Getting Started

So you want to get serious with vvvv? You've come to the right place!

Depending on where you come from you may at some point want to read one of the following guides:

- Introduction for Creative Coders
- Introduction for .NET programmers
- Introduction for vvvv beta users

Download

When downloading vvvv you can choose between two versions:

- Stable: Use for production
- Preview: Use for testing latest features

You may also want to use the third-party Gamma Launcher for conveniently downloading and using multiple versions side by side.

Finding Help

When you start vvvv for the first time, you're greeted by the HelpBrowser which is your entry point into the world of Tutorials, Examples and more.

For completeness, here we duplicate links to some important bits:

Tutorials

- vvvvTv: Our youtube channel has the official Tutorials and HowTo videos
- Beginner video tutorials (by chk)
- Intermediate video tutorials (by TobyK)
- The NODE Institute Winter 2023 (various topics)
- The NODE Institute Summer 2023 (various topics)
- NODE20 workshop recordings: Recordings of 30 webinars held during NODE20
- Graphics video tutorials (by Takuma Nakata)
- Chinese video tutorials (by RED)

Examples

- VL.TheBigBang: A tutorial series for absolute beginners, covering all nodes and techniques to get you started (by chk)
- VL.GenerativeGestaltung: A collection of examples following the Generative Gestaltung v2 book (by phlegma)

- VL.ExtendedTutorials: Extended tutorial help patches, covering common questions as learners move from beginner towards intermediate (by TobyK)
- Nodevember2022: A collection of examples made following the prompts of Nodevember (by CeeYaa)
- Genuary2023: A collection of examples made following the prompts of Genuary (by CeeYaa)

Chapters

This reference is further structured into the following chapters:

Chapter	Content
Development Environment	About the different components (menues, windows) of vvvv
Language	About the language VL
Libraries	An overview of all available node libraries
Extending vvvv	For developers to write their own nodes and libraries
Best Practice	Learn the best practices for specific topics

Connect

- Forum: Any questions left? Get them answered here
- Chat: Want to talk with other vvvv users? Here is where we meet
- Blog: Follow this one, to make sure not to miss any news
- Mastodon: Follow for more bite-size news than in the blog
- Pixelfed: Share screenshots of your work with fellow patchers using Show and Tell
- Vimeo: Share your project documentations
- GitHub: Ready to contribute code? All of vvvv's libraries are open-source!

Introduction for Creative Coders

What you can do with the various existing creative coding frameworks, is often roughly the same. What's mostly different is the way in which things are done. vvvv has a history spanning since the early twothousands. With the newer version "vvvv gamma" you get to use the results of many years of research into creating a convenient and fast to use visual live-programming environment.

What people are using vvvv for

vvvv is used in a wide range of projects covering topics like: generative-design, interaction-design, data-visualization, computer-vision, VR, show-control, physical-computing, machine-learning and generally for any kind of quick prototyping. Have a look at the Gallery to see some specific examples.

Depending on where you come from, vvvv offers different benefits:

Coming from other visual programming environments

If you're familiar with visual creative coding environments, like:

- cables
- Isadora
- Max
- PD
- TouchDesigner
- Vuo

... then the following could be interesting for you:

Vast, modular, open-source library of nodes

Browse the Library section to get an overview of the availability of a vast range of different libraries for vvvv, most of which are open-source (some not yet, but only for organisational reasons). Apart from the VL.CoreLib the three biggest to date are:

- VL.Stride: for 3d rendering, shader programming, VR,... based on Stride
- VL.Skia: for 2d rendering, vector graphics export,... based on Skia
- VL.OpenCV: for computer vision,... based on OpenCV
- VL.Fuse: a collection of GPU tools and libraries to use with VL.Stride
- VL.Elementa: a UI widget library for VL.Skia

Further there is a big focus on supporting loads of Devices and Protocols out of the box.

Export Windows applications

Any program you build with vvvv can be exported as a proper Windows application. Other platforms are planned.

Comfortable licensing model

vvvv is free for non-commercial use. Simply download, install and run it:

- No questions asked
- No copy-protection
- No feature limitations
- No mandatory registration
- Pay per size of your business (Individual, Freelancer or SME, Big Fish)
- You own the last version you paid for
- Yearly or monthly payment options

As you're starting to use vvvv for commercial projects, you simply buy a per-developer-seat license.

Coming from text based coding

If you're familiar with text based creative coding, like:

- Cinder or OpenFrameworks
- Nannou or rin
- Processing or p5.js
- OpenRNDR

... then the following could be interesting for you:

Save time

vvvv does not have the classic time-consuming edit-compile-run cycle. For every change you make to your vvvv program, you'll get instant results! We call this **live-programming** and are using a state hot-reload approach: Every change is instantly compiled under the hood without you having to trigger it. If there is an error, you get an in-place indicator or tooltip about the problem and you can fix it without the need to restart your program or loose state.

Further, with its rich set of Libraries that only need to be connected, vvvv allows you to prototype a lot of common basic scenarios in no time.

Use familiar techniques

VL (the language used in vvvv) is not your ordinary visual language where you only can work with readymade, but hard to extend blocks: It comes with loops and if-regions, allows for recursions, lets you define and instantiate your own datatypes (even generic ones) and define and implement interfaces. This means you're not limited to dataflow programming but can also apply your object-oriented

programming skills. Further you can make use of delegates and observables and execute parts of your program asynchronously. All without writing a single line of code.

Write your own nodes

True, certain things, like low-level algorithms, are sometimes easier to write in text. Nodes can be written using plain C# or F#, without the need for any vvvv specific boilerplate code. That's why you can also directly use any of the .NET libraries hosted by the NuGet package manager.

What you may miss

Compared to one or the other frameworks listed above, you may miss:

- vvvv is still Windows only, which is planned to change at some point
- vvvv cannot export to mobiles or the web. This is not totally out of the question, but not on the roadmap as of now

Getting Started

- Watch these Tutorials and then these Tutorials to learn the very basics
- See if there are upcoming live online courses at The NODE Institute
- And the Recordings of NODE20 Webinars for 90 additional hours of learning
- As you have more specific questions, find them answered in the HowTo's

Press `F1` in vvvv to open the Helpbrowser. There you'll find numerous example and help patches on various topics. Also: With any node selected press F1 to see its dedicated help patch.

Any questions left? Get help and support by the developers and a welcoming global community in the forum or chat.

Introduction for vvvv beta users

Over the years people have been using vvvv beta to realize more and more complex projects, often maxing out its capabilities. New programming concepts and techniques had to be introduced to keep it up to date with today's user requirements.

At some point we came to the conclusion that the quantity and scope of such changes would create a lot of overhead in development. Chances were that implementing these changes would compromise performance and stability and disappoint users because they would have to adjust their workflows to new paradigms. So instead we decided to start from scratch, and build VL as a new, completely independent visual language. This gave us much more freedom and flexibility in creating VL in its own pace, rather than having to build new features into the already proven vvvv beta.

Creating VL, our mission has always been to incorporate as much of the good stuff from vvvv beta as possible and add new features to solve problems that people get stuck on when using vvvv beta. Ultimately we want all vvvv beta users to feel at home in VL while also introducing new audiences to and convincing skeptics of the wonders of visual programming.

With the integration of VL into vvvv beta, users are able to keep their patching habbits while exploring the features of VL bit by bit, as needed. When you hit the limits using vvvv beta, we now recommend that you check to see if VL might be able to resolve these issues. The following is a non-exhaustive list of common issues that are/will be solved by VL:

When to use VL

Your huge patch has become quite a mess over time and you find it hard to read and maintain it

vvvv beta does not have much of an idea about the structure of a patch. The only way to structure things are subpatches but even those are only useful for us humans, vvvv beta actually ignores them and sees just one big patch. The fact that vvvv beta doesn't offer better ways to organize your patches makes it hard to use vvvv beta to build big, well structured programs. VL has many features built-in to tackle exaclty that problem: Documents can explicitly reference each other, you can create custom datatypes and operations (think object-oriented programming). All these help you improve readability and maintainability of your programs.

Parts of your patch need to only run once to initialize or need to be initialized only at a later point during runtime

To initialize parts of your patch once on startup, in vvvv beta you'd send a bang to certain parts to evaluate them once. Even when not evaluated though, those parts are part of the vvvv beta program and waste at least some of your precious CPU cycles and memory. VL allows you to more cleary run parts

only once (using constructors in datatype patches) and more clearly run whole parts of a patch only on demand.

You want to offload parts of your patch to separate threads

Large patches can become computationally expensive and vvvv beta does not allow you to use the full power of your PC by being inherently single-threaded. Using VL you can define regions of your program that you want to run asynchronously to the main patch thus using multiple CPUs in parallel.

You need to react asynchronously to an input device

External input devices often send their data at a higher or lower rate than the framerate you want your patch to run at. vvvv beta has only one mainloop and any data coming in first has to be matched to that before being able to handle it. With timing critical-devices this can cause problems where your only chance so far was to write a c# plugin. In VL all external input comes via "observables" that are executed asynchronously and allow you to deal with incoming data at their rate before synching it to the mainloop.

You need to use an external .net library

Using any .NET library is already possible in vvvv beta by writing a c# plugin. This requires you to change to text-programming and follow the vvvv beta plugin-interface which introduces a bit of overhead both in development-time and computation. Using VL, chances are that you can access the same .NET library directly, without any such overhead and still keep programming visually all the way.

Main differences between vvvv beta and VL

At first glance VL looks rather familiar to vvvv beta users: There are gray nodes, links and IOBoxes. Patching works similar in both. The most obvious differences a regular vvvv beta user will stumble upon are:

- LOOPS: there is no automatic spreading. instead, VL only has explicit loops (for now)
- TYPES: VL is more picky about types which also leads to more flexibility
- PADS: instead of FrameDelay nodes in VL you can use named pads to denote the handing of state from one frame to the next
- NODEBROWSER: VL has different kinds of nodes which makes it a bit different to navigate the nodebrowser

These are only the most striking differences that former vvvv beta users will detect, VL has many more features to be disovered and applied as needed. More details about these and some more of the things that will look and sound familiar to vvvv beta users can be found in the following chapters.

The User Interface

Main Menu

There is no middleclick main menu in VL. Instead all main functions hide behind the little gray `Quad Menu` in the topleft corner. Next to the quad menu is the Document Menu with entries concerning the active document. See Navigating a Project.

The NodeBrowser

The VL NodeBrowser is a totally different beast. See The NodeBrowser.

Inspektor

There is no global Inspektor in VL as of yet. Instead an Inspektor pops up next to inspectable elements on demand. MiddleClick an Input/Output, Pad or IOBox or rightclick on its label -> Configure to bring up its Inspektor.

Finder

See Finders.

TTY Renderer

There is a Console which you an open via `Quad Menu` -> `Windows` -> `Log`.

Docking Patches

In VL all open patches are docked by default. A session of open tabs cannot be saved.

Open a nodes patch

Where in beta you'd rightclick a node to open its corresponding patch, in VL you rightclick -> Open.

Patching

Removing a link to a pin in vvvv beta would copy the current value into the pin. This is not happening in VL.

The Document Structure

.v4p versus .vl

In vvvv beta each patch has its own .v4p file. This is different with VL. Here many patches can be collected within a single .vl file or VL document, as we call it. Therefore small VL projects typically only have one VL document, even if they consist of multiple patches.

Then there are different types of patches in VL. You can learn about them in the Patches section.

The Process Patch is the one that most closely correspond to what you're familiar with from vvvv beta. Watch HowTo Make a Node to learn how you can wrap a group of nodes in a new patch.

The Language

Nodes

In vvvv beta all nodes look the same. In VL we distinguish between Process and Operation nodes:

Image:Process node (LFO) vs. Operation node (Distance) vs. Member Operation node (Any)

- Process nodes are *stateful* and have a darker bar underlining the pins
- Operation nodes are *stateless* and don't have pin-bars

Being stateful allows a Process node to store data between frames. Stateless Operation nodes are simple functions that can only operate on data they receive via their inputs.

In that sense all nodes in vvvv beta were potentially stateful but there wasn't any simple way to find out whether they actually were or not.

The reason for why it is important to distinguish is that it greatly improves the way you can build your patches by making deliberate decisions about which parts of your patch holds state and which do not. This improves readability and most of all simplifies debugging as certain types of problems can only occur with state involved. So with a runtime/logic problem at hand it is always wise to first start looking into Process nodes.

Naming Conventions

Naming conventions have slightly changed and are now as follows:

Name (Version1 Version2 ..) [Category.Subcategory]

when in vvvv beta it was:

Name (Category Version1 Version2)

Operations

In vvvv beta each patch defines exactly one operation. In VL a patch can define any number of operations. Each operation has a user-specified name and version and inherits the category of its patch.

Image:Multiple operations in a patch

IOBoxes vs Pins

In vvvv beta an IOBox can be used to set or display values. By giving an IOBox a descriptive name you turn it into an input- or output-pin of its patch. In VL we distinguish between IOBoxes and Pins. While

IOBoxes can still be used to set or display values you now use explicit Pin elements to specify Inlets and Outlets for operations.

Image:IOBox vs. Pin

Just like in vvvv beta create an IOBox by pressing the middle mousebutton while making a connection. If instead you `CTRL` + Leftclick while making a connection you create an Inlet or Outlet.

Both Pins and IOBoxes can be configured via a middleclick on them. Input Pins can also be given a default value.

Image:Configuring a Pin or IOBox

Another note on pins is that in vvvv beta you're used to when removing a link that goes into an input pin of a primitive type (value, string, color, enum) that the input pin will store the last values that came in via the link. This is not happening in VL, where input pins cannot store values.

Values

In vvvv beta there is only one numerical type. It is called simply "Value" and it is internally represented by a Float64 (which you as a user hardly ever have to worry about). Any output of type Value can be connected to any input of type value even though they may have different subtypes, like Integer, Boolean, or Bang.

In VL there are many different types for values:

- Boolean
- Byte
- Integer32
- Integer64
- Float32
- Float64

and for now you can only connect from lower to higher precision, eg. from Integer32 to Float32 (or Float64) but not the other way round.

Vectors

In vvvv beta there is no difference between a 2/3/4-dimensional vector and a spread with 2/3/4 slices. In VL we have explicit types for vectors, ie: Vector2, Vector3 and Vector4.

Spreads

In vvvv beta there are spread generators (like LinearSpread,...) spread operators (like GetSlice, Zip,...) and spread sinks (like Bounds, Mean, ...). The same and more are available in VL with the additional

advantage that in VL all spread operations are always available for all datatypes without adding more nodes to the NodeBrowser. See Generics.

In vvvv beta every connection between two pins is a spread. A spread can have 0, 1 or more slices but still it would be a spread in all cases.

In VL this is more defined: You need to understand a few things:

- There is a difference between a single value and a spread with a single value.
- When we talk of a spread of Integer32, strings or colors, we write: Spread, Spread, Spread
- We can also have multidimensional Spreads, like Spread<Spread>
- A Spread is only one type of collection. Another common type of collection would be a Dictionary or a HashSet, but we can imagine many different types of collections. Anyway for a start you'll mostly use Spreads.

