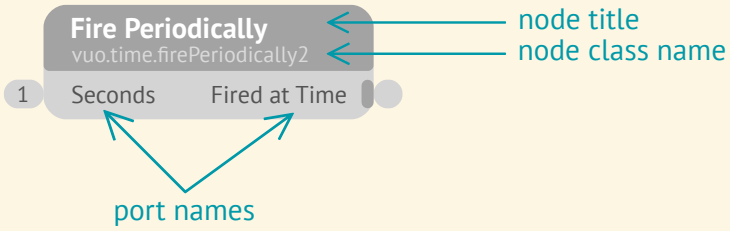
Or suppose you're creating a composition that applies a color effect to a movie. You might use the **Play Movie** node to bring the movie into the composition, the **Adjust Image Colors** node to change the movie's color, and the **Save Images to Movie** node to save the color-changed movie to a file.



Part of the process of creating a composition is taking your idea of what it should do and breaking that down into smaller tasks, where each task is carried out by a node. Each node in Vuo has a specific job that it does. Some nodes do simple jobs, like adding numbers or checking if two pieces of text are the same. Other nodes do something complex, like receiving a stream of video from a camera, finding a barcode in an image, or turning a 3D object into a wiggly blob. You can browse through a list of all the nodes available in the Node Library (the panel along the left side of the Vuo editor window) or the online node documentation.

When you start making a composition, often the first thing you'll do is pick a node from the Node Library. You can search the Node Library for what you want to do (for example, a search for "movie" brings up a list of nodes for playing, inspecting, and saving movies) and then drag the nodes you want onto the composition canvas.

You can learn about a node by looking at its title, node class name, and port names, which are pointed out in the illustration below. For a detailed description of how the node works, you can look at the node's documentation, which appears in the Node Documentation Panel in the lower part of the Node Library. Many nodes come with example compositions (listed in the node's documentation) that demonstrate the node in action.



2.3 Events are what cause nodes to execute

Let's think again about creating a composition that applies a color effect to a movie. Your first step might be to drop a **Play Movie** node, an **Adjust Image Colors** node, and a **Append to Movie** node onto the canvas. Then what? How do you tell the composition that, first, you want **Play Movie** to bring the movie into the composition, second, you want **Adjust Image Colors** to apply the effect, and third, you want **Append to Movie** to save the movie to a file? The way that you control *when* nodes do their job and *how* information flows between them is with **event**s.

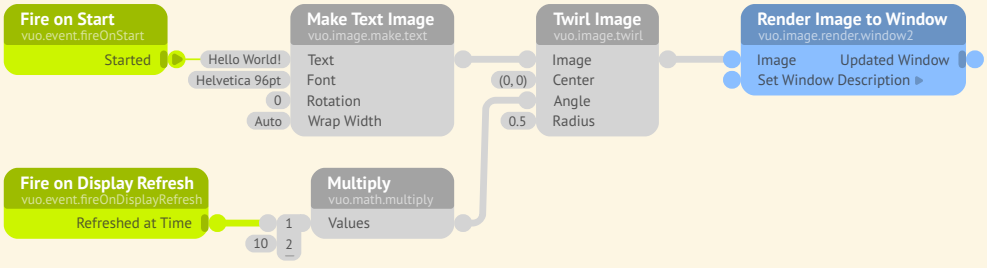
</> Note for text programmers

Vuo is event-driven. The events are generated by trigger ports, and the event handlers are implemented by the nodes executed as the event travels through the composition.

Here's a composition that simply displays some text on a window:

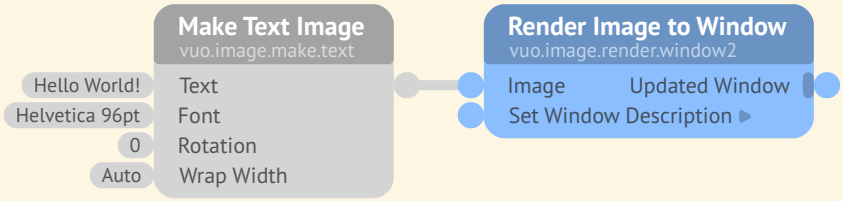
How do events come into play in this composition? This composition involves a single event that causes the text to render as soon as the composition starts running. The event is **fire**d (originates) from the **trigger port** called **Started** on the **Fire on Start** node. (A trigger port is a special kind of port, which you can recognize by the thick line along its left side.) The event travels to the **Make Text Image** node, causing that node to **execute** (do its job). The event then travels onward to the **Render Image to Window** node, causing it to execute as well. From the **Make Text Image** node to the **Render Image to Window** node, the event carries with it the image that was created by **Make Text Image** and will be rendered by **Render Image to Window**.

Here's a variation on that composition that involves multiple events:



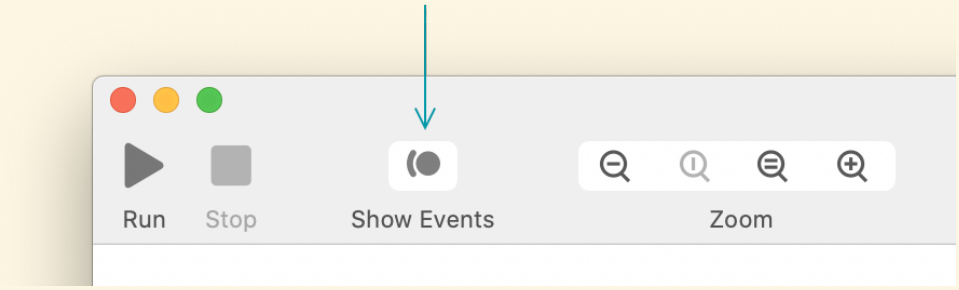
This composition displays an animation of the text becoming more and more twirled as time passes. It still has the event fired from the **Fire on Start** node's **Started** port when the composition starts. It also has events being fired from another trigger port: the **Fire on Display Refresh** node's **Refreshed at Time** port. Unlike the **Started** port, which fires only once, the **Refreshed at Time** port fires 60 times per second (or whatever your computer display's refresh rate is). Unlike the event from **Started** port, which is useful for doing something once, the events from the **Refreshed at Time** port are useful for doing something continuously, such as displaying an animation that changes smoothly over time. In the composition above, each of those 60 times per second that the **Refreshed at Time** node fires an event, that event (along with a piece of information that says how long the composition has been running) travels to the **Multiply** node. The event (along with the result of multiplying numbers) travels to the **Twirl Image** node. Finally, the event (along with the twirled image) travels to the **Render Image to Window** node. As the event travels along its path, it causes each node to execute in turn, and carries information with it from one node to the next.

Here's a composition that *doesn't* display text in the window (can you guess why?):



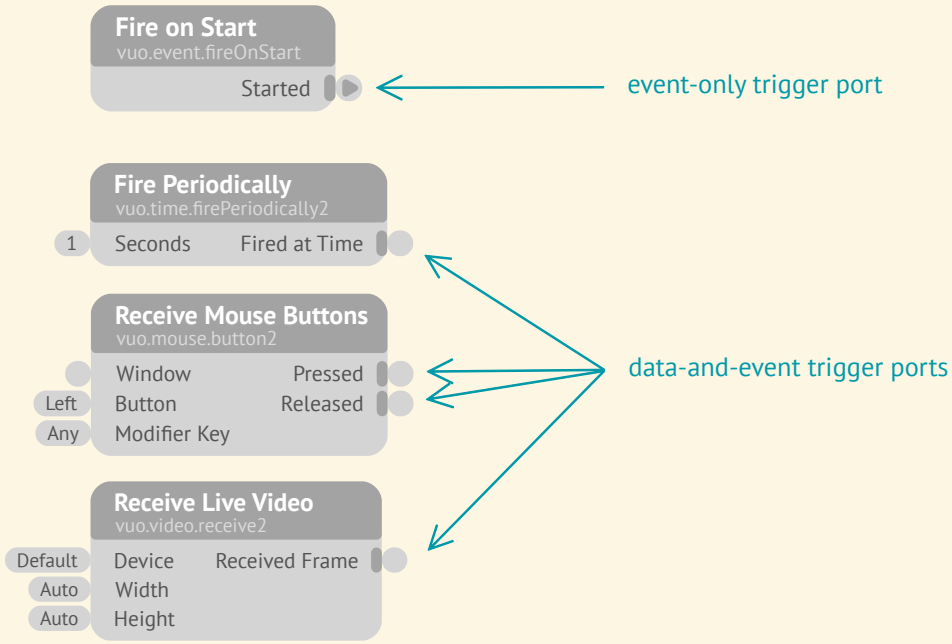
This composition doesn't have any events going into the **Make Text Image** node. Without any incoming events, the **Make Text Image** node never executes and never passes an image along to the **Render Image to Window** node. So no text is displayed. If you want a node to execute, make sure you feed it some events!

If you'd like to watch the events moving through a composition, you can do that by clicking the Show Events button in the toolbar. As the composition runs, you can see the events being fired from trigger ports, and you can trace the path of the event by watching each node change color as it executes.



2.4 Trigger ports fire events and sometimes data

As you just saw, events are fired from trigger ports, which are special ports that some nodes have. Here are some examples of trigger ports:



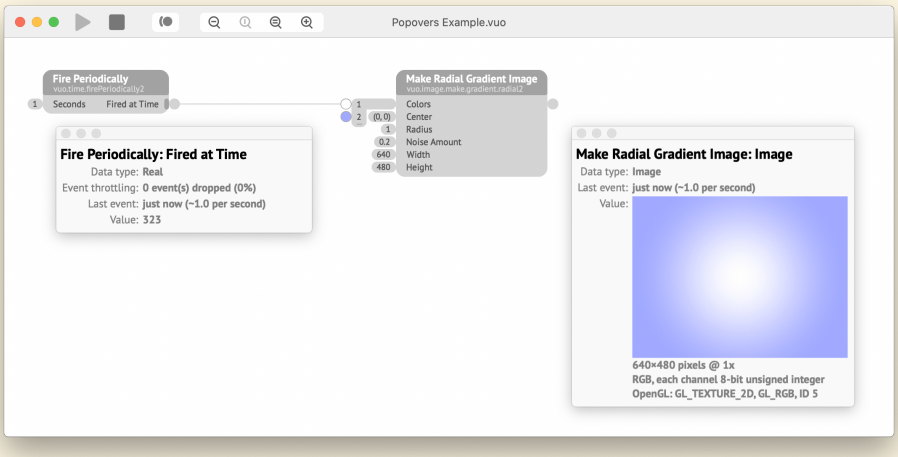
The **Started** trigger port on the **Fire on Start** node fires a single event when the composition starts running. The **Fired at Time** trigger port on the **Fire Periodically** node fires events at a rate determined by the node's **Seconds** port. The **Pressed** trigger port on the **Receive Mouse Buttons** node fires an event each time the mouse button is pressed, and the **Released** trigger port fires an event each time the mouse button is released. The **Received Frame** trigger port on the **Receive Live Video** node fires events as it receives a stream of images from a camera.

Some trigger ports, like **Started**, fire just events. Other trigger ports, like **Pressed**, **Released**, and **Received Frame**, fire **data** (a piece of information) along with each event. The **Pressed** and **Released** ports fire the coordinates of the point where the mouse was pressed or released. The **Received Frame** port fires the video frame received from the camera. This data travels along with the event to the next node. When that node executes, it can use the data to do its job (such as drawing a shape at the given coordinates, or extracting an image from the given video frame).

Nodes with trigger ports are often responsible for bringing information into the composition from the outside world, such as video, audio, device input, and network messages. These nodes can be a

good starting point when creating a composition. You can see a list of all nodes with trigger ports by searching the Node Library for “trigger” or “fire”.

As just mentioned, one way to watch what trigger ports are doing in a composition is to run the composition with Show Events enabled. Another way is to click on the trigger port, which opens a view called the Port Popover. As the composition runs, the Port Popover shows how recently the trigger port fired an event and what data (if any) came with the event.



Tip

If you click on the Port Popover, it becomes a small window that you can leave open as you continue working and perhaps open other Port Popovers.

2.5 Events and data travel through cables

Let’s take yet another look at this composition that displays text in a window:

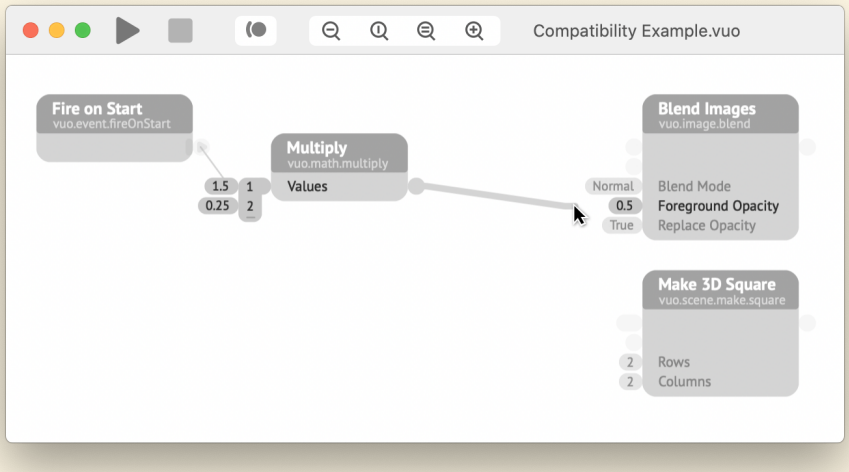
The lines connecting the nodes are called **cable**s. Cables are the conduits that data and events travel through.

In the composition above, an event travels along the cable from the **Started** trigger port of the **Fire on Start** node to the **Text** port of the **Make Text Image** node. An event and data travel along the cable from the **Make Text Image** node to the **Render Image to Window** node’s **Image** port. Notice the difference between the two cables: the first cable is thinner since it only carries events (an **event-only cable**), while the second cable is thicker since it carries both events and data (a **data-and-event cable**).

Often it helps to think of cables as pipes that data and events flow through. Like water flowing through a pipe, events and data flow through the cable from one end to the other, always in the same direction. Extending the water analogy, you can think of trigger ports as being **upstream** and the nodes that their events flow to as being **downstream**.

But, unlike water flowing through a pipe, events and data travel as discrete packets instead of a continuous flow. Another way to think of a cable is as a one-way, one-lane road on which each event is a car. On some roads (data-and-event cables), each car carries a piece of data.

You can create a cable by dragging the mouse from one port to another. While you're dragging, the ports that you're allowed to connect the cable to are highlighted. If you're not allowed to connect a cable from one port to another, it's because the two ports have different, incompatible types of data. For example, you can't connect a port whose data is a number to a port whose data is a 3D model.



2.6 Events and data enter and exit a node through ports

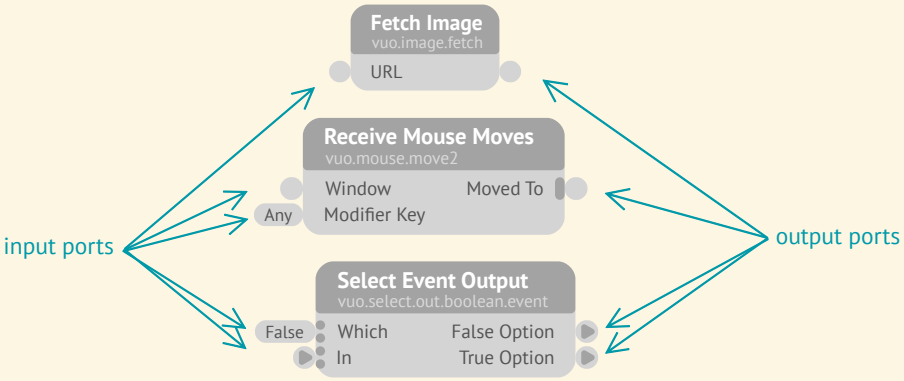
When an event (and possibly data) is fired from a trigger port and travels along a cable, what happens when it reaches the port on the other end of the cable?

That port on the other end is called an **input port**. You can think of it as a portal that inputs (receives) information into the node.

In the above composition, the **Text** input port of the **Make Text Image** node inputs an event, which causes the node to execute. The **Image** input port of the **Render Image to Window** node inputs the event and an image. When the node executes, it uses that image to do its job of rendering an image to a window.

You may have noticed that, in the above composition, some input ports have data that's attached to the port rather than coming in through a cable. The **Text** input port has the data "Hello World!", and the **Font** input port has as its data a description of a Helvetica font. These are called **constant value**s because they don't vary the way that data coming through a cable can. Like data coming in through cables, constant values are also used by the node when it executes. If a port has a constant value, you can edit it by double-clicking on it.

After a node executes, it outputs (sends) information through its **output port**s. The information outputted — events and possibly data — can then travel along cables from the output ports to other input ports.



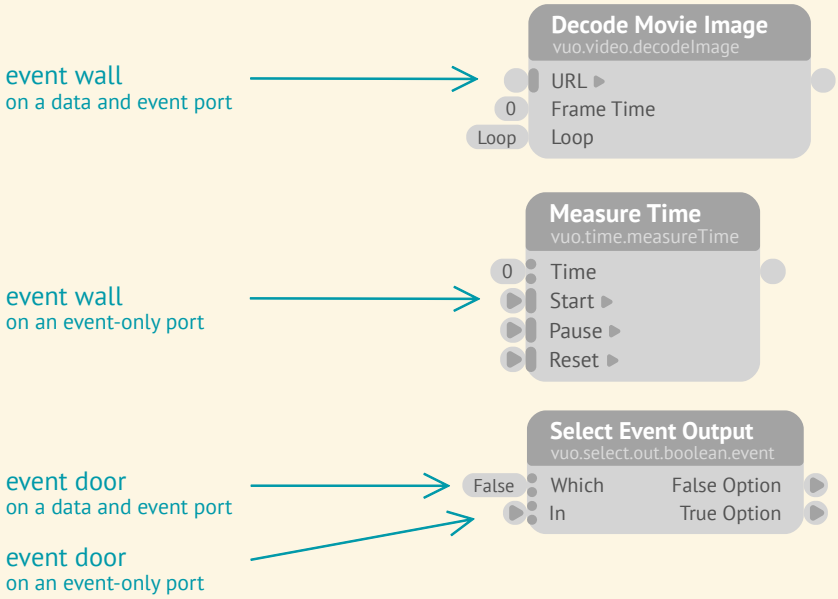
On most nodes, every event that comes in through one or more inputs ports goes out of all of the output ports. But there are a couple of exceptions.

One exception is trigger ports. Although trigger ports are output ports, events that come in through input ports are never outputted through them. Trigger ports can only fire new events, not transmit existing events.

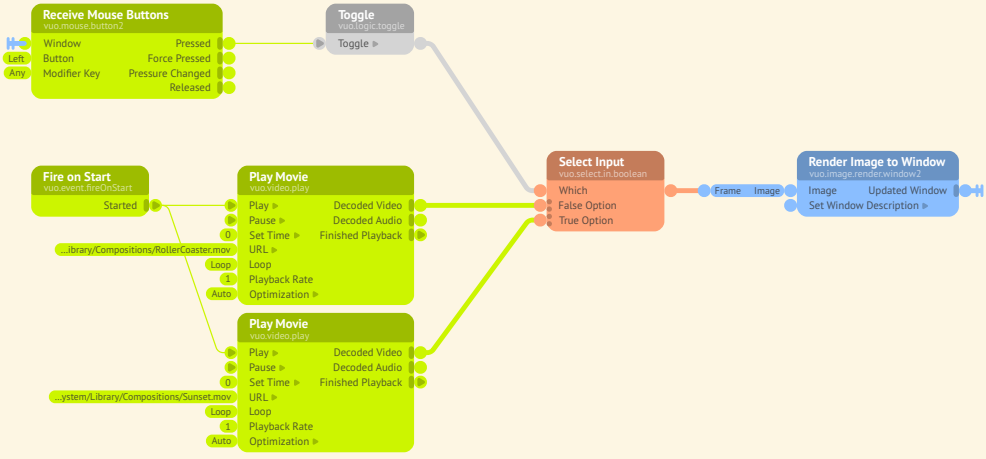
The other exception is for nodes whose input ports have thick lines along their right side, which are called **event wall**s and **event door**s. If an event comes into a node only through an input port with a wall, then the event won't go out any of the node's output ports. If an event comes in only through an input port with a door, then the event may or may not go out of some or all of the node's output ports — the exact behavior depends on the node, and is explained in the node's documentation.

Tip

Both for trigger ports and for walls and doors, the thick line is a hint to remind you that events may be blocked.



The composition below, Select Movie (File > Open Example > Select), demonstrates one way that event doors can be useful. This composition displays one of two movies at a time, switching between them each time the mouse is pressed. The doors on the **Select Input** node's **False Option** and **True Option** input ports allow the node to let the stream of events and video frames from one movie through while blocking the stream from the other movie.

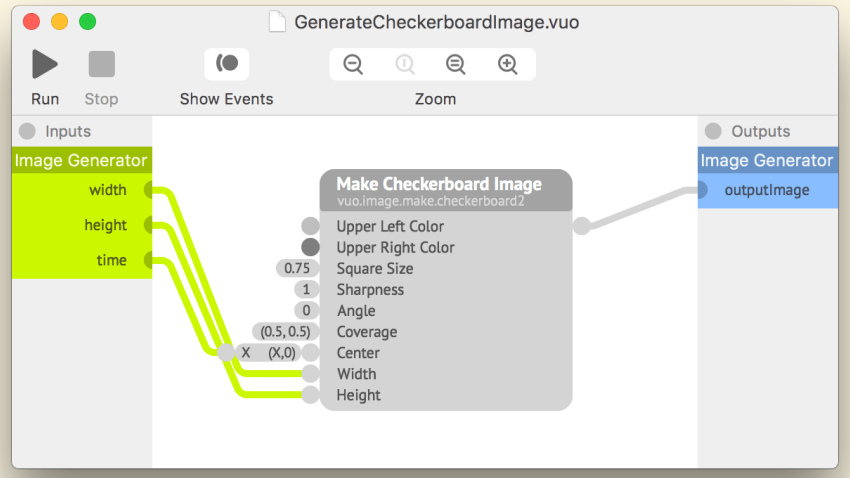


If you're not sure if a node is letting events through or blocking them, you can enable Show Events or look at Port Popovers to see where events are flowing.

2.7 Events and data enter and exit a composition through published ports

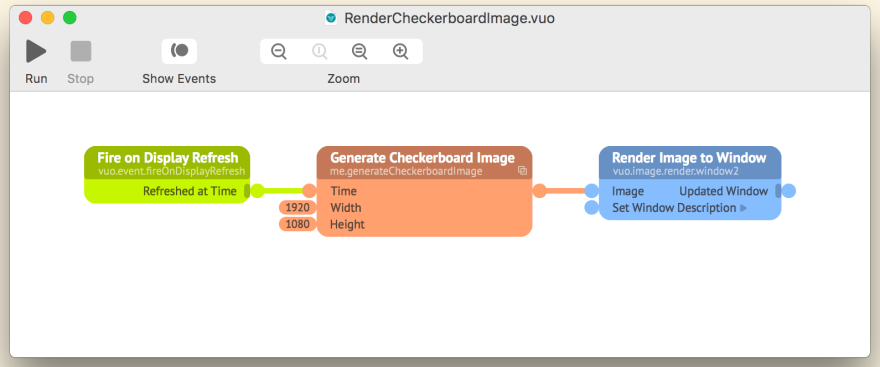
Earlier, you learned that a composition is made up of nodes, each of which is a building block that has a specific job to perform. If you think about it, the composition as a whole also has a specific job to perform. It's like a node, but on a larger scale. A composition can even be used as a building block within another composition or another application.

Just as a node can input and output information through its ports, a composition can input and output information through **published port**s. If a composition has published ports, Vuo shows them in sidebars along the left and right sides of the composition canvas.



Above is an example of a composition with published ports: Generate Checkerboard Image (File > Open Example > Image Generator). It inputs events and data through published input ports called **width**, **height**, and **time**. It outputs events and data through a published output port called **outputImage**.

You can use this composition as a building block, called a *subcomposition*, inside of another composition. Below is what that looks like — the published input and output ports of the composition become the input and output ports of a node. (You'll learn more about subcompositions in Using subcompositions inside of other compositions.)



Because this composition has a certain set of published ports, making it an *image generator*, you can use it in other special ways. You can install it as a plugin for a VJ application that supports Vuo plugins. You can run it in Vuo to see a preview of the video stream it would generate in a VJ application. You can export a movie of the video stream. (More about image generators is in Making compositions fit a mold with protocols.)

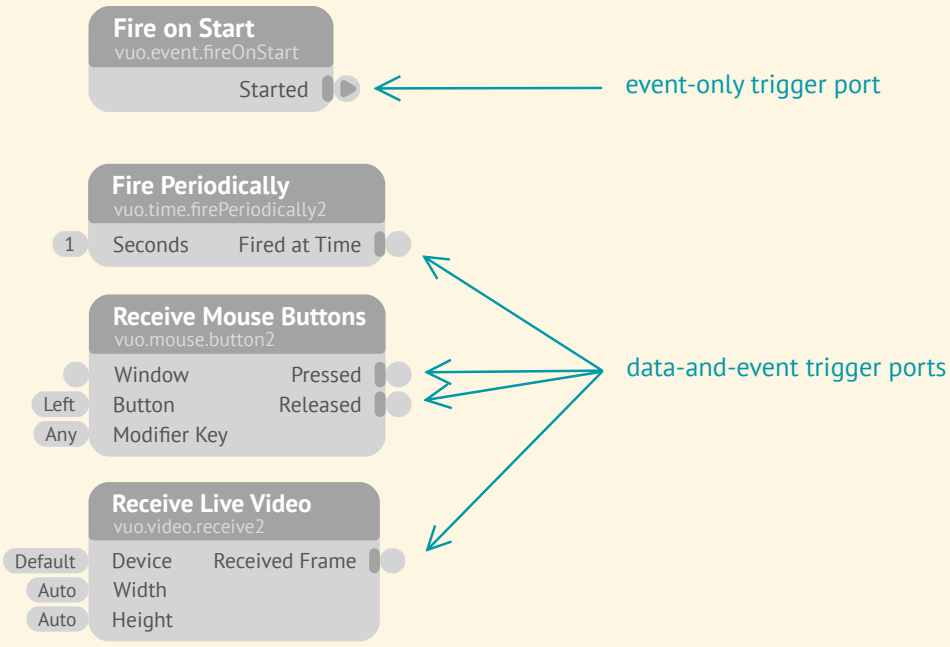
In all of these different ways of executing the composition — as a subcomposition, within a VJ application, as a preview, for a movie export — events and data enter the composition through its published input ports, flow through the composition, and exit through the published output ports.

3 How events and data travel through a composition

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.

3.1 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 **Fire on Display Refresh**

node's **Refreshed at Time** trigger port fires a rapid series of events, so that you can use these events to display graphics in rapid sequence as an animation.

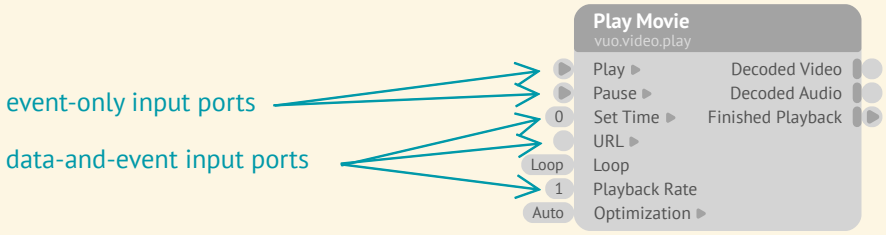
Other trigger ports fire events in response to things happening within the composition. For example, the **Fire on Start** node's trigger port fires an event when the composition starts. The **Fire Periodically** node's trigger port fires events at a steady rate while the composition is running. A node's 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 Run slow parts of the composition in the background.)

Some nodes block events until a certain condition is met. The node **Became True**, for example, only lets an event through when the condition changes from false to true. These nodes are not trigger nodes, since they don't create events, but they control when events are output.

3.2 How events travel through a node

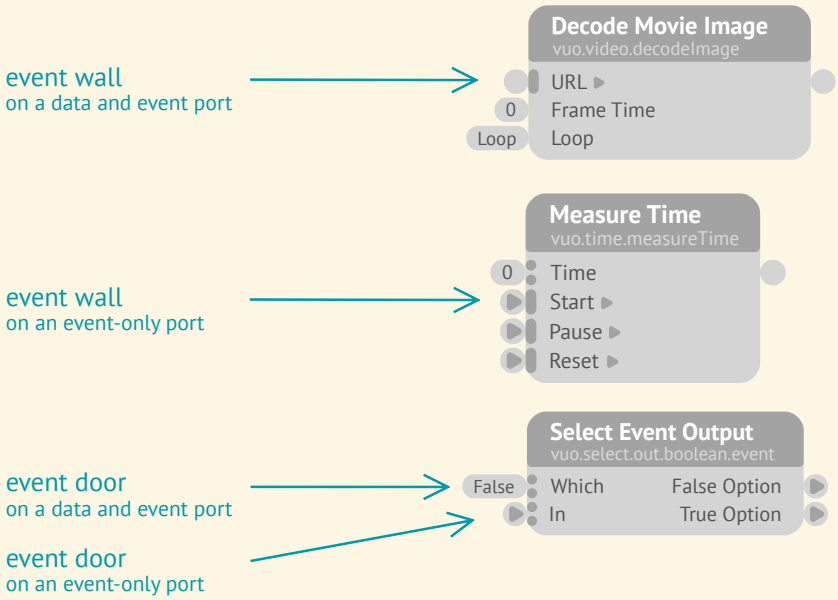
An event can come into a node through cables connected to one or more of its input ports. When an event reaches the node's input ports, the node executes, and it may or may not send the event through its output ports.

3.2.1 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.

3.2.1.1 Event walls and doors Some nodes, like the ones 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. 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.



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.

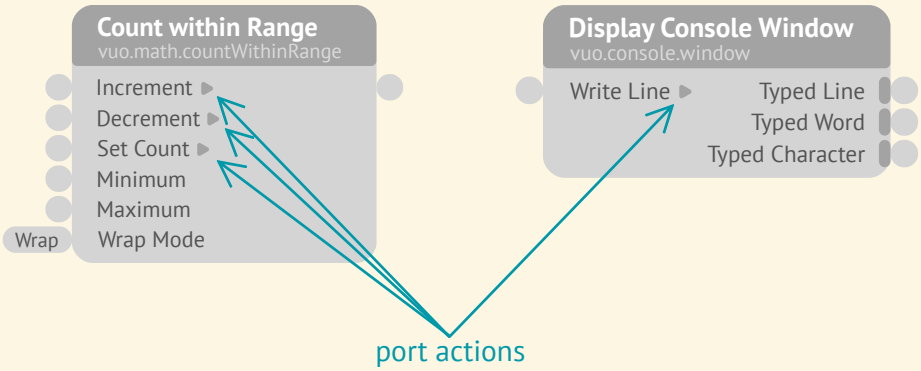
Ports that always block events have a solid semi-circle (like the **URL** port above) or a solid chevron (like the **Start** port above). This is called an **event wall**. The node must receive an event from another port without an event wall for the results of the node's execution to be available to other nodes.

Ports that sometimes block events have a broken semi-circle (like the **Which** port above) or a broken chevron (like the **Time** port above). 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 Output** node, the value at the **Which** port will determine whether the data-and-event coming into the **In** port will be transmitted to the **Option 1** port or the **Option 2** port.

The manual section How events travel through a composition has more information on how events move through a composition.

3.2.1.2 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 **Set Count** 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 **Write Line** 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.