TODO: Link to Collections.Spreads docu!

Spreading

In vvvv beta every node can automatically be spreaded, meaning the node is executed for every slice on its inputs. This convenience feature is not (yet) available in VL. We're still thinking about implementing something similar though.

Image:Loop around a vl node vs. implicit spreading in vvvv beta

Bin Size

Bin Sizes are vvvv beta's workaround to no having multi-dimensional spreads. As with a Bin Size pin associated with a spreaded pin you can specify how the spreads individual bins are to be interpreted by a node. Since in VL spreads can now contain other spreads the concept of Bin Sizes is no longer necessary.

Framedelay

In vvvv beta there were two reasons to use framedelays

- to make sure one thing happens after the other
- to store a value for the next frame

In VL you'll only use a FrameDelay node for the first use-case. If your patch needs to store a value for the next frame you're creating a datatype patch. In this case you can now use Properties to much better structure your programs.

Adding pins to nodes

Nodes like +, *, Cons, that have a dynamic pin-count can have pins added/removed by selecting them and pressing `CTRL` `+` or `CTRL` `-`.

Evaluation

Evaluation in vvvv beta is framebased. Each frame the whole graph is being evaluated *lazily* starting from sinks, like Renderers or Writers:

There are some nodes that don't evaluate inputs under certain circumstances. A Switch (Input) for example only evaluates the one input that currently is switched to. A S+H only evaluates its data source, if you really want to sample in that frame. So vvvv beta takes care that not everything is evaluated, but only the part that is necessary.

VL however always evaluates all unless you force it not to evaluate certain parts.

Let's say you have a VL node in vvvv beta. The whole VL patch will get evaluated. And not only that: all nodes in there will get completely evaluated. So if there is a patch behind a node this whole patch gets evaluated also.

One reason to do so was to get the system very much compatible with .NET. So if you would export a vl doc later on, you would encounter no special types like Lazy, but just the basic type that you also see in your patch.

The only way to force VL not to compute certain areas all the time is to use regions. There are different regions, but basically they all fence the inner part from the outer part in a way that the region itself can decide when to call the inner part.

A *loop* or an *if* region will not execute when the iteration count is 0 or the condition is false. Some nodes come with sugar for a hidden if region. Those nodes come with an apply pin. If apply is off, the node doesn't get evaluated.

Nodes

Here we're having a look at individual nodes that work differently in VL in regards to vvvv beta:

IOBox (String)

In vvvv beta an IOBox (String) connected to a pin that takes a filename or directory, automatically adapts to it so that you can right-click on it and get an open-dialog to choose a file or directory accordingly.

In VL there is an IOBox (Path) to handle file and directory paths but it doesn't know (yet) which you'd want to choose. So by default a rightclick is opening a FileDialog. To get a DirectoryDialog you have to press `Shift` while rightclicking!

Cons

In vvvv beta you can connect single values as well as spreads to a cons node. Since VL distinguishes between single values and spreads (see the section on Spreads above) there are now two nodes:

- Cons: combines single values to a spread
- Concat: concatenates multiple spreads to a single spread

In case you want to combine single values with spreads you first have to convert the single values to spreads using the ToSpread node. See "Adding Pins to Nodes" above to learn how to increase/decrease the pin count for those nodes.

Map and MapRange

Instead of having one node with an enum to choose between different map-modes, in VL there are now distinct nodes for doing only:

- Clamp
- Mirror
- Wrap
- Map

And besides the simple Map node there are distinct nodes for applying map in combination with a specific map-mode:

- MapClamp
- MapMirror
- MapWrap

InputMorph

Looking for the InputMorph (Animation) node? This one is now called Lerp which is short for Linear Interpolation.

But beware of two differences:

- While on the InputMorph the first input is the morph factor ("Switch" pin), the respective "Scalar" input on the Lerp node is its rightmost!
- The InputMorph can have any number of inputs and morph between all those, while the Lerp only has two inputs!

Quaternions

There is one difference to take care of: The multiplication order is inverted. VL uses the textbook convention of the quaternion multiplication whereas vvvv beta is using the DirectX implementation which deviates from the official convention so that quaternions behave in the same way as rotation matrices. But most 3D engines nowadays use the textbook convention and we comply in order to make it easier to read foreign documentation/code that also uses quaternions.

VL Templates

When creating a VL node from vvvv beta you have the choice between two available templates for cloning them via `Ctrl` + Leftclick

- Template (Value)
- Template (Value Stateless)

Cloning either creates a new .vl document. The first template then creates a ProcessDefinition in that document while the stateless template creates an OperationDefinition with the given name, category and version.

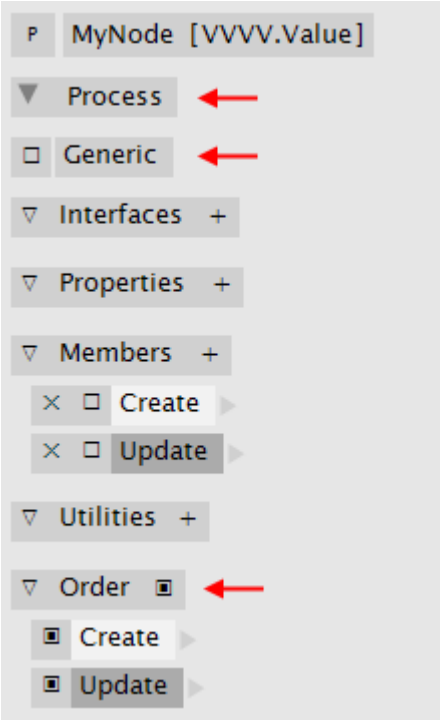
Make VL nodes show up in vvvv beta

So cloning a template creates a new .vl document and one nodefinition (process or operation). In order to make additional nodes created in vl show up in vvvv beta, they must be:

- non-generic
- in the category VVVV

and either:

- an OperationDefinition
- a ProcessDefinition with ProcessNode enabled



Preparing a vl process for use as a node in vvvv beta

Introduction for .NET programmers

VL is a visual programming language for .NET. It combines dataflow programming with features known from object-oriented programming. vvvv is the editing environment for VL.

Versions 5.x of vvvv are using .NET6 Versions 6.x of vvvv are using .NET8

With direct access to all of .NETs libraries you can basically use it as just another .NET language like C# or F#. But since most of those libraries were not created with dataflow in mind, we've curated a library for you that is much more comfortable to use and is the default referenced library when creating a new VL document.

Language Features

If you're familiar with programming in C#, VL should feel quite familiar, apart from the fact that it is visual. Here are a few things things that will be new to you though:

Spreads

The main collection type in VL is called **Spread** and its individual entries are called **Slices**. The Spread is an immutable collection with one special feature: If you're calling `GetSlice(Index)` on it, with an index that exceeds its count, instead of an error you get the slice with the index taken modulo the count.

E.g. if you have a spread with 5s slice and you're asking for slice 7, you get slice 2.

Foreach with Multiple Inputs

Where in C#, a foreach loop can only iterate over one collection at a time, in VL you can iterate over many collections at the same time. The number of iterations executed, is determined by the collection with the smallest count.

Renamings

In general we try to do as little renaming as possible when importing data types. But for collections we took the liberty to do the following renamings from the original ones:

- All immutable .NET collections drop the *Immutable* pre-fix since it's the default in VL
- All mutable .NET collections get a *Mutable* pre-fix
- IEnumerable is called Sequence

Getting Started

Here are a few workshop recordings particularly suited if you have a background in .NET:

- Introduction to vvvv For Coders

- Using .NET NuGets
- Object Oriented Patching
- Introduction to Reactive Patching
- Turning a .NET library into a VL library
- Talk introducing vvvv to .NET developers

Then head back to the Overview for more sources of learning content.

You may also want to find out how you can extend vvvv and have a look at our Demo Library including C# and F# examples of writing your own nodes for VL.

Any questions left? Get them answered in the forum or chat.

C# Concepts

... and how to express them in VL.

foo++

How to translate an expression like the following to VL:

```
var foo = 1;
foo++;
```

First we need to agree that the above is only a shortcut to writing:

```
var foo = 1;
foo = foo + 1;
```

Then the below patch should be read as: The lower foo pad corresponds to the left side of the assignment (foo =) and the upper foo pad corresponds to the initialized variable foo (var foo = 1). So:

```
foo (lower pad) = foo (upper pad) + 1;
```

.Note how the link from the IOBox to the foo pad is white, meaning it is assigned to the constructor

.Shortcut for + 1: Use the Inc node

Nullable

When referencing an external library, you may encounter input or output pins of type **Nullable**. To deal with them you need to reference the **System.Runtime** assembly from the GAC.

This gives you access to the nodes HasValue and Value to read from nullable outputs. To set a value to an input that requires a Nullable, it is enough to put a CastAs node in between the value and the nullable input.

Note

CastAs only shows up with the **Advanced** aspect enabled in the nodebrowser.

Variables

Lambda

Observable

See Reactive.

Task

Enumerator

When referencing an external library, you may encounter collection types, that do not inherit from Sequence and as such cannot simply be used with VL's ForEach loop.

Most likely those collections will still support an Enumerator. Here is how you can deal with an enumerator in VL:

MoveNext, Current

C# Keywords

... and how to express them in VL.

Legend

- **not-supported** This keyword does not have an equivalent in vl. If you find it necessary to use its functionality, you can still write C# code that can be used in vl, see Writing Nodes in C#.
- **no-inheritance** VL does not support class inheritance and therefore also not the concept that comes with this keyword.

abstract

C# Reference

```
abstract class Foo
{
  abstract public int Bar();
}
```

{no-inheritance}

as

C# Reference

```
object foo = 1;
var foo = foo as int;
```

Use the *CastAs* node.

base

C# Reference
{no-inheritance}

break

C# Reference

```
for (int i = 0; i < 10; i++)
{
  if (i % 2 == 0)
    break;
  Console.Beep();
}
```

See Special Loop Pins for how to use a **Break** output, to break out of a loop in vl.

case

C# Reference
See switch

catch

C# Reference
See try

checked

C# Reference
{not-supported}

class

C# Reference
See Datatype Patch.

const

C# Reference
{not-supported}

continue

C# Reference
{not-supported}

decimal

C# Reference

default

C# Reference
See switch

delegate

C# Reference
See Delegates.

do

C# Reference
See while

double

C# Reference
Called *Float64* in vl, part of category *Primitives*.

Note

Only showing in the nodebrowser, if the *Advanced* aspect is activated.

else

C# Reference
See if

enum

C# Reference

```
enum Alignment {Left, Middle, Right};
```

While existing enums can certainly be used in VL, one thing you cannot yet do in VL, is define a custom enum. If you need a custom enum, for now you'll have to define it using C# code. See Writing Nodes in C# for how to do this.

event

C# Reference
Instead of events, VL uses a similar concept called **Observables**. See Reactive for details.

explicit

C# Reference
{not-supported}

extern

C# Reference
{not-supported}

finally

C# Reference
See try

fixed

C# Reference

{not-supported}

float

C# Reference

Called *Float32* in vl, part of category *Primitives*.

for

C# Reference

See Loops.

foreach

C# Reference

See Loops.

goto

C# Reference

{not-supported}

if

C# Reference

See Conditions.

implicit

C# Reference

{not-supported}

in

C# Reference

{not-supported}

int

C# Reference

Called *Integer32* in vl, part of category *Primitives*.

interface

C# Reference

See Interfaces.

internal

C# Reference

is

C# Reference
{not-supported}

lock

C# Reference
{not-supported}

long

C# Reference
Called *Integer64* in vl, part of category *Primitives*.

Note

Only showing in the nodebrowser, if the *Advanced* aspect is activated.

namespace

C# Reference
The concept of a *namespace* is called Category in VL.

new

C# Reference

```
var date = new DateTime(2002, 12, 24);
```

The new keyword denotes a constructor, meaning the operation that creates a new instance of a object. In VL all constructors of classes and records are called **Create**.

The equivalent of creating an instance of the DateTime class, in VL looks like this:name:

null

C# Reference

Part of the category **Primitive.Object**

Note

Only showing in the nodebrowser, if the *Advanced* aspect is activated.

operator

C# Reference

out

C# Reference

override

C# Reference
{no-inheritance}

params

C# Reference

private

C# Reference

protected

C# Reference

{no-inheritance}

public

C# Reference

readonly

C# Reference
{not-supported}

ref

C# Reference

return

C# Reference

sbyte

C# Reference

Called *Integer8* in vl, part of category *Primitives*.

Note

Only showing in the nodebrowser, if the *Advanced* aspect is activated.

sealed

C# Reference

{not-supported}

short

C# Reference

Called *Integer16* in vl, part of category *Primitives*.

Note

Only showing in the nodebrowser, if the *Advanced* aspect is activated.

sizeof

C# Reference

stackalloc

C# Reference

{not-supported}

static

C# Reference

{not-supported}

struct

C# Reference

switch

C# Reference

VL does not have a *switch* statement yet. See Conditions for workarounds.

this

C# Reference

throw

C# Reference

try

C# Reference

See Exception Handling.

typeof

C# Reference

uint

C# Reference

Called *Integer32 (Unsigned)* in vl, part of category *Primitives*.

Note

Only showing in the nodebrowser, if the *Advanced* aspect is activated.

ulong

C# Reference

Called *Integer64 (Unsigned)* in vl, part of category *Primitives*.

Note

Only showing in the nodebrowser, if the *Advanced* aspect is activated.

unchecked

C# Reference

{not-supported}

unsafe

C# Reference

{not-supported}

ushort

C# Reference

Called *Integer16 (Unsigned)* in vl, part of category *Primitives*.

Note

Only showing in the nodebrowser, if the *Advanced* aspect is activated.

using

C# Reference

virtual

C# Reference

{no-inheritance}

void

C# Reference

volatile

C# Reference

{not-supported}

while

C# Reference

VL doesn't have a *while* loop yet. See Loops for an easy workaround.