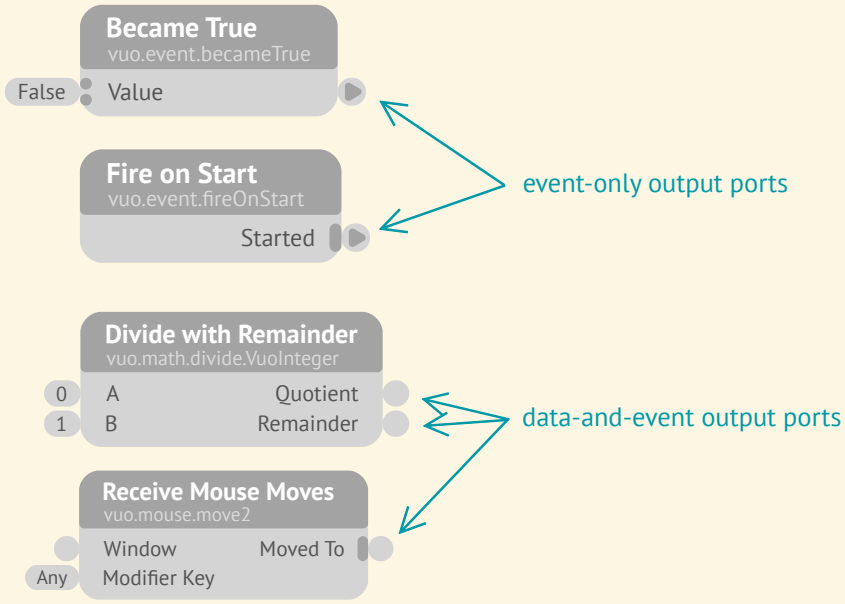
Looking again at the **Count within Range** node, you can see that the node has some input ports with port actions and some without. For the ports without port actions — **Minimum**, **Maximum**, and **Wrap Mode** — the node will output the same number regardless of whether the event causing the node 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. When the **Write Line** 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.

3.2.2 Output ports

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



3.2.2.1 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.

3.3 How events travel through a composition

Note for text programmers

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

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

3.3.1 The rules of events

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

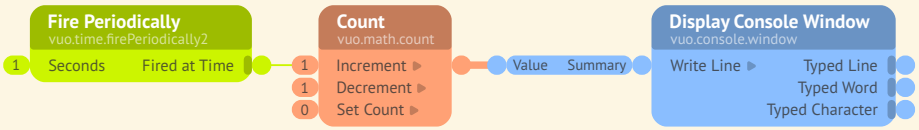
1. **An event travels forward through cables and nodes.** Along each cable, it travels from the output port to the input port. Within each node, it travels from the input ports to the output ports (unless it's blocked). An event never travels backward or skips around.
2. **One event can't overtake another.** 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 those rules apply to some actual compositions.

3.3.2 Straight lines

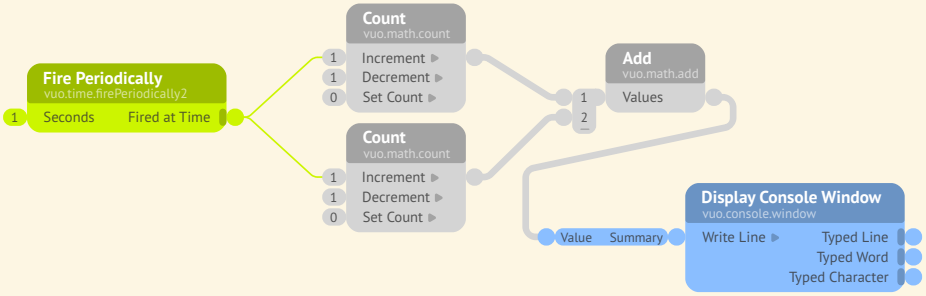
The simplest event flow in a composition is through a straight line of nodes, like the composition below.



In this composition, the **Fired at Time** trigger port fires an event every 10 seconds. 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.

3.3.3 Splits and joins

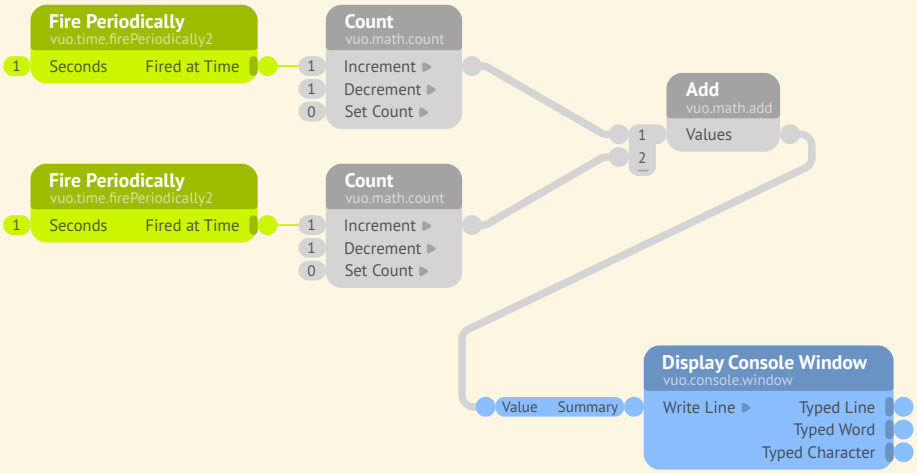
When you run a composition in Vuo, multiple nodes can execute at the same time. This takes advantage of your multicore processor to make your composition run faster.



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**.

3.3.4 Multiple triggers



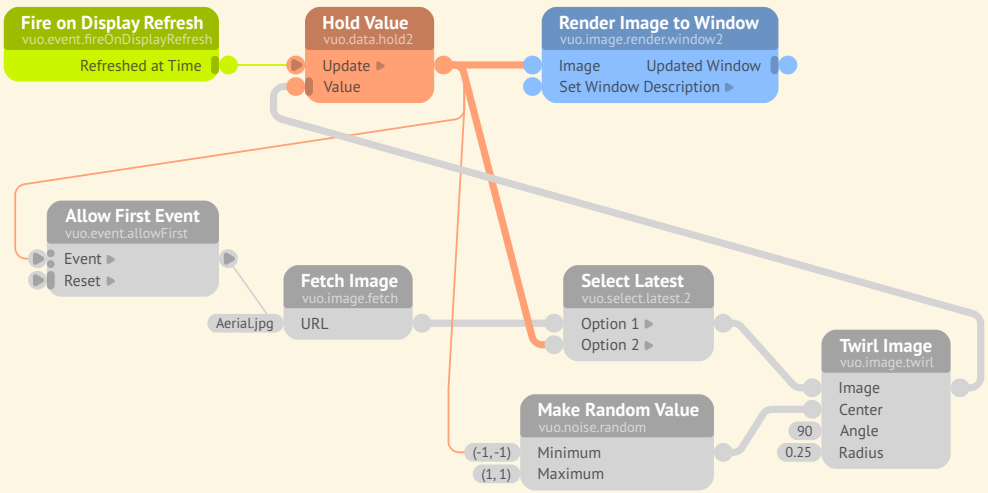
In this composition, the **Add** node executes each time either **Fire Periodically** node fires an event. If one of the **Add** node's inputs receives an event, it doesn't wait for the other input. It goes ahead and executes.

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.

Compare this composition to the one above it. Since in this composition the **Fire Periodically** nodes can execute in either order, or at the same time, the results are unpredictable. When you want to ensure events are executed by separate nodes at the same time, use the *same* event.

3.3.5 Feedback loops

You can use a **feedback loop** to store and build upon a result over time. This example composition demonstrates (File > Open Example > Data > Twirl Image Repeatedly):



Starting with the image from **Fetch Image**, this composition adds another twirl to the image with each display refresh. Over time, the entire image accumulates twirls upon twirls.

The orange and gray nodes, and the cables between them, comprise the feedback loop. Let's focus on the part of the loop that stores and repeatedly processes the image: **Hold Value**, **Select Latest**, and **Twirl Image**. Each time **Fire on Display Refresh** fires an event:

- The event enters the **Hold Value** node's **Update** input port.
- The **Hold Value** node executes, outputting the event plus the image produced the previous time around the loop (or an empty image if this is the first time around).

- The event plus image enters the **Select Latest** node's **Option 2** input port.
- The **Select Latest** node executes, outputting the event plus the image from **Hold Value** (or the original image from **Fetch Image** if this is the first time around).
- The event plus image enters the **Twirl Image** node's **Image** input port.
- The **Twirl Image** node executes, outputting the event plus an image with one additional twirl.
- The event plus image hits the **Hold Value** node's **Value** input port.
- The **Hold Value** node executes, although it doesn't have any visible effect. It doesn't output any event or data because the event is blocked at the input port's event wall.

With each event fired from **Fire on Display Refresh**, these steps repeat and the image gains another twirl.

For each event from **Fire on Display Refresh**, when exactly does the **Hold Value** node output an image to **Render Image to Window**? The first time **Hold Value** executes, the second time, or both times? The answer is: Only the first time.

- The first time **Hold Value** executes:
  - The event hits the **Update** input port.
  - The node executes.
  - The event plus data travels out of the output port through all connected cables, including the one to **Render Image to Window**.
- The second time **Hold Value** executes:
  - The event hits the **Value** input port.
  - The node executes.
  - Nothing further happens. The event is blocked by the wall on the input port.

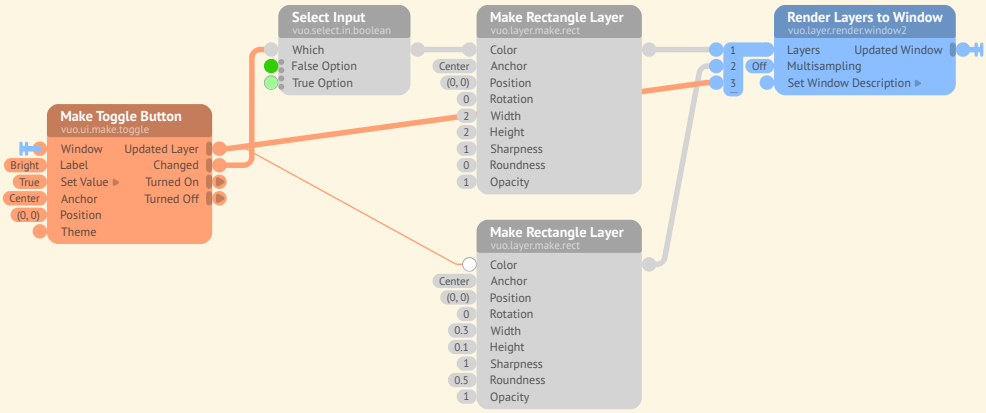
3.3.6 Summary

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.

3.4 How data travels through a composition

Most often, data and events travel together. In most compositions, including the example below, the majority of cables are data-and-event (thick) cables. Whenever an event travels through one of these cables, it's accompanied by a piece of data — like the color that travels from **Select Input** to **Make Rectangle Layer**.



> **Tip**
>
> The antenna symbols in this composition indicate a hidden cable from the **Updated Window** output port to the **Window** input port. To hide a cable, right-click on it and select Hide.

When an event and its companion piece of data reach a node's input port, the event causes the node to do its job, while the data affects how the node does its job. (This is explained further in How compositions process data.)

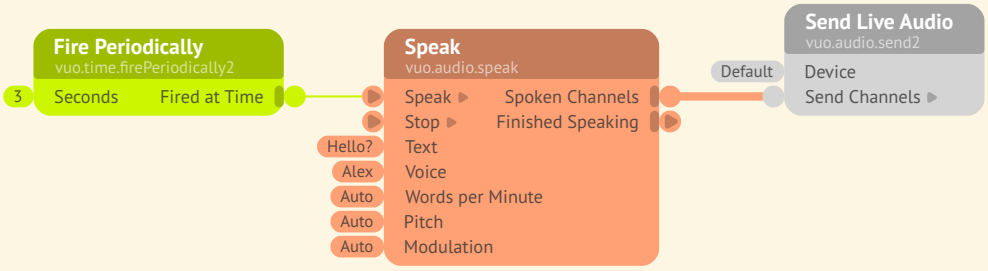
Data generally doesn't travel *through* a node in the same way that an event does. Instead, the node, informed by its input data, produces other data as output.

3.4.1 Ignoring data

Sometimes you don't want the data that a node outputs. You just want the events.

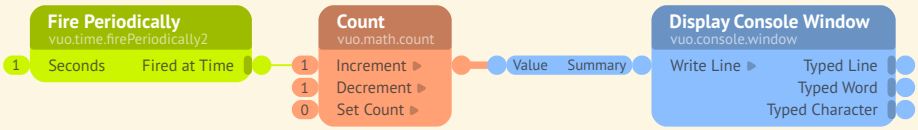
One example is the composition below. The **Fire Periodically** node's trigger port fires an event along with data — the number of seconds since the composition started — every 3 seconds. The **Speak** node doesn't need or want that data. It just needs the event.

To create an event-only cable like the one below, start dragging from the **Fired at Time** port (pulling out a data-and-event cable), then drop the end of the cable onto the **Speak** input port. When you connect a data-and-event cable to an event-only port, the cable automatically becomes event-only.



It's also possible to connect an event-only cable between a pair of data-and-event ports. This can be useful with the **Count** node, as shown below. Each time the **Count** node executes, it adds the amount in its **Increment** port to the total that it outputs. Let's say you want to count up by 1 every 10 seconds. To control the timing, you can use the events from the **Fire Periodically** node's trigger port, but you need to ignore the data from that port. You can accomplish this with an event-only cable.

To create this event-only cable, start dragging from the **Fired at Time** port. Hold down ⇧ (Shift) to change the cable from data-and-event to event-only, then drop the end of the cable onto the **Increment** port.



Alternatively, you can drag the cable from the **Fired at Time** port and drop it into the title area of the **Count** node. Dropping the end of a cable onto a node's title area changes the cable to event-only and connects it to the node's first non-walled port.

New in Vuo 2.0

3.4.2 Data flow without an event

There are only two cases in which data can travel without an event: from a *drawer* to its attached node and from a *published input port* through directly connected cables. Both are explained later, in the section Inputting data.

In all other cases, the only way that data can travel through a composition is when accompanied by an event.

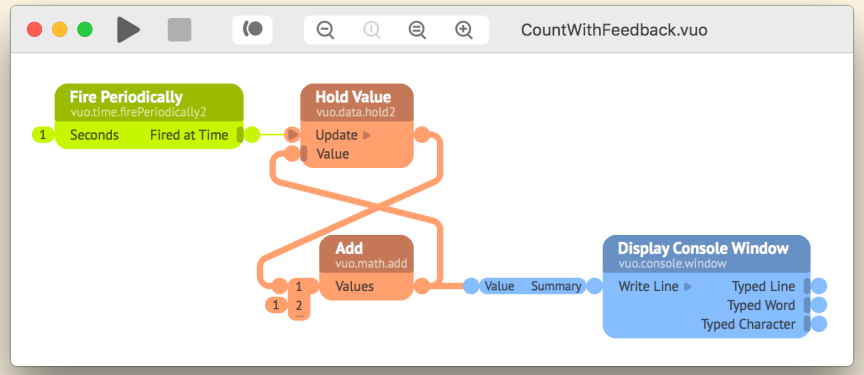
3.5 Solving problems with event flow

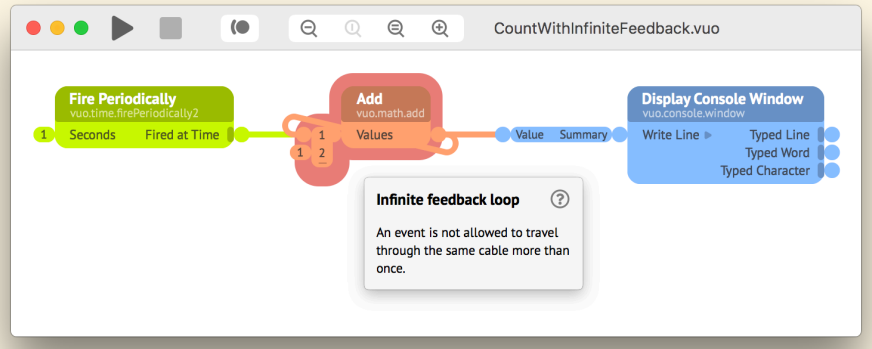
Events are a powerful tool, as they make it possible for you to control exactly when each node in your composition executes. However, events fired at the wrong place or time can lead to problems. This section covers several problems you might encounter and the ways that Vuo can help you identify and fix them.

3.5.1 "Infinite feedback loop" error

One of the rules of events is that an event can travel through each cable at most once. This rule comes into play when your composition has a feedback loop.

As you learned earlier in this section, feedback loops are a useful construct for accumulating changes over time. You just need to regulate the flow of data through the loop. Here are examples of a working (regulated flow) and a non-working (unregulated flow) feedback loop:





In the working feedback loop:

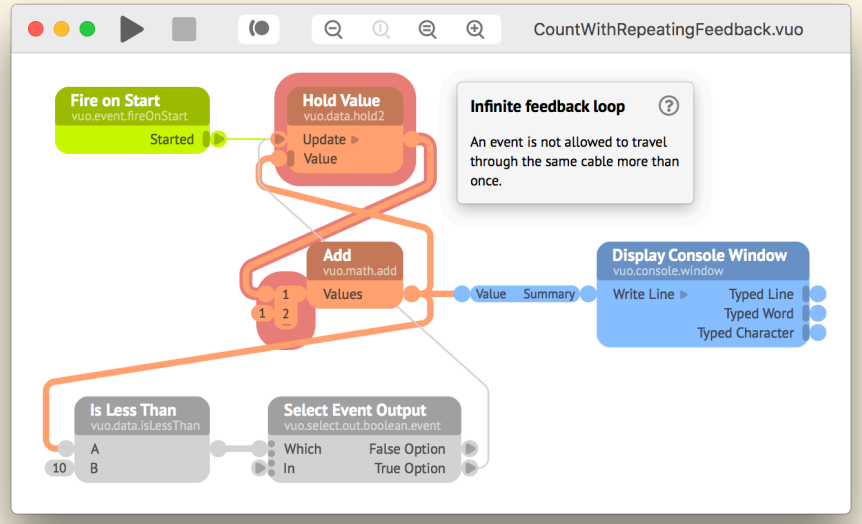
- The event enters the **Hold Value** node's **Update** input port and travels through to the output port.
- The event enters the **Add** node's input port and travels through to the output port.
- The event hits the **Hold Value** node's **Value** input port and is blocked by the event wall.

In the non-working feedback loop, there's no wall to block the event from looping through the **Add** node over and over. Vuo reports an **infinite feedback loop** and doesn't allow the composition to run.

Comparing the two compositions above, you can see that one way to fix an infinite feedback loop is to insert a node with an event wall, such as **Hold Value**.

Another way to fix an infinite feedback loop is to get rid of the feedback loop. In the composition above, you could replace the feedback loop with a **Count** node. The **Enqueue** node and the **Blend Image with Feedback** node can also take the place of some feedback loops.

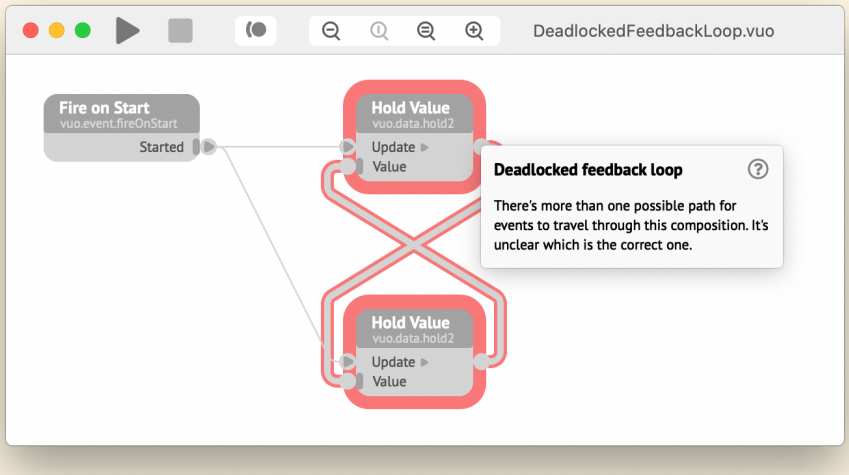
Rarely, you might encounter situations where Vuo reports an infinite feedback loop even though you can logically reason that the loop is finite — like in this composition:



The event would only circle the loop 10 times, so it's not "infinite". Nevertheless, you'll need to re-structure your composition to avoid the error. When iterating through a fixed number of items, you can use a **Build List**, **Process List**, or **Spin Off Events** node. (Alternatively, you can circumvent the error by inserting a **Spin Off Event** node in the loop, but be aware that you could create an actual infinite feedback loop that causes your composition to freeze.)

3.5.2 "Deadlocked feedback loop" error

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 needs 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.

3.5.3 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 enqueuing 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 enqueuing 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 enqueuing events.

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.

4 How compositions process data

Data is information such as numbers, text, and images. Nodes use the data in their input ports to control how they do their job. The goal of a composition is almost always to create or transform data in some way.

4.1 Data types

Numbers, text, and images are all examples of data — but they're not all the same type of data. You can do things with a number (such as calculate the square root) that wouldn't make sense with a sentence of text. Similarly, you can do things with an image (such as applying a kaleidoscope filter) that wouldn't make sense with a number.

In Vuo, data is categorized by **data type**. A node can only input and output certain data types that make sense with the job that the node does, such as calculating numbers or filtering images.

4.1.1 Basic data types

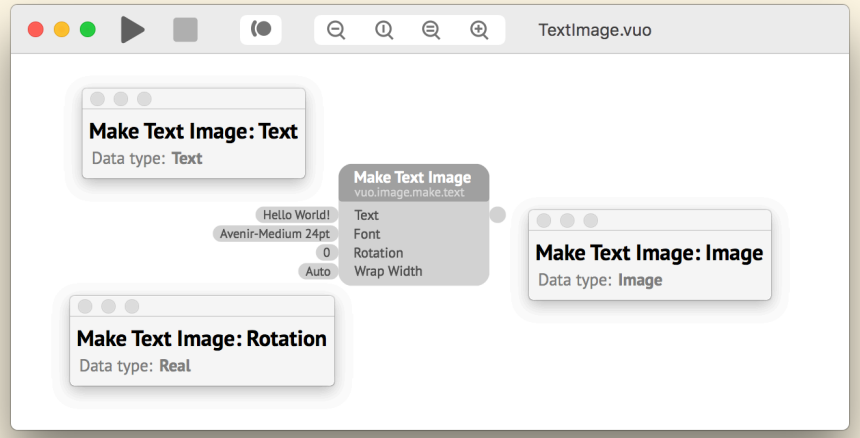
Here are the most common data types in Vuo:

Name	Examples	Description
Integer	-5; 0; 103	A positive or negative whole number
Real	-1.2; 0.0; 33.333	A positive or negative decimal number
Text	*Thank you!*	A sequence of characters
Boolean	*true*; *false*	"Yes" (*true*) or "no" (*false*)
2D Point	(0.1, -1.5)	A position in 2-dimensional space
3D Point	(0.1, -1.5, 0.8)	A position in 3-dimensional space
4D Point	(0.1, -1.5, 0.8, 1.0)	A position in 4-dimensional space
Color		A combination of hue, saturation, and lightness
Image		A rectangular grid of pixels
Layer		A 2D shape or image that can be stacked with others
Scene Object		A 3D shape that can be placed with others in a scene

If your computer is configured to use a comma instead of a period for the decimal mark (System Preferences > Language & Region), then Vuo displays numerical types accordingly.

Vuo has dozens of other data types, many of them specific to certain tasks (such as processing audio or receiving keyboard input). You'll learn about those data types in the process of learning how to perform the tasks.

You can see which data type a port has by clicking on the port to open its port popover.



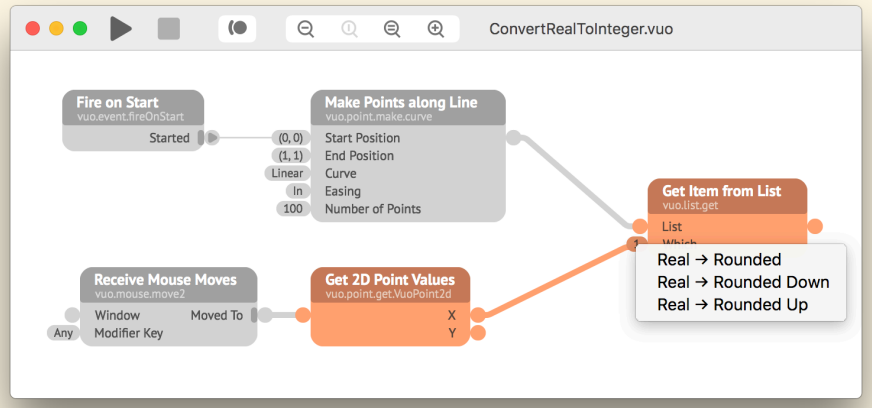
When you start dragging a cable from a port, Vuo shows you which ports you can connect the other end of the cable to — ports that have a compatible data type — by fading out all other ports. Ports that remain opaque have the same data type as the original port. Ports that are slightly faded have a data type that is different but related, so it's possible to convert from one data type to the other.

4.1.2 Type-converter nodes

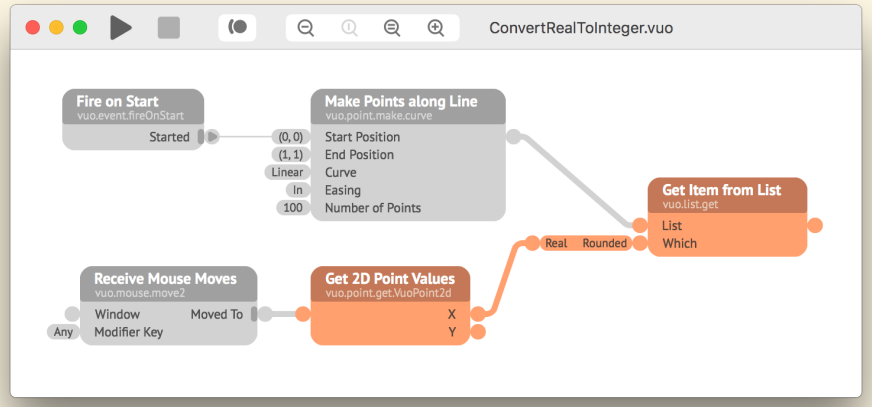
The two ports at either end of a cable always have the same data type. If you need to connect an output port of one data type to an input port of a different but related data type, you can insert a **type-converter node**. A type-converter node's job is to translate data from one type to another.

When you drop a cable endpoint onto a port of a different but compatible data type, either Vuo will ask you to choose which type-converter node to use or, if there's only one type-converter node available for that pair of data types, Vuo will go ahead and insert it.

For example, if you want to connect a Real output port to an Integer input port, you can choose the **Round**, **Round Up**, or **Round Down** node to convert the Real (number with a decimal point) to an Integer (number without a decimal point).



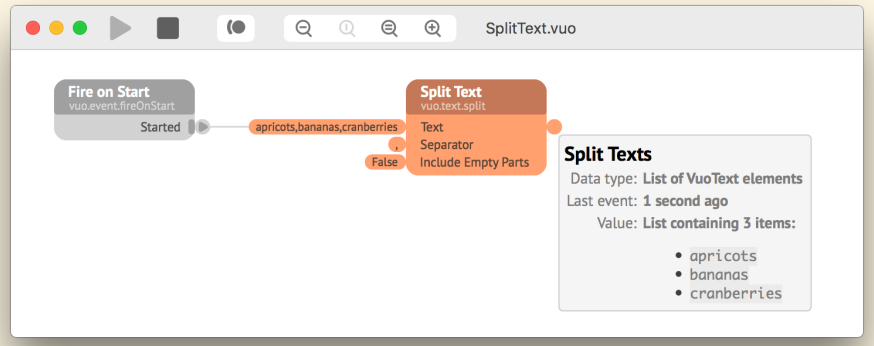
Vuo inserts the type-converter node in a collapsed form to save space. You can still click on the node to see its uncollapsed form and description in the Node Documentation Panel.



4.1.3 List data types

For every single-value data type in Vuo, there's a corresponding list data type.

For example, the **Split Text** node inputs a single Text and separates it into parts. Each part is a Text. The node outputs the collection of parts as a List of Text.

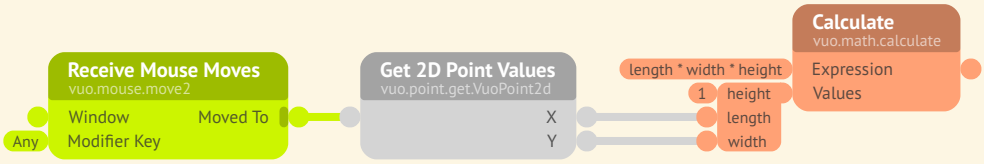


A **list** is a sequence of data items. Lists are useful when you want to work with a collection of data items instead of just one.

4.1.4 Dictionary data types

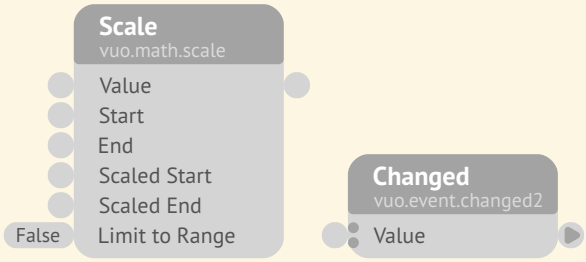
With a list, each item is identified by its position in the sequence. With a different kind of collection called a **dictionary**, each item is instead identified by a name or *key*.

For example, the **Calculate** node's **Values** input port has a dictionary data type, specifically Dictionary of Text keys and Real values. The keys are the names of variables in a math expression. The values are the numerical values that the node should substitute in place of the variables to calculate the result.



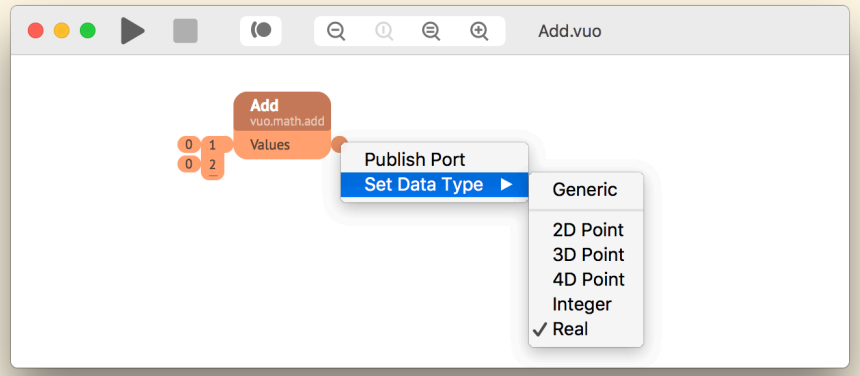
4.1.5 Ports with changeable data types

Some nodes can work with many different types of data. For example, the **Scale** node can scale Reals, 2D Points, 3D Points, or 4D Points. The **Changed** node can input a Text value, an Image, a Color, or many other types of data.



When using nodes that are flexible about the type of data they can work with, you can choose the data type that you need for your composition.

To see the data types that a port can be changed to, right-click on the port and look at the `Set Data Type` submenu. (Only ports with changeable types have a `Set Data Type` submenu.)



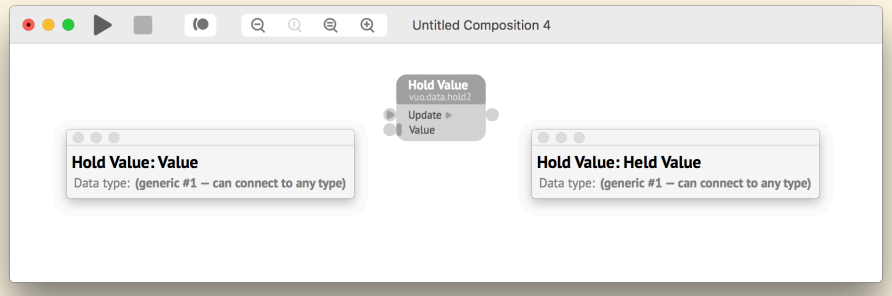
Changed in Vuo 2.0

The Set Data Type menu now appears for all changeable data types, not just generic data types. You no longer need to change the data type back to generic before changing it to a different type.

4.1.5.1 Generic data types

The `Generic` menu item means that the port has a **generic data type** — a stand-in for when the port's data type hasn't been decided yet.