C# Contextual Keywords

... and how to express them in VL.

add

alias

async/await

by

descending

dynamic

equals

from

get

global

group

into

join

let

nameof

on

orderby

partial

remove

select

set

value

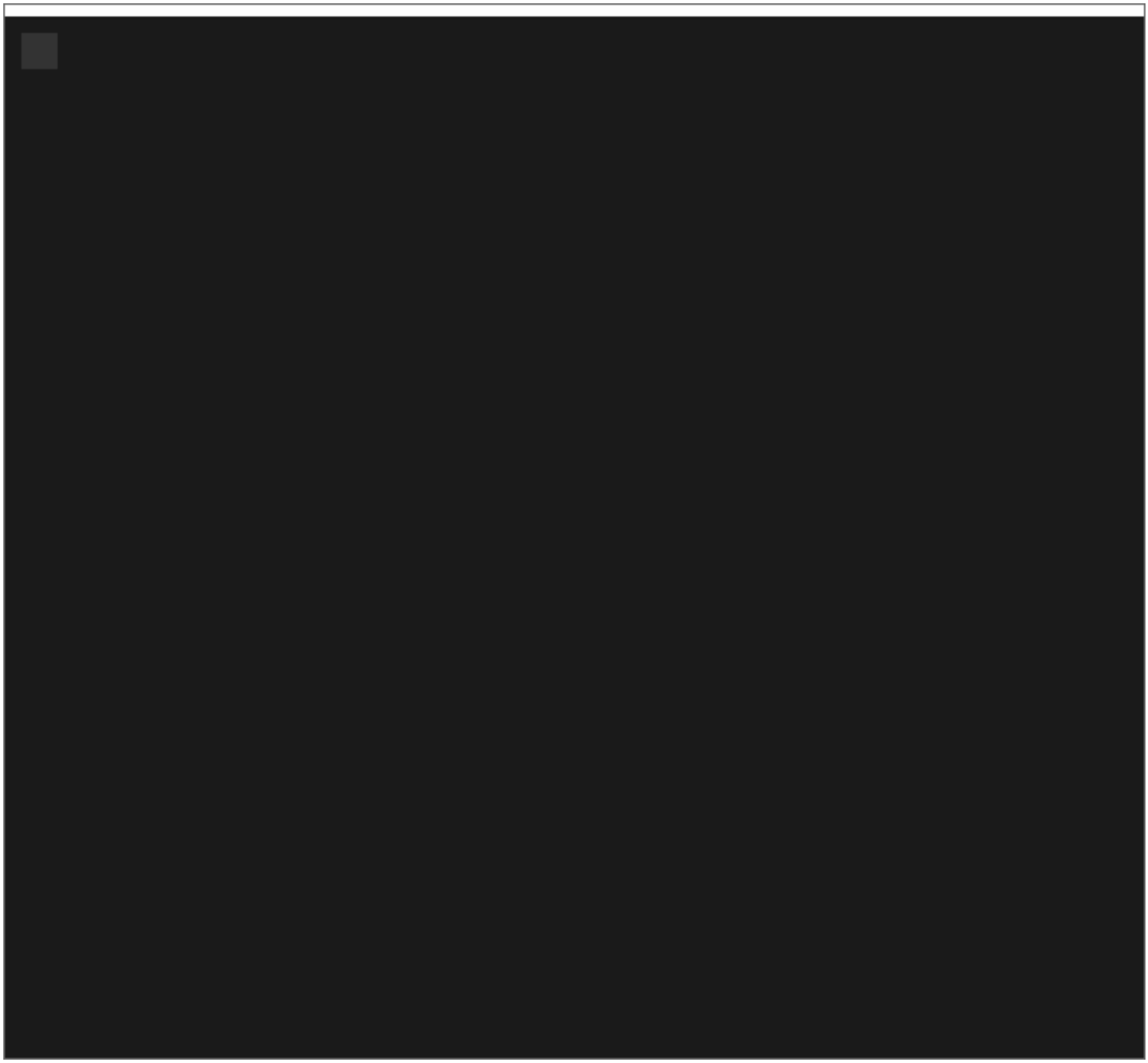
var

when

where

yield

The Development Environment



vvvv - distraction free development since 2002

Overview

The Quad menu

The Quad menu collects all global commands.

It gives you quick access to:

- Recent Sketches: For quickly accessing most recent .vl documents, saved in the default location
- The Helpbrowser
- Extensions
- Managing Nugets
- All Documents: A listing of all currently open .vl documents
- Run, Step, Pause, Stop, Restart all running applications
- Exporting Applications
- Additional Windows like the Documentation Window and the Solution Explorer

The document menu

The document menu collects all commands relevant for the active document.

It gives you quick access to:

- Configuring the documents Dependencies
- The documentes application patch
- The documents definition patch
- Saving, reloading, closing the document
- Jumping to the document in the Explorer

Document Tabs

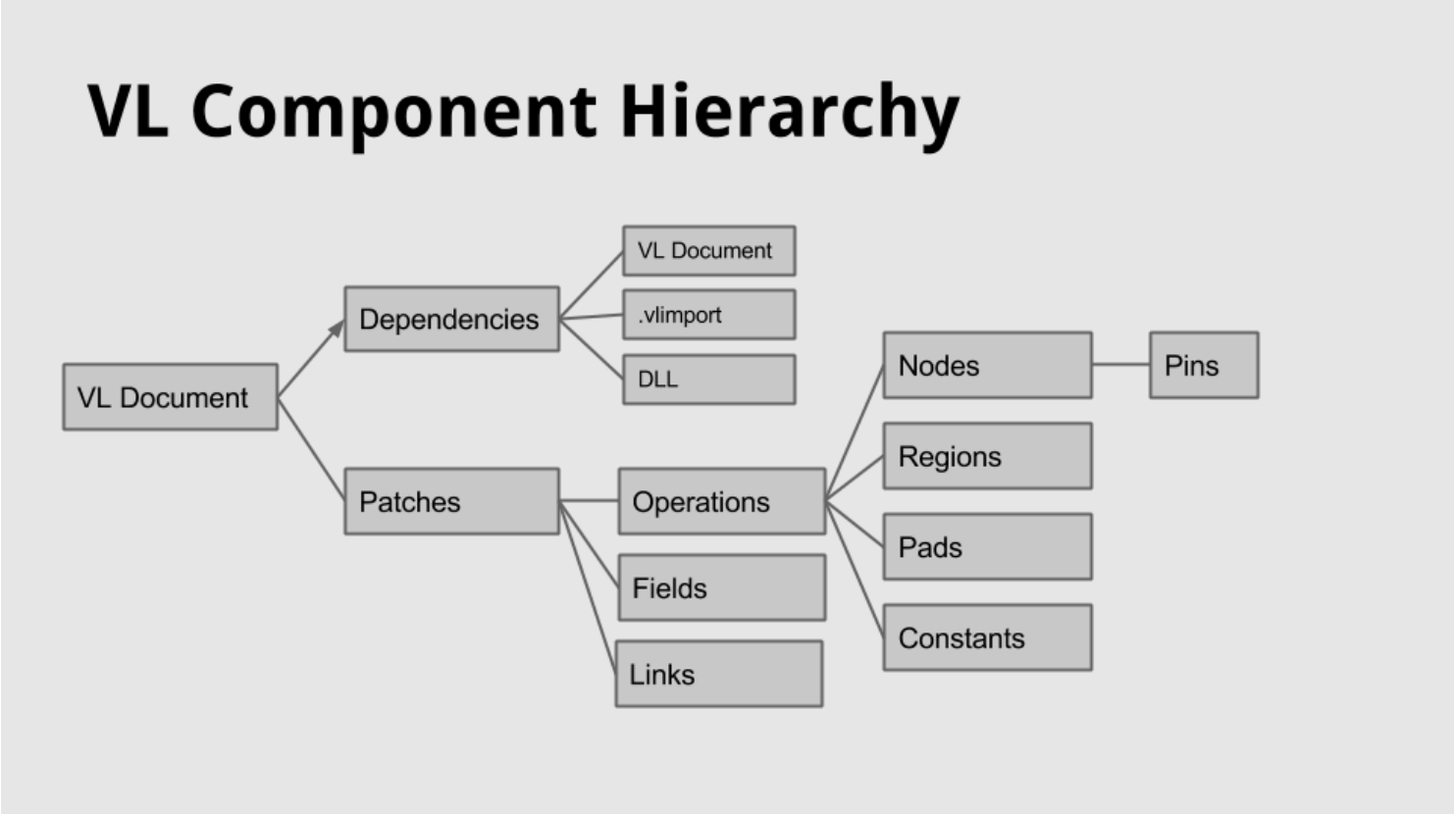
Shows titles of all open patches. All tabs with white titles belong to the active document. Tabs with dim titles belong to other documents.

The Hamburger

Gives you quick access to:

- Settings including a link to the backup directory
- Themes
- Licensing Terms including a link to the vvvv License Store
- The About dialog including
  - Version and licensing info for vvvv itself and all used open source libraries
  - Links to this documentation, the Changelog and Roadmap

Project Structure

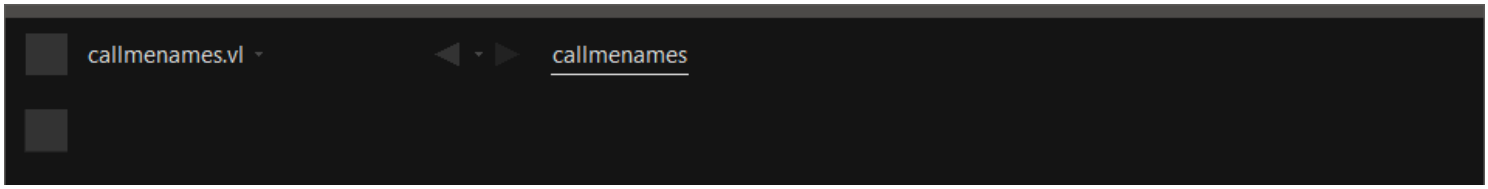


Components

Navigating a Project

A project in VL typically consists of a single .vl document that holds one ore more patches. In addition a document can depend on other documents or Nugets whose nodes it can access.

In the main menubar you always see the filename of the document you're currently working on, ie. the "Active Document".



The main menubar with the Active Document "callmenames.vl"

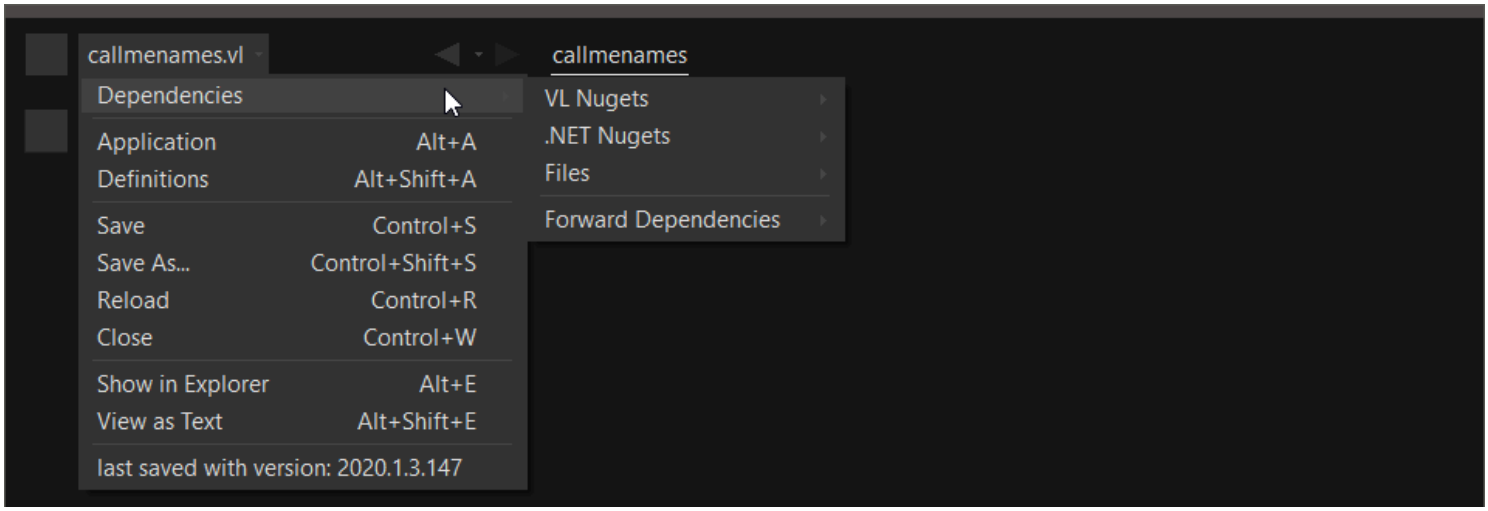
Active Document Menu

Clicking on the active document opens its menu.

Dependencies

A document can reference different types of dependencies:

- VL Nugets
- .NET Nugets
- Files

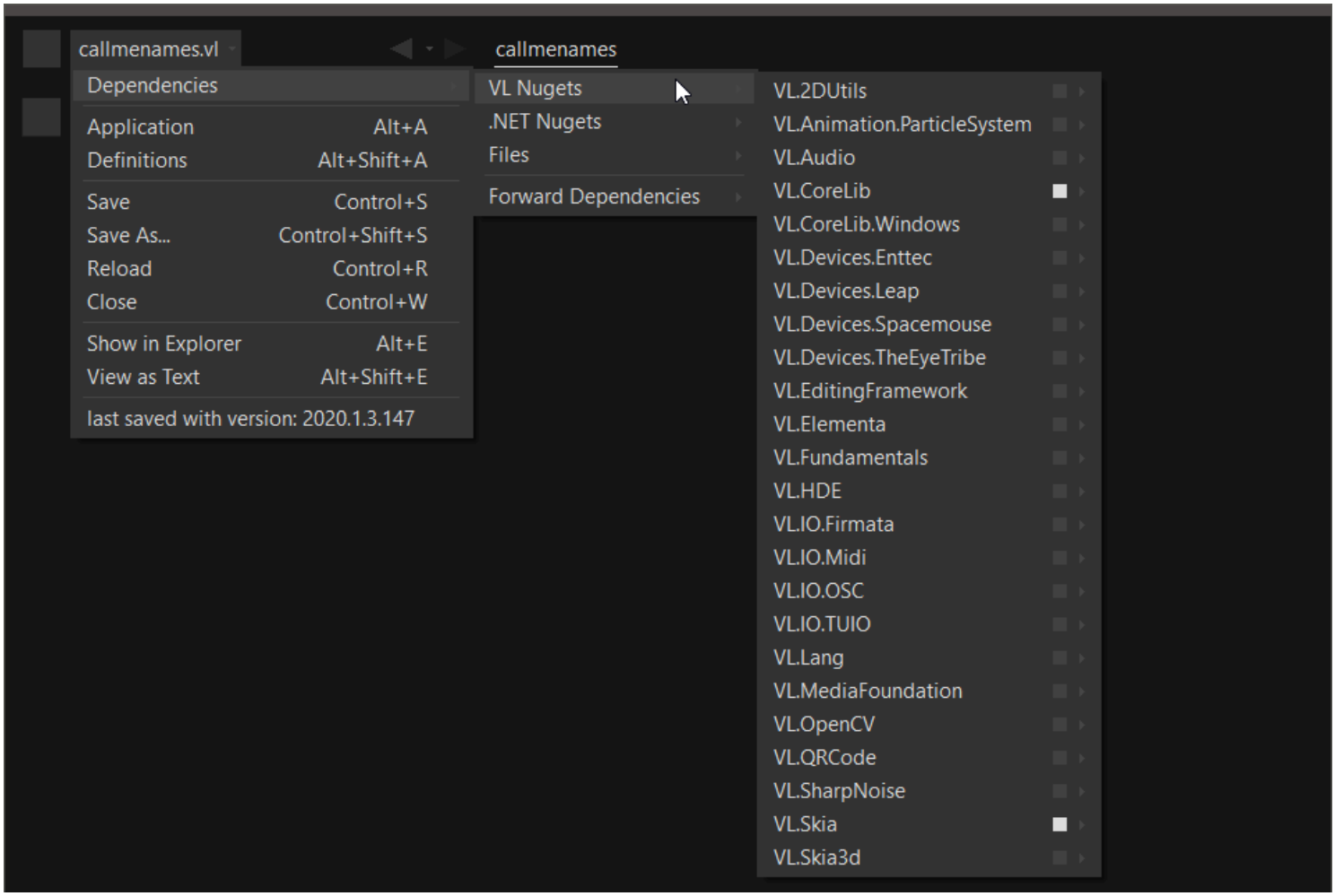


Document's dependencies

VL Nugets

Navigate to "VL Nugets" to see a list of all available Nugets exposing nodes for VL. Each Nuget is a collection of documents (.vl, .dll, ...) that provide nodes for a document.