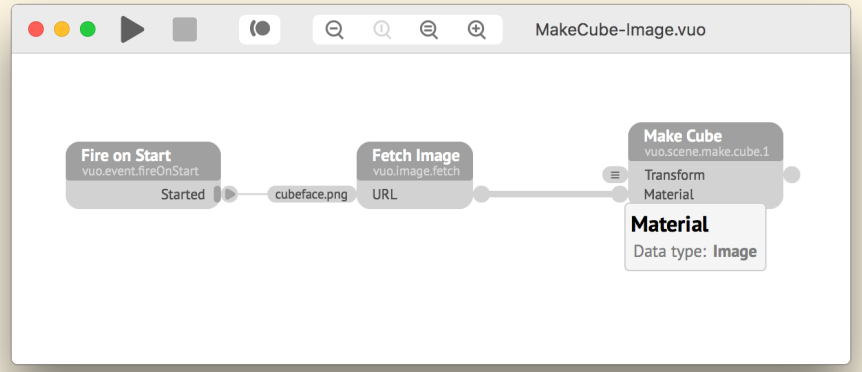
For example, when you first put a **Hold Value** node on the canvas, its **Value** input port and **Held Value** output port both have a generic data type.



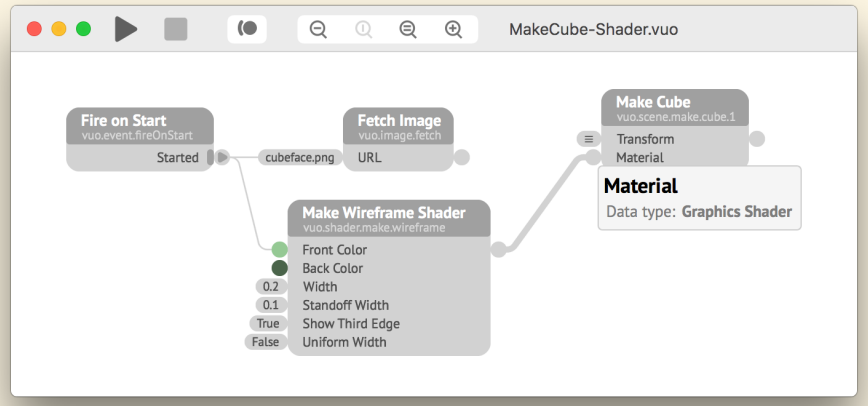
4.1.5.2 Changing the data type When a port has a changeable data type, one way to change it is to use the `Set Data Type` submenu. Another is to connect a cable.

When you start drawing a data-and-event cable from a port with a changeable data type (or indeed from any port), Vuo fades out the ports with incompatible data types, leaving the ports that the cable can potentially connect to. Here are some examples of the kinds of connections you can make.

Connect an unchangeable port to a generic port — The **Make Cube** node has a **Material** input port that controls how the sides of the cube are painted. The port has a changeable data type that starts out generic and can be changed to Color, Graphics Shader, or Image. If you put a **Make Cube** node on the canvas and connect a cable carrying Image data to it, then the port's data type changes from generic to Image.



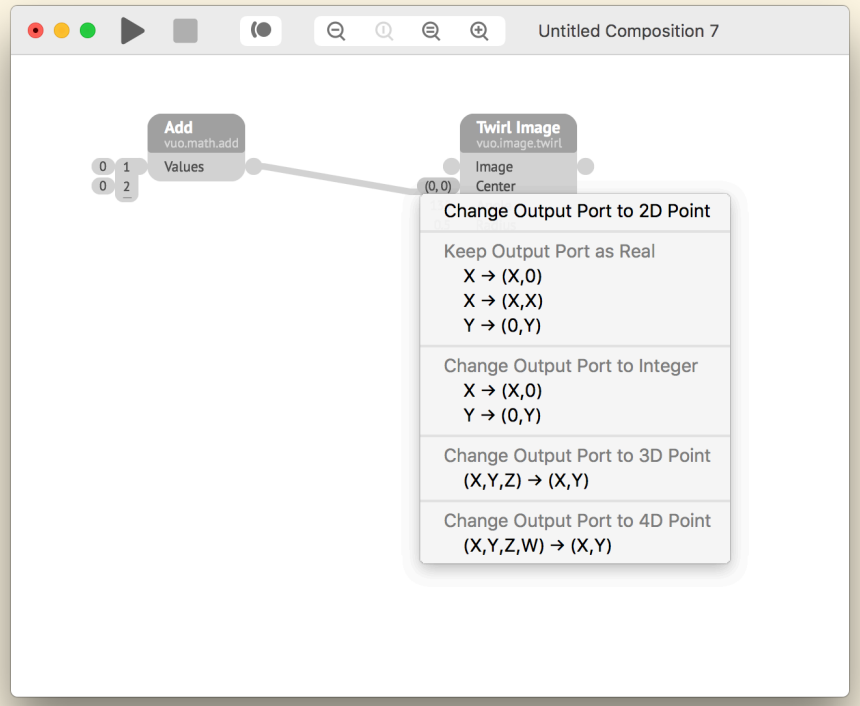
Connect an unchangeable port to a changeable port — Following on the previous example, suppose you change your mind and decide you'd like to paint the cube with a Graphics Shader instead of an Image. If you connect a cable between the **Material** port and a port of type Graphics Shader, then the **Material** port's data type changes from Image to Graphics Shader.



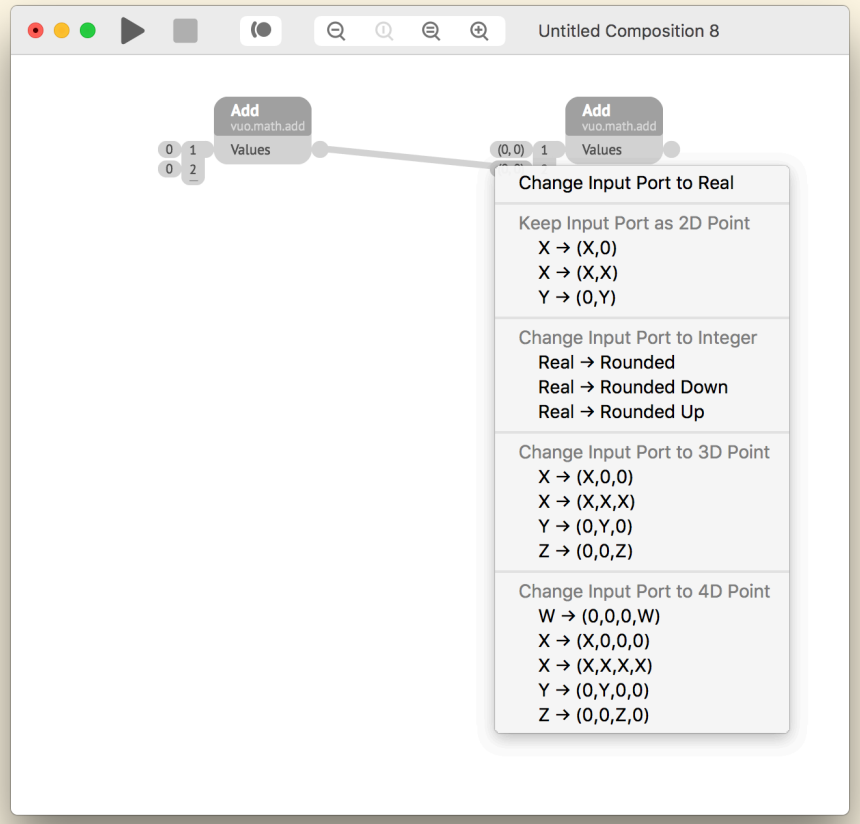
Sometimes there are different ways that you could make the connection depending on what you're trying to accomplish, so Vuo presents some suggestions and asks you to choose. For example, if you're connecting an **Add** node's Real output port to a 2D Point input port, you might want to change the **Add** node's output port from Real to 2D Point. Or you might want to keep the **Add** node's output port a Real and use it to control the X-coordinate of the 2D Point. You can choose either of those options (among others) from the menu presented when you drop the cable onto the 2D Point port.

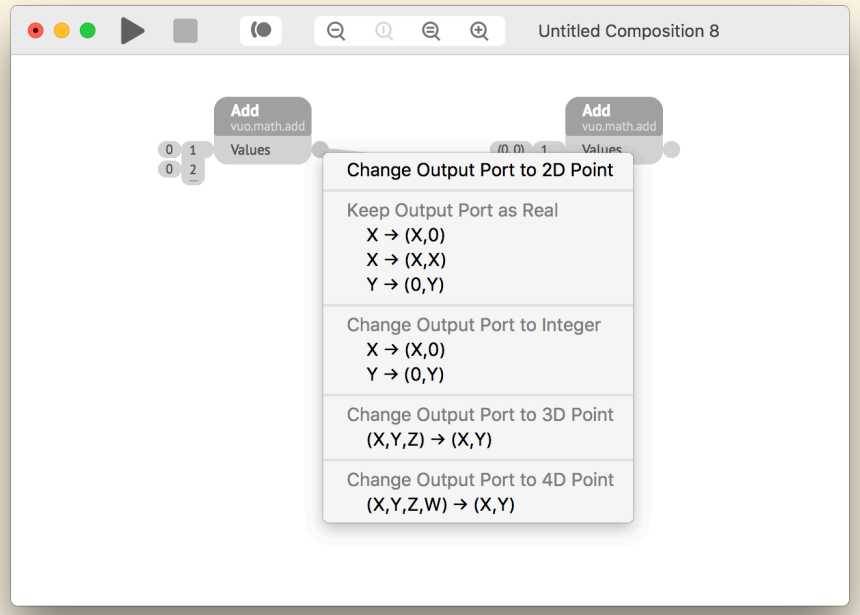
Changed in Vuo 2.0

This menu now gives you the option to change the port's data type.



Connect a changeable port to a changeable port — When connecting two ports that both have changeable data types, you're likely to have even more options. If you drag a cable from an **Add** node's Real output port and drop it onto a different **Add** node's 2D Point input port, you can opt to change the input port's data type to Real or insert various type converters. If you drag a cable in the other direction, from the input port to the output port, you instead have the option to change the *output* port's data type to 2D Point.

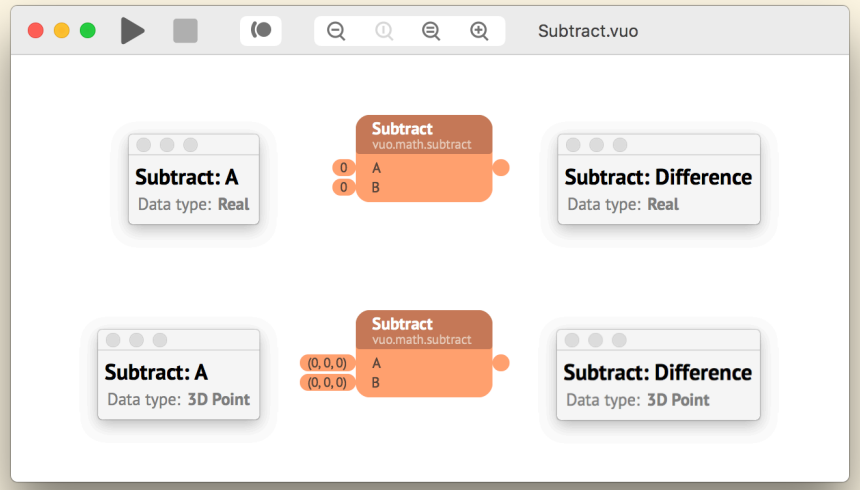

Untitled Composition 8
Add
vuo.math.add
Values
0
1
0
2
Add
vuo.math.add
Values
(0, 0)
1
Change Input Port to Real
Keep Input Port as 2D Point
X → (X,0)
X → (X,X)
Y → (0,Y)
Change Input Port to Integer
Real → Rounded
Real → Rounded Down
Real → Rounded Up
Change Input Port to 3D Point
X → (X,0,0)
X → (X,X,X)
Y → (0,Y,0)
Z → (0,0,Z)
Change Input Port to 4D Point
W → (0,0,0,W)
X → (X,0,0,0)
X → (X,X,X,X)
Y → (0,Y,0,0)
Z → (0,0,Z,0)



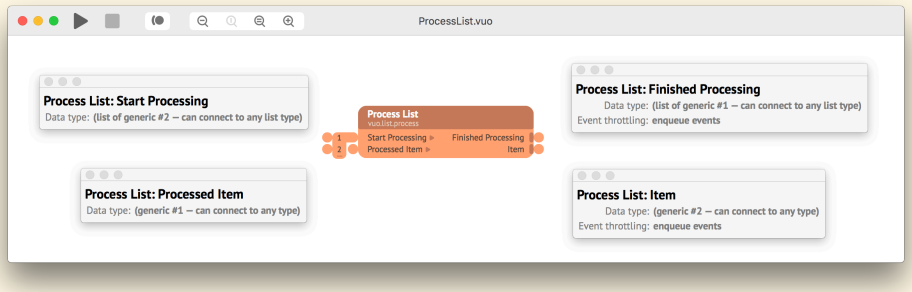
4.1.5.3 Groups of ports that share the same data type

When you connect cables between two ports of changeable data types, their data types become synchronized. If you change one port's data type, then the other port's data type automatically changes to match.

Some nodes have ports that are synchronized with each other. The **Subtract** node is an example. It can subtract two Integers, resulting in another Integer; two 3D Points, resulting in another 3D Point; and so on. When you change the data type of one of the node's ports, that automatically changes the data types of the other ports as well.



On a few nodes, one group of ports shares the same data type and a separate group of ports shares another data type. The **Process List** node is an example. The **Start Processing** port has a list data type, and the **Item** port outputs the items of that list. So if **Start Processing** has type List of Text, **Item** must have type Text. Similarly, the **Finished Processing** port has a list data type, and its items must match the type of **Processed Item**. But **Processed Item** doesn't have to match the type of **Item**.



You can see which ports on a node share a data type by selecting Set Data Type › Generic for each port, then opening the port popovers and observing the numbers on the generic data types (*generic #1*, *generic #2*, etc.).

4.2 Inputting data

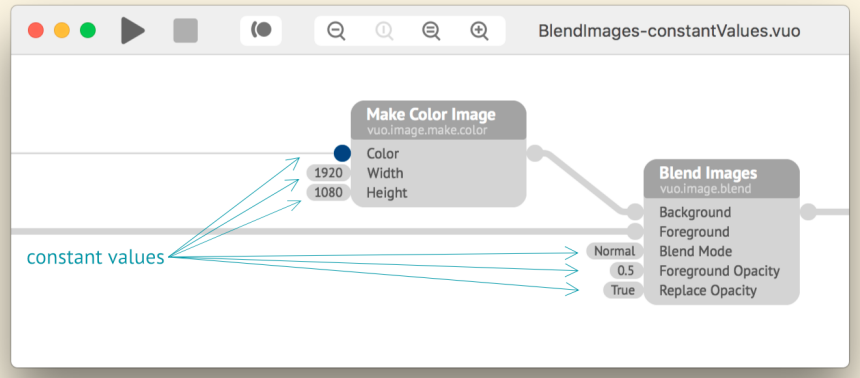
You've learned how data flows through a composition (How events and data travel through a composition). But how does the data get into the composition in the first place?

There are two main ways. One is to pull in data from the world outside the composition — files on your computer, input devices such as a mouse or video camera, information communicated over a network, and so on. A good starting point to learn about this is to search the Node Library for nodes whose titles begin with *Receive*.

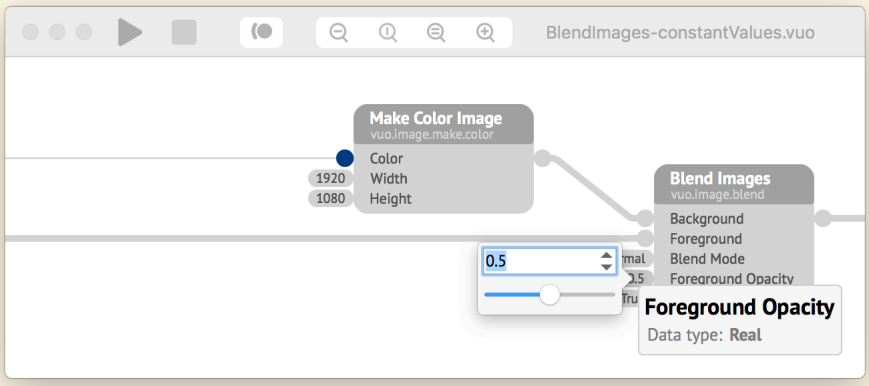
The other way to get data into a composition is to enter it yourself. This section explains how.

4.2.1 Editing data in a node's input port

If an input port doesn't have a data-and-event cable connected to it, then it has a **constant value**. Rather than being replaced with new data coming in from a cable, the port's value remains the same as the composition runs.



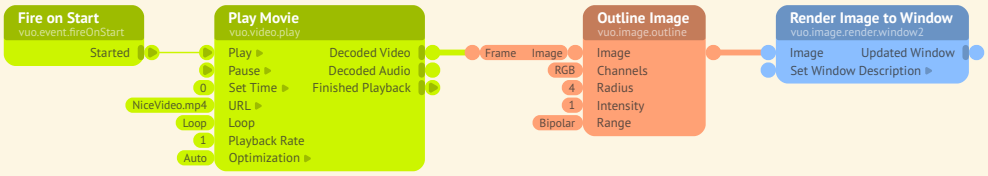
For many data types, the constant value is displayed alongside the input port. You can double-click on the constant value to open an **input editor** and edit the value.



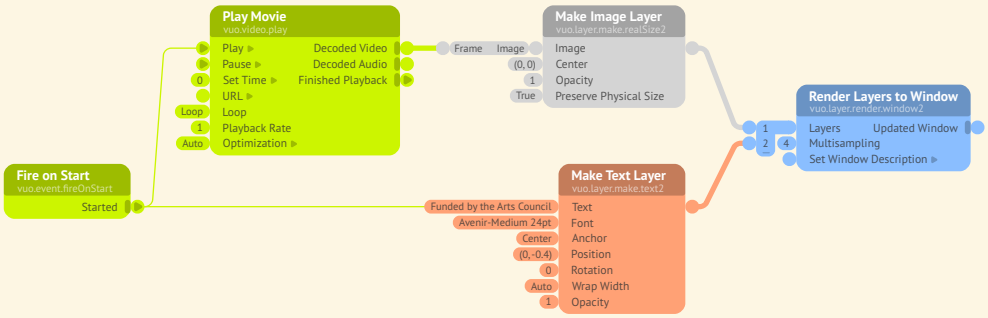
Different data types have different kinds of input editors. Some data types aren't editable. Double-clicking on them doesn't open an input editor. The only way to change their value is by connecting a cable.

You can close most input editors (keeping the edits) by clicking on the canvas or pressing ↵. In input editors for Text data, since the ↵ key is taken, you can enter a linebreak with ⌥↵. You can cancel edits by pressing ⎋.

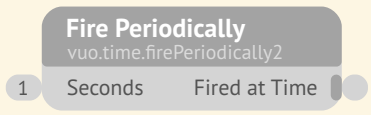
After you edit an input port's data, the new data will take effect the next time the node executes. If there's already a steady stream of events flowing through the node, like the **Outline Image** node below, the new data will naturally enter into the flow.



If events only rarely hit the node, like the **Make Text Layer** node below, then, in order to see the results of the new data, you'll have to either restart the composition or fire an event into the node manually. To fire an event manually, right-click on one of the node's input ports and select Fire Event.



If you edit an input port value on a node that has a trigger port, the new data will take affect immediately. You don't have to fire an event into the node. For example, after you edit the **Seconds** input port of a **Fire Periodically** node, the node immediately adjusts the rate at which it fires.



4.2.2 Editing data in a published input port

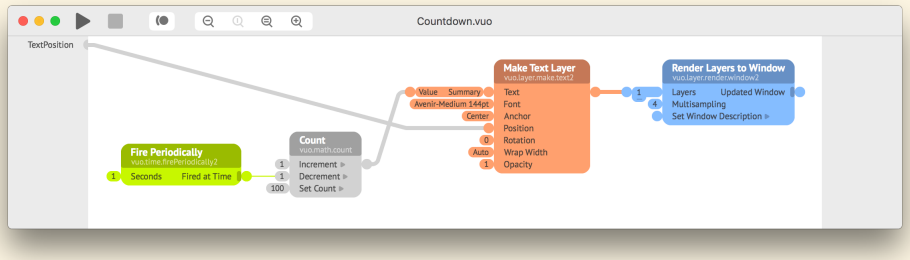
Like an input port on a node, a published input port can have a constant value. (The exception is protocol published input ports, which are explained in Making compositions fit a mold with protocols.)

You can edit a published input port's constant value by double-clicking on the port, which brings up an input editor.

If the published input port has a numerical data type, you can also edit the input editor's range. Right-click on the published port and go to Edit Details... . Suggested Min and Suggested Max are the recommended lower and upper bounds of the data value. If both are set, then the input editor will have a slider, a text field, and up and down arrows. Otherwise, the input editor will only have a text field and up and down arrows. Suggested Step controls the step size of the up and down arrows.

If you're running the composition as a standalone composition (not a subcomposition), after you edit a published input port's data, the new data will immediately flow through any cables directly connected to the published input port — but no farther. This is a rare case in which data flows without an event.

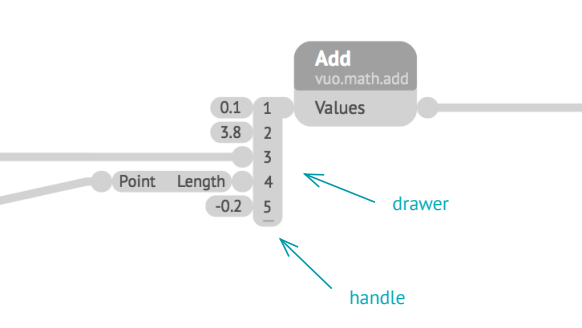
For example, in the composition below, if you change the value of **TextPosition** from (0, 0) to (1, 1), the value of the **Position** port on **Make Text Layer** immediately changes to (1, 1). But the **Make Text Layer** node doesn't execute and doesn't affect any nodes downstream — so the text in the window remains at (0, 0) for the moment. The next time **Fire Periodically** fires an event, the event hits **Make Text Layer**, causes the node to execute with the new **Position** value, and travels onward along with the resulting data to **Render Layers to Window** — so the text in the window now appears at (1, 1).



To learn about constant values of published input ports in subcompositions, see this section.

4.2.3 Inputting lists

When an input port has a list data type, you can either input the list as a whole (by connecting a cable to the list port) or input each list item separately via the **drawer** attached to the port.



You can change the number of list items by dragging the drawer's handle (bar along the bottom) up or down, or by right-clicking on the drawer and selecting `Remove Input Port` or `Add Input Port`.

4.2.4 Inputting dictionaries

The **Calculate** and **Calculate List** nodes each have a drawer attached to their dictionary input port. The keys of the drawer adjust automatically when you edit the **Expression** input port's data. (You can't resize this drawer like you can a list drawer.)

On other nodes, dictionary input ports don't have drawers. Instead, you can connect a cable from the output of a **Make Dictionary** node.

5 How nodes can be used as building blocks

Nodes are the building blocks of Vuo, which you can assemble in any way you can think of to create compositions. When you download Vuo, it comes with a large set of nodes that support 2D and 3D graphics, video, audio, networking, user interaction, and more. If you've purchased Vuo Pro, then you have some bonus nodes available to you. Whether you're using Vuo Community Edition or Vuo Pro, you can also download nodes by third-party developers to add to your collection.

5.1 Finding out what nodes are available

Vuo has a list of all nodes called the Node Library. (If you don't see it, go to View > Node Library > Show Node Library.) You can skim through the Node Library to see what's available, or use the search bar at the top. For example, if you're wondering if Vuo has nodes for working with hues, search for "hue" and you'll find several nodes related to color. For search tips, see Searching for nodes.

For a complete list of built-in nodes, you can go to the online node documentation.

For even more nodes, you can visit the node gallery. There, members of the Vuo community share nodes that they've created.

5.2 Learning how to use a node

Each Vuo node has documentation, or in other words, a description of how it works. You can view this description in the Node Documentation Panel (lower panel of the Node Library) after clicking on the node in the Node Library or on the composition canvas.

Besides the documentation for individual nodes, there's also documentation for node sets. At the top of the Node Documentation Panel, most nodes have a link to their node set's documentation. For example, the **Make 3D Object** (`vuo.scene.make`) node has a link for `vuo.scene`, which provides documentation that applies to nodes throughout the `vuo.scene` node set.

Documentation both for nodes and for node sets is available in the online node documentation.

Besides documentation, many nodes also come with example compositions, which demonstrate use of the node within a composition. For nodes that have them, the example compositions are listed near the bottom of the Node Documentation Panel.

5.3 Pro nodes

Pro nodes are only available in Vuo Pro, not in Vuo Community Edition. If a node is Pro, then the Node Documentation Panel says so at the bottom. If you try to open a composition containing Pro nodes using Vuo Community Edition, then you'll be warned that you won't be able to run the composition.

If you plan to share a composition that contains Pro nodes, keep in mind that Vuo users without Vuo Pro can't run the composition. If you want others to be able to use your composition even if they don't have Vuo Pro, consider exporting it to an app.

Pro nodes can be used when running compositions inside of another application (such as a VJ app), as long as Vuo Pro has been activated on the computer running the application.

5.4 Deprecated nodes

As Vuo grows and changes with each version, new nodes are added while some older nodes become **deprecated**, or obsolete. When a node is deprecated, that means there's now a better way to accomplish that node's job.

When a node becomes deprecated, compositions that contain the node will continue to work for the time being. However, the node may stop working or be removed in a future version of Vuo. In compositions that you want to continue using for the long term, it's a good idea to replace deprecated nodes.

New in Vuo 2.0

To find all deprecated nodes in a composition, go to Edit ⟩ Find ⟩ Find... and type *deprecated* into the search box.

To replace a deprecated node, the first thing to try is to right-click on the node and go to Change To. If the first menu item is a node with the same title but a different node class name — for example, you've clicked on a **Make Image with Shadertoy** (`vuo.image.make.shadertoy`) node and the menu lists **Make Image with Shadertoy** (`vuo.image.make.shadertoy2`) — then the replacement is simple. First, select that menu item to insert the new version of the node. Second, run your composition and modify it as needed to work correctly with the new version of the node. For example, you might need to adjust the input port values because the new version handles them differently. Consult the node documentation to understand the differences.

If the Change To menu doesn't list an obvious replacement, try searching the Node Library for a node with the same title. If you find one, add it to your composition in place of the deprecated node. As above, consult the node documentation to understand the differences between the new version and the deprecated version. Be sure to run your composition and adjust it as needed.

If you still haven't found the new node(s) to replace the deprecated node with, check the release notes or ask the community.

5.5 The built-in nodes

This section gives an overview of some of Vuo's built-in nodes. The purpose is to give you a sense of what you can accomplish with the built-in nodes and where to start. For more details, see the node and node set documentation.

5.5.1 Graphics/video

Vuo comes with many different nodes for working with graphics. These can be roughly divided into 2D and 3D graphics (along with some nodes to convert between them).

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

For **3D models and meshes**, the `vuo.scene` node set is your starting point. It lets you load or build 3D objects, warp them, and arrange them within a scene, which you can render in a window or image. When building 3D objects, two additional node sets are helpful: `vuo.transform` for positioning, rotating, and scaling an object, and `vuo.shader` for painting a pattern or material on an object.

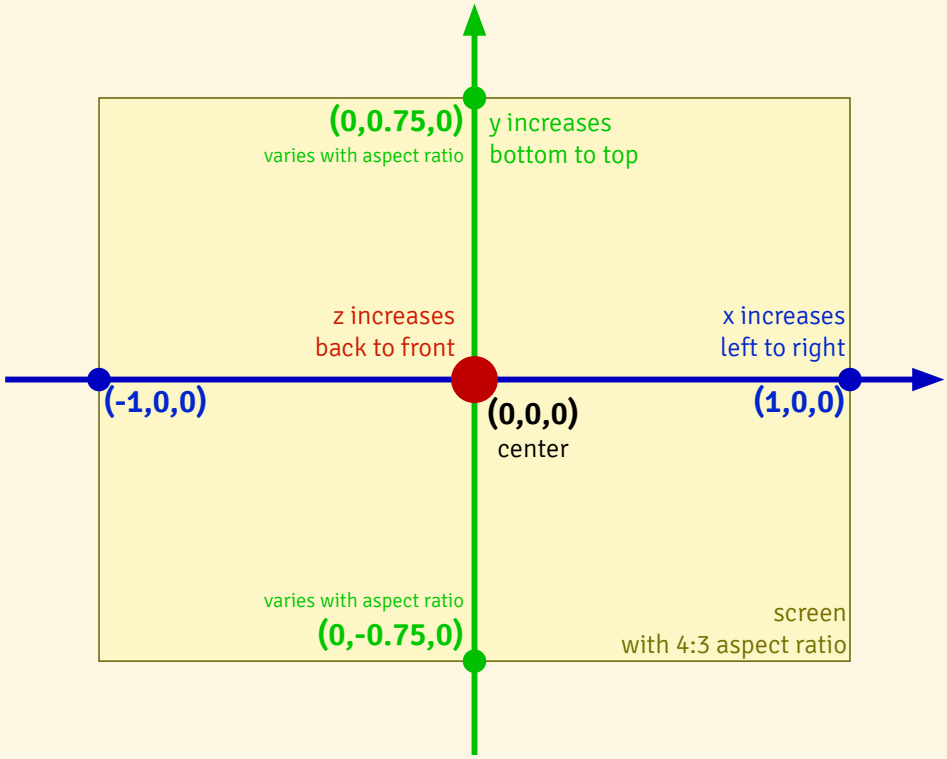
When working with 2D or 3D animations, the `vuo.motion` node set lets you control the **path and speed of a moving object**.

For **video**, the `vuo.video` node set handles playing movies and receiving video from cameras. When working specifically with the cameras on a Kinect, you can use the `vuo.kinect` node set. If you want to send and receive video between Vuo compositions and other applications, there's the `vuo.syphon` node set.

Make Quad Layer and related nodes in the `vuo.layer` node set support **projection mapping**.

5.5.1.1 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 https://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.

5.5.2 Sound/audio

The `vuo.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 it aloud.

5.5.3 User input devices

There are many built-in nodes you can use to make your compositions interactive, including:

- `vuo.mouse` for getting input from a mouse or trackpad
- `vuo.keyboard` for getting input from keys typed or pressed
- `vuo.hid` for getting input from a USB Human Interface Device (HID)
- `vuo.leap` for controlling a composition with hand and finger movements from a Leap Motion device
- `vuo.osc` for remotely controlling a composition via a TouchOSC interface on a phone or tablet
- the **Filter Skeleton** node for getting input from Delicode NI mate 2

5.5.4 Music and stage equipment

Your compositions can control and be controlled by music and stage equipment — such as keyboards, synthesizers, sequencers, and lighting — using several common protocols:

- `vuo.osc` for receiving OSC messages
- `vuo.midi` for sending and receiving MIDI events
- `vuo.artnet` (pro) for sending and receiving Art-Net messages

The `vuo.bcf2000` nodes interface with the Behringer BCF2000 MIDI controller.

5.5.5 Applications

Applications that send or receive messages via the OSC, MIDI, or Art-Net protocol can communicate with your composition if you use the `vuo.osc`, `vuo.midi`, or `vuo.artnet` nodes.

Your composition can send video to and receive video from other applications via Syphon using the `vuo.syphon` node set.

With the `vuo.app` node set, your composition can launch other apps and open documents in them.

5.5.6 Sensors, LEDs, and motors

The `vuo.serial` nodes allow your composition to connect to serial devices, including programmable microcontrollers like Arduino. Via the Arduino, your composition can receive data from sensors, and send data to control LEDs and motors.

5.5.7 Displays

Two node sets let you fine-tune how a composition's windows are displayed on the available screens. The `vuo.screen` node set provides information about the available screens. The `vuo.window` node set controls how each window is displayed, including its aspect ratio and whether it's fullscreen.

5.5.8 Files

Your composition can open files on your computer's filesystem or download them from the internet using "fetch" nodes, such as **Fetch Image**, **Fetch Data**, and **Fetch XML Tree**.

Your composition can save files to your computer's filesystem using "save" nodes, such as **Save Image**, **Save Data**, and **Save Images to Movie**.

For opening, manipulating, and saving XML and JSON files, there's the `vuo.tree` node set. And for CSV and TSV files, there's the `vuo.table` node set.

The `vuo.file` nodes enable your composition to interact with your computer's filesystem.

5.5.9 Internet

With the `vuo.rss` nodes, your composition can download RSS feeds.

To retrieve data from an XML or JSON web service, you can use the `vuo.tree` nodes.

5.6 Adding nodes to the canvas by dropping files

Dragging and dropping an audio, video, image, scene, or projection mesh file onto the canvas will create a node with that file as input. For example, when you drag an image onto the canvas, Vuo will create a **Fetch Image** node with the file path entered in the **URL** input port. The default will be the file's relative path. To enter an absolute path, hold down the option key when dragging the file on the canvas.

5.7 Creating a node

You can expand the things that Vuo can do by adding nodes to your Node Library. There are several ways to create your own nodes:

- By turning a group of nodes and cables into a single node. See Using subcompositions inside of other compositions.
- By writing GLSL code within Vuo. See Turning graphics shaders into nodes.
- By writing C/C++/Objective-C code in a text editor or IDE. See Developing node classes and types for Vuo.

5.8 Installing a node

You can download nodes created by other people and add them to your Node Library. These include nodes found in the Node Gallery, subcompositions found in the Composition Gallery, and fragment shaders in Interactive Shader Format (ISF).

Changed in Vuo 2.0

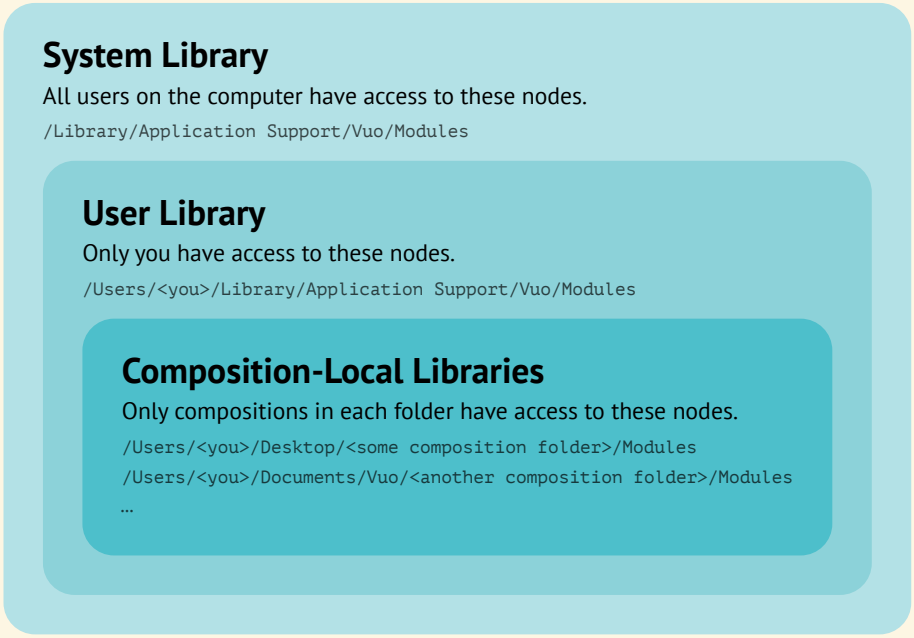
You no longer have to relaunch Vuo after installing nodes.

5.8.1 Installing a node the quick way

If the node to install is a `.vuonode` file, you can just double-click on the file to install it. The node gets installed in the User Library folder (explained in the next section). You can begin using the node right away, without having to relaunch Vuo.

5.8.2 Making a node available to all compositions

Depending on who should have access to the node — which compositions, and which user accounts on the computer — there are different places to install the node.



If the node is one that you expect to use in many compositions, you can install it in the **User Library folder** or the **System Library folder**. To access these folders, go to:

- Tools > Open User Library in Finder
  - Or in Finder: Hold down the Option key and select the Go > Library menu option. From there, go to 🗀Library‣Application Support‣Vuo‣Modules.
- Tools > Open System Library in Finder
  - Or in Finder: In the top-level folder on your hard drive, go to 🗀Library‣Application Support‣Vuo‣Modules.

You'll typically want to choose the User Library folder, since yours will be the only user account on your computer that needs access to the node. Use the System Library folder only if you have administrative access and you want all users on the computer to have access to the node.

To install the node, just place the node file in the User Library or System Library folder. If Vuo is running, it will automatically detect the file, and the node will appear in the Node Library momentarily. Otherwise, the node will appear in the Node Library the next time you launch Vuo.

5.8.3 Making a node available to one or a few compositions

New in Vuo 2.0

While some nodes are generally useful, others are more specialized. They may only make sense within the context of a certain composition. Thus, you may not want them to appear in your Node Library when you're working on unrelated compositions.

You can make a node available only to selected compositions by installing it in a **Composition-Local Library**. For example, let's say you have a composition called `Reptiles.vuo` saved to your Desktop. You have a subcomposition called `me.crocodile.vuo` and a node called `me.tortoise.vuonode` that you need only for `Reptiles.vuo`. You can create a folder called 🗀`Modules` on your Desktop and place `me.crocodile.vuo` and `me.tortoise.vuonode` in that folder.

A Composition-Local Library is a folder called 🗀`Modules` located in the same folder as a composition. When you have that composition open, the nodes in the Composition-Local Library appear in the Node Library.

If there's more than one composition in the same folder as 🗀`Modules`, all of those compositions can "see" and make use of the nodes in that Composition-Local Library.

A subcomposition within a Composition-Local Library may contain other nodes that are installed in the same Composition-Local Library. It may also contain nodes that are installed in the User Library or System Library folder.

If you install a subcomposition in a Composition-Local Library and later decide that you want to make the node available to all compositions, you can open the subcomposition and go to File ❯ Move to User Library.

5.8.4 Uninstalling a node

To uninstall a node, delete or move the node file out of the Library folder in which it's installed.

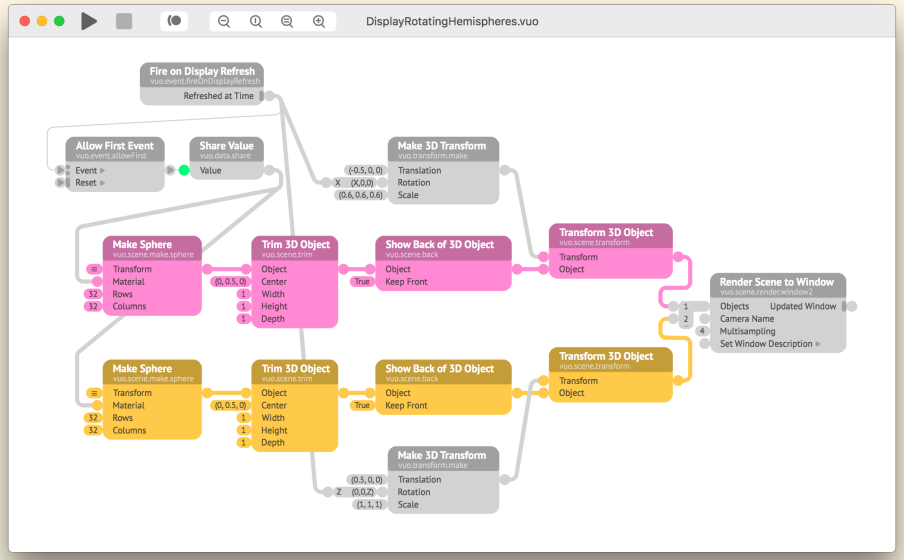
To find out where the node is installed, right-click on it in the Node Library and go to Show in Finder.

Be aware that once you've uninstalled a node, compositions that contain the node will no longer work.

6 Using subcompositions inside of other compositions

If you find yourself copying and pasting the same group of nodes and cables into many compositions, you may want to turn those nodes and cables into a **subcomposition**. A subcomposition is a composition that can be used as a node inside of other compositions. A subcomposition saves you the effort of having to recreate the same nodes and cables over and over. They're packaged neatly inside a node, which you can drag from the Node Library onto your canvas just like any other node.

Let's walk through an example. Suppose you often draw hemispheres (half spheres) in your 3D compositions, and it would be convenient to have a **Make Hemisphere** node in your Node Library. The first step is to identify the nodes and cables that you want to package into a subcomposition.

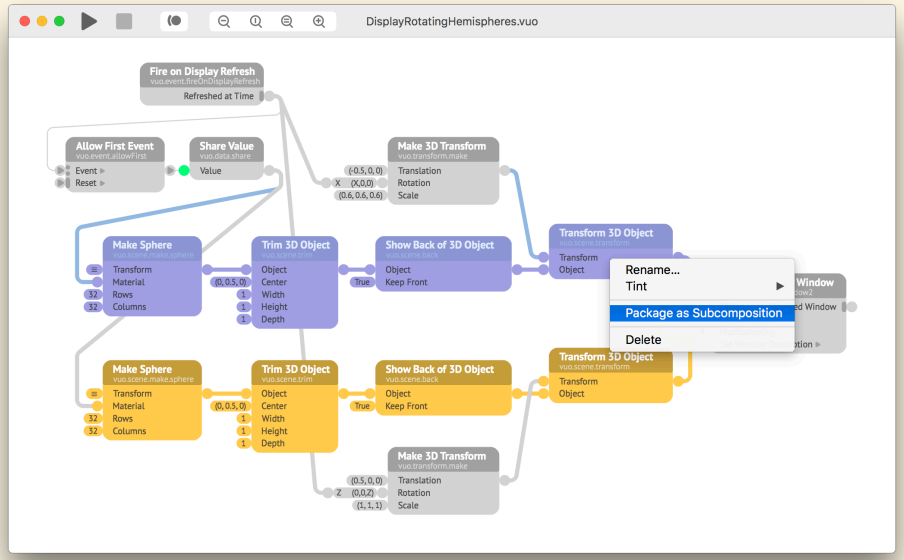


The composition above renders two rotating hemispheres to a window. (The **Trim 3D Object** node cuts off half of the sphere. The **Show Back of 3D Object** node makes the inside of the sphere visible.)

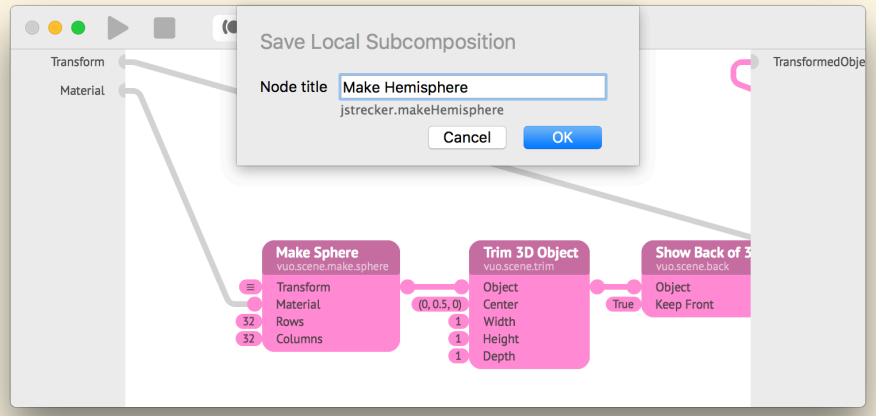
In other compositions, you may want to create any number of hemispheres. The hemispheres could have different rotations, positions, sizes, and colors. They could be rendered to a window or an image. So, for the subcomposition, let's choose a piece of the composition that's flexible enough to be used in all of these scenarios: the nodes and cables tinted magenta.

To turn these nodes and cables into a subcomposition, select them, then right-click on them and go to Package as Subcomposition.

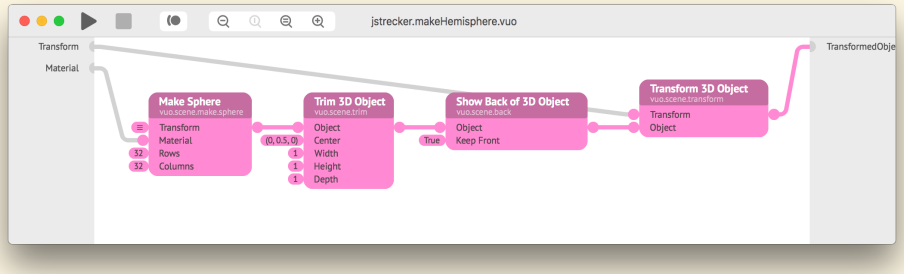
New in Vuo 2.0

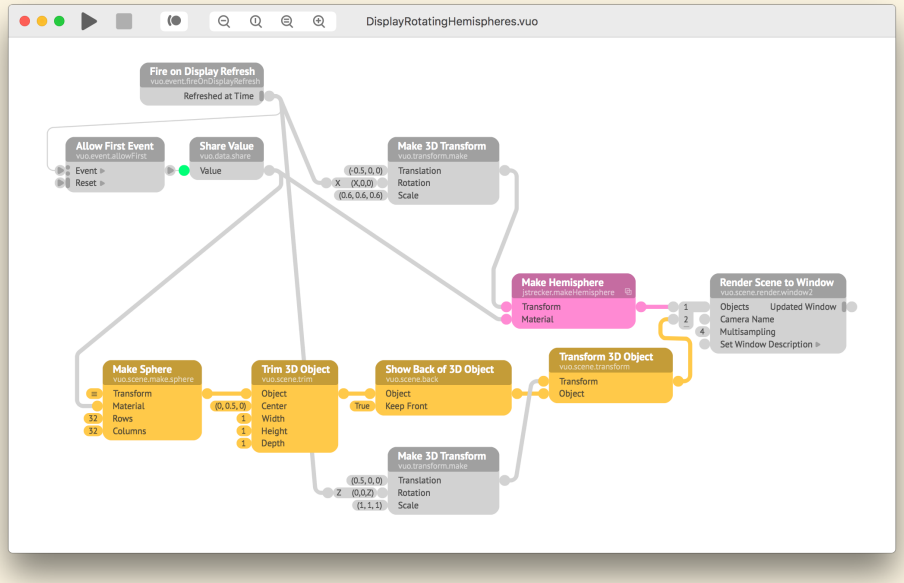


This extracts the selected part of the composition into a new window, where you're asked to pick a name for the subcomposition.



Having named the subcomposition, you now have two windows: one with the subcomposition and one with the original composition, in which the selected part has been replaced with a subcomposition node.



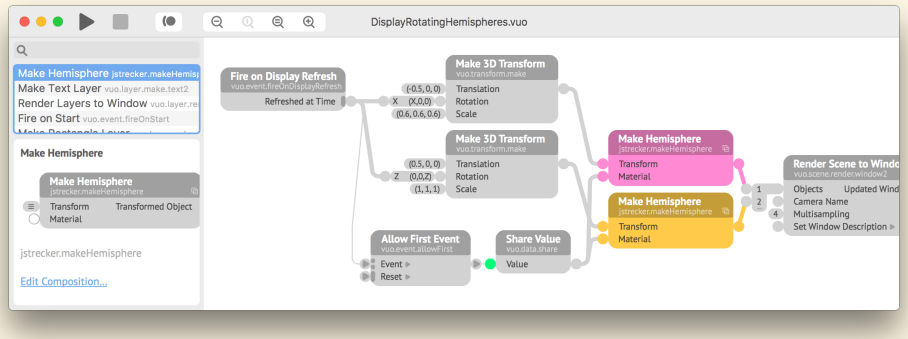


In the original composition, the nodes and cables outside of the selected part (which was packaged into a subcomposition) connected to the selected part at three points of contact:

- the **Transform** input port of **Transform 3D Object**,
- the **Material** input port of **Make Sphere**, and
- the **Transformed Object** output port of **Transform 3D Object**.

When the selected part was packaged into a subcomposition, these three points of contact became published input and output ports within the **Make Hemisphere** subcomposition. Correspondingly, they became input and output ports on the **Make Hemisphere** node.

Now that you have a **Make Hemisphere** node available, you can search for it in the Node Library and add more instances of it to your composition.



6.1 Making a subcomposition available to other compositions

Package as Subcomposition installs the subcomposition in a Composition-Local Library. This makes the subcomposition appear in the Node Library only when the composition it came from (or another composition in the same folder) is open. If you want the subcomposition to appear in the Node Library for all compositions, click on the subcomposition window and go to File > Move to User Library.

For an explanation of Composition-Local and User Library folders, see Installing a node.

6.2 Reasons to use subcompositions

The **Make Hemisphere** subcomposition illustrated one motivation for using subcompositions: to avoid recreating the same composition pieces over and over again. A subcomposition enables you to assemble a composition piece once and reuse it many times. If you notice a problem with the subcomposition or want to improve it, you only have to make the change in one place to have it apply everywhere the subcomposition is used.

Another reason you may want to use subcompositions is to better organize large compositions to make them more readable. You can replace a complex network of nodes and cables with a subcomposition that has a descriptive title and a clearly defined set of inputs and outputs.

A third reason for using subcompositions is to share your work with others in a modular format. When you create a composition piece that other people might like to use inside of their compositions, you can package it as a subcomposition that others can install in their Node Libraries.

6.3 Creating a subcomposition

In the example above, we created the **Make Hemisphere** subcomposition by selecting nodes and cables within a composition, right-clicking on them, and going to Package as Subcomposition. This installed the subcomposition in a Composition-Local Library.

If you want to create an empty subcomposition in a Composition-Local Library, you can right-click on the canvas and go to Insert Subcomposition.

New in Vuo 2.0

If instead you want to create a subcomposition in the User Library folder, you can open a new or existing composition and go to File > Save to User Library.

6.3.1 Naming a subcomposition

When you turn an already-saved composition into a node, the node's title derives from the composition's file name. A composition file called `Scribble.vuo` or `scribble.vuo` would be turned into a node titled **Scribble**. A composition file called `Solve Anagram.vuo` or `SolveAnagram.vuo` would be turned into a node titled **Solve Anagram**.

If you haven't yet saved the composition file, Vuo prompts you to enter a node title.

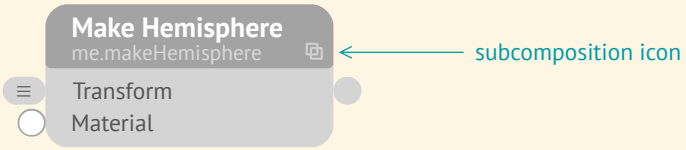
The node's class name is your vuo.org account name followed by a period followed by a lower-camel-case version of the node title — for example, `me.scribble` or `me.solveAnagram`. In the File > Save to User Library dialog, you can customize the prefix — for example, you could group your subcompositions into categories such as `me.interaction.scribble` or `me.algorithm.solveAnagram`.

After turning a subcomposition into a node, if you want to change the node's title, open the subcomposition and go to Edit > Composition Information....

New in Vuo 2.0

If you want to change the node's class name, rename the installed subcomposition file. Do this by right-clicking on the subcomposition node in the Node Library and choosing the menu item Show in Finder, finding your installed subcomposition in that folder (for example, `me.scribble.vuo`), and renaming the file. Be careful renaming a subcomposition, because any compositions that refer to the subcomposition by its old name will have an error until you substitute in the new version of the subcomposition.

6.4 Editing a subcomposition



A subcomposition node has an icon in its top-right corner, indicating that there's a composition inside that you can edit. You can open that composition by double-clicking on the subcomposition node on the canvas, or by right-clicking on the subcomposition node on the canvas or in the Node Library and choosing Edit Composition....

When you save changes made to the subcomposition, the changes apply everywhere the subcomposition is used. If the subcomposition is installed in the User Library or System Library folder, this means every instance of the subcomposition node in every composition that uses it.

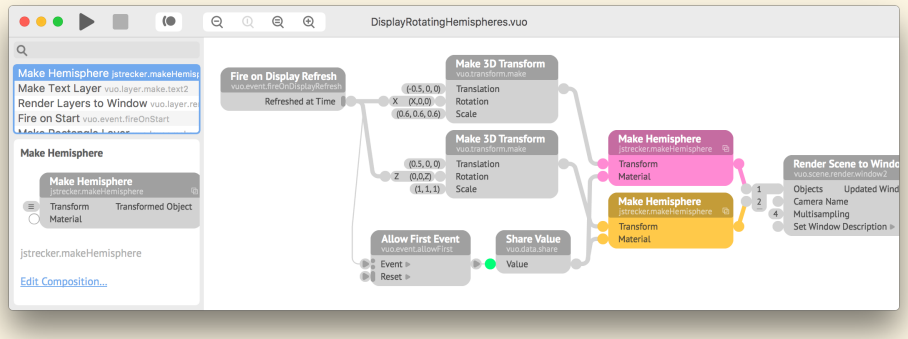
If the subcomposition is installed in a Composition-Local Library, changes likewise affect every instance of the subcomposition node in every composition that uses it. However, the compositions are limited to those in the same folder as the Composition-Local Library. It's possible to have different variants of a subcomposition (same node class name, different contents) installed in multiple Composition-Local Libraries — in which case, changes to one variant of the subcomposition don't affect compositions that use a different variant.

6.5 Watching events and data inside a subcomposition

New in Vuo 2.0

When you open a subcomposition via a node on the canvas — by double-clicking on the node or by right-clicking and choosing Edit Composition... — you can monitor the data and events flowing through the subcomposition just as you would in a regular composition — by opening port popovers and enabling Show Events.

For example, in the composition below, if you double-clicked on the magenta (upper) **Make Hemisphere** node to open the subcomposition, then opened a port popover within the subcomposition, you'd see the events and data that are flowing through the magenta instance of the subcomposition.



If instead you wanted to see the events and data that are flowing through the tangerine (lower) instance of **Make Hemisphere**, you would double-click on that node.

Be aware that if you open a subcomposition via the Node Library, even if a composition that contains the subcomposition is running, you won't see any data or events flowing through the subcomposition. You need to open it via a node on the canvas instead.

6.6 How events travel through a subcomposition

Changed in Vuo 2.0

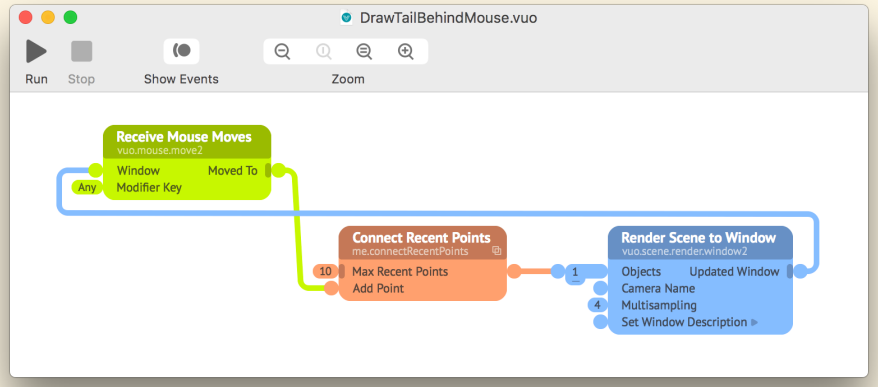
An event now only enters a subcomposition through the published input ports that it hit, not all published input ports.

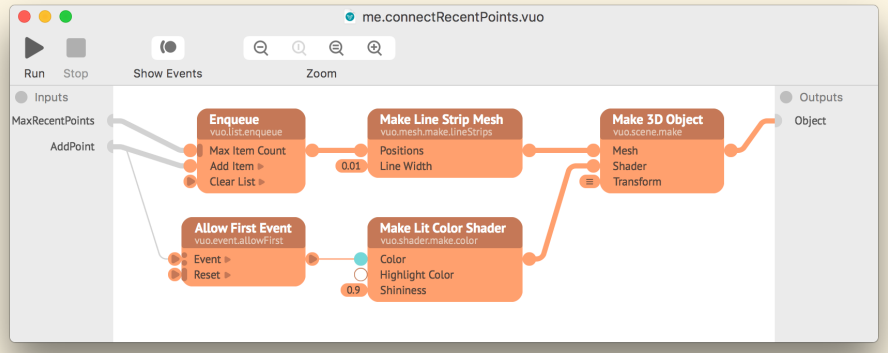
6.6.1 Events into a subcomposition

When an event hits an input port of a subcomposition node, it travels into the subcomposition through the corresponding published input port.

When an event hits multiple input ports of the subcomposition node, it travels in through all of the corresponding published input ports simultaneously.

To illustrate, here's a composition that uses a subcomposition node called **Connect Recent Points** to draw a series of connected line segments behind the mouse cursor as it moves. Below that is the **Connect Recent Points** subcomposition.

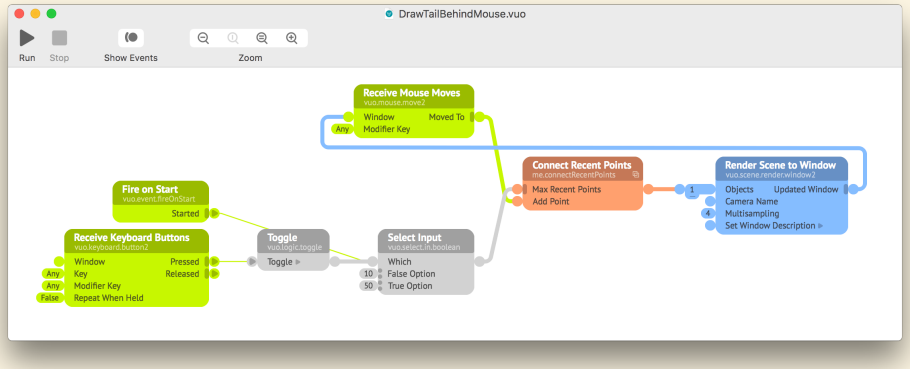




When an event hits the **Add Point** input port of the **Connect Recent Points** node (in the first composition above), it travels into the subcomposition (second composition above) through the **AddPoint** published input port. The event hits the **Enqueue** node and travels onward through the subcomposition.

You might be wondering about the **Max Recent Points** input port, which has no incoming events. We'll talk more about ports with constant values in a moment, but for now just know that the constant data does enter the subcomposition through the **MaxRecentPoints** published input port and reach the **Enqueue** node.

Here's a modification of the first composition above that allows the user to toggle between a short tail and a long tail by pressing any key.



Now events come in through the **MaxRecentPoints** published input port whenever the user presses a key and the **AddPoint** published input port whenever the user moves the mouse.

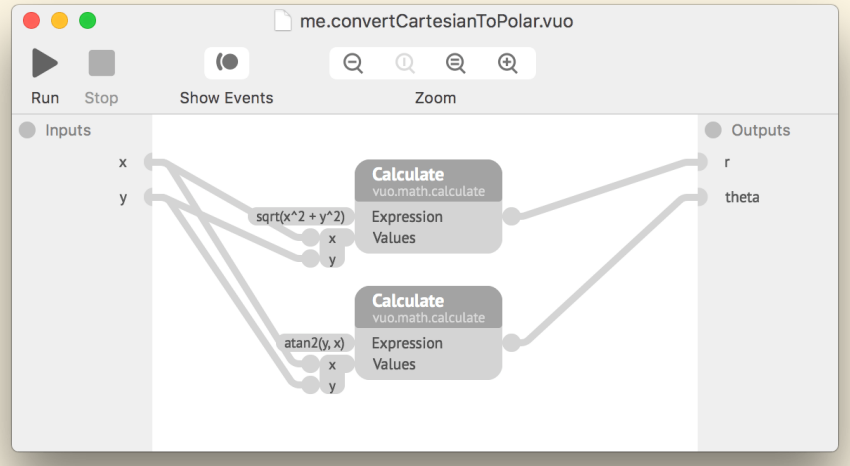
Changed in Vuo 2.0

An event now only exits a subcomposition through the published output ports that it hit, not all published output ports.

6.6.2 Events out of a subcomposition

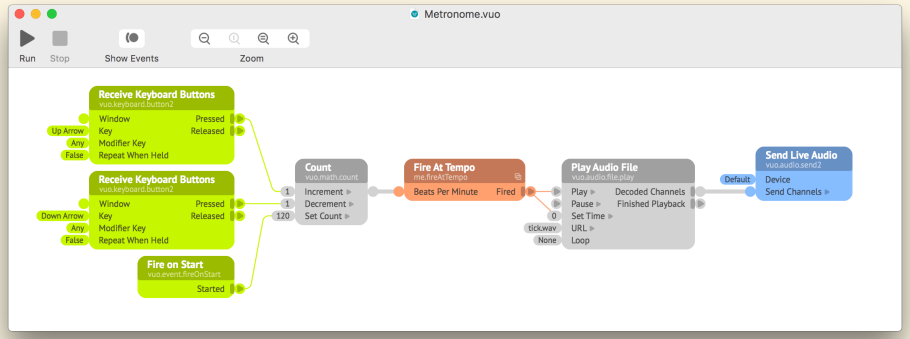
If an event reaches a published output port of a subcomposition, it travels out of the corresponding output port of the subcomposition node.

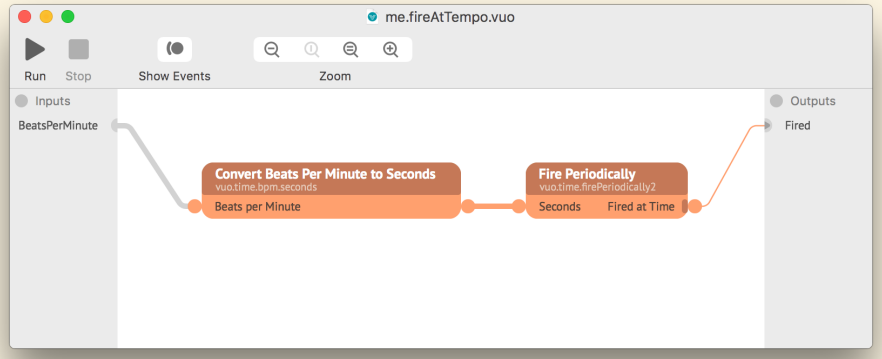
If an event into a subcomposition node reaches multiple published output ports of the subcomposition, it travels out of all of the corresponding output ports of the subcomposition node simultaneously. For example, in the subcomposition below, even though the **Calculate** nodes can execute concurrently and may not output their values at exactly the same time, the **Convert Cartesian To Polar** subcomposition node always outputs the event from its **R** and **Theta** ports simultaneously.



If an event that comes in through a subcomposition's published input ports doesn't reach any of the subcomposition's published output ports (because of wall or door ports within the subcomposition), then the event doesn't come of out any of the subcomposition node's output ports. The subcomposition node blocks the event.

A subcomposition can fire events, as demonstrated below. The **Fire At Tempo** subcomposition node is set to fire at a rate of 120 beats per minute.





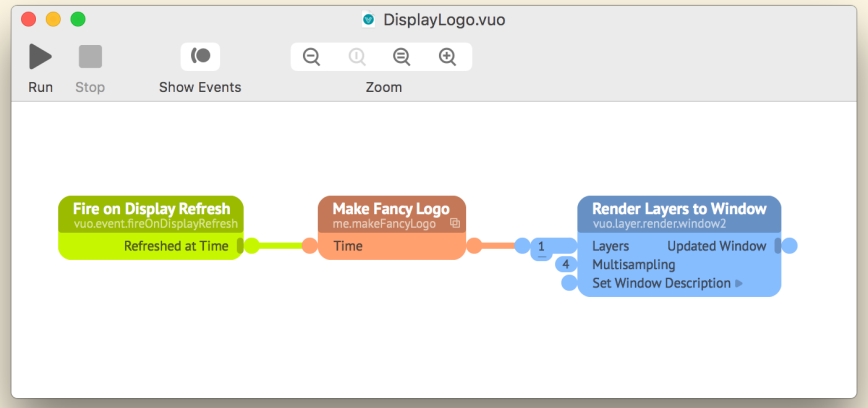
Changed in Vuo 2.0

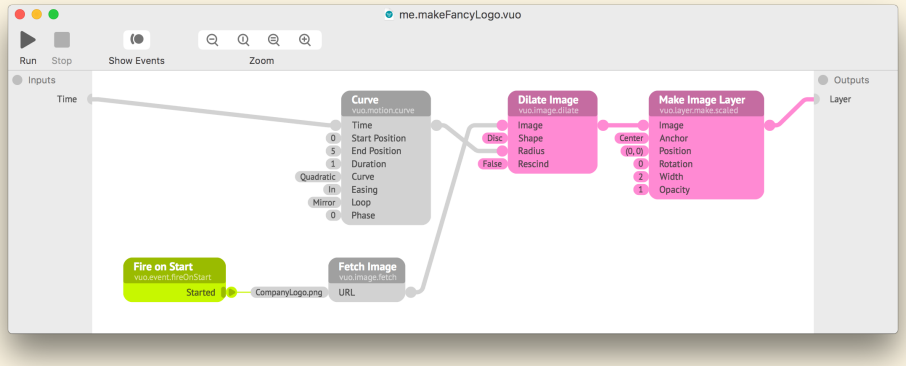
Events from triggers within a subcomposition don't exit the subcomposition if they overlap with events from published input ports.