A version in brackets next to a Nuget means that the currently loaded version of the Nuget is different to the version that was originally referenced.
> meaning the referenced version being smaller,
< meaning the referenced version being bigger
than the currently loaded version.



Available Nugets

Rightclick a Nuget to select it. Selected Nugets provide access to all their nodes via the nodebrowser in that document.

.NET Nugets

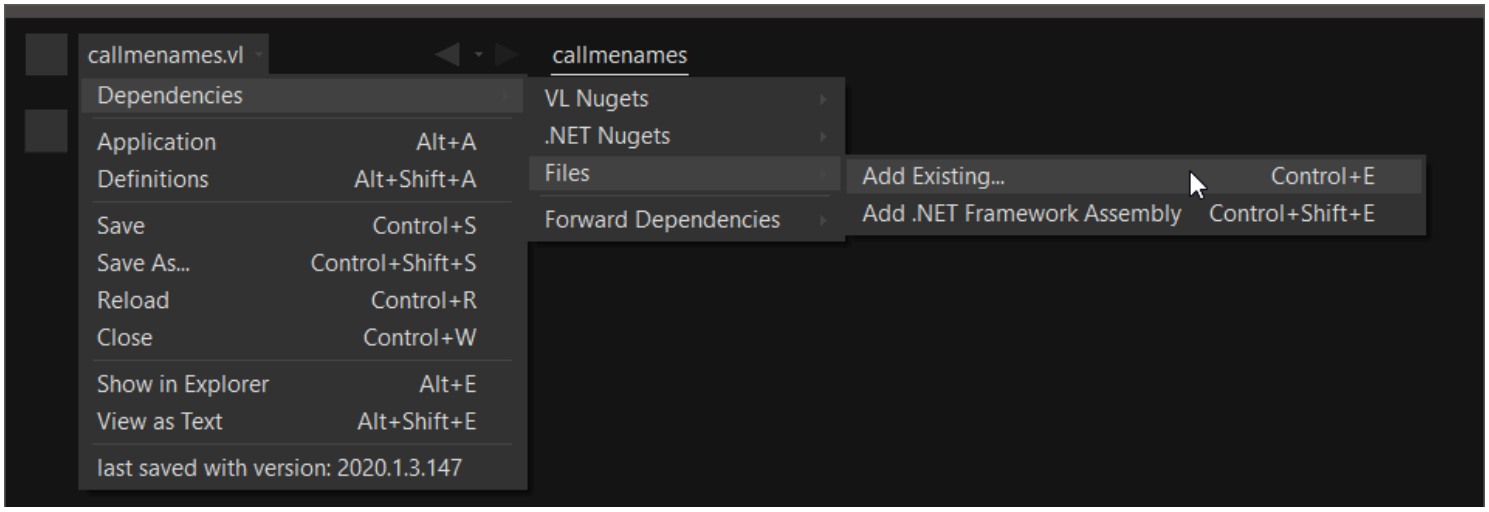
In the ".NET Nugets" section you find all Nugets that are not tailored for VL. That means you can still absolutely use them, but depending on the complexity of the library this may be a bit more advanced endeavor.

Files

In addition to Nugets you can also reference individual files of the following types:

- .vl

- .dll
- .csproj



Add Files

Choose "Add Existing..." to select a file via the file browser. All nodes that are exposed by those files will be available in your active document via the nodebrowser.

Forward Dependencies

In this section you see a listing of all Nugets and files combined. Here you can specify if the nodes of a specific dependency will be forwarded or not.

If a dependency is not forwarded, its nodes will only be visible in the current document. They will not be seen by a document that references the current document.

If a dependency is forwarded, its nodes will also be seen by any other document that references the current document.

Application

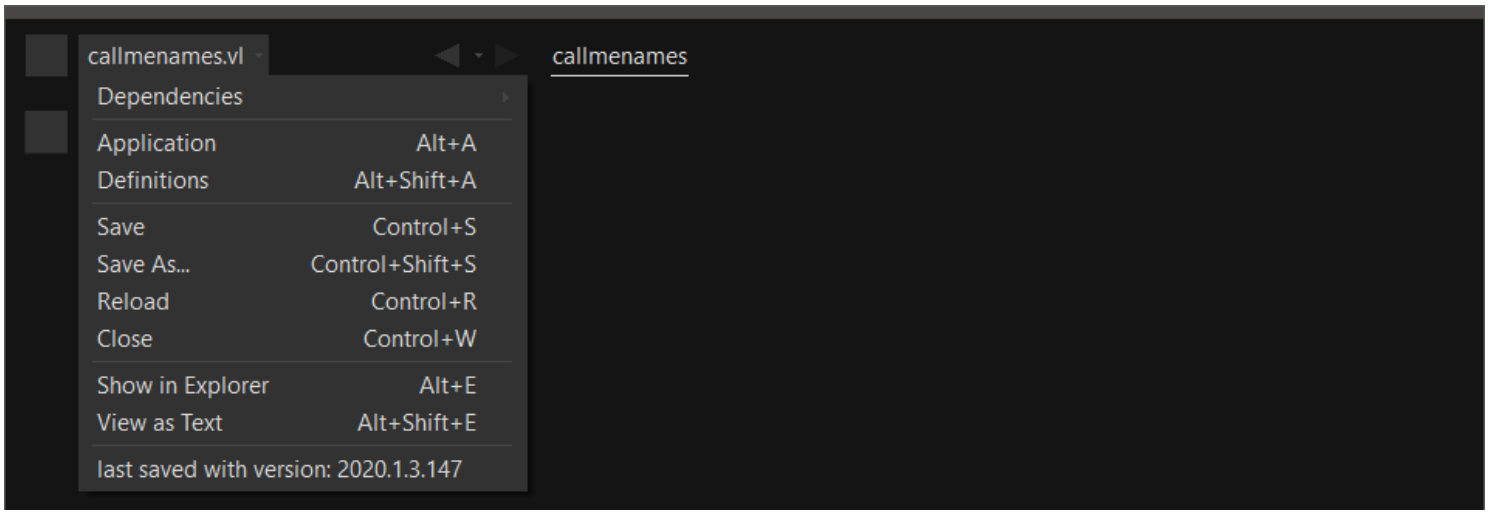
See Application Patch.

Definitions

See Definitions Patch.

Last saved with version

At the bottom of the menu you see the VL version the document was last saved with. If there is a little green or red icon before the document name, this means that the document has been saved with a version different to the one currently running.



In this example gamma version 2020.1.3.147 is running

Green

A little green symbol next to the document's name is a hint that the document was upgraded to the currently running version, which is usually just fine. Still this hint is given so you know that if you save the document in this version it may have troubles being loaded in an older version.

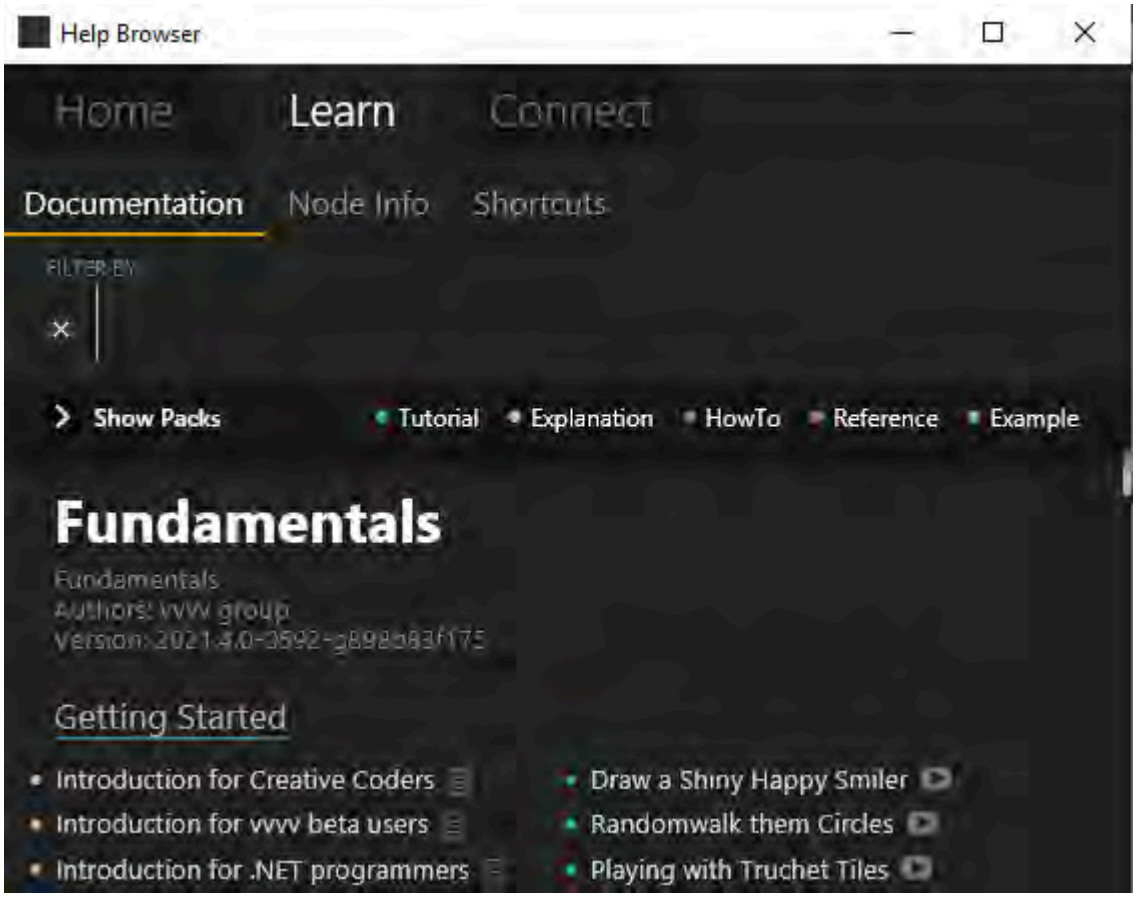
Red

A red symbol is a warning that the document was last saved with a newer version and therefore things may not look/work as expected. In that case consider running a newer version of VL to open this document.

Finding Help

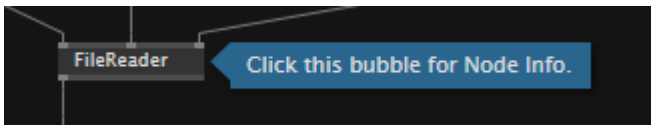
Help Browser

Press `F1` while no node is selected in the patch to open the help browser. Or choosing it from the main menu `Help Browser`. Go to the `Learn` section to search for help content among all the installed nugets. These entries will lead you to help patches, videos or written documentation:



Help Patches

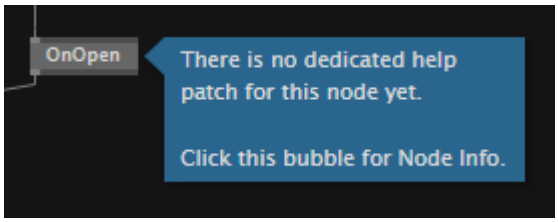
Press `F1` on a selected node to open its help patch. If a help patch is found for the node, it opens and marks the node you were interested in with a bubble.



The bubble indicating which node the help patch was opened for

Clicking the bubble will open the help browser showing the Node Info.

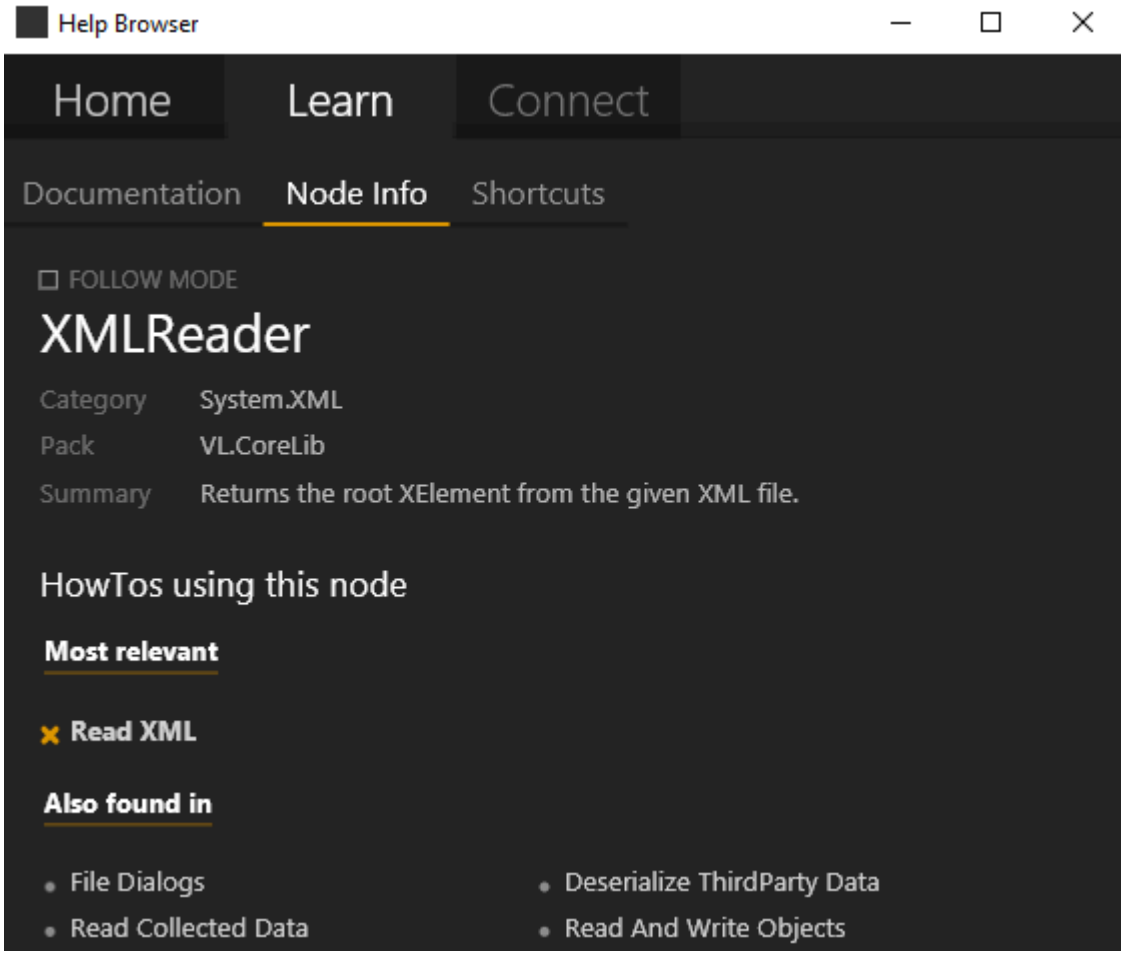
In case no help patch is found for a node, a bubble will pop up next to the node, indicating the fact that the node does not have a dedicated help patch. You can then still click the bubble to show the Node Info.