If events fired from within the subcomposition can overlap with events from the published input ports, only the events from the published input ports will exit through the published output ports. The subcomposition node will transmit events but not fire events.

The subcomposition above, `me.fireAtTempo`, *does* fire events from its published output port. That's because there's no overlap between the events coming in through the **BeatsPerMinute** published input port (which are blocked at the **Fire Periodically** node's input port) and the events fired from the **Fire Periodically** node's output port.

The subcomposition below *does not* fire events from its published output port. That's because the event fired from the **Fire on Start** node travels along the same path as the events coming in from the published input port (nodes tinted magenta). The event fired from **Fire on Start** travels within the subcomposition but is blocked from exiting the subcomposition. Meanwhile, events that come in through the published input port do exit through the published output port.



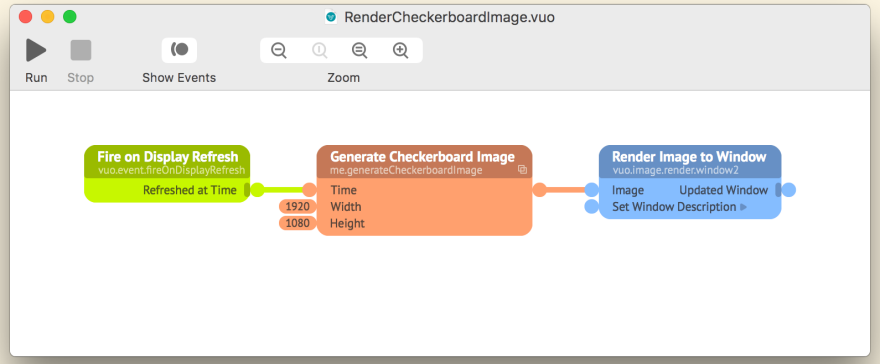


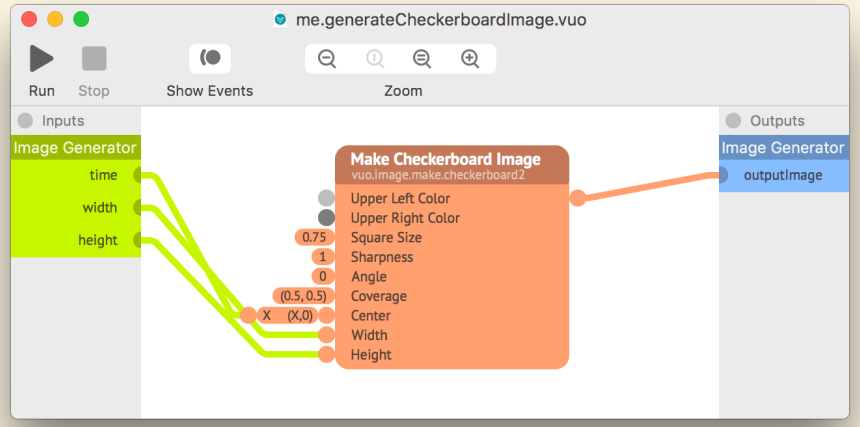
6.6.3 Constant input port values

If an input port on a subcomposition node has a constant value with no cables going into it, the constant value travels into the subcomposition through the corresponding published input port when the composition starts and whenever you edit the constant value.

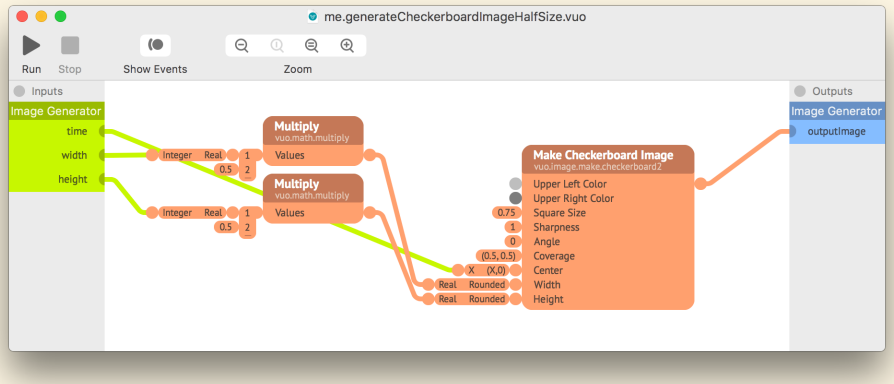
The data travels from the published input port to any input ports that are directly connected to it by a cable. This is a rare case in which data can travel without an event. The data reaches the input ports on nodes but does not cause the nodes to execute.

In the example below, the **Generate Checkerboard Image** subcomposition's **width** and **height** input ports are set to the constant values 1920 and 1080. The subcomposition outputs a 1920x1080 image.

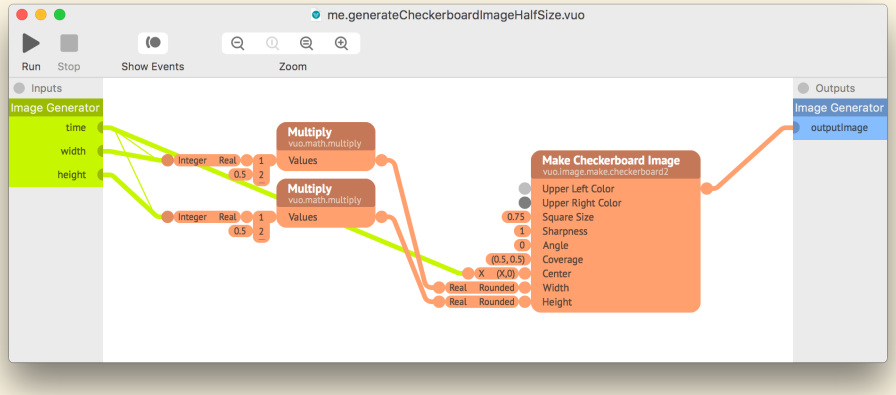




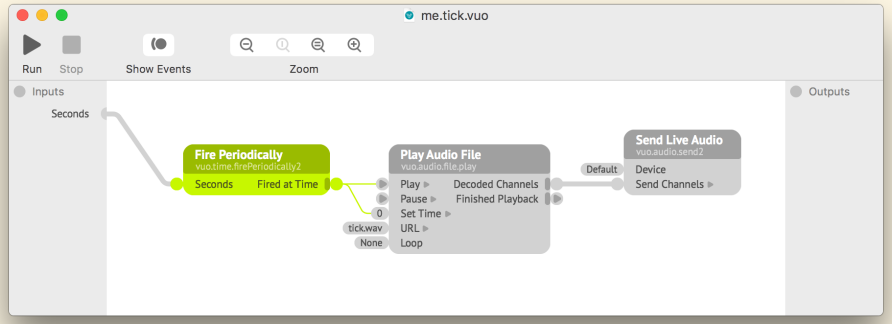
It's important to realize that the data travels only through the cable from the published input port to the next node, no farther. With the below variation on **Generate Checkerboard Image**, the subcomposition no longer outputs an image of size specified by **width** and **height**. Why? Because the constant values 1920 and 1080 only flow as far as the **Multiply** node's input ports. They don't cause the **Multiply** node to execute and pass its data along to the **Make Checkerboard Image** node.



To fix the composition above, you could add an event cable from the **time** published input port to each of the **Multiply** nodes. This would ensure that the **Multiply** nodes provide the halved width and height to **Make Checkerboard Image** whenever **Make Checkerboard Image** needs them, even if **width** and **height** have constant values.



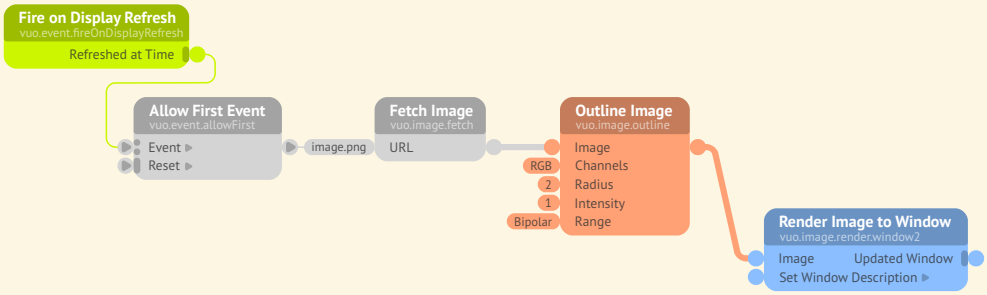
As long as the published input port connects directly to the node's input port, you can use a constant value to control a node that fires events. In the example below, editing the subcomposition node's **Seconds** input port would affect the firing rate of the **Fire Periodically** node inside of it.


me.tick.vuo
Run
Stop
Show Events
Zoom
Inputs
Seconds
Outputs
Fire Periodically
vuo.time.firePeriodically2
Seconds
Fired at Time
Play Audio File
vuo.audio.file.play
Play
Pause
0
tick.wav
None
Set Time
URL
Loop
Decoded Channels
Finished Playback
Default
Send Live Audio
vuo.audio.send2
Device
Send Channels

7 Making compositions fit a mold with protocols

You may have noticed that a lot of Vuo's nodes fall into groups where the nodes in the group are, in some sense, interchangeable. For example, what are some nodes that can fill in the blank in the composition below?

There are numerous possibilities: **Adjust Image Colors**, **Outline Image**, **Make Cartoon Image**, and **Reduce Haze**, just to name a few. Any node that inputs an image and outputs a modified version of that image will fit.



The idea of a **protocol** is to give a name to the kinds of nodes or compositions that can fill in a blank. A protocol defines a list of input and output ports, with certain names and data types.

Protocols are something you need to know about when exporting a composition as a movie, screen saver, or plugin (Exporting compositions) or using a composition in a VJ application.

You can find compositions exemplifying each protocol under File > Open Example > Image Generator, Image Filter, and Image Transition.

7.1 Image Filter protocol

The **Image Filter** protocol is for compositions that alter an image (or stream of images). For example, an Image Filter could add a special distortion effect to a video clip.

7.1.1 Published input ports

Name	Data type	Required/Optional	Description
image	Image	Required	The original image.
time	Real	Required	A number that changes over time, used to control animations or other changing effects.
duration	Real	Optional	For FxPlug: The length, in seconds, of the clip.
framerate	Real	Optional	For FxPlug: The framerate of the project, in frames per second.
frameNumber	Integer	Optional	For FxPlug: The number of frames since the beginning of the clip, starting at 0.
quality	Real	Optional	For FxPlug: The rendering quality or level of detail.

time and **quality** are further explained later in this section.

7.1.2 Published output ports

Name	Data type	Description
outputImage	Image	The altered image.

7.2 Image Generator protocol

The **Image Generator** protocol is for compositions that create an image (or stream of images). For example, an Image Generator could create a special animation for the opening sequence of a video.

7.2.1 Published input ports

Name	Data type	Required/Optional	Description
width	Integer	Required	The requested width of the image, in pixels.
height	Integer	Required	The requested height of the image, in pixels.
time	Real	Required	A number that changes over time, used to control animations or other changing effects.

offlineRender	Boolean	Optional	For movie export: *true* if the composition is being exported to a movie and *false* otherwise.
motionBlur	Integer	Optional	For movie export: The number of frames rendered per output frame. 1 means motion blur is disabled; 2, 4, 8, 16, 32, or 64 means motion blur is enabled.
duration	Real	Optional	For movie export and FxPlug: The length, in seconds, of the movie/clip.
framerate	Real	Optional	For movie export and FxPlug: The framerate of the movie/project, in frames per second.
frameNumber	Integer	Optional	For movie export and FxPlug: The number of frames since the beginning of the movie/clip, starting at 0.
quality	Real	Optional	For FxPlug: The rendering quality or level of detail.
screen	Screen	Optional	For screen savers: Which display the screen saver is running on. (macOS runs a separate instance of the composition on each display.)
preview	Boolean	Optional	For screen savers: *true* when the screen saver is running in the System Preferences preview thumbnail.

time and **quality** are further explained later in this section.

7.2.2 Published output ports

Name	Data type	Description
outputImage	Image	The created image. Its width and height should match the **width** and **height** published input ports.

7.3 Image Transition protocol

New in Vuo 2.0

The **Image Transition** protocol is for compositions that transition from one image (or stream of images) to another. For example, an Image Transition could crossfade from one scene in a movie to the next scene.

7.3.1 Published input ports

Name	Data type	Required/Optional	Description
startImage	Image	Required	The image to transition from.
endImage	Image	Required	The image to transition to.
progress	Real	Required	A number from 0 to 1 for how far the transition has progressed. At 0, the transition is at the beginning, with only **startImage** showing. At 0.5, the transition is halfway through. At 1, the transition is complete, with only **endImage** showing. When previewing the composition in Vuo, the mouse position left to right controls **progress**.
time	Real	Required	A number that changes over time, used to control animations or other changing effects. **time** is independent of **progress**.
duration	Real	Optional	For FxPlug: The length, in seconds, of the transition.
framerate	Real	Optional	For FxPlug: The framerate of the project, in frames per second.
frameNumber	Integer	Optional	For FxPlug: The number of frames since the beginning of the transition, starting at 0.
quality	Real	Optional	For FxPlug: The rendering quality or level of detail.

time and **quality** are further explained later in this section.

7.3.2 Published output ports

Name	Data type	Description
outputImage	Image	The resulting image.

7.4 Time

The **time** published input port, which appears in multiple protocols, has a slightly different meaning depending on the context.

- In most situations, including when previewing a protocol-compliant composition in Vuo, **time** is the number of seconds since the composition started running.
- When exporting a movie, **time** is the number of seconds from the start of the movie to the beginning of the current frame.
- In an exported FxPlug plugin, **time** is the number of seconds since the start of the clip (for generators and effects) or transition.

7.5 Quality

Another published input port common to multiple protocols is **quality**. In exported FxPlug plugins, this port's value is the requested rendering quality or level of detail, from 0 (low quality / faster performance) to 1 (high quality / slower performance).

In Final Cut Pro X, **quality** is always 0.5.

In Motion, the Render > Normal and Draft settings correspond to value 0.5. The Render > Best setting corresponds to value 1.0. More information about render quality is in the Motion documentation.

Changed in Vuo 2.0

Added the Export submenu.

7.6 Creating a protocol composition

To create a composition that conforms to a protocol, choose one of the options under File > New Composition from Template > Protocol or Export. If you plan to export the composition (to create a movie or screen saver, for example), then your best option is to pick from the Export submenu. These menu items automatically add the optional protocol published ports relevant to the chosen export type.

If you've already started working on a composition and want to make it conform to a protocol, go to Edit > Protocols and choose a protocol.

7.7 Editing a protocol composition

If you didn't get the optional protocol published ports automatically by choosing a menu item under File > New Composition from Template > Export, you can still add them later. You add them in the same way that you would add a non-protocol published port. Be sure to set the published port's name and data type exactly as they appear in this manual. Names are case-sensitive.

7.8 Running a protocol composition

When you run an Image Generator, Image Filter, or Image Transition composition with the Run button, Vuo feeds data and events into the published input ports and displays the published output image in a window. This makes it easy to preview how the composition will look when run inside of an exported product or another application.

You can change the images the Vuo feeds into protocol published input ports. For an Image Filter composition, drop an image file onto the running composition's window to change the image being filtered. For an Image Transition composition, drop an image file onto the left or right half of the window to change the start or end image, respectively.

Changed in Vuo 2.0

You no longer need to block unnecessary published input events with Allow Changes nodes.

7.9 How events travel through a protocol composition

Whether you're running a protocol-compliant composition while exporting a movie, within an exported product such as a screensaver, or inside of another application, the same basic rules apply for how data and events enter through the published input ports:

- A data-and-event published input port transmits its data and event whenever the data changes.
  - The first event enters through every published input port.
  - Subsequent events enter only through published input ports whose data has changed, generally speaking.
  - However, there are some exceptions. Depending on which data type the port has and who is running the composition (for example, Vuo or another application), the published input port may transmit every event.
- An event-only published input port never transmits an event.
- The next event comes in through the published input ports only after the composition has finished processing the current event and any events spun off from it.
  - Events spun off are those output by **Spin Off Event**, **Spin Off Events**, **Spin Off Value**, **Build List**, and **Process List**.
  - The composition has finished processing an event when the event either has reached the published output ports or has been blocked within the composition.

Changed in Vuo 2.0

You no longer need to ensure that exactly one event reaches the published output ports for each event from the published input ports.

Image Generator, Image Filter, and Image Transition compositions are typically expected to do their jobs at a steady rate — receiving events and providing output images at evenly spaced time intervals. Triggers within a composition may fire additional events, but those events don't affect the host (movie exporter, exported product, or other application) running the composition. The composition continues to provide output images at the same rate that it receives events.

8 Exporting compositions

You may want to export a Vuo composition...

- to create a finished product, such as a video, image, or app,
- to create a plugin for another application, such as video editing or VJ software,
- to share your work with people who don't have Vuo installed.

This section covers the many ways that you can export a Vuo composition to another format.

8.1 Exporting an image

If you want to capture an image of a composition, you can either take a screenshot (open the Preview app and go to File 〉 Take Screenshot) or use the **Save Image** node (see the node's description for details).

8.2 Exporting a movie

Vuo offers several ways to create a movie from a composition:

- For an easy way to record the graphics displayed in a window, in the composition's menu go to File 〉 Start Recording.
- For the highest-quality rendering, make your composition use the Image Generator protocol, and in Vuo go to File 〉 Export 〉 Movie...
- To control the movie export from within your composition, use the **Save Images to Movie** node or the **Save Frames to Movie** node. (See each node's description for details.)
- To control the movie export from the command line, use the `vuo-export` command-line tool. (See Exporting a composition on the command line for details.)

8.2.1 Recording the graphics in a window

To record a movie:

- Run a composition that shows at least one window.
- If your composition has more than one window, click on the one you want to record to make it the active (frontmost) window.
- Go to File 〉 Start Recording. This immediately starts recording the movie.
- Let the composition run for as long as you want to record the movie. You can interact with the composition while it's recording.
- Go to File 〉 Stop Recording... This immediately stops recording the movie and presents a save dialog.
- In the save dialog, choose the file where you want to save your movie.

When you start recording, the graphics showing in the window at that moment are added as a frame in the movie. After that, each time the window being recorded renders some graphics — in other words, each time the **Render Image to Window**, **Render Layers to Window**, or **Render Scene to Window** node receives an event — a frame is added to the movie. If your composition is rendering about 60 frames per second, then your movie will play back at about 60 frames per second. If your composition renders once, then waits 10 seconds, then renders again, your movie will do the same — show the first frame for 10 seconds, then show the second frame.

The dimensions of the rendered movie match the dimensions of the window's graphics area at the moment when you start recording. If you resize the window while the recording is in progress, then the recorded images will be scaled to the movie's dimensions.

If your composition has multiple windows, then the active (frontmost) window at the time when you went to File 〉 Start Recording will be the one recorded. Only the content displayed within the window's graphics area — not the window's title bar, not the cursor, and not any audio — will be recorded in the movie.

Although recording from a composition window is an easy way to create a movie, and allows you to interact with the composition while the recording is being made, it does limit the quality of the movie. Recording a movie in real time means that your computer has to do extra processing, beyond just running the composition. Depending on how powerful your computer is, this may slow the composition down or make it render choppily, and do the same to the recorded movie.

The most reliable way to avoid slowness or choppiness is to export a movie from an Image Generator composition, as described in the next section. But if you do want to record from a composition window, here are some ways to improve the quality of your recording:

- Avoid doing other processor-intensive things on your computer (such as running other compositions) while the recording is in progress.
- Limit the size of the window that you record. (Larger windows require more processing power.)
- Avoid resizing the window during a recording. (Scaling the movie frames after the window has been resized requires more processing power.)

8.2.2 Exporting a movie from an Image Generator composition

Another way to create a movie from a composition is with File > Export > Movie... . Instead of recording a composition in real time, this option runs the composition invisibly and takes as long (or short) as needed to render each movie frame. The resulting movie has a precise frame rate and no dropped frames. You can choose the start and end time, frame rate, and dimensions. Optionally, you can add antialiasing and motion blur (if you have Vuo Pro).

To export a movie:

- Go to File > New Composition from Template > Export > Movie. This creates a composition that conforms to the Image Generator protocol.
- Add nodes to the composition to make it output a stream of images.
- Go to File > Export > Movie... .
- In the dialog that appears, choose the movie file to output to and the other settings for your movie.
- Click the Export button.

8.3 Exporting a screen saver

New in Vuo 2.0

You can turn your Vuo compositions into screen savers that will run on macOS 10.11 and later.

- In Vuo, go to File > New Composition from Template > Export > Screen Saver. This creates a composition that conforms to the Image Generator protocol.
- Add nodes to the composition to make it output a stream of images.
- Go to File > Export > Mac Screen Saver.
- When the export is complete, relaunch System Preferences.
- In System Preferences, go to Desktop & Screen Saver > Screen Saver and find your screen saver.

8.3.1 Sharing screen savers

You can share the screen savers you've created in Vuo with other people, even if they don't have Vuo. To find a screen saver that you've exported:

- In Finder, hold down ⌥ and go to Go > Library.
- In that folder, navigate to 🗀 `Screen Savers`.
- Locate the screen saver (a `.saver` file).

When you send the screen saver to someone else, here's how they can install it:

- Right-click on the `.saver` file and choose Open.
- In the dialog that warns that the file is from an unidentified developer, click Open.
- In the dialog that asks if you want to install the screen saver, click Install.

Alternatively, the person installing the screen saver can navigate to the 🗀 `Screen Savers` folder as above and drop the `.saver` file in there.

8.4 Exporting an FxPlug plugin

New in Vuo 2.0

You can turn your Vuo compositions into custom effects, transitions, and generators for Final Cut Pro X and Motion.

8.4.1 Video effects

- In Vuo, go to File > New Composition from Template > Export > FxPlug > Effect. This creates a composition that conforms to the Image Filter protocol.
- Add nodes to the composition to make it alter the input image and output the result.
- Go to File > Export > FxPlug.
- When the export is complete, relaunch Final Cut Pro.
- In Final Cut Pro, find the plugin in the Effects Browser under Video > Vuo.

8.4.2 Transitions

- In Vuo, go to File > New Composition from Template > Export > FxPlug > Transition. This creates a composition that conforms to the Image Transition protocol.
- Add nodes to the composition to make it combine the input images and output the result.
- Go to File > Export > FxPlug.
- When the export is complete, relaunch Final Cut Pro.
- In Final Cut Pro, find the plugin in the Transitions Browser under Vuo.

8.4.3 Generators

- In Vuo, go to File > New Composition from Template > Export > FxPlug > Generator. This creates a composition that conforms to the Image Generator protocol.
- Add nodes to the composition to make it output a stream of images.
- Go to File > Export > FxPlug.
- When the export is complete, relaunch Final Cut Pro.
- In Final Cut Pro, find the plugin in the Titles and Generators sidebar under Vuo.

8.4.4 Category and name

By default, plugins exported from Vuo are installed in Final Cut Pro under the Vuo category. To choose a different category, before exporting go to Edit > Composition Information..., click on the Exporting tab, and enter an FxPlug Group.

In Edit > Composition Information..., under the General tab, you can change the name of the plugin displayed in Final Cut Pro.

8.4.5 Parameters

When creating a composition to be exported as an FxPlug, the composition will have the published ports required by the protocol. It may also include some of the optional published ports for FxPlug listed in Making compositions fit a mold with protocols.

You can add non-protocol published input ports as well. These appear in the Inspector panel in Final Cut Pro.

Non-protocol published input ports with the following data types are controllable in Final Cut Pro or Motion:

Data type	Notes
Real	
2D Point	
3D Point	
4D Point	
Integer	
Boolean	

Text	
Color	
Image	Available in Motion but not Final Cut Pro
Option types	Data values edited in Vuo using menus

Published input ports that have menu input editors in Vuo may behave the same or differently in Final Cut Pro, depending on the port's data type. You can check the port's data type by clicking on the port to open its popover. If the data type is something other than Integer — for example, the Blend Mode type of the **Blend Images** node's **Blend Mode** port — then Final Cut Pro will present a menu. If the data type is Integer, then Final Cut Pro will present a slider to select by number, with 0 corresponding to the first menu item.

You can adjust the default, minimum, and maximum values for a parameter in Final Cut Pro by editing the published input port in Vuo. Right-click on the published input port and select Edit Value... to change the default value or Edit Details... to change the minimum and maximum.

8.4.6 Image scaling

When Final Cut Pro provides images to your composition via the Image Filter's `image` input port, or the Image Transition's `startImage` and `endImage` input ports, Vuo assigns the image a **Scale Factor** based on the scale of the image relative to the Motion Template. Some image filter nodes use the Scale Factor to keep effects looking consistent between Final Cut Pro's Better Performance and Better Quality modes (see the `vuo.image` node set documentation for a list of these nodes). All image filter nodes give their output image the same scale factor as the input image.

Vuo creates Motion Templates with resolution 5120x2700. If, for example, your Final Cut Pro project is 1920x1080, the input image's Scale Factor will be 0.375 (1920/5120) in Better Quality Mode, or 0.1875 (1920/5120/2) in Better Performance Mode. If you use a `Blur Image` node with radius 100 points, it will be scaled to 37.5 pixels in Better Quality Mode, or 18.75 pixels in Better Performance mode, resulting in the same effective blur amount in both modes.

8.4.7 Sharing plugins

You can share FxPlug plugins you've created in Vuo with other people, even if they don't have Vuo. There are two files associated with a plugin. To find them:

- FxPlug file

  - In Finder, hold down ⌥ and go to Go > Library.
  - In that folder, navigate to Plug-Ins > FxPlug.
  - Locate the plugin (a `.fxplug` file).

- Motion template
  - In Finder, go to Go > Home.
  - In that folder, navigate to Movies > Motion Templates.
  - Within the Effects, Generators, or Transitions folder, locate the file for your plugin.

You can send these two files to someone else, who can install them in the same location on their computer. Both files are needed for the plugin to work.

8.4.8 Uninstalling plugins

To uninstall a plugin that was exported from Vuo:

- Quit Final Cut Pro and Motion.
- Locate the two files described in the previous section, and throw them in the Trash.

8.4.9 Maintaining compatibility between plugin versions

Each FxPlug has a unique identifier (UUID). Vuo generates this unique identifier based on your composition's Bundle Identifier, which you can set in Edit > Composition Information..., under the Exporting tab. Final Cut Pro X uses this identifier to keep track of which plugin provides the effects for clips on the timeline.

For example, say you release version 1.0 of a plugin, and people start using it in their Final Cut Pro X projects. Then you want to release version 1.1 of your plugin which fixes a small issue — if the new version has the same Bundle Identifier as version 1.0, then the updated plugin will automatically be used throughout people's existing Final Cut Pro X projects when they install the plugin. You can even change the name of the plugin (Edit > Composition Information..., under the General tab), and it will continue to be associated with people's existing Final Cut Pro X projects.

Then, say, you want to release version 2.0 of your plugin, which differs significantly from version 1. You don't want the updated plugin to automatically be used in people's existing Final Cut Pro X projects, since the changes in 2.0 might cause unwanted appearance changes to their projects. In this case, you should change the Bundle Identifier — for example, add a `.2` suffix, as in `com.mycompany.myplugin.2` — so people will be able to continue using the old version of your plugin in their existing projects, while also being able to use the new version of your plugin when they pick it from Final Cut Pro X's Effects, Transitions, or Generators browser.

8.5 Exporting an FFGL plugin

New in Vuo 2.0

You can turn your Vuo compositions into FFGL (FreeFrame 1.6+) plugins that can be loaded by many VJ apps on macOS, including Resolume Avenue, Resolume Arena, Magic Music Visuals, VDMX, and Isadora 3.

FFGL plugins exported from Vuo can run only in 64-bit apps. Most macOS apps these days are 64-bit, although some older VJ apps are still 32-bit. You can check your app's documentation to see if it's 64-bit or 32-bit.

You can create three kinds of FFGL plugins in Vuo: *sources* using the image generator protocol, *effects* using the image filter protocol, and *blend modes* using the image transition protocol. To learn how to use protocols, see Making compositions fit a mold with protocols.

After exporting an FFGL plugin, you may need to restart your VJ app for the plugin to become available.

8.5.1 Sources

- In Vuo, go to File > New Composition from Template > Export > FFGL > Source. This creates a composition that conforms to the Image Generator protocol.
- Add nodes to the composition to make it output a stream of images.
- Go to File > Export > Mac FFGL Plugin.

8.5.2 Effects

- In Vuo, go to File > New Composition from Template > Export > FFGL > Effect. This creates a composition that conforms to the Image Filter protocol.
- Add nodes to the composition to make it alter the input image and output the result.
- Go to File > Export > Mac FFGL Plugin.

8.5.3 Blend modes

- In Vuo, go to File > New Composition from Template > Export > FFGL > Blend Mode. This creates a composition that conforms to the Image Transition protocol.
- Add nodes to the composition to make it combine the input images and output the result.
- Go to File > Export > Mac FFGL Plugin.

8.5.4 Name

Vuo names the plugin according to the Name field in Edit > Composition Information... under the General tab. Since FFGL limits plugin names to 16 characters, Vuo shortens the name if needed.

8.5.5 Parameters

In addition to the published ports required by the Image Filter or Image Generator protocol, you can create other published input ports to appear as parameters in your VJ app.

You can use the following data types for non-protocol published input ports for FFGL plugins:

Data type	Scaled range	Shortened name	Notes
Real	0 to 1	16 characters	
2D Point	(0,0) to (1,1)	14 characters	
3D Point	(0,0,0) to (1,1,1)	14 characters	
4D Point	(0,0,0,0) to (1,1,1,1)	14 characters	
Integer	0 to 1	16 characters	
Boolean		16 characters	
Text		16 characters	
Color		14 characters	
Image		16 characters	Not supported in Resolume

Since FFGL limits numeric values to the range 0 to 1, Vuo automatically scales parameter values from that range to the range you've specified in your composition (by right-clicking on the published input port, going to Edit Details..., and editing Suggested Min and Suggested Max). For example, if you have a Real published input port with Suggested Min -10 and Suggested Max 10, your VJ app will show a slider from 0 to 1. Your composition will get an input value of -10 when the slider is at 0, -5 when the slider is at 0.25, and 10 when the slider is at 1.

Since FFGL limits parameter names to 16 characters, Vuo shortens names if needed. For 2D, 3D, and 4D Point published input ports, Vuo creates a separate parameter for each coordinate (X, Y, Z, or W) and appends a space and the coordinate name to the parameter name. To fit in these 2 extra characters, Vuo shortens the rest of the name to 14 characters. Similarly, for Color published input ports, Vuo creates a separate parameter for each channel (R, G, B, or A).

8.5.6 Sharing plugins

You can share FFGL plugins you've created in Vuo with other people, even if they don't have Vuo. To find an FFGL plugin that you've exported:

- In Finder, hold down ⌥ and go to Go > Library.
- In that folder, navigate to 🗀 `Graphics > FreeFrame Plug-Ins`.
- Locate the plugin (a `.bundle` file).

You can send this file to someone else, who can install it in the same location on their computer.

8.5.7 Uninstalling plugins

To uninstall an FFGL plugin that was exported from Vuo, locate the `.bundle` file as described in the previous section and throw it in the Trash.

8.6 Exporting an application

Using the File > Export > Mac App... menu item, you can turn your composition into an macOS application (*.app* file).

When exporting a composition that refers to files on your computer (such as images, scenes, or movies), typically Vuo will know to copy those into the exported app. If you've added these files to your composition by dragging them onto the canvas (without holding down ⌥) — creating a node such as **Fetch Image** or **Play Movie** — then the files will automatically be copied into the exported app. In fact, Vuo will automatically copy files and folders for all relative paths found in ports named **URL**, **URLs**, or **Folder** on nodes that read files.

If you've held down ⌥ while dragging a file onto the canvas, or if you've typed an absolute path into the input editor for a URL, then Vuo won't copy the file into the exported app. This is useful if you want to refer to a file that you know will be in a certain location on every computer that runs the app, such as an image that comes with the operating system.

In some cases, you may want a file to be copied into the app, but Vuo may not be able to figure this out. This may happen, for example, if your composition uses an **Append Text** node to construct relative file paths out of smaller pieces. If Vuo doesn't copy your files into the exported app automatically, then you can copy them yourself. For example, if your composition uses a file called `image.png`:

- Place `image.png` in the same folder as your composition (.vuo file).
- Go to File > Export > Mac App... and create `MyApp.app`.
- Right-click on `MyApp.app` and choose Show Package Contents.
- In the package contents, go to the 🗀 `Contents‣Resources` folder. Copy `image.png` into that folder.

9 Turning graphics shaders into nodes

New in Vuo 2.0

If you're familiar with the programming languages C/C++ and GLSL, you can create your own graphics nodes using Vuo's SDK.

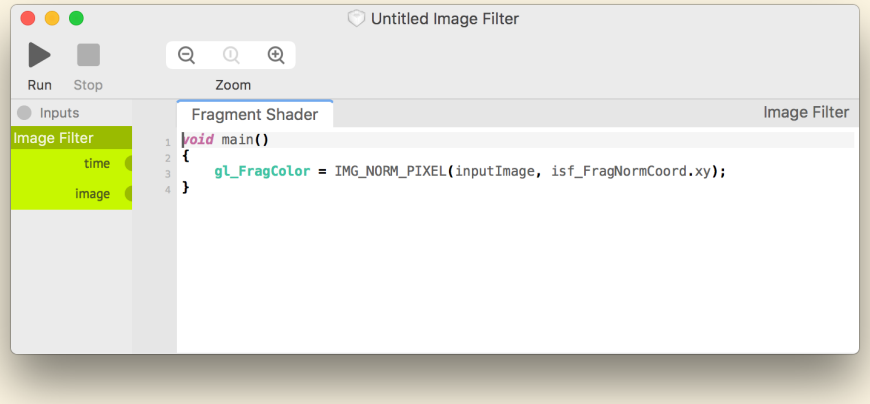
For certain kinds of graphics nodes, there's a shortcut. You don't have to write C/C++ boilerplate, just GLSL. And you don't have to use the Vuo SDK; you can edit the code without leaving the Vuo application.

Vuo can turn GLSL fragment shaders in Interactive Shader Format (ISF) into nodes. The ISF code's inputs and output are automatically turned into input and output ports on the node.

9.1 Creating an ISF node

There are two ways to begin developing an ISF node in Vuo. One is to start with an ISF fragment shader that you've already written or downloaded. Save the file to the same location that you would install a node. Then find the node in your Node Library, right-click on it, and go to Edit Shader….

The other way is to start from scratch. Go to File › New Shader › Image Filter, Image Generator, or Image Transition. (Making compositions fit a mold with protocols explains Image Filters, Image Generators, and Image Transitions.) This opens a window with a small template as a starting point.

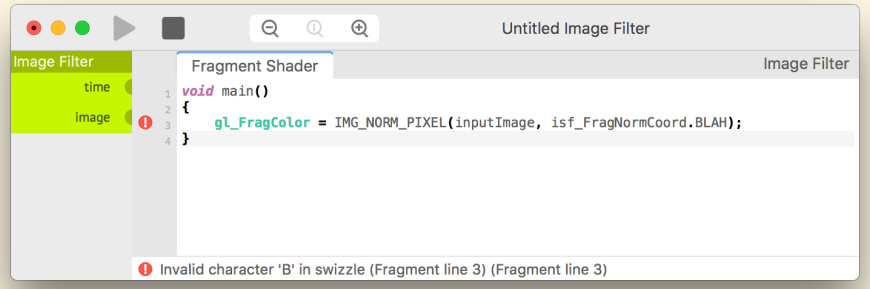


9.2 Editing an ISF node

The ISF code's inputs appear as published input ports in the left sidebar. As with published input ports in a composition, you can edit their default values and suggested ranges, rename them, and add more. The published input ports correspond to variables in the source code (as explained later in this section). If you rename a published input port, you also need to rename the corresponding variable.

You can preview the shader using the Run button. You can edit the shader's published input ports and GLSL code while it's running. Changes to a published input port's value and details take effect immediately. For other kinds of changes, after editing, go to Run > Restart to make your changes take effect in the running preview.

If there are any problems with your shader that prevent it from compiling, the error messages are displayed at the bottom of the window, with the relevant lines marked alongside the code.



To change the title, keywords, description, and other metadata of the ISF node, go to Edit > Composition Information....

If you'd prefer, you can edit the ISF source code in a text editor of your choice. Be aware that changes will only take effect in Vuo when you save the file. You won't see errors reported in your text editor, only in the Console app. If you're using TextEdit, be sure to disable Smart Quotes.

9.3 Saving an ISF node

To be able to use a shader as a node, you'll need to save it to one of the locations where you would install a node.

You can also edit and run shaders in Vuo without installing them as nodes. These shaders can be saved anywhere on your computer.

9.4 How ISF source code translates to a Vuo node

To be recognized by Vuo, the ISF source code must have a file with extension `.fs` containing GLSL code for a fragment shader.

(Optionally, the ISF source code may also have a vertex shader in a `.vs` file. This is an experimental feature in Vuo and may not work as expected.)

9.4.1 Node metadata

As with subcompositions and other custom nodes, the file name becomes the node class name. For example, an ISF file called `me.image.squiggle.fs` becomes a node with class name `me.image.squiggle`.

The keys and values in the JSON-formatted comment at the beginning of the ISF file are translated to the Vuo node as follows.

ISF key	Vuo node characteristic	Notes
`LABEL`	Title	Shown at the top of the node.
`DESCRIPTION`	Description	Shown in the Node Library.
`CREDIT`	Appended to description	Shown in the Node Library.
`VSN`	Version	Shown in the Node Library.
`KEYWORDS`	Keywords	Used when searching the Node Library.

9.4.2 Ports

In most cases, the input and output ports on the Vuo node correspond to the items listed under `INPUTS` and `OUTPUTS` in the ISF file's JSON-formatted comment.

ISF key	Vuo port characteristic	Notes
`NAME`	Internal name	Used when saving a composition to file.
`LABEL`	Display name	Shown on the node.
`TYPE`	Data type	See the next section for details.
`DEFAULT`	Initial/default constant value	For input ports only.

`MIN`, `MAX`, `STEP`	Suggested minimum, maximum, and step value	For input ports only. Used in the input editor.
`VALUES`, `LABELS`	Menu items	For integer input ports with a fixed set of options. Used in the input editor.

If an ISF input has `"TYPE"="size"`, it is turned into two integer input ports on the Vuo node: **Width** and **Height**.

If an ISF file provides no way to determine the output image's size — no input with `"TYPE":"image"` or `"TYPE"="size"` — then input ports **Width** and **Height** are automatically added to the Vuo node.

If an ISF file lacks an output with `"TYPE"="image"`, an output port called **Output Image** is added automatically to the Vuo node.

One Vuo input port is unusual in that it's not determined by the `INPUTS` and `OUTPUTS` (or lack thereof) in the JSON-formatted comment, but rather by the content of the GLSL code. That is the **Time** port. In any ISF shader, a uniform called `TIME` of type float is automatically declared. If you use the `TIME` uniform anywhere in your GLSL code, an input port called **Time** is added to your Vuo node automatically.

9.4.3 Data types

Vuo supports most ISF data types plus some additional data types specific to Vuo.

ISF data type	Vuo data type	Vuo-specific?
`event`	Boolean	no
`bool`	Boolean	no
`long`	Integer	no
`float`	Real	no
`color`	Color	no
`image`	Image	no
`point2d`	2D Point	no
`point3d`	3D Point	yes
`point4d`	4D Point	yes
`colorDepth`	Image Color Depth	yes
`size`	Converted to two Integer ports	yes

`bool[]`	List of Boolean	yes
`long[]`	List of Integer	yes
`float[]`	List of Real	yes
`point2d[]`	List of 2D Point	yes
`point3d[]`	List of 3D Point	yes
`point4d[]`	List of 4D Point	yes
`color[]`	List of Color	yes

9.4.4 Output image size and color depth

If the Vuo node created from an ISF shader has input ports **Width** and **Height**, the output image's size is set by these ports. Otherwise, the output image's size is the same as the image in the first populated image port — in other words, the top-most image port whose popover shows a value other than "(no image)".

If the Vuo node has an input port of type Image Color Depth, the output image's color depth is set by that port. Otherwise, the output image's size matches the image in the first populated image port.

9.4.5 Coordinates

Although not part of the ISF 2.0 specification, to be consistent with many official and unofficial examples of ISFs, Vuo treats inputs of type 2D point specially. If an input has type 2D point and does not have `MIN` and `MAX` specified, then the input port value is scaled from normalized coordinates to pixel coordinates when used as a uniform in the GLSL code. For example, if an input port has value (1.0, 0.5) and the output image is to be 1000 x 800 pixels, then the uniform has value (1000, 400).

3D and 4D points are not scaled.

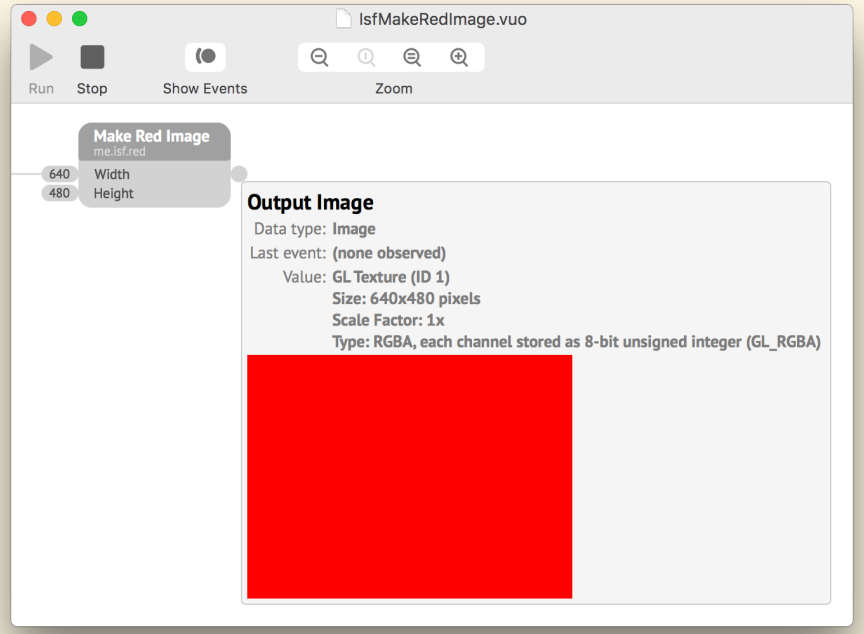
9.4.6 Examples

The examples below focus on how ISF source code translates to Vuo node characteristics, with minimal GLSL code. (For examples with more interesting GLSL code, see the ISF website.) After each ISF source listing is the Vuo node that it creates.

Listing 1: Input and output ports are added automatically.

```
/*{
   "LABEL":"Make Red Image"
}*/

void main()
{
  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

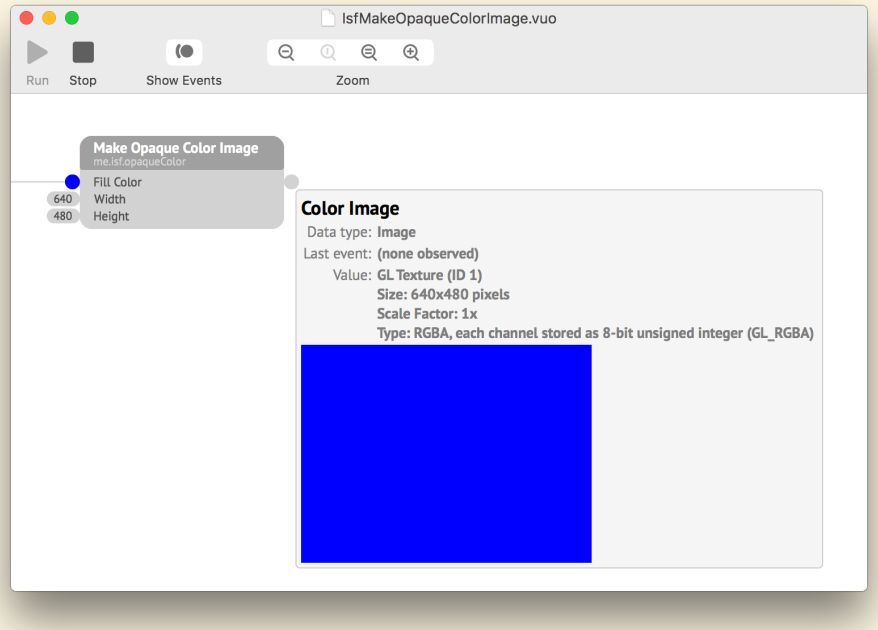


Listing 2: Input and output ports are specified in the ISF code.

```
/*{
   "ISFVSN":"2.0",
   "TYPE":"IMAGE",
   "LABEL":"Make Opaque Color Image",
  "INPUTS":[
    {
      "NAME":"fill",
      "LABEL":"Fill Color",
      "TYPE":"color",
      "DEFAULT":
      {
        "r":0.0,
```

```
        "g":0.0,
        "b":1.0,
        "a":1.0
      }
    },
    {
      "TYPE":"size"
    }
  ],
  "OUTPUTS":[
    {
      "NAME":"colorImage",
      "TYPE":"image"
    }
  ]
}*/

void main()
{
  gl_FragColor = vec4(fill.rgb, 1.0);
}
```

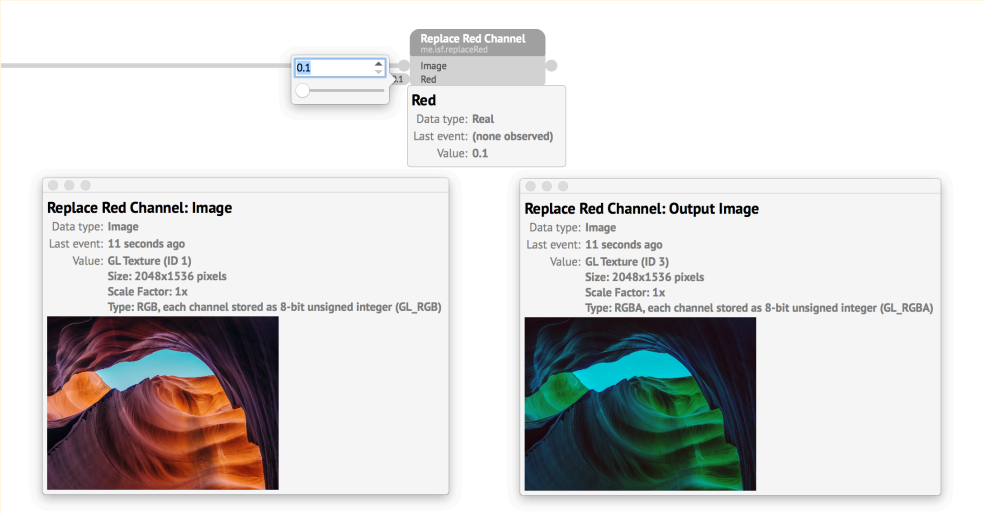


Listing 3: An input port with suggested minimum and maximum values.

```
/*{
  "LABEL":"Replace Red Channel",
```

```
  "INPUTS":[
    {
      "NAME":"inputImage",
      "TYPE":"image"
    },
    {
      "NAME":"red",
      "TYPE":"float",
      "MIN":0.1,
      "MAX":0.9,
      "DEFAULT":0.5
    }
  ]
}*/

void main()
{
  gl_FragColor = vec4(red, IMG_THIS_NORM_PIXEL(inputImage).gba);
}
```

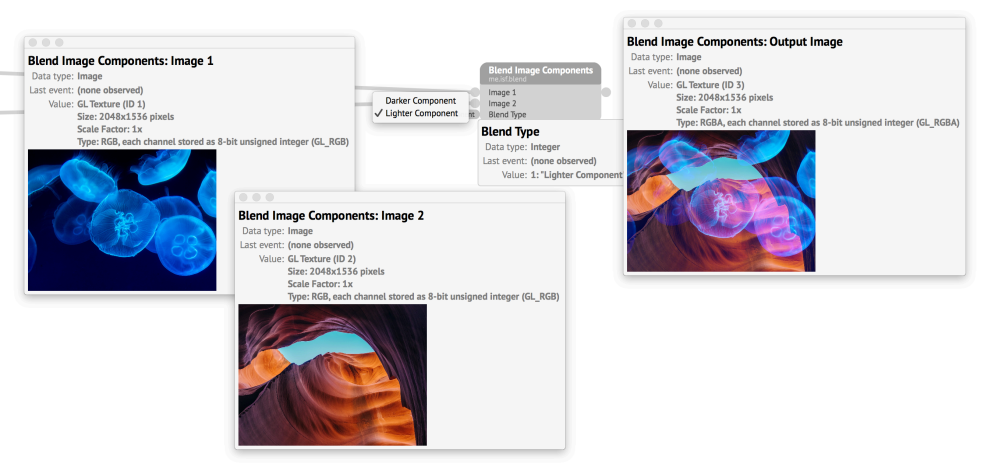


Listing 4: A menu input port.

```
/*{
   "LABEL":"Blend Image Components",
  "INPUTS":[
    {
      "NAME":"image1",
      "TYPE":"image"
    },
    {
      "NAME":"image2",
      "TYPE":"image"
```

```
    },
    {
      "NAME":"blendType",
      "TYPE":"long",
      "VALUES":[0, 1],
      "LABELS":["Darker Component", "Lighter Component"]
    }
  ]
}*/

void main()
{
  vec4 color1 = IMG_THIS_NORM_PIXEL(image1);
  vec4 color2 = IMG_THIS_NORM_PIXEL(image2);
  gl_FragColor = (1-blendType) * min(color1, color2) + blendType * max(color1, color2);
}
```



9.5 Supported ISF features

Vuo recognizes most fragment shaders that conform to the ISF 2.0 specification.

9.5.1 Functions

Vuo supports these ISF-specific functions:

Function	Description

`vec4 IMG_PIXEL(image, vec2)`	The color of a pixel in an image, using pixel-based coordinates.
`vec4 IMG_NORM_PIXEL(image, vec2)`	The color of a pixel in an image, using normalized coordinates.
`vec4 IMG_THIS_PIXEL(image)`	The color of the pixel that the fragment shader is currently executing on.
`vec4 IMG_THIS_NORM_PIXEL(image)`	The color of the pixel that the fragment shader is currently executing on.
`vec2 IMG_SIZE(image)`	The size of an image, in pixels.

Vuo also supports these Vuo-specific functions in ISF code:

Function	Description
`int LIST_LENGTH(list)`	The number of items in a list that was declared in `INPUTS`.

9.5.2 Uniforms

Vuo supports these ISF-specific uniforms:

Uniform	Description
`vec2 RENDERSIZE`	The size of the output image, in pixels.
`float TIME`	The time since the composition started, in seconds.
`float TIMEDELTA`	The time since the previous frame was rendered, in seconds. For the first frame, this is 0.
`vec2 isf_FragNormCoord`	The normalized coordinates of the pixel that the fragment shader is currently executing on.
`vec4 DATE`	The current date and time: year, month, day, and seconds since midnight.
`int FRAMEINDEX`	0 for the 1st frame, 1 for the 2nd frame, 2 for the 3rd frame, and so on.

9.5.3 Unsupported

Vuo does not currently support:

- multiple passes/buffers (`PASSES` key and `PASSINDEX` uniform)
- image file loading (`IMPORTED` key)
- audio input (`audio` and `audioFFT` data types)
- `IDENTITY` key

10 The Vuo editor

10.1 The Node Library

When you create a composition, your starting point is always the **Node Library** (View › Node Library › 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'll 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.

10.1.1 Docking and visibility

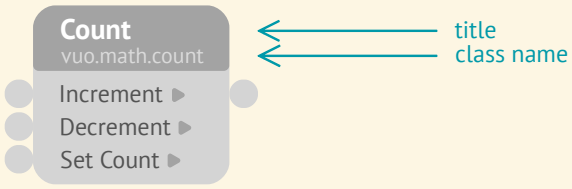
By default, the node library is docked within each open composition window. You can undock the node library by dragging or double-clicking its title bar. While undocked, only a single node library will be displayed no matter how many composition windows are open. You can re-dock the node library by double-clicking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

You can hide the node library by clicking the circle within its title bar, or by pressing ⎋ once (if the search bar is empty) or twice (first to clear the search bar, then to close the node library). Once hidden, you can re-display it by selecting View › Node Library › Show Node Library or using ⌘↵. The same command or shortcut, ⌘↵, will put your cursor in the node library's search box.

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

10.1.2 Node names and node display

Each node has two names: a title and a class name. The **node title** is a quick description of a node's function; it's the most prominent name written on a node. The **node 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.

The Editing node settings section explains how to change node titles.

10.1.3 Node Documentation Panel

The node library makes the complete set of Vuo core 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. In addition to the Vuo core nodes, if you have access to pro nodes, you'll see those displayed.

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

10.1.4 Searching for nodes

At the top of the Node Library is a search bar. When you type in a search phrase, matching nodes will show up in the Node Library. Pressing ⎋ while in the search bar will clear out your selection and show the entire Node Library, as will deleting your search term by pressing ⌫.

You can search by node title, node class name, and a variety of keywords:

Type of search	Example search phrase	Example search results
Node title	*radial blur*	**Blur Image Radially**
Node class name	*vuo.mouse*	**Receive Mouse Moves** (vuo.mouse.move2), **Receive Mouse Buttons** (vuo.mouse.button2), and others
Port name	*month*	**Get Date-Time Values**, **Make Date-Time**, and others
Keyword	*trigonometry*	**Calculate Sine**, **Calculate Cosine**, and others
Nodes with triggers	*trigger*, *fire*, or *bang*	Nodes that have one or more trigger ports
Input and output nodes	*i/o*	Nodes whose title begins with *Receive* or *Send*
Image filters	*filter*	Nodes that input one image and output a filtered image
Image generators	*generator*	Nodes that produce an image
Image transitions	*transition*	Nodes that input two images and output an in-between image
Pro nodes	*pro* or *premium*	Nodes that are only available with a Pro license
Subcompositions	*subcomposition* or *source:.vuo*	Nodes that are subcompositions
Community nodes	*source:.vuonode*, *source:.fs*, or *source:.c*	Nodes installed as `.vuonode`, `.fs`, or `.c` files

The search finds all nodes that match on the title *or* class name *or* keyword. For example, if you search

for *filter*, the results include **Filter Skeleton** (which has *filter* in the title and class name), **Allow First Event** (which has *filter* as a keyword), and **Adjust Image Filters** (which is an image filter). If you wanted to narrow the search to nodes like the last one, you could search for *image filter*.

If the search doesn't find a node that you expect to see:

- Make sure you're using a version of Vuo that has the node.
  - If the node was recently added, make sure you're using the latest version of Vuo.
  - If the node has been deprecated, instead use the node that is meant to replace it.
- If the node is a Pro node, make sure you've activated Vuo Pro.
- If the node is a subcomposition or a community node, review the steps for installing a node.

10.2 Working on the canvas

10.2.1 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 ↵ 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.

You can add certain commonly used nodes — such as **Share Value**, **Hold Value**, and **Allow Changes** — to your composition without a trip to the node library. Just right-click on the canvas and select an item from the Insert Node menu.

10.2.2 Drawing cables to create a composition

You can create a cable by dragging from a node's output port to a compatible input port or from a node's output port to a compatible input 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'll need to delete and add back the cable.

10.2.3 Adding a comment

You can add a comment to a composition by using the Edit > Insert Comment menu option, or by right-clicking on the canvas and selecting Insert Comment from the menu.

When you create a comment, you will be in editing mode, and can start typing the comment's text. You can press ⌥↵ to continue on a new line. To end editing, either press ↵ or click outside the comment area. To reenter editing mode, double click within the comment area or right-click on the comment and pick Edit... from the menu.

Text inside a comment can accommodate Markdown formatting. This is especially useful if you want to include a link, or make the text larger by using headings.

To change the color of a comment, right-click on the comment and pick Tint from the menu.

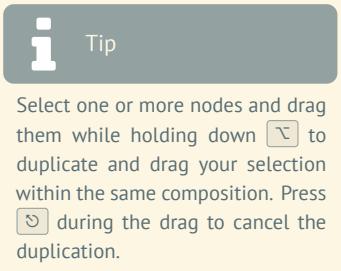
To move a comment, hover over the top edge of the comment and drag the handle that appears. To resize a comment, hover over the bottom-right corner of the comment and drag the handle that appears.

To select a comment, click on the comment's text, click near the top edge of the comment, or rubberband-select the top edge of the comment. (Clicking or rubberbanding the part of the comment below the text does not select the comment, so when you place nodes in that area you can easily rubberband-select the nodes.)

10.2.4 Copying and pasting nodes, cables, and comments

You can select one or more nodes or comments, 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.



10.2.5 Deleting nodes, cables, and comments

Delete one or more nodes, cables, and/or comments from your canvas by selecting them and either pressing ⌫ or right-clicking one of your selections and selecting Delete 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.

10.2.6 Rearranging nodes, cables, and comments

Tip

Hold down ⇧ while pressing an arrow key to move the nodes even faster.

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

10.2.7 Replacing nodes

Sometimes after adding a node to a composition, you might decide to replace it with something a little different. For example, you might like to replace a **Crop Image** node with a **Resize Image** node. If there are already cables hooked up to the original node, it can be inconvenient to disconnect each one and connect it to the new node. Fortunately, there's an alternative.

Right-click on the original node and go to the Change To menu. Here, you can select from a list of similar nodes that can be substituted in without breaking any cable connections.

10.2.8 Editing node settings

You can change a node's title (displayed at the top of the node) by double-clicking or hovering over the title and pressing ↵, then entering the new title in the node title editor that pops up. You may save or dismiss your changes by pressing ↵ or ⎋, respectively, just as you would using a port's input editor. You can also select one or more nodes from your canvas and press ↵ to edit the node titles for each of the selected nodes in sequence. If you delete the title and don't enter a new title, the node will default to its original title.

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.

10.2.9 Editing port values and settings

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 pressing ↵.) When the input editor is open, press ↵ to accept the new value or ⎋ 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.

10.2.10 Finding nodes on the canvas

Using Edit 〉 Find 〉 Find..., you can search the canvas for nodes matching a search phrase.

Type of search	Example search phrase	Example search results
Node title	*blur image rad*	**Blur Image Radially**
Node class name	*vuo.mouse*	**Receive Mouse Moves** (vuo.mouse.move2), **Receive Mouse Buttons** (vuo.mouse.button2), and others
Port name	*month*	**Get Date-Time Values**, **Make Date-Time**, and others
Subcompositions	*.vuo*	Nodes that are subcompositions
Community nodes	*.vuonode*, *.fs*, or *.c*	Nodes installed as .vuonode, .fs, or .c files
Deprecated nodes	*deprecated*	Nodes that are deprecated

10.2.11 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 > Actual Size / Zoom In / Zoom Out / Zoom to Fit menu options, or a pinch gesture on your trackpad, 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. You can also grab the workspace by holding down the spacebar while dragging.

Hold down ⇧ while pressing an arrow key to scroll even faster.

10.2.12 Publishing ports

A composition's published ports are displayed in sidebars, which you can show and hide using the menu View > 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 Delete 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 Port... . You can reorder published ports (except those that are part of a protocol) by dragging the name of a published port up or down in the sidebar.

For published ports with numerical data types (integers, real numbers, 2D points, 3D points, and 4D points), you can modify the behavior of their input editors by right-clicking on the published port in the sidebar and selecting Edit Details... . The Suggested Min and Suggested Max determine the range of values provided by the input editor's slider or spinbox (arrow buttons). The Suggested Step controls the amount by which each click on a spinbox button increments or decrements the value.

10.2.13 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 from Template, and select the protocol you want. Typically, a protocol is used when running a Vuo composition inside another application, such as a VJ or video postproduction app. 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.

10.3 Running a composition

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

10.3.1 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 from Template, 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.

10.3.2 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 ⌘ while left-clicking on the trigger port. If the trigger port carries data, it outputs its most recent data along with the event.

You can also fire an event directly into an input port (as if it had an incoming cable from an invisible trigger port). To do this, you can right-click on the input port and choose Fire Event, or you can hold down ⌘ and left-click on the input port.

If you've already manually fired an event, you can fire another event through the same port by going to Run > Re-fire Event. This fires an event through the trigger port or input port that most recently had an event manually fired.

10.3.3 Understanding and troubleshooting a running composition

Vuo has several helpful tools, such as the **Show Events mode** and **port popover**s to help you understand in more depth what is happening in your composition. These features can help you see exactly what events and data are flowing through your composition. For more information on troubleshooting steps see the section on Troubleshooting.

10.4 Editing composition information

You can enter metadata about a composition by going to Edit 〉 Composition Information... . The information you enter in this dialog can help people understand how to use your composition or products exported from it, as well as provide a reminder to yourself.

Some of the fields in this dialog — Name, Version, Description, Copyright, Homepage Link, and Documentation Link — are displayed in the Node Library's lower panel when the composition is open. If you export an app from the composition, these fields are shown in the app's About dialog.

The Keywords field is used when the composition is installed as a subcomposition. You can search the Node Library for the subcomposition using the keywords.

The Icon for Exported App, Bundle Identifier, and FxPlug Group fields are used in apps or plugins exported from the composition.

The License field is saved in the composition (.vuo) file. You can view it by opening the file in a text editor.

10.5 Working with subcompositions

With a subcomposition, you can use a composition as a node within other compositions. For more on what subcompositions are and why to use them, see Using subcompositions inside of other compositions.

10.5.1 Installing a subcomposition

To turn part of an existing composition into a subcomposition, select those nodes and cables within the composition, then go to the Edit 〉 Package as Subcomposition menu item.

To turn an entire composition into a subcomposition, create or open the composition and select the File 〉 Move to User Library menu item. (If your composition has not yet been saved, the menu item will read Save to User Library, and you'll be prompted to enter a title for your node.) The subcomposition node will immediately be listed and highlighted within your Node Library for use within other compositions.

To insert an empty subcomposition into a composition, select the Edit 〉 Insert Subcomposition menu item.

To install a subcomposition that you've downloaded, see Installing a node.

10.5.2 Editing a subcomposition

There are several ways to edit a subcomposition after it has already been installed:

- Right-click on the subcomposition node, either within the Node Library or on the canvas, and select the Edit Composition... context menu item.
- Double-click on the body of the node on the canvas.
- Select the node on the canvas and press ⌘↓.
- Click the "Edit Composition..." link in the node library documentation panel.

10.5.3 Uninstalling a subcomposition

To remove an installed subcomposition, right-click on the subcomposition node within the Node Library and select the Show in Finder context menu item. Locate the `.vuo` file matching the name of your subcomposition and remove it from the folder.

10.6 Keyboard Shortcuts

Vuo has keyboard shortcuts for working with your composition.

In the keyboard shortcuts below, these symbols represent keys in macOS:

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

10.6.1 Working with composition files

Shortcut	Definition
⌘N	New Composition
⌘O	Open Composition
⇧⌘O	Open the most recent composition
⌥⌘O	Open a random example composition
⌘↓	Open the composition contained in the selected subcomposition node
⌘S	Save Composition
⇧⌘S	Save Composition As
⌘W	Close Composition

10.6.2 Controlling the composition canvas

Shortcut	Definition

⌘↵	Show Node Library
⎋	Clear Node Library search box, then close Node Library
⌘=	Zoom In
⌘-	Zoom Out
⌘9	Zoom to Fit
⌘0	Actual Size
⇧ Double-click on comment	Zoom to fit the comment
Spacebar Drag	Move the canvas viewport
⌘1	Set canvas transparency to None
⌘2	Set canvas transparency to Slightly Transparent
⌘3	Set canvas transparency to Very Transparent
⌘4	Show or hide published ports

10.6.3 Creating and editing compositions

Shortcut	Definition
⌘A	Select all
⇧⌘A	Select none
⌘C	Copy
⌘V	Paste
⌘X	Cut
⌘Z	Undo
⇧⌘Z	Redo
⌘F	Find
⌘G	Find Next
⇧⌘G	Find Previous
⌫	Delete
⌘I	Composition Information
⌥ Drag near input port	Duplicate the cable connected to the input port.

⌥ Drag selected components	Duplicate the selected nodes, cables, and comments.
⌥ while rubberband selecting	Select all cables within the rubberband area, not just those connected to selected nodes.
⇧ while dragging cable	Change the data-and-event cable being dragged to event-only.
↑↓←→	Move the selected nodes, cables, and comments 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.
⌥↵	Open a Text input editor and press ⌥↵ to add a linebreak.

10.6.4 Creating and editing shaders

Shortcut	Definition
⌘5	Show or hide GLSL/ISF Quick Reference

10.6.5 Running compositions (when the Vuo editor is active)

Shortcut	Definition
⌘.	Stop
⌘R	Run
⇧⌘R	Restart
⌘ Click	Do this on an input port or a trigger port to manually fire an event.
⌘T	Re-fire Event

10.6.6 Running compositions (when the composition is active)

Shortcut	Definition

⌘Q	Stop the composition
⌘F	Toggle between windowed and fullscreen
⌥⌘E	Toggle recording the composition's graphical output to a movie file

10.6.7 Application shortcuts

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

11 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 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.

11.1 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
- Install the package file (double-click on it in Finder)
- Open the `/Library/Developer/Vuo` folder

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, run this command:

```
echo "export PATH=\$PATH:/Library/Developer/Vuo/framework" >> ~/.bash_profile
```

- Close and re-open the Terminal window

11.2 Getting help

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

11.3 Rendering a composition on the command line

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

Listing 5: Rendering a composition

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

`vuo-render` can output either raster (PNG) or vector (PDF or SVG) 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 6: Rendering a Vuo composition using Graphviz

```
1 dot -Grankdir=LR -Nshape=Mrecord -Nstyle=filled -Tpng -oRenderTextLayer.png RenderTextLayer.vuo
```

11.4 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 7: Compiling a Vuo composition

```
vuo-compile --output RenderTextLayer.bc RenderTextLayer.vuo
```

Then, turn the `.bc` file into an executable:

Listing 8: Linking a Vuo composition into an executable

```
vuo-link --output RenderTextLayer RenderTextLayer.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.