The bubble indicating there is no help patch for this node

Node Info

Node Info is available for every selected node. Next to information about the node it also shows you patches using this node. Click the entries in the list to open the patches relevant for the selected node.



Forum

Get help and support by the developers and a welcoming global community in the forum.

Chat

Talk to developers and fellow vvvv users in the chat.

Keyboard Shortcuts

Quad Menu

Description	Action
New Document	`Ctrl` `N`
Open Document	`Ctrl` `O`
Save All Documents	`Alt` `Ctrl` `S`
Run	`F5`
Step	`F6`
Pause	`F7`
Stop	`F8`
Export...	`F10`

Document Menu

Description	Action
Open Application Patch	`Alt` `A`
Open Definitions Patch	`Alt` `Shift` `A`
New Patch (=Process Node at Cursor)	`Ctrl` `P`
New Process Definition in Definitions	`Ctrl` `Shift` `P`
Save Active Document	`Ctrl` `S`
Save Active Document As...	`Ctrl` `Shift` `S`
Reload Active Document discarding unsaved changes	`Ctrl` `R`
Close Active Document	`Ctrl` `W`
Show Active Document in Explorer	`Alt` `E`
View Active Document as Text	`Alt` `Shift` `E`
Add Existing Dependency	`Ctrl` `E`
Add .NET Framework Assembly	`Ctrl` `Shift` `E`

Basics

Description	Action
General *middleclick* alternative	`ALT` + leftclick
Navigate back in Tab history	`Ctrl` `<` or `Alt` `Left`
Navigate forward in Tab history	`Ctrl` `>` or `Alt` `Right`
Navigate one level Up in application	`Ctrl` `^` (where `^` is the Key below `ESC`)

Description	Action
Navigate one level Up in definition	`Ctrl` `Shift` `^` (where `^` is the Key below `ESC`)
Copy screenshot of active patch to clipboard and save it as PNG next to its .vl document	`Ctrl` `2`
Take screenshot of active patch and save it as SVG next to its .vl document	`Ctrl` `Shift` `2`
Duplicate selected nodes/pads keeping	`Ctrl` `D`
Duplicate selected nodes/pads keeping incoming links	`Ctrl` `Shift` `D`

Pan & Zoom

Description	Action
Pan the patch	Rightdrag in an empty area
Zoom the patch	Mousewheel or `Ctrl` `+` & `Ctrl` `-`
Reset pan and zoom of the patch	`Ctrl` `0`
Zoom the tooltip	`Ctrl` while using the Mousewheel

Alternative Pan & Zoom

If you prefer to use the mousewheel for panning vertically instead of zooming change the "Mouse wheel zooms" setting to false to get the following behavior:

Description	Action
Pan the patch vertically	Mousewheel
Pan the patch horizontally	`Shift` + Mousewheel
Zoom the patch	`Ctrl` + Mousewheel

Selecting elements in a patch

Description	Action
Select an Element	left click
Add an element to the selection	`Ctrl` + leftclick
Force to include links in a marquee selection despite nodes or pads are already part of it	`Space`
Force to only select links when making a marquee selection	`L`

Nodes

Description	Action
Bring up the NodeBrowser to choose node to create	Double leftclick in an empty area of a patch or on a link

Description	Action
Replace a node	Double leftclick it to bring up the NodeBrowser and choose a different node. You may need to press ESC to remove existing choices
Move a node (or selection of nodes) into or out of a region	Press `Space` while draging a node
Assign an operation-node to an operation	Rightclick -> Assign -> (operation)
Remove the operation-assignment of a node	Rightclick -> Assign -> Clear assignment
Align selected nodes	`Ctrl` `L`
Line up selected nodes	`Alt` `L`
Evenly distribute selected nodes between left/top-most and right/bottom-most node in selection	`Ctrl` `Shift` `L`
Evenly distribute selected nodes taking the gap between the left/top-most two nodes as a measure for the rest	`Ctrl` `ALT` `L`
Increase count of input pins for nodes like +, Cons, ...	`Ctrl` `+`
Decrease count of input pins for nodes like +, Cons, ...	`Ctrl` `-`
Expose Pins	`Ctrl` `K`
Move selected nodes	`Arrows`
Move selected nodes faster	`Shift` `Arrows`
Assign a Help Flag	`Ctrl` `H`

Pin Groups

In general, the `Shift` key means second group and the `Alt` key means output group. However, if the node has no input pin groups, the `Alt` key is optional. If the node has more pin groups, the additional ones can be found in the context menu of the node.

Description	Action
(Input Group 1) Add Pin	`Ctrl` `+`
(Input Group 1) Remove Pin	`Ctrl` `-`
(Input Group 2) Add Pin	`Ctrl` `Shift` `+`
(Input Group 2) Remove Pin	`Ctrl` `Shift` `-`
(Output Group 1) Add Pin	`Ctrl` `Alt` `+`
(Output Group 1) Remove Pin	`Ctrl` `Alt` `-`
(Output Group 2) Add Pin	`Ctrl` `Alt` `Shift` `+`

Description	Action
(Output Group 2) Remove Pin	`Ctrl` `Alt` `Shift` `-`

Pads

Description	Action
Create via Nodebrowser	double leftclick in patch, type name of pad, choose *Pad*
Create while linking	Finish with `Shift` + leftclick in the patch
Bake current type annotation	`Ctrl` `T`
Clear type annotation	`Ctrl` `Shift` `T`
Create Create/Split operations	`Ctrl` `K`
Create property accessors operations (Get/Set)	`Ctrl` `Shift` `K`

IOBoxes

Description	Action
Create via Nodebrowser	right doubleclick in the patch
Create while linking	Middleclick (or `ALT` + leftclick) in the patch
Reset to default	`ALT` + rightclick
Edit the value	Double leftclick
IOBox (Value): Change value	Rightdrag up/down
IOBox (Value): Change value finer	`Ctrl` + Rightdrag up/down to change value with stepsize divided by 10
IOBox (Value): Change value finer	`Shift` + Rightdrag up/down to change value with stepsize divided by 10
IOBox (Value): Change value coarser	`Alt` + Rightdrag up/down and combine with `Ctrl` and/or `Shift` to multiply stepsize by 10 or 100
IOBox (String): Bring up FileOpenDialog	`Ctrl` + Rightclick
IOBox (String): Bring up DirectoryOpenDialog	`Shift` + Rightclick
IOBox (Path): Bring up FileOpenDialog	Rightclick
IOBox (Path): Bring up DirectoryOpenDialog	`Shift` + Rightclick
IOBox (Color): Change brightness	Rightdrag up/down
IOBox (Color): Change hue	Rightdrag left/right
IOBox (Color): Change saturation	`Ctrl` + Rightdrag up/down
IOBox (Color): Change the alpha channel	`Shift` + Rightdrag up/down

Links

Description	Action
Create an IOBox while linking	Finish with middleclick (or `Alt` + leftclick) in the patch
Create an input or output pin while linking	Finish with `Ctrl` + leftclick in the patch
Create a pad while linking	Finish with `Shift` + leftclick in the patch
Create a node while linking	Finish with a double leftclick
Insert a link point while linking	Click in an empty area in a patch
Insert a link point into an existing link	Press down on the link and drag a new point away
Remove a link point from an existing link	Click to select the point and then press `Delete`
Create an accumulator input or output proxy in IF or LOOP while linking	Finish with `Ctrl` + leftclick in the region
Create a splicer input or output proxy in LOOP while linking	Finish with `Ctrl` `Shift` + leftclick in the region
Insert a IOBox into an existing link	Double rightclick the link
Delete a link	Middleclick it or select it and press `Delete`
Start a new link from the same source after finishing a connection	Finish with middleclick
Assign a link to an operation	Rightclick -> Assign -> (operation)
Remove the operation-assignment of a link	Rightclick -> Assign -> Clear assignment
Insert a pad into an existing link	`Shift` + double leftclick or double rightclick the link
Insert a node into an existing link	Double leftclick the link
Show a links tooltip to see its current value and the operation it is assigned to	Press `Ctrl` while hovering the link
Force a connection to a datahub that would otherwise not accept it	`Space`
Force to include links in a marquee selection despite nodes or pads are already part of it	`SPACE`

Finders

For more details, see Finders.

Description	Action
Look for strings in the active patch	`Ctrl` `F`

Description	Action
Globally search for symbols (documents, patches, operations, pads)	`Ctrl` `Shift` `F` or `Ctrl` `,`

Frames

Description	Action
Create a Frame from Marquee selection	Hold `Alt` while making a marquee selection
Create a Frame in viewspace from Marquee selection	Hold `Alt` `Shift` while making a marquee selection
Force to include frames in a marquee selection despite nodes or pads are already part of it	`SPACE`
Surround selected nodes with a frame	`Alt` `F`
Surround selected nodes with a frame in viewspace	`Alt` `Shift` `F`
Toggle visibility of all frames in a patch	`Ctrl` `Alt` `F`
Take a screenshot of the selected frame	`Ctrl` `2`
Take a screenshot of all frames in a document at once	`Ctrl` `5`
Start/stop a screenshot recording of the selected frame	`Ctrl` `4`
Screenshot of marquee selection to clipboard	Hold `S` while making a marquee selection

Tooltips

Description	Action
Copy content when showing an error	`Ctrl` `Shift` `C`
Toggle showing more infos for infos, warnings or errors	`Space`
Detach from currently inspected instance	`Tab`

Runtime

Description	Action
Run	`F5`
Step	`F6`
Pause	`F7`
Stop	`F8`
Restart	`F9`
Restart editor extensions	`Shift` `F9`

Windows

Description	Action
Help Browser	`F1`

Description	Action
Inspector	`Ctrl` `I`
Debug Views	`Ctrl` `F2`
Gobal Channels	`Ctrl` `F3`
Application Exporter	`F10`
Patch Documentation	`Ctrl` `M`
Solution Explorer	`Ctrl` `J`
Make window topmost	`Ctrl` `T`

VL.Stride

The following shortcuts are working with either a VL.Stride SceneWindow or RenderWindow active:

Description	Action
Copy screenshot of render window to clipboard	`Ctrl` `2`
Toggle Performance Meter	`F2`
Toggle Profiler	`F3`
Toggle Helper View	`F4`
Toggle Fullscreen	`F11` or `Alt` `Return`

If the profiler (`F3`) is enabled and the window is active:

Description	Action
Cycle Page (FPS, CPU, GPU)	`F5`
Togle Sort Mode (Name, Timinig)	`F6`
Decrease Result Page	`F7`
Increase Result Page	`F8`
Jump to Result Page (1, 2, 3, 4, 5)	`1` , `2` , `3` , `4` , `5`
Slow down Refresh Time	`-`
Speed up Refresh Time	`+`

VL.Skia

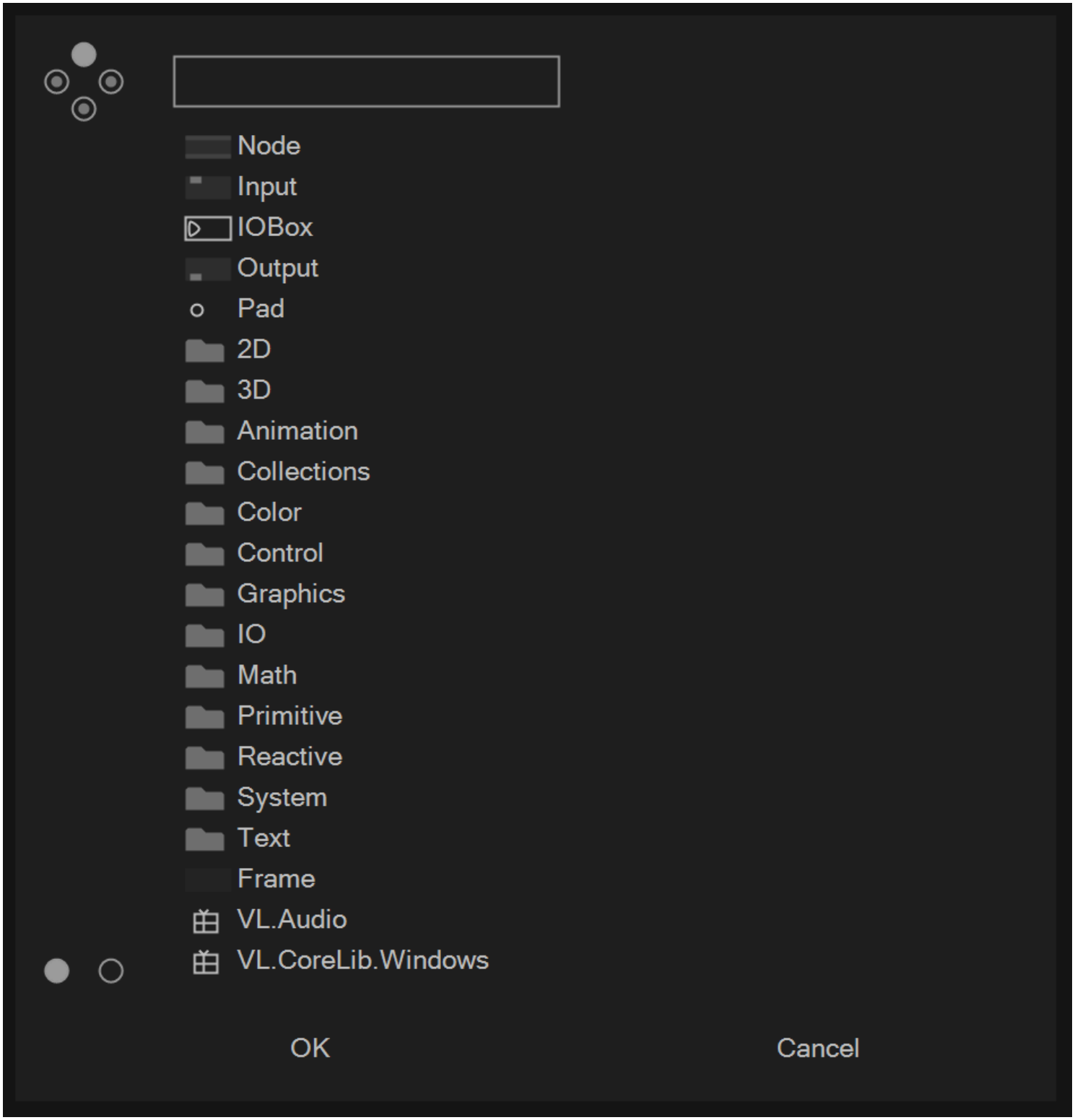
The following shortcuts are working with a VL.Skia Renderer active:

Description	Action
Copy screenshot of render window to clipboard	`Ctrl` `2`
Toggle Performance Meter	`F2`

The NodeBrowser

When opening the NodeBrowser via a left double-click anywhere on a patch what you see is:

- Language primitives (Node, Input, IOBox,...)
- A list of top-level node categories (2D, 3D, Animation,...)
- A list of available nugets (VL)



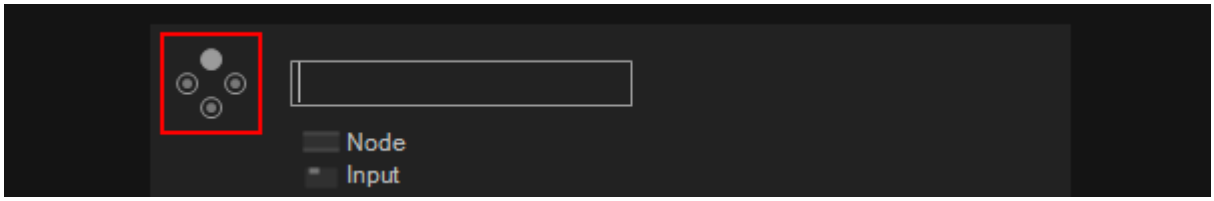
You can also bring up the NodeBrowser:

- While linking by double leftclicking to create a node that is connected to the link at hand
- By double leftclicking on an existing link to insert a node there
- By double leftclicking an existing node to replace it

Filtering nodes

The circles on the left side are filters that let you quickly include or exclude a certain group of nodes. Hover them with the mouse to see their purpose and click to toggle them on or off:

- Include high level nodes (the standard set)
- Include potential future nodes (Experimental)
- Include low level nodes (Advanced)
- Include obsolete, old nodes (Obsolete)



The bottom two circles are two more filters:

- Include internal nodes (those that are only visible inside this document)
- Include external nodes (from referenced .dlls and NuGets)

If you see a dot inside of a circle, it means that, given the current search term you would get additional nodes listed, if this filter was on.

The defaults for those filters can be set via the Settings in the "Advanced" section.

Finding Nodes

There are different ways to navigate the list of available nodes:

- By category
- By tag

In both cases it helps to be familiar with the icons:

 Category

 NuGet Package

Nodes:

 Process Node

 Operation Node

Types:

 Record

 Class

 Enumeration

 Interface

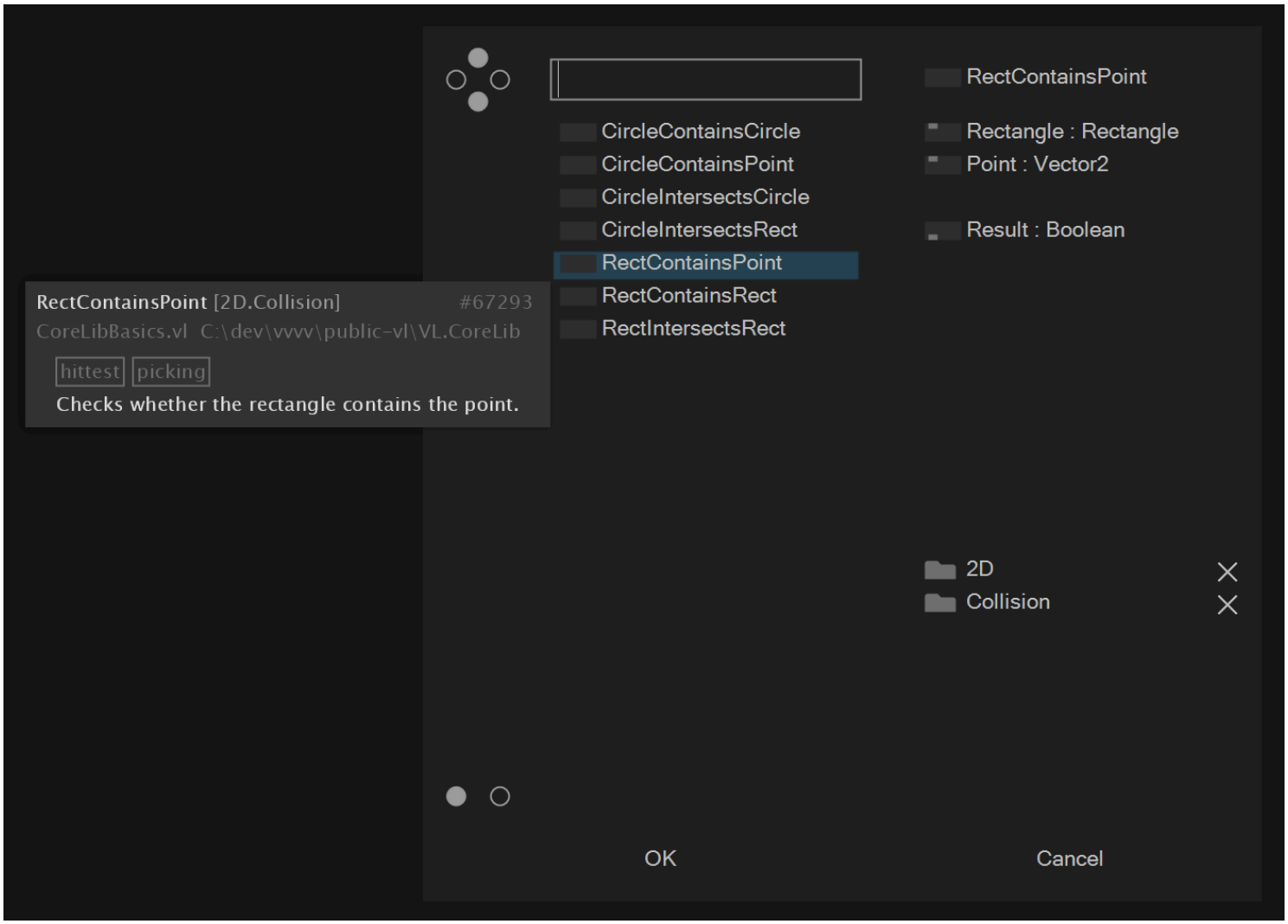
Hovering any entry that represents a node will show you the nodes Inputs and Outputs in the topright corner of the NodeBrowser and a tooltip will show you the description associated with the node if available.

When you're clicking an entry that represents a node two things may happen:

- If your selection is unambiguous the node will be created
- There are situations where a selection is ambiguous in which case the NodeBrowser prompts you to specify more details by choosing from the offered options.

Search By Category

Nodes in VL are organized in a hierarchical structure of categories. Click any of the categories to enter it.



Note that when entering a category, a tag appears in the bottom right part of the NodeBrowser. The listing is now filtered by this tag. Choose another category from the listing to refine your search or remove a tag by clicking on the X button next to it. Pressing the ESC button always removes the last tag.

Search By Tag

Enter any word to search for it.

Language Primitives

The language primitives are written in *italic*:

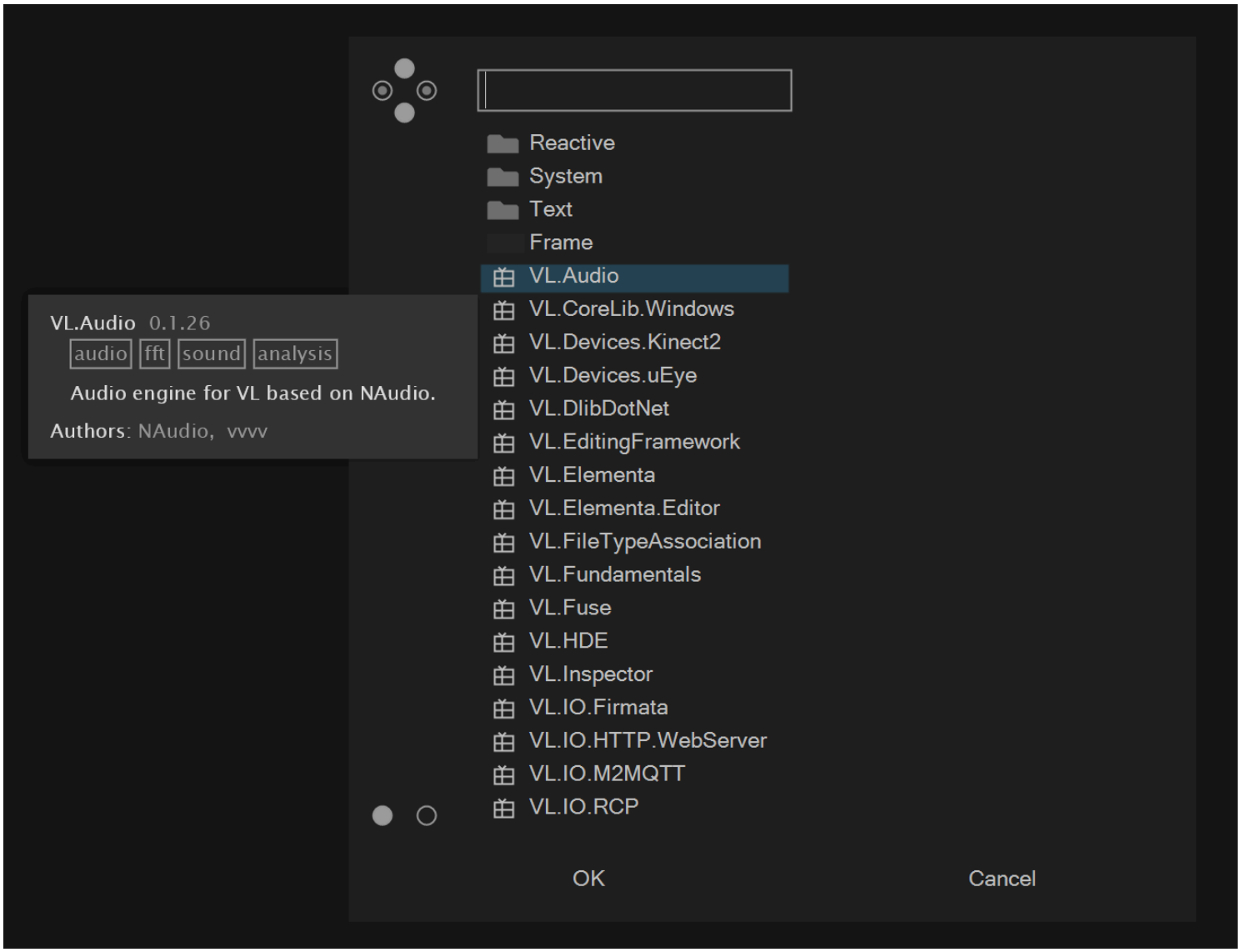
- IOBox
- Pad
- Pins (Input, Output)
- Patch Definition
- Canvas
- Record Definition
- Class Definition
- Operation Definition

- Delegate
- Regions (Repeat, Foreach, If)

Many of the primitives can be given a name directly after choosing to create them via the NodeBrowser. For Pins and Pads it also works the other way round: type a name into the NodeBrowser and then click Pin/Pad to create the respective element with already the desired name set.

Available Nugets

Nugets that are available for VL but not referenced by the active document show up in the NodeBrowser from where you can quickly reference them via a single click. After the nuget is referenced all its nodes show up in the NodeBrowser.



If you want to get rid of a nuget again you have to uncheck it in the documents list of dependencies.

Nodes that only differ by Signature

We typically distinguish nodes by Name, Version or Category. But in some cases there are several nodes that share all of the above. When in the process of designing the surface of a library we typically give

nodes a Version to distinguish nodes that only differ by a detail, but in other cases we import nodes from a dll without naming each of the overloads differently.

In these cases the user needs to select the node by choosing a signature: 

We have two modes to let the user select the node:

- choose a node signature. This is like choosing several pins at once.
- choose single pins in a multiple choice fashion

Choose a node signature

In our case we see three entries corresponding to the three available nodes: 

Choose single pins

This is how the workflow for choosing single pins works:

- If there is a node that is clearly simpler than all others in terms of its signature, this node will be placed.
- Double-clicking on the node will show you pins that you can opt into
- Selecting such a pin will again close the node browser if there is one node that is clearly simpler than any other with that pin configuration.
- Double-clicking the node will allow you to see if there are more pins that you could add, and which pins you already committed to. You can cancel pin choices individually in order to find the overload you need. This workflow can be helpful in cases where many overloads with many pins show up. You don't need to keep track of all the different variations. Just tell the node browser which pins you want to work with and you'll be presented pins that are still an option.