11.5 Running a composition on the command line

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

Listing 9: Running a Vuo composition

```
./RenderTextLayer
```

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

Listing 10: Running a Vuo composition

```
vuo-debug ./RenderTextLayer
```

11.6 Exporting a composition on the command line

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

Listing 11: Exporting a Vuo composition to a movie

```
vuo-export movie --output GenerateCheckerboardImage.mov GenerateCheckerboardImage.vuo
```

Listing 12: Exporting a Vuo composition to an application

```
vuo-export macos --output RenderTextLayer.app RenderTextLayer.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 > Mac App... menu item in the Vuo editor. See the section Exporting an application for more information.

12 Common patterns - “How do I...”

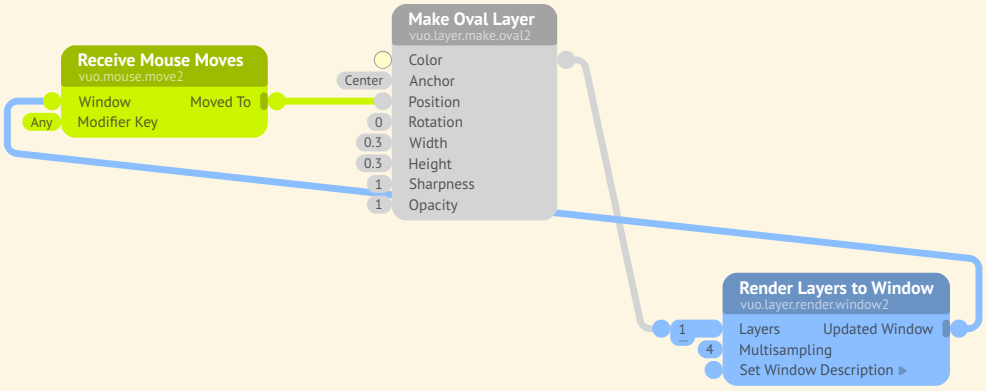
If you’re trying to figure out how to accomplish something in Vuo, one good starting point is the Node Library search bar. For example, if you want to make a random list of things, search the Node Library for “random” to find relevant nodes like **Make Random List** and **Shuffle List**. Another good starting point is the example compositions for each node set, found under File > Open Example.

Some problems you might want to solve with Vuo aren’t specific to one node or node set. Certain patterns come up again and again, whether you’re making compositions to display graphics, play audio, or anything else. This section covers these general patterns. Reviewing these patterns can help you create compositions more quickly and easily.

12.1 Do something in response to user input

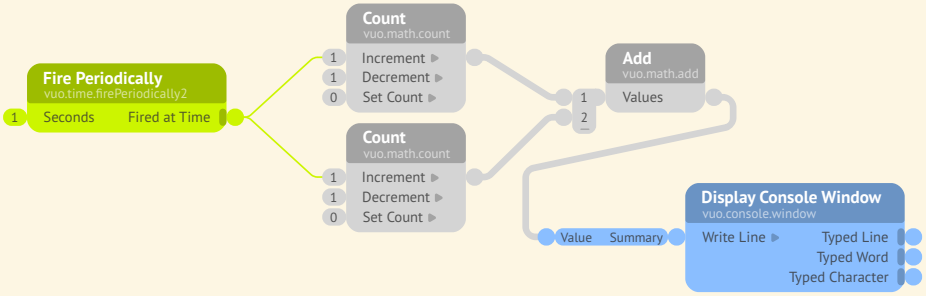
Since Vuo is event-driven, this is easy. Most nodes that get user input have a trigger port that fires an event each time new input comes in. To make something happen in response to that event, just connect a cable from the trigger port to the nodes that make it happen.

Here’s an example that makes a circle follow the mouse cursor as the user moves the mouse around.



12.2 Do something after something else is done

This is often quite easy, too, because of Vuo’s rules for event flow. If you want one node to execute before another, you can just draw a cable from the first node to the second node. In the composition below, for each event from **Fire Periodically**, the two **Count** nodes always finish executing before the **Add** node begins executing.



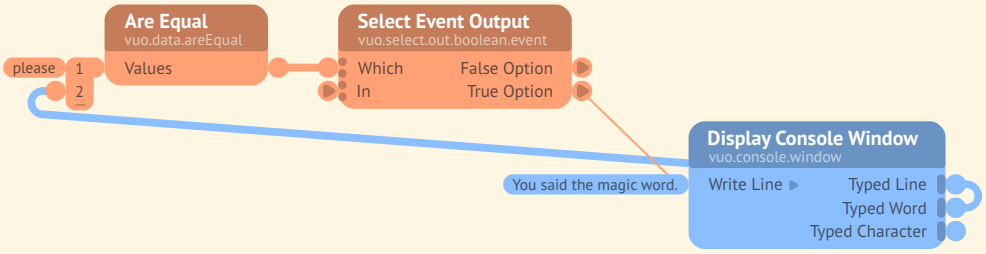
Sometimes you might need to enforce a "do something after something else is done" rule that's more complicated than putting nodes in a sequence, as above. For example, you might want a composition to do something only after the user has typed a certain word. The next section explains how to check for conditions like that and do something when they're fulfilled.

12.3 Do something if one or more conditions are met

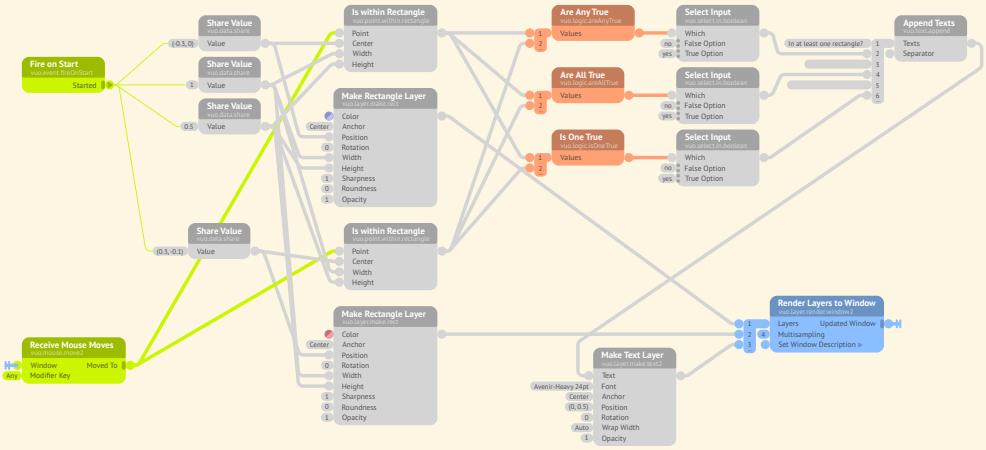
Vuo has a data type that represents whether a condition is met: the Boolean data type. If a node has a Boolean port, that port's value can be one of two things: *true* or *false*. *True* means "yes, the condition is met". *False* means "no, the condition is not met".

When checking if conditions are met, you'll often be working with nodes that have a Boolean output port. Many such nodes have a title that starts with "Is" or "Are", like **Is Greater than** and **Are Equal**.

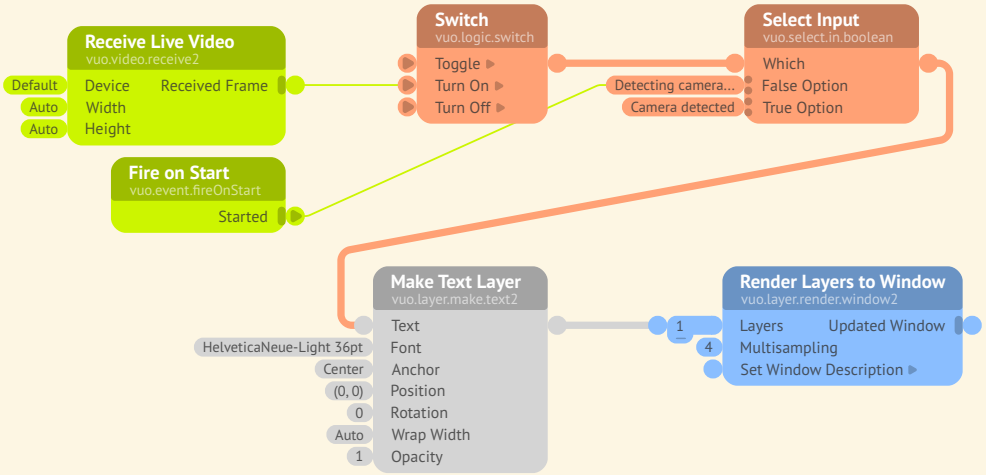
Here's an example that writes a message on the console window when the user types the word "please".



Below is an example (File > Open Example > Logic > Is Mouse Within Intersecting Rectangles) that checks two conditions: is the mouse cursor within the blue rectangle? is it within the red rectangle? The **Are Any True** node says yes (*true*) if the mouse is within at least one of the rectangles. The **Are All True** node says yes if the mouse is within both rectangles. The **Is One True** node says yes if the mouse is within one rectangle and not the other.



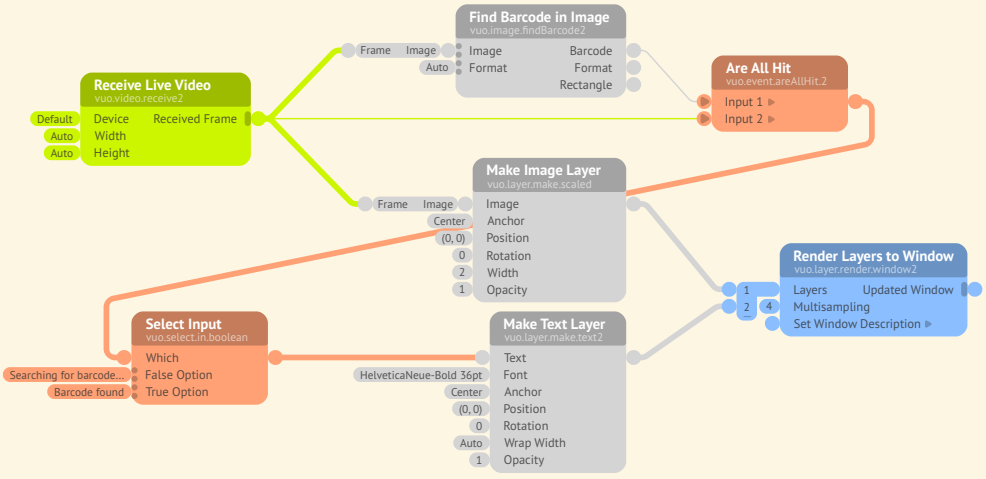
Here's one more example. It demonstrates how conditions can be used to coordinate between nodes downstream of different triggers. The composition displays the message "Camera detected" once it starts receiving input from the user's video camera, that is, once the **Receive Live Video** node's trigger port starts firing events. The events from that trigger port change the **Switch** node's output to *true*, indicating to the rest of the composition that "Camera detected" should be displayed.



12.4 Do something if an event is blocked

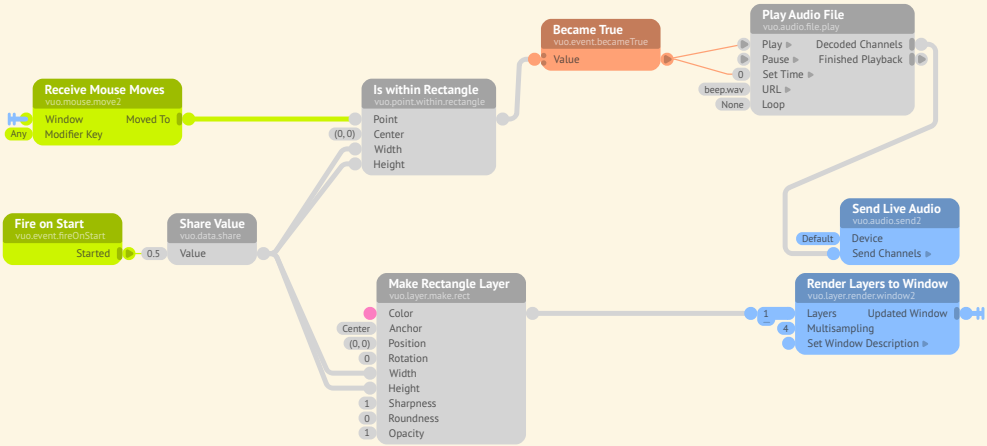
Nodes that have an event door on an input port can let some events through and block others. If you want to do something different depending on whether the event was let through or blocked, you can use an **Are All Hit** node.

Below is an example that checks if a barcode was found in an image. Since the **Find Barcode in Image** node blocks events when no barcode is found, the **Are All Hit** node is used to check whether the event was blocked. **Are All Hit** outputs *false* if **Find Barcode in Image** blocks the event and *true* otherwise.

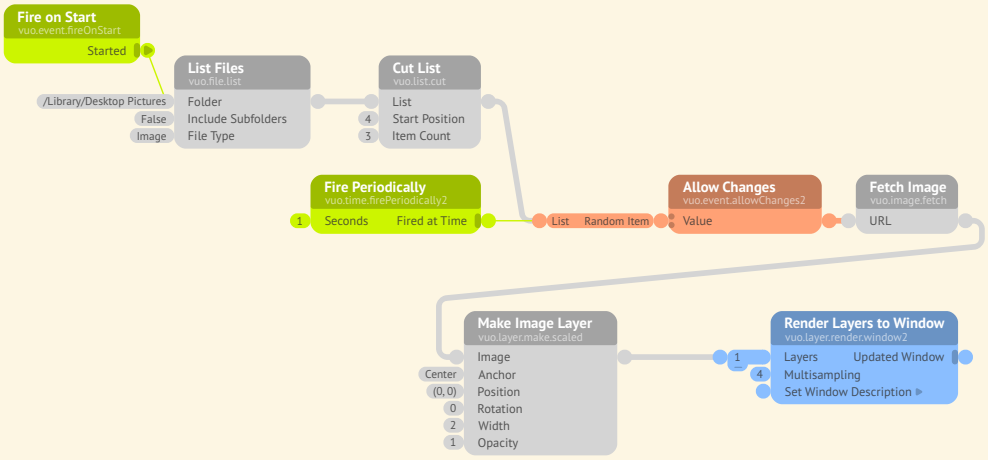


12.5 Do something if data has changed

Several nodes check if data has changed in a certain way and only let an event through if it has: **Changed**, **Increased**, **Decreased**, **Became True**, and **Became False**. In the composition below, the **Became True** node outputs an event each time the output of **Is Within Rectangle** changes from *false* to *true*, emitting a sound effect each time the mouse cursor enters the square.



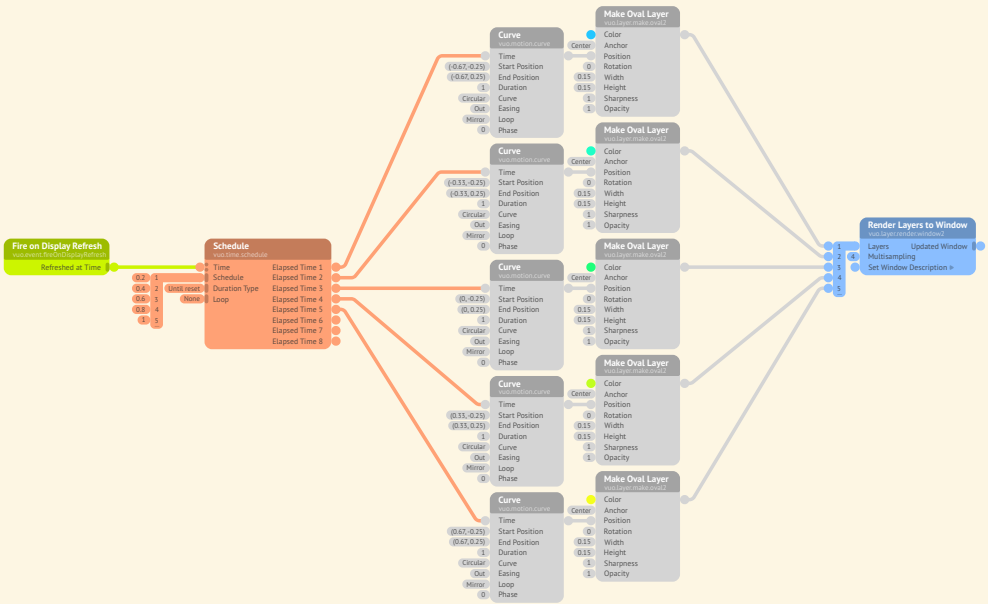
Like **Became False** and the other nodes just described, the **Allow Changes** node only lets an event through if the data has changed. But **Allow Changes** is different because it passes the data through along with the event. This can be useful when your composition does something time-consuming or processor-intensive with the data, and only needs to do that work when the data changes. For example, this composition periodically picks a large image file to load, but avoids reloading the same image file if it’s picked twice in a row.



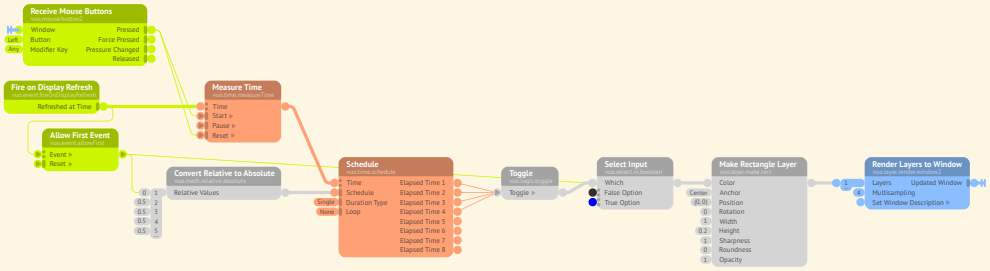
12.6 Do something after an amount of time has elapsed

Sometimes, you may want a composition to do something immediately in response to an event. Other times, you may want it to wait until a certain amount of time has passed to do something — for example, launch an animation, start a video, or display a message.

This composition (File > Open Example > Time > Animate On Schedule) launches a series of animations. At 0.2, 0.4, 0.6, 0.8, and 1 second after the composition starts, it sets in motion the next in a series of circles. The bouncing movements of the circles are staggered because each **Elapsed Time** port of the **Schedule** node outputs a time that's 0.2 seconds after the previous **Elapsed Time** port's value.



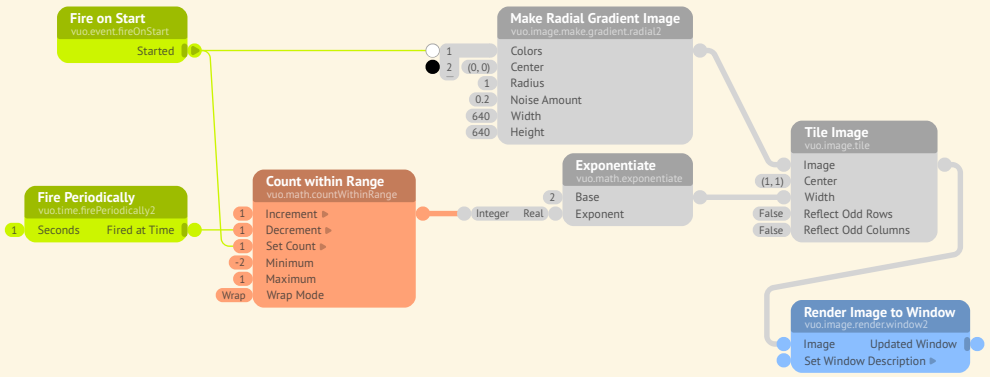
Instead of scheduling things relative to the start of the composition, the composition below (File > Open Example > Time > Flash On Mouse Press) schedules things relative to the most recent mouse press. When the mouse is pressed, the rectangle's color changes to blue, then gray, then blue, then gray. Why does the **Schedule** node in this composition schedule things relative to the most recent mouse press, instead of relative to when the composition started, as in the previous example? Because the **Schedule** node's **Time** input port gets its data from the **Measure Time** node, which outputs the time elapsed since the mouse press.



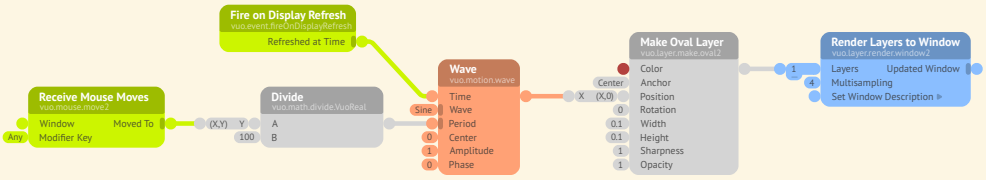
12.7 Do something repeatedly over time

If you want a composition to do something every N seconds, there are several nodes that fire events at a steady rate. The **Refreshed at Time** trigger port the **Fire on Display Refresh** node fires every time the computer display refreshes, which is usually about 60 times per second. For a faster or slower rate, you can use the **Fire Periodically** node.

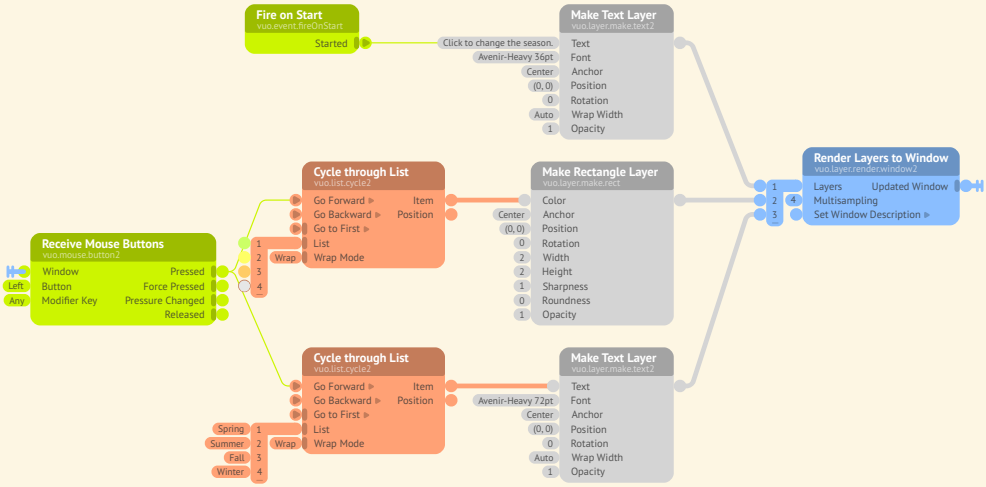
The composition below uses a **Fire Periodically** node to change the width and number of tiled copies of an image every 3 seconds. This composition actually has two kinds of repetition over time. One is the change in tile width that occurs every 3 seconds because of the **Fire Periodically** node. The other is that the tile width repeats itself every 12 seconds. It goes from 2, to 1, to 0.5, to 0.25, and then back to 2. This wrapping-around of the tile width is done by the **Count within Range** node.



Count within Range is one of many ways to cycle through a series of numbers. Another is the **Curve** node when its **Loop** port is set to *Loop* or *Mirror*. And another is the **Wave** node. The composition below (File > Open Example > Motion > Wave Circle) uses the **Wave** node to make a circle move back and forth.



If you want to cycle through a series of things other than numbers, you can use **Cycle through List**. Here's an example (File > Open Example > List > Cycle Seasons) that uses **Cycle through List** nodes to cycle through colors and texts, displaying the next one each time the mouse is pressed.



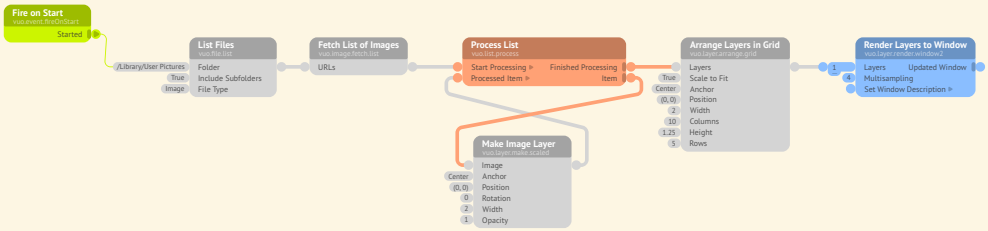
12.8 Do something to each item in a list

The previous section showed how to do something with each list item in turn, using a **Cycle through List** node. For each event the **Cycle through List** node receives, it outputs one list item. If instead you want an event to do something to all list items, you can use the **Process List** node.

Note for text programmers

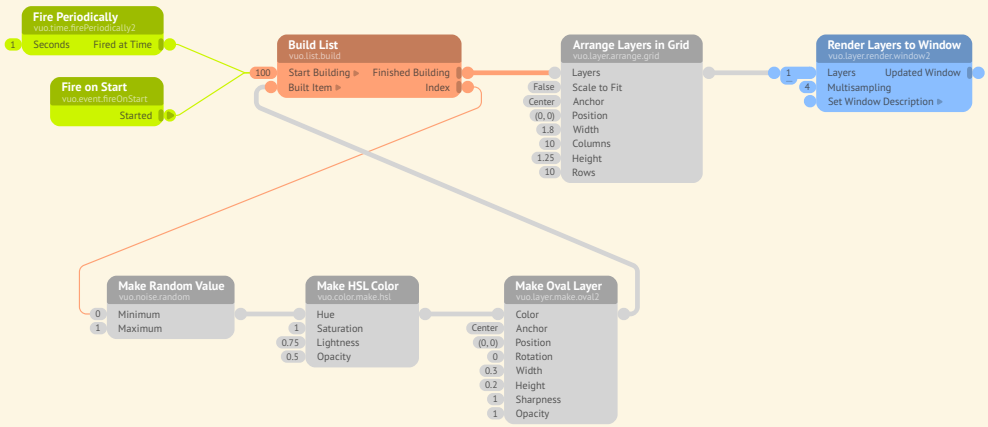
Process List and **Build List** are Vuo's general-purpose nodes for iteration. They're similar to text programming constructs such as loop control structures and foreach and apply functions.

Here's an example (File > Open Example > List > Display Grid Of Images) that turns a list of images into a list of layers using **Process List**. When **Process List** gets an event and list of images into its **Start Processing** port, it rapidly fires a series of events through its **Item** port, one event for each image in the list. The image and event go through the **Make Image Layer** node, and the created layer and event go into the **Process List** node's **Processed Item** port. Once that port has received as many events as **Item** fired, the **Finished Processing** port fires an event with the accumulated list of created layers.



12.9 Create a list of things

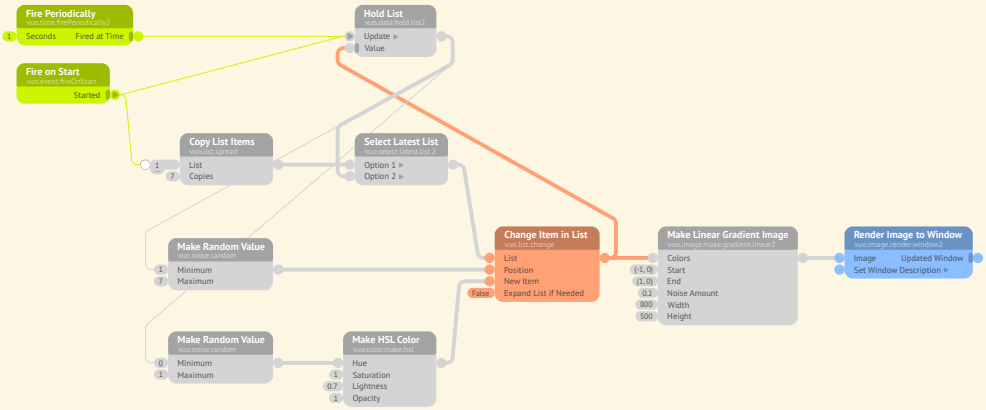
If you don't have a list to start with, one way to create one is with the **Build List** node. **Build List** looks a lot like **Process List**. The difference is that the **Build List** node's **Start Building** port inputs an integer (the number of list items to create) instead of a list, and the **Index** port rapidly fires a series of integers (from 1 to the number of list items) instead of input list items. Here's an example (File > Open Example > List > Display Rainbow Ovals) that uses the **Build List** node to display a grid of 100 different-colored ovals.



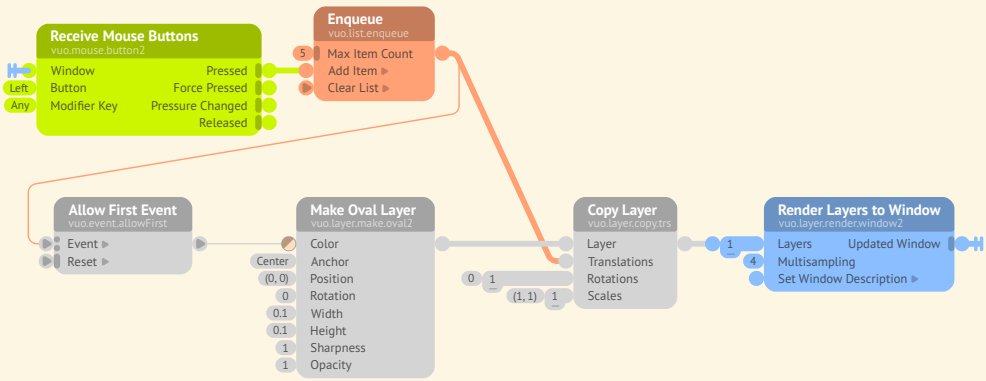
Build List and **Process List** are general-purpose tools. Vuo also provides some simpler, more specialized ways to create certain types of lists. These include **Make Random List** to make a list of random numbers or points, **Copy Layer** and **Copy Scene** to duplicate a 2D or 3D object, and **Enqueue**, which is explained in the next section.

12.10 Maintain a list of things

Sometimes you may want not only to create a list, but also to hold onto it and make changes to it over time. One way to do that is with a feedback loop, as in the example composition below (File > Open Example > List > Replace Colors In Gradient). It maintains a list of colors, randomly changing one of them every 1 second.



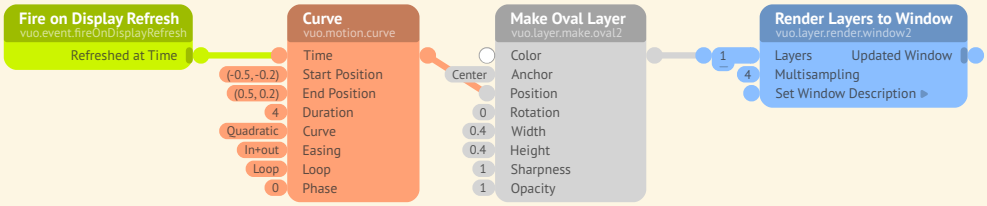
Another way you might want to maintain a list is to accumulate a queue of items over time, using the **Enqueue** node. A queue in this node is like a queue of people waiting in line. It's first-in-first-out, meaning that new items get added to the end of the line, and the item that's been waiting in line the longest is the next one that can leave the queue. Here's an example that uses **Enqueue** to remember the positions of the 5 most recent mouse presses.



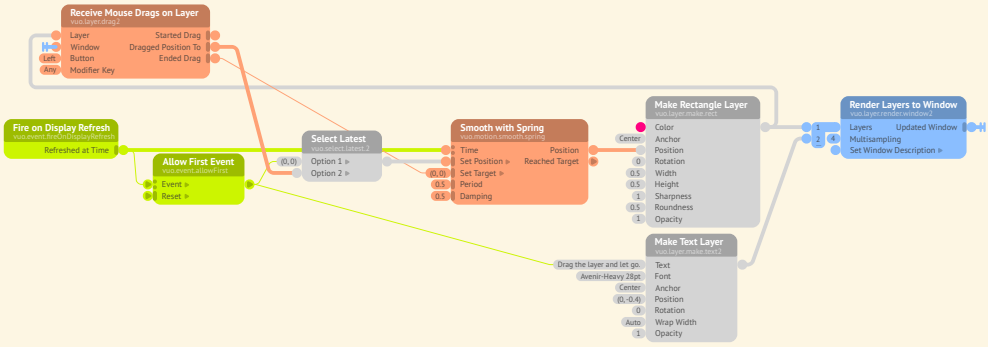
12.11 Gradually change from one number/point to another

Earlier, under "Do something repeatedly over time", the **Curve** and **Wave** nodes were mentioned as ways to cycle through a series of numbers or points. You can also think of these nodes as ways to

gradually change from one number or point to another. Here's an example that uses a **Curve** node to gradually move a circle from one point to another. Since the **Curve** port is set to *Quadratic* and the **Easing** port is set to *In + Out*, the circle starts moving slowly, picks up speed, and then slows down as it reaches its destination.



Another way to gradually change from one number or point to another is with the "Smooth" nodes — **Smooth with Duration**, **Smooth with Inertia**, **Smooth with Rate**, and **Smooth with Spring**. Here's an example (File > Open Example > Motion > Spring Back) that makes a square spring back to the center of the window when the user drags and releases it.

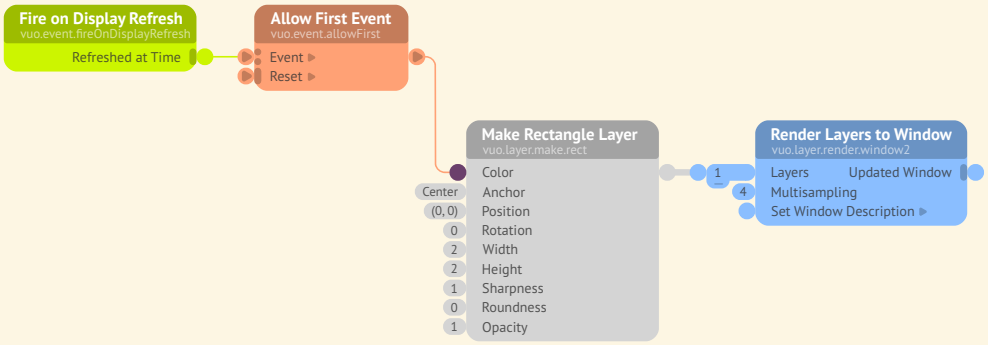


12.12 Set up a port's data when the composition starts

When a composition starts running, its data-and-event input ports start out with some initial data — either the port's constant value, if you've used the input editor to set one for the port, or the port's default value. An input port with an incoming data-and-event cable stays at its default value until the first data-and-event comes in through the cable. Sometimes you may want to send certain data with that first event so that the port will start off with the right value.

A simple way to do that is with a **Fire on Start** node. In the **Smooth with Spring** example in the previous section, the **Fire on Start** node fires an event that sets up the data for two input ports. One is the **Align Layer to Window** node's **Layer** input port, which gets the layer created by **Make Text Layer**. The other is the **Smooth with Spring** node's **Set Position** input port, which gets initialized to (0,0). The **Select Latest** node helps out here by sending (0,0) to the **Set Position** port for the **Fire on Start** event and, after that, the current mouse position each time the **Receive Mouse Drags on Layer** fires an event.

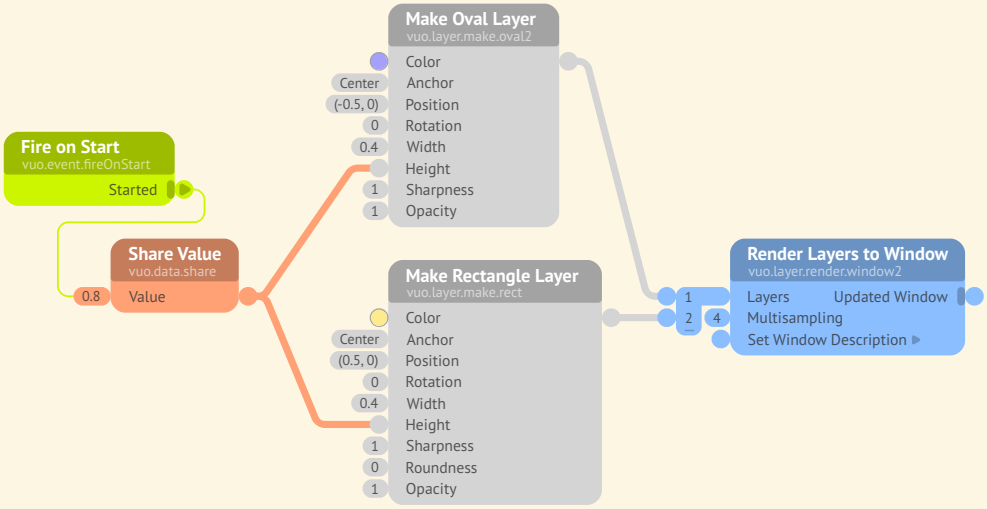
Using **Fire on Start** to set up data is pretty simple, but it has one weakness: the **Fire on Start** node's trigger isn't coordinated with other triggers in the composition. If you're trying to use **Fire on Start** together with the **Refreshed at Time** port of **Fire on Display Refresh**, you might see a momentary flicker or adjustment in graphics as the composition starts. That's because the **Fire on Start** event and the first **Refreshed at Time** event are setting up different parts of the graphics at slightly different times. So how can you avoid the flicker? Instead of **Fire on Start**, which fires its own event, use **Allow First Event**, which can borrow the event fired from **Refreshed at Time**. Here's an example.



12.13 Send the same data to multiple input ports

If you have several input ports in your composition that all need to stay in sync with the same data, then it's usually a good idea to feed cables to all of them from a single output port. But what if the data isn't coming from an output port — what if it's a constant value? In that case, you can use a **Share Value** node to set the constant value in one place and propagate it from the **Share Value** node's output port to all connected input ports.

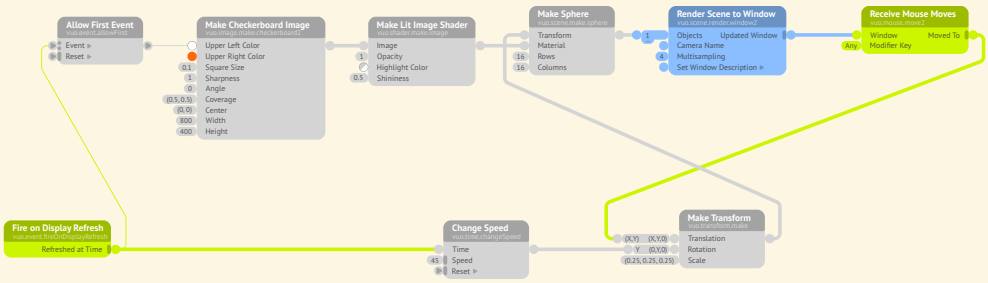
Here's an example that draws two shapes, all of the same height. You could accomplish the same thing without the **Share Value** node by using input editors to individually set the **Height** input ports to 0.8. The advantage of using **Share Value** is that, if you change your mind and decide the height should be 1.0 instead, you only have to edit it on the **Share Value** node's input port instead of on all connected input ports.



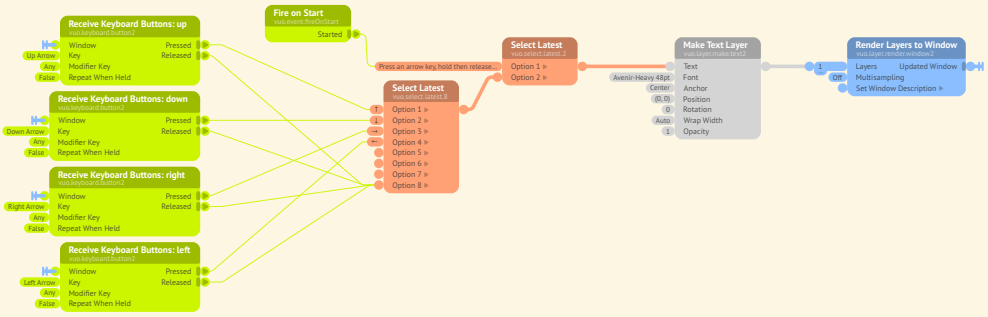
12.14 Merge data/events from multiple triggers

When you have streams of events from multiple triggers flowing through your composition, usually those streams of events have to merge somewhere in the composition.

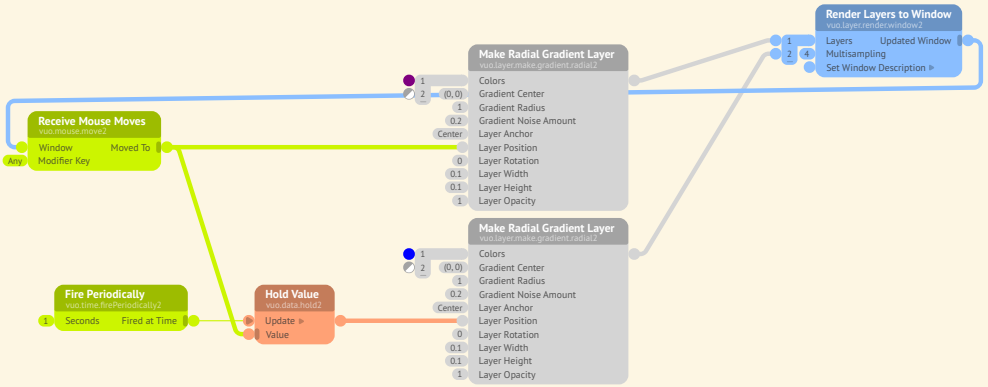
Sometimes the streams of events just naturally overlap, as in the example below (File > Open Example > Scene > Move Spinning Sphere). The events fired from the **Refreshed at Time** port on **Fire on Display Refresh** and the events fired from the **Moved To** port on **Receive Mouse Moves** both travel through the **Make Transform** and **Make Sphere** nodes to the **Render Scene to Window Node**.



Other times, you may want to merge the event streams more intentionally. Here's an example (File > Open Example > Select > Show Arrow Presses) that takes input from key presses on different arrow keys, and displays a message for each one. The **Select Latest** node lets the events from each arrow key through.