Activate this workflow by choosing 

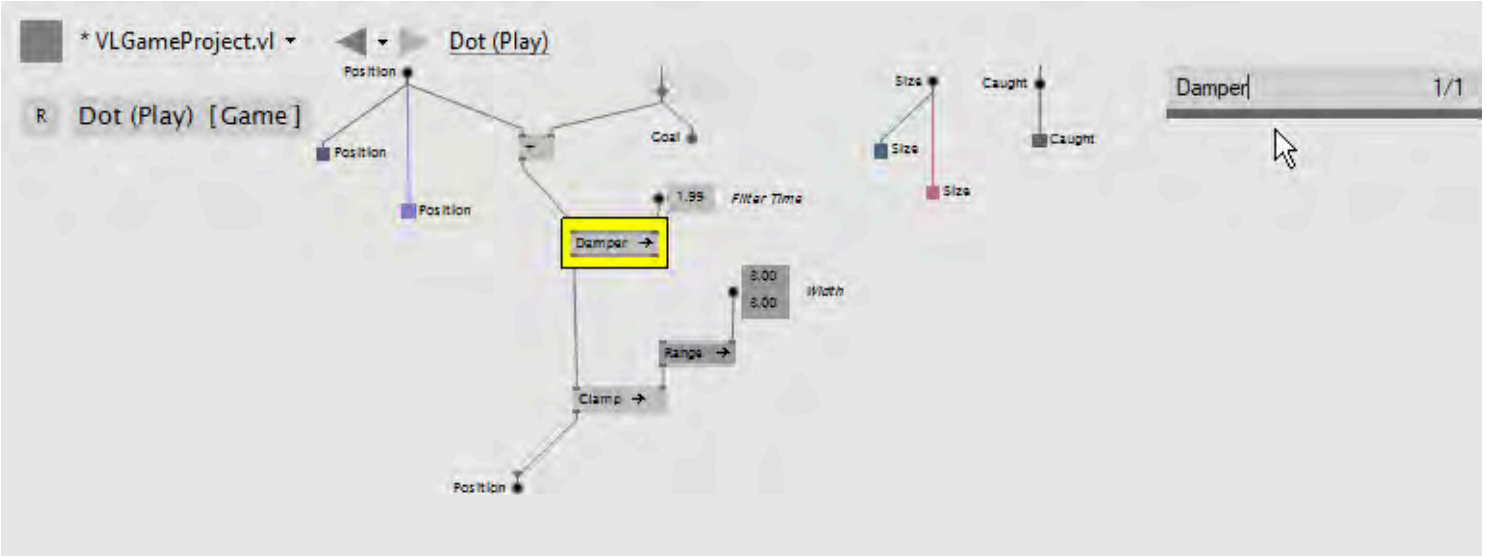
Finders

The development environment offers 2 different ways to find stuff quickly

- Finder: Searches for strings in the active patch
- SymbolFinder: Searches for documents, patches, operations, pads

Finder

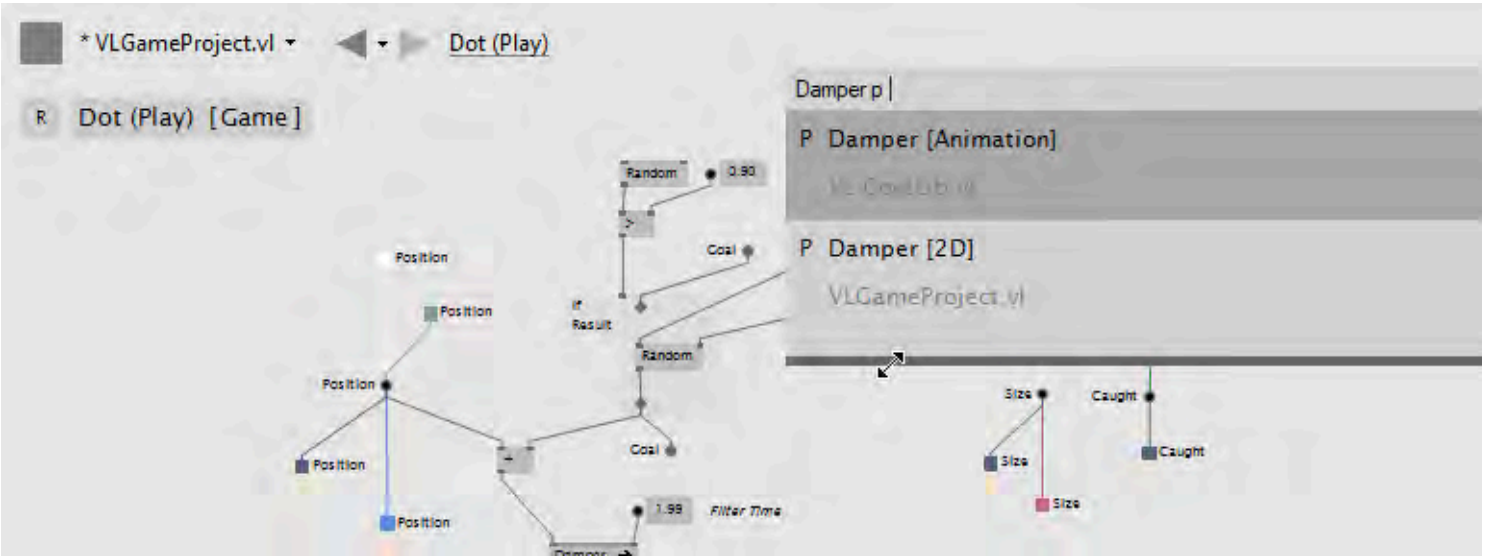
Press `Ctrl` `F` in any patch to search for local occurrences of strings. Type a string and then use `F3` or `Enter` to navigate through the results. Press `ESC` to hide the results again.



Finder

SymbolFinder

Press `Ctrl` `Shift` `F` or `Ctrl` `,` to globally search for any symbol including the given string.



Symbol Finder

Use the shortcuts below followed by a " " (space) to narrow down your search:

Shortcut	Description
d	Search for documents only
p	Search for patches (types, processes) only
m	Search for member operations only
u	Search for utility operations only
f	Search for fields only

Debugging

Press `Ctrl` `F2` to open the following debug windows

- Build result
- App Health
- Log

Stepping Execution

Description	Action
Run	`F5`
Step execution	`F6`
Pause execution	`F7`
Stop execution	`F8`
Restart	`F9`

The execution of VL can be paused by pressing `F7`. Pressing `F6` repeatedly, causes the execution to make one step at a time. To get back into run mode press `F5`.

You can also choose to auto-pause whenever an error occurs and jump to the node that threw the exception by activating the *Pause on error* Setting.

IOBoxes

Use IOBoxes connected upstream to display incoming values.

Image:IOBox used to inspect an upstream value

Tooltips

Hover a pin with the mouse to see a tooltip with its name and type.

Image:Tooltip of a pin showing its name and type

If the type is a collection, like Sequence, Spread,... you'll also see the collections number of elements in square brackets and the value of the first three elements in that collection.

Image:Tooltip of a collection type

Note

If you're observing a pin in a patch that is instantiated multiple times there is no way yet to know to which instance this value belongs!

Timings

The tooltip shows timings when *Show Timings* is activated in the Settings.

Image:Tooltip shows timings on nodes

Image:Tooltip shows timings on Datatype definition

Write to Console

The Log window can be opened via `Quad > Windows > Log` or by pressing `Ctrl` `F1`.

Use the advanced `Write` and `WriteLine` nodes from the `System.Console` category to write debug infos to the Log.

Pressing the `Backspace` key with the Console focused clears it.

Warnings and Errors

Image:A node showing an error

Image:A link showing a warning

Image:A pin showing a warning

To get rid of pin-warnings after you've acknowledged them, press `Ctrl` `E`.

Attaching Visual Studio

In case you encounter errors that are not traceable through techniques mentioned above, you can also attach to your patches with Visual Studio.

Note

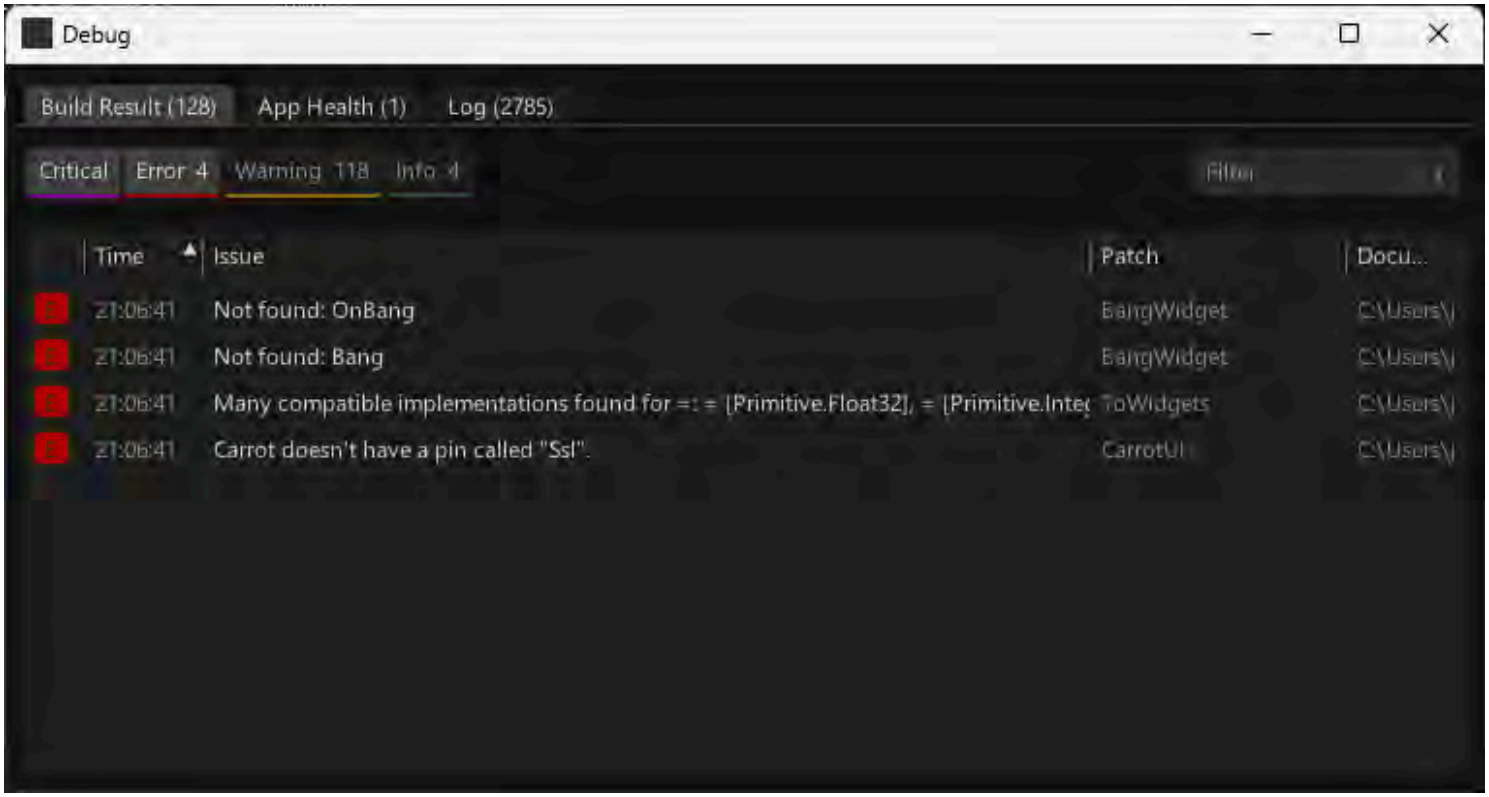
Beware, this is not meant for the casual user!

But if you're quite familiar with C# programming, you can do the following:

- Start vvvv.exe with the `--debug` commandline argument
- Run Visual Studio
- Attach to vvvv.exe
- Wait for the exception to occur
- This will bring up the C# code of your patch and jump to the location where the error occured. Here you can also set break points. This may help you figure out the source of the problem.

Build Result

Here you'll see errors, warnings and infos that came up as a result of the build process. Whenever vvvv is building your patches (ie. whenever you make a change), this list is cleared and you'll get a fresh view of the latest issues.



Most commonly, you'll see errors relating to:

- Missing dependency references or nodes
- Typing issues on links, pins, or pads

Double-clicking an entry brings you to the source of the message: right in the patch, where you'll find the same issue visualized as a red or yellow node, link, pin, or pad. While the patch is the place that allows you to understand and fix a problem, the Build Result view is supposed to help you:

- Zoom out and understand the big picture of build issues
- Navigate and find the root issues

Keeping this list in check is the basis of a healthy app.

Remarks

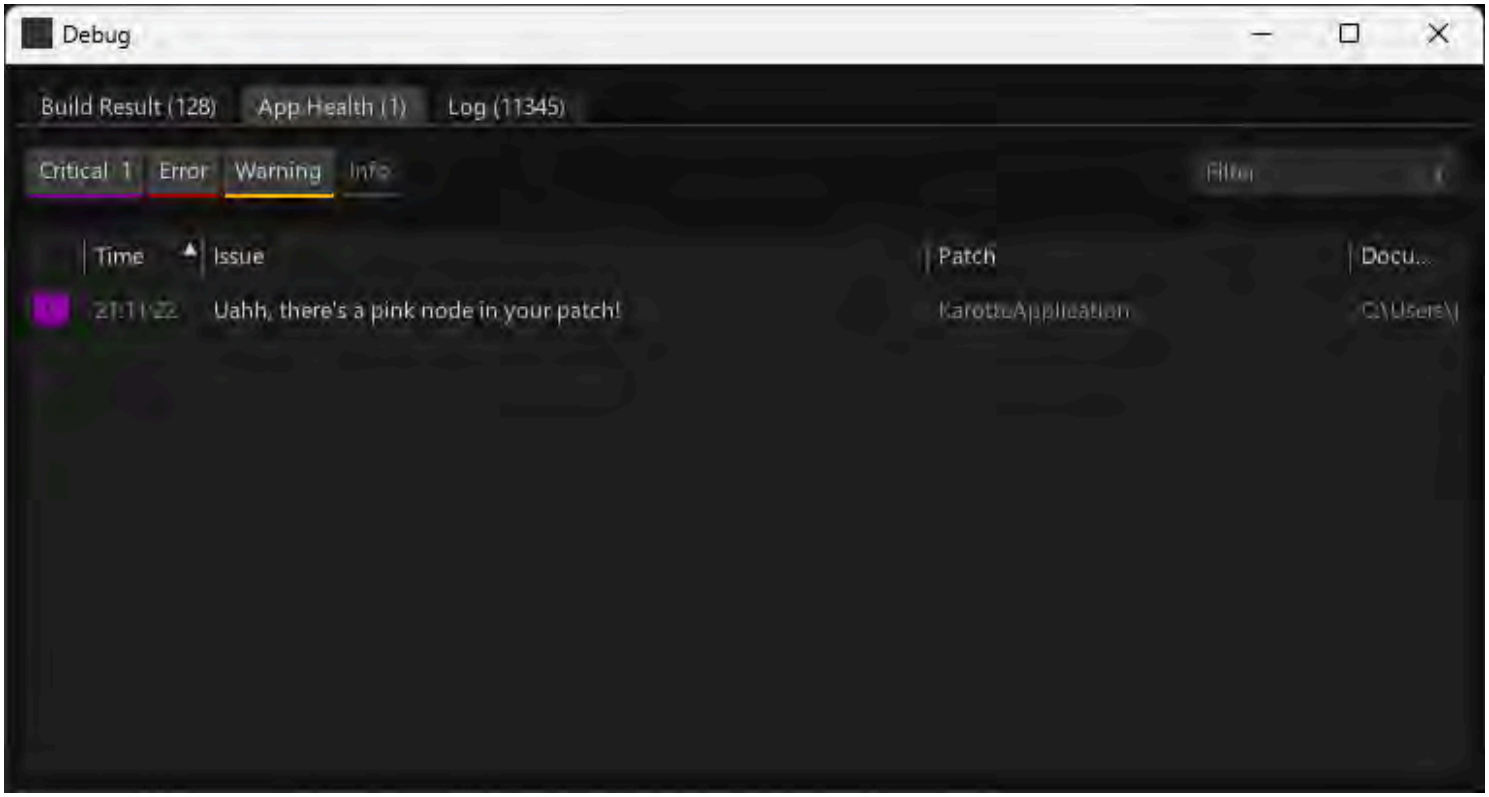
- Typically only the Errors are of interest
- vvvv can build patches with errors. This allows for faster iterations and refactorings. You don't need to fix every error to test a new idea. If you are in the process of such a refactoring you might see a lot of errors. The Build Result view reminds you of those, but don't feel bad. Sometimes when

developing bigger patches, this is just part of working, a proof that things are moving. Decide for yourself when it's about tackling those errors again. For sure, when testing your app, searching for a bug is easier if there are no build errors left

- Warnings: There are too many of them typically and it's quite hard to get rid of them. So it's a bit tricky to take them seriously. However, they still have a purpose which is trying to help you with reasoning about what your app does. Especially when it comes to object mutability and order of execution

App Health

Here you'll see an overview of issues coming from your running application. These can be exceptions (the pink nodes) or come from a `Warn [VL.Session]` node that you can even use yourself in patches to indicate a problem.