Here's an example that shows a different way of merging two event streams. This composition (File > Open Example > Data > Store Mouse Position) draws two gradients that each follow the mouse cursor a bit differently. The purple (upper) gradient stays with the mouse all the time. The violet (lower) gradient only updates every 1 second. For the lower gradient, the event streams from **Receive Mouse Moves** and **Fire Periodically** merge at the **Hold Value** node. Unlike the composition in the previous example, which let both event streams through, this composition lets one event stream through and blocks the other. However, the data left by the blocked event stream (from **Receive Mouse Moves**) gets picked up and carried along downstream by the other event stream (from **Fire Periodically**).



Note for text programmers

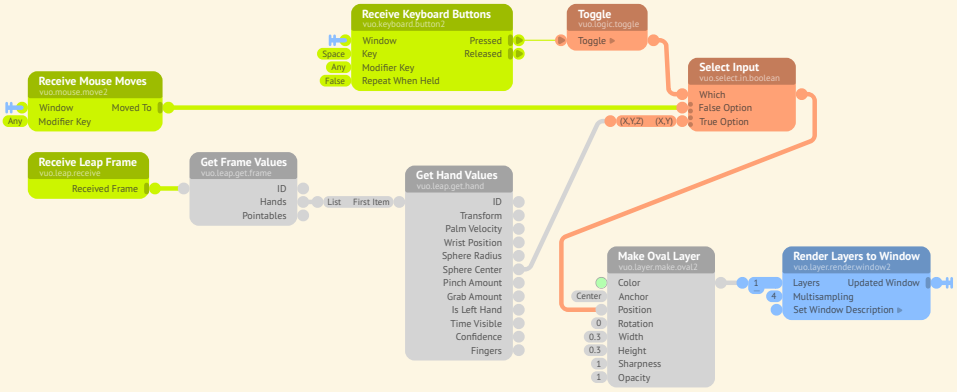
Vuo's **Select Input** and **Select Output** are similar to `if/else` or `switch/case` statements.

12.15 Route data/events through the composition

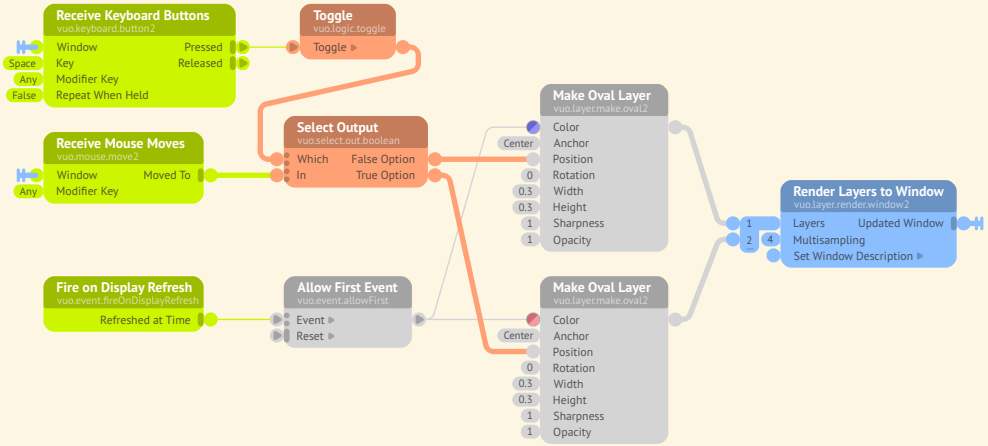
In the last example in the previous section, events from the **Receive Mouse Moves** node's trigger were always blocked at the **Hold Value** node, and events from the **Fire Periodically** node's trigger were always allowed through. Instead of always blocking one trigger's events and always letting another trigger's events through, what if you want to switch between the event streams?

Here's an example with a keyboard control that switches the data-and-event stream that controls a circle's position. When the user presses the space bar, setting the **Select Input** node's **Which** port to *true*, the circle is controlled by the Leap Motion device. When the user presses the space bar again,

setting the **Which** port to *false*, the circle is controlled by the mouse. Whichever data-and-event stream is *not* controlling the circle at a given time is blocked at the **Select Input** node.



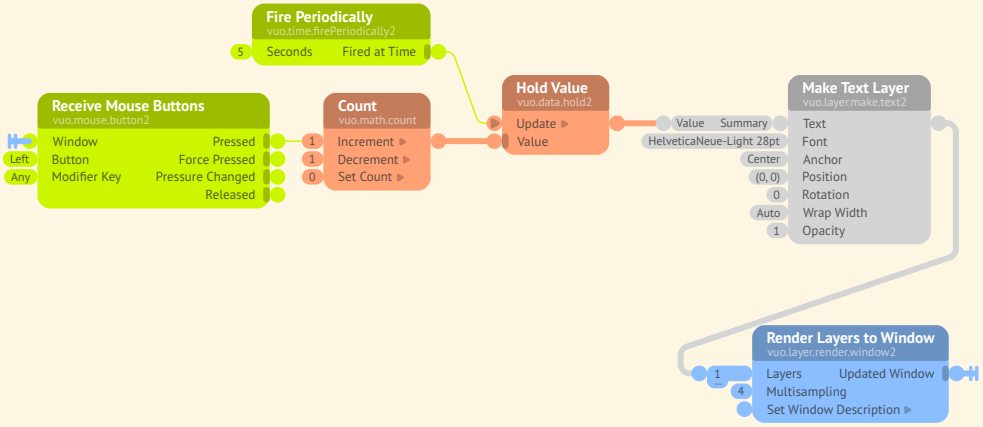
Instead of taking multiple event streams and picking one to let through, as in the previous example, what if you have a single event stream and want to pick one of several downstream paths to route it to? Below is an example of that. The space bar toggles between two circles. Whichever circle is chosen at a given time is controlled by the mouse. This works because the **Select Output** node routes the data-and-event stream from **Receive Mouse Moves** through just one of its output ports at a time.



12.16 Reuse the output of a node without re-executing the node

Some nodes change their output every time they're executed. The **Count** node is an example. If you feed an event into any of its input ports — **Increment**, **Decrement**, or **Set Count** — the node outputs a count that's different from the previous count (except of course in special cases, like doing **Set Count** when the node is already at that count). What if you don't want to change the count, and you just want to output the current count?

Here's an example that increments a count each time the user presses a mouse button, and displays the current count in a window every 5 seconds. (This same pattern could be applied to practical situations, such as a sensor incrementing a count each time a person passes through a doorway and the count periodically being sent over a network to monitor the building's occupancy.)

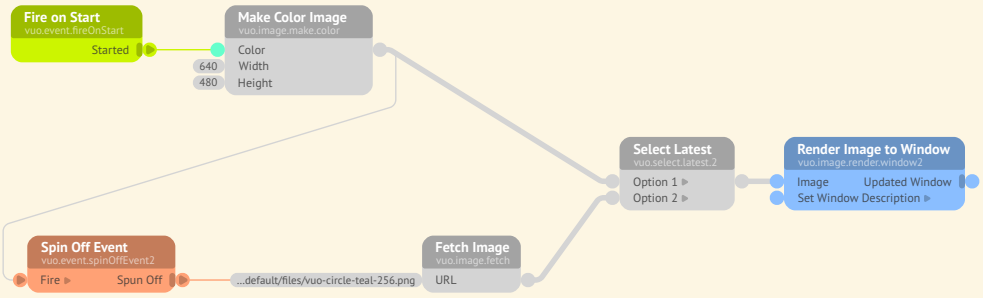


The key to this composition is the **Hold Value** node. Each time the **Count** node outputs a value, the **Hold Value** node holds on to it (in other words, stores it). Every 5 seconds, an event from **Fire Periodically** hits the **Hold Value** node's **Update** port, and the **Hold Value** node outputs the count that it's storing.

12.17 Run slow parts of the composition in the background

Different parts of the composition can be executing simultaneously. If you have multiple triggers firing events through the composition, events from both triggers can be traveling through the composition at the same time. This fact comes in handy if you want a composition to start working on a slow task and do something quicker in the meantime.

Here's an example (File > Open Example > Event > Load Image Asynchronously). The slow task, in this case, is to download an image from the internet. Immediately after this composition starts running, it starts downloading the image and, in the meantime, fills the window with a solid color. The **Spin Off Event** node is what allows the download to happen in the background. If **Spin Off Event** weren't there, then the **Select Latest** node would wait for both **Make Color Image** and **Fetch Image** to complete before it executed. But, thanks to **Spin Off Event**, the **Fetch Image** node is now executed by a different event than the **Make Color Image** node, so **Select Latest** can go ahead and execute as soon as **Make Color Image** is complete.


Fire on Start
vuo.event.fireOnStart
Started
Make Color Image
vuo.image.make.color
Color
640
Width
480
Height
Select Latest
vuo.select.latest.2
Option 1
Option 2
Render Image to Window
vuo.image.render.window2
Image
Updated Window
Set Window Description
Spin Off Event
vuo.event.spinOffEvent2
Fire
Spun Off
...default/files/vuo-circle-teal-256.png
Fetch Image
vuo.image.fetch
URL

13 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.

13.1 Helpful tools

Vuo provides several helpful tools for troubleshooting:

- **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 popover**s 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 description**s 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.

When watching data with the **Display Console Window** node, you can use the **Allow Changes** node to filter out repeated data.

There are nodes to help with troubleshooting, too. One is **Display Console Window**, which shows a text window that your composition can write text on. You can use **Display Console Window** to observe values that are hard to see in port popovers because they're changing too rapidly. To find other nodes that can help with troubleshooting, search the Node Library for "debug" or "troubleshoot".

13.2 Common problems

13.2.1 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?

Using the tools provided by Vuo, 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.

13.2.2 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 isn't executing. 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 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.

13.2.3 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 - "How do I...".

13.2.4 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.

13.2.5 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 **Fetch Image** node that downloads an image from the internet. Change your composition so those nodes receive fewer events. (See the section Common patterns - "How do I…".)
  - 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 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 - "How do I…".)
  - Export the composition to an application. When run as an application instead of in the 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.

13.2.6 Vuo slows down when my computer heats up.

Some Mac systems, including recent MacBook Pros, aren't designed to adequately dissipate the heat they generate when under heavy load, so macOS drastically slows down the system in order to generate less heat. This is called *thermal throttling*. This behavior may affect the performance of your Vuo compositions.

You can monitor macOS's thermal throttling by opening Terminal.app and running this command: `pmset -g thermlog`. It will automatically update when the status changes. When the `CPU_Speed_Limit` value is less than 100, thermal throttling is active.

To mitigate this, consider trying some of the following options:

- Reset your Mac's SMC to recalibrate its thermal management profile, which may change when macOS decides to apply thermal throttling.
- Improve your Mac's heat dissipation:
  - Ensure your Mac is in a cool room and placed on a cool surface out of direct sunlight.

  - Ensure your Mac's air inlets and egresses are unobstructed.
  - Hire a qualified technician to disassemble your Mac and clean the dust from its fans.
  - Use an external fan to draw hot air away from your Mac.
  - Install 3rd-party software to increase your Mac's internal fan speeds.
- Reduce the CPU and GPU usage of your composition:
  - Reduce the image and mesh resolutions.
  - Reduce the framerate.
  - Reduce the number or quality of image filters.

13.2.7 Various compositions won't run

If compositions fail to start, the problem could be that you have a node installed that for some reason prevents Vuo from running compositions. The node might be outdated or broken, or it might trigger a latent bug in Vuo. In any case, you can check if this is the problem by uninstalling all nodes, including subcompositions. See Installing a node.

If uninstalling all nodes enables your compositions to run again, the next step is to figure out which node was the problem. You can do this efficiently by first reinstalling half of your nodes. If the problem returns, then the problematic node must be in that half, so uninstall half of them and see what happens. Otherwise, reinstall half of the remaining nodes. Continue working by halves until you've narrowed the problem down to one node. If you downloaded the node from another Vuo community member, you can contact that person for help. If the problem is a node or subcomposition that you created, you can contact Vuo Support.

13.3 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 **Fire on Display Refresh** node's **Refreshed at Time** trigger port, use a **Fire Periodically** node's **Fired at Time** 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 (in Finder, 🗀`Applications > Utilities > Console.app`) while running your composition. Some nodes send console messages when they have problems.
- Try rearranging your nodes and cables so you can see the flow of events more clearly. If your nodes and cables are nicely laid out, then it can be easier to spot problems.
- Don't hesitate 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.
- You're welcome to ask questions. For help, go to Vuo Support.

14 Contributors

Vuo is built and maintained by Team Vuo and the Vuo Community. Everyone is encouraged to contribute toward improving Vuo.

14.1 Contributors

Below is an alphabetical list of the people who have contributed to bringing Vuo to fruition.

- .lov.
- 2bitpunk
- 3lab_tv
- AdrienM1
- ajm
- akashaslc
- alexmitchellmus
- Anthony
- architek1
- ariam
- atompowered
- automatone
- a_o
- baksej
- balam
- Benedikt
- bLackburst
- bmellen
- Bodysoulspirit
- Bonemap
- botnotbot
- carlitos
- casdekker
- conanp
- cremaschi
- cwilms-loyalist
- cwright
- cymaspace

- David
- ddelcourt
- destroythings
- DetAndreTeatret
- DiGiTaLFX
- Doro
- dumski
- e.duchemin
- eganpc
- ellington
- emervark
- errol
- eseftel
- Eurotrash
- franz
- fRED
- gabe
- gautjac
- George_Toledo
- Gillieron
- iason
- Illuminator
- inadvisable
- inx
- jayeazy
- Jérôme Lanon
- jersmi
- jinyaolin
- jmcc
- Jobok31
- joeladria
- Joëlle
- johnnykuo
- jokkeheikkila
- jstrecker
- jte2384
- jungbas
- jvolker
- Kewl

- khenkel
- kingluma
- Kino
- kokot
- kozistan
- krezrock
- Landscaper
- lechbialek
- lhepner
- lipoqil
- Luiz Andre
- manuel_mitasch
- marcorexo1
- marioepsley
- MartinusMagneson
- mattgolsen
- meno
- micpool
- microlomaniac
- miramon9
- mixfilet
- mkegan
- mnstri
- mradcliffe
- mrray
- mutable
- p8guitar
- pbourke
- Pianomatic
- prackvj
- pyramus
- raphael
- rbetin
- richardbyers
- rmercuri
- robaiello
- ryandmonk
- sala28
- Salvo

- savienojums
- sboas
- Scratchpole
- seanradio
- shakinda
- Sigve
- SimHacker
- sinemod
- sinsynplus
- smokris
- Steamboy
- steinboy
- stromqvist
- synnack
- Taco Circus
- teaportation
- tfrank
- timwessman
- tivonrice
- tmoles
- tobyspark
- unfenswinger
- unicode
- useful design
- video piglet
- visiophone
- vjsatoshi
- volkerku
- WARP
- wmackwood
- Xavier dev
- xoanxil
- zwei-p
- zzkj

Thanks to our contributors!

14.2 Software Vuo uses

- Apple Csu
- Apple dyld
- Apple ld64
- BeatDetektor
- Clang
- Conan
- Discount
- DocBook
- Doxygen
- FFmpeg
- FreeImage
- Gamma
- Ghostscript
- Graphviz
- Hap
- JSON-C
- LLVM
- LaTeX
- Leap Motion
- NewTek NDI®
- Open Asset Import
- OpenSSL
- Pandoc
- Qt
- RtAudio
- RtMidi
- Snappy
- Squish
- Syphon
- YCoCg-DXT
- ZXing
- csgjs-cpp
- glib
- http-parser
- libcsv
- libcurl
- libfacedetection

- libffi
- libfreenect
- libintl (gettext)
- libjpeg-turbo
- liblqr
- libusb
- libxml2
- muParser
- nginx
- oscpack
- pngquant
- zlib
- ØMQ

14.3 Resources Vuo uses

- PT Sans

Glossary

cable A line connecting nodes; the conduit that data and events travel through 36

composition A document you create in Vuo 31

Composition-Local Library A folder containing nodes that are available only to compositions located next to the folder 80

constant value Data in an input port that doesn't have a connected data-and-event cable. 38, 68

coordinate system A way to represent a position in 2D or 3D using numbers 74

data A piece of information 35

data type The format of a piece of information, such as numeric or textual 57

data-and-event cable A cable that carries both events and data 36

deadlocked feedback loop A **feedback loop** where it's impossible for an event to travel through all the cables leading up to a node before reaching the node itself 55

deprecated Obsolete or outdated 73

dictionary A set of data items that can be looked up by name 60

downstream Nodes that execute after other nodes 37

drawer An attachment to a port that lets you input each item of a list or dictionary separately 70

drop 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) 56

enqueue events The trigger port will keep firing events regardless of whether the downstream nodes can keep up 56

event Controls when nodes do their job and how information flows between nodes 33

event door May or may not allow an event to go out any of the node's output ports (exact behavior depends on the node, and is explained in the node's documentation) 38, 44

event throttling Controls whether a trigger port will **enqueue events** or **drop events** 55

event wall Prevents an event from going out any of the node's output ports 38, 44

event-only cable A cable that carries only events, not data 36

execute Perform a specific job 33

feedback loop A group of nodes connected by cables forming a loop, causing the group's latest output to be affected by the group's prior output 49

fire Originate an event 33

generic data type a stand-in used when a port has a changeable data type and the data type hasn't been decided yet 61

Image Filter A **protocol** for altering an image 95

Image Generator A **protocol** for creating an image 96

Image Transition A **protocol** for transitioning from one image to another 97

infinite feedback loop A **feedback loop** that a single event could loop through repeatedly without being blocked by an **event wall**. 54

input editor A widget for setting the value of an input port 68

input port Receives information into a node 37, 43

list A sequence of data items 60

node A building block that performs a specific job 31

node class name A categorical name that reveals specific information about a node, shown directly below the node's title 122

node description Tells you how a node is supposed to work; appears in the **Node Documentation Panel** whenever you select the node in the Node Library or on the canvas 156

Node Documentation Panel The lower section of the **Node Library**, which describes the general purpose of the node as well as details that will help you use it 123

Node Library The panel or floating window in Vuo's user interface that lets you explore and use Vuo's **node**s 122

node title A quick description of a node's function, shown at the top of a node 122

output port Sends information out of a node 38, 45

port action A port that causes the node to do something different when it receives an event than it does when any other input port receives an event 45

port popover A panel that shows a port's current value, shown when you click on a port 130, 156

Pro node A node that is only available in Vuo Pro 73

protocol A predetermined set of published ports with certain names and data types 95

published port Receives or sends data outside the composition 40

Scale Factor The ratio of pixels per linear point. For example, Scale Factor 2 means there are 2 pixels per linear point, or 4 pixels per square point. 106

Show Events mode Lets you watch the events flow through your composition 130, 156

subcomposition A composition that can be used as a node inside of other compositions 81

System Library folder A folder containing nodes that are available to any composition opened by any user logged into the computer 79

trigger port A port that **fire**s events 33, 42

type-converter node A node that translates data from one type to another 58

upstream Nodes that execute before other nodes 37

User Library folder A folder containing nodes that are available to any composition opened by the user currently logged into the computer 79

Vuo Coordinates Vuo's specific **coordinate system**, where the center of the rendering area is represented by (0,0) for 2D graphics or (0,0,0) for 3D graphics 74

yank zone The section of the cable with the extra-bright highlighting when hovering over it, which lets you drag the cable away from an input port to which it is currently connected 126