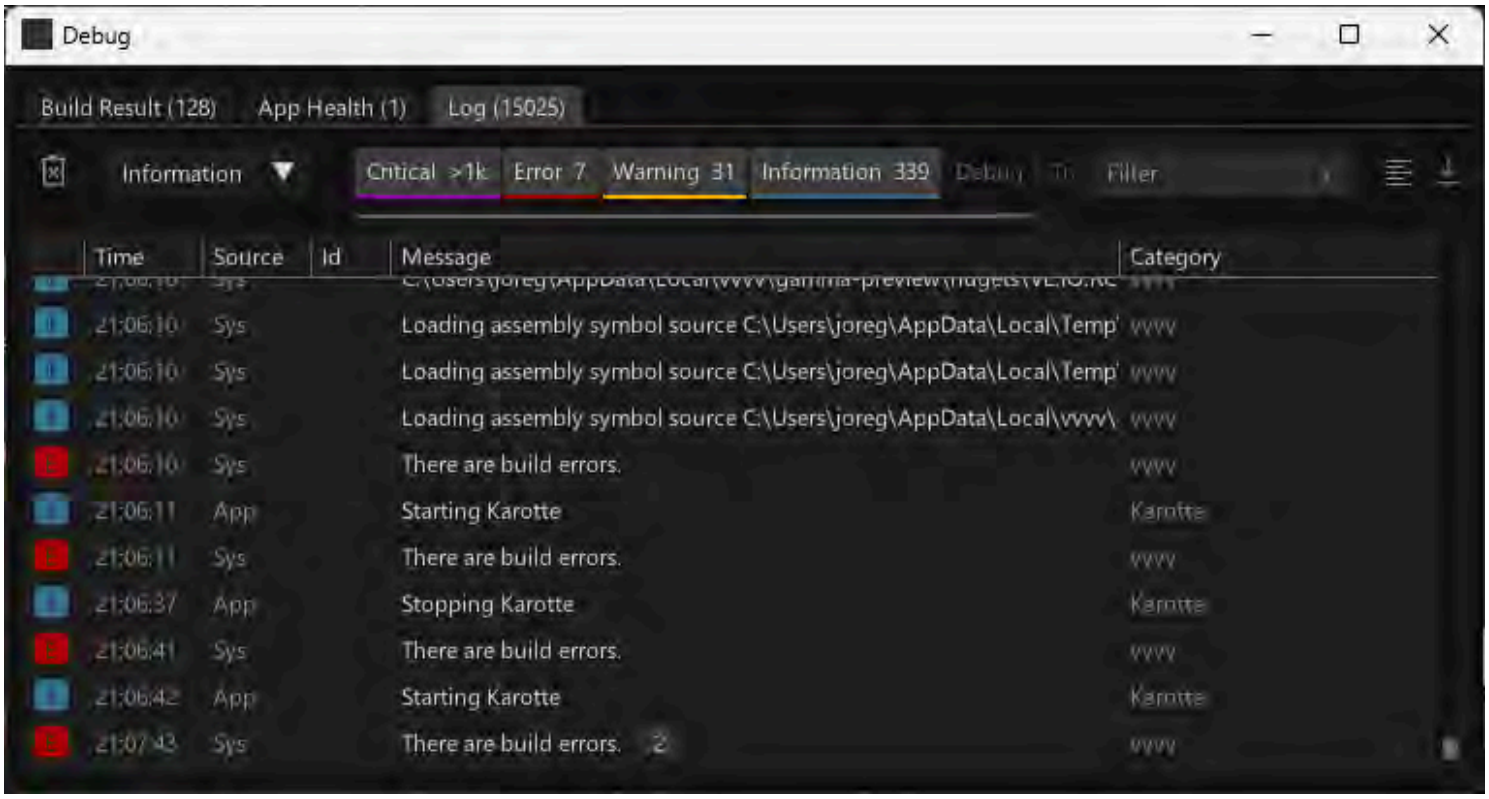


Exceptions are critical and should be taken seriously. Your application crashed. Errors, Warnings and Infos are messages from the library developer. The app didn't crash, but there is something fishy, which you should probably take another look at.

Note that same as the Build Result view, also this view is ephemeral in that it always only shows the current state of affairs. If you're looking for a history of issues your app ever had, consult the Log.

Log

The log is collecting all messages over time and shows you the whole history, until you clear it or the buffer runs over (Buffer length defaults to 5000 but can be configured via the Settings).



Severity Threshold

Every log message has one of the following severities assigned:

- Critical
- Error
- Warning
- Information
- Debug
- Trace

Via the severity pulldown (defaulting to "Information") you can set a hard log-level threshold, meaning that any messages with a lower severity level than the chosen threshold will be ignored.

Filtering Severities

All log-messages that pass the severity threshold will be visible in the list. To filter for certain severities you can toggle the severity buttons right next to the severity threshold pulldown. Pro-tip: Rightclick a severity to solo it!

System vs. App messages

The log-messages you see in the list view can be coming either from the system (vvvv itself) or from your app. To distinguish between those, see the "Source" column.

Logging from your patches

To create log messages from your patches use the Log [System.Logging] node.

Log Providers

The interesting thing with logging is that you can also route logs to any logging provider you prefer. In a real-life project you may want to log certain messages to a file and others to a cloud service. Anything is possible, see at "HowTo Configure logging providers" in the helpbrowser.

Log UI in your own application

When exporting your application, by default you've now lost the ability to view your log messages, since the above Log window is part of vvvv itself and not your app.

Yes you still have all the possibilities to use any thirdparty log providers but you may also want to have a log window as part of your application, see "HowTo Use the log view in an exported application" for how to set this up.

Log issues during startup of vvvv

If you encounter issues already while vvvv itself is starting up, run vvvv.exe using the `--log` commandline argument and then inspect the vvvv.log file being created in:

```
%UserProfile%\Documents\vvvv\gamma\
```

Extensions

Editor extensions are little tools that can be installed to enhance vvvv.

Examples:

- The Key & Mouse display: Shipping with vvvv. Displays key and mouse actions in the desktops lower left corner. Useful for recording tutorials or during live workshops.
- Pipette: A desktop color picker. Hover the mouse over any pixel of the desktop and press ESC to get the pixels color as a hex-string in the clipboard (which you can paste into any color IOBox)
- TUIO Simulator and Monitor: Are you working on a TUIO multitouch project but don't have the touch device at hand for testing? This simulator and monitor will be your friends.
- Spout Monitor: Are you relying on textures coming in via Spout from other programs? Use this monitor to get a quick overview of textures currently shared on your system.

Finding and installing extensions

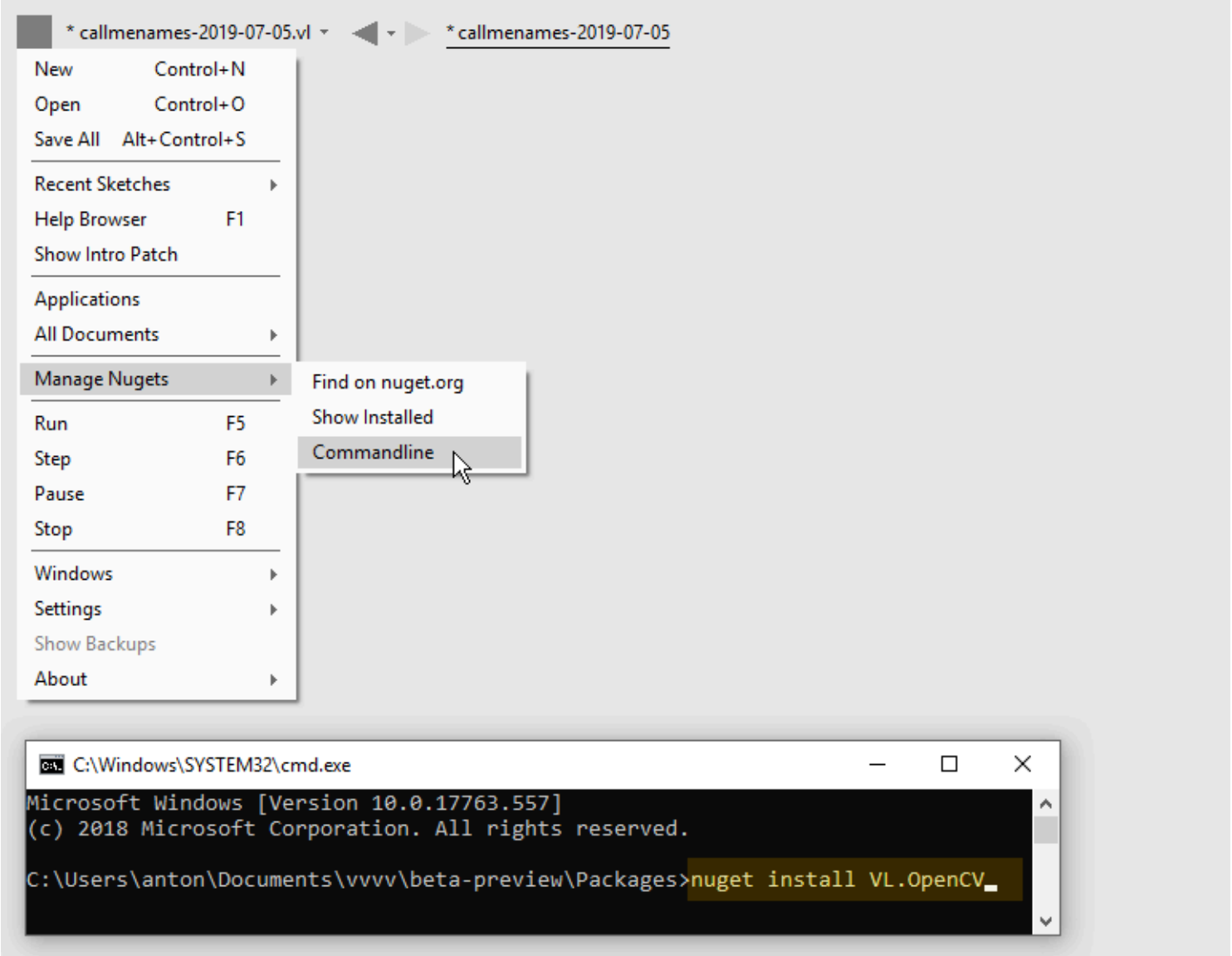
Extensions are distributed as NuGets and as such can be included with any VL NuGet you install. There are also NuGets that only include extensions. Those can easily be found on nuget.org.

Once found, extensions are installed like any other NuGet.

Creating extensions

Extensions are only patches, so it is easy to create your own. See Creating Editor Extensions.

Managing NuGets



Installing a nuget via the commandline

Find on nuget.org

Opens a webbrowser and lets you search for NuGets in the online repository. Note that for now, after you've found what you're looking for, you'll still have to install a NuGet manually via the commandline.

Show installed

Opens a file explorer at the path all your nugets are installed locally. By default this is in your users directory:

```
AppData\Local\vvvv\gamma\nugets
```

but can be changed using the commandline argument "nuget-path", see Commandline Arguments.

Commandline

Opens a commandline from which you can run nuget commands.

Installing the latest version of a NuGet

To install the latest version of a NuGet simply run `nuget install`, like e.g:

```
nuget install vl.opencv
```

After installing a NuGet via the commandline it is not yet automatically referenced by the current document! It is now merely available among the NuGets via the Dependencies menu. But if an installed NuGet contains any help files, those will already show up in the HelpBrowser.

Note

To update to the latest version of a NuGet you have already installed, simply run the install command again. Your package directory can contain multiple versions of the same nuget and VL will always use the newest. Older versions have to be removed manually if no longer needed.

Installing a pre-release version

Sometimes you may want to install a "pre-release" version of a package. To install those, add the argument `-pre` when running the install command, like e.g:

```
nuget install vl.audio -pre
```

Installing a specific version

Sometimes you may want to install a specific version rather than just the latest. In this case, use the `-version` argument when running the install command, like e.g:

```
nuget install vl.opencv -version 2.1.0
```

Note

vvvv will always load the newest version of a NuGet it finds installed. So in case you want to run a specific older version of a package (downgrade), make sure you don't still have newer versions of it and its dependencies around.

To remove versions of a NuGet, go to Show Installed and delete respective folders from there. If deletion fails, make sure to close vvvv first.

In the NuGet Compatibility Chart we are collecting packages and their recommended versions for specific vvvv releases.

Troubleshooting

If running the `nuget install` command returns with an error hinting at the nuget not existing but you're certain that it does and you spelled it correctly, there is an off-chance that your NuGet.Config file is corrupt. You'll find it in:

```
C:\Users\..\AppData\Roaming\NuGet
```

Rename the existing file to have a backup just in case. Then try to run the install again, which will recreate a working version of NuGet.Config automatically.

Show & Tell

The Show & Tell extension allows you to quickly share screenshots of your renderings with the world.

Press `Ctrl` `3` to open it.

When open, it intercepts any screenshot you make. Use either:

- `Win` `Shift` `S`: The default windows shortcut to take a screenshot
- `Ctrl` `2`: Works on Skia and Stride windows

Screenshots augmented with some text can then be posted to our common madewithvvvv account. In addition you can configure ShowAndTell to post to one of your own Pixelfed or Mastdon accounts and thus build a beautiful portfolio of your work.

Configuring Accounts

Press the `gear` button to open the configuration page.

The common "madewithvvvv" account

This is the default account that anyone can always post to. If you want your images to be associated with your user name, provide your vvvv.org login credentials instead of guest/guest.

Custom Accounts

If you have either a Pixelfed or Mastdon account, you can also post to those.

Press the `Add Account` button and specify:

- A name that identifies the account for you
- The instance (without http://, e.g.: pixelfed.de)
- The access token

Pixelfed access token

Note

For issues we're still investigating, this does currently not work with the pixelfed.social instance! There are many other instances to choose from.
In your Pixelfed account, go to `Settings > Applications > Create New Token`

Fill out the form, make sure to enable the `Write` scope and press `Create`. This will prompt you with an access token that you copy-paste into the account configuration.

Mastodon access token

In your Mastodon account, go to `Preferences > Development > New Application`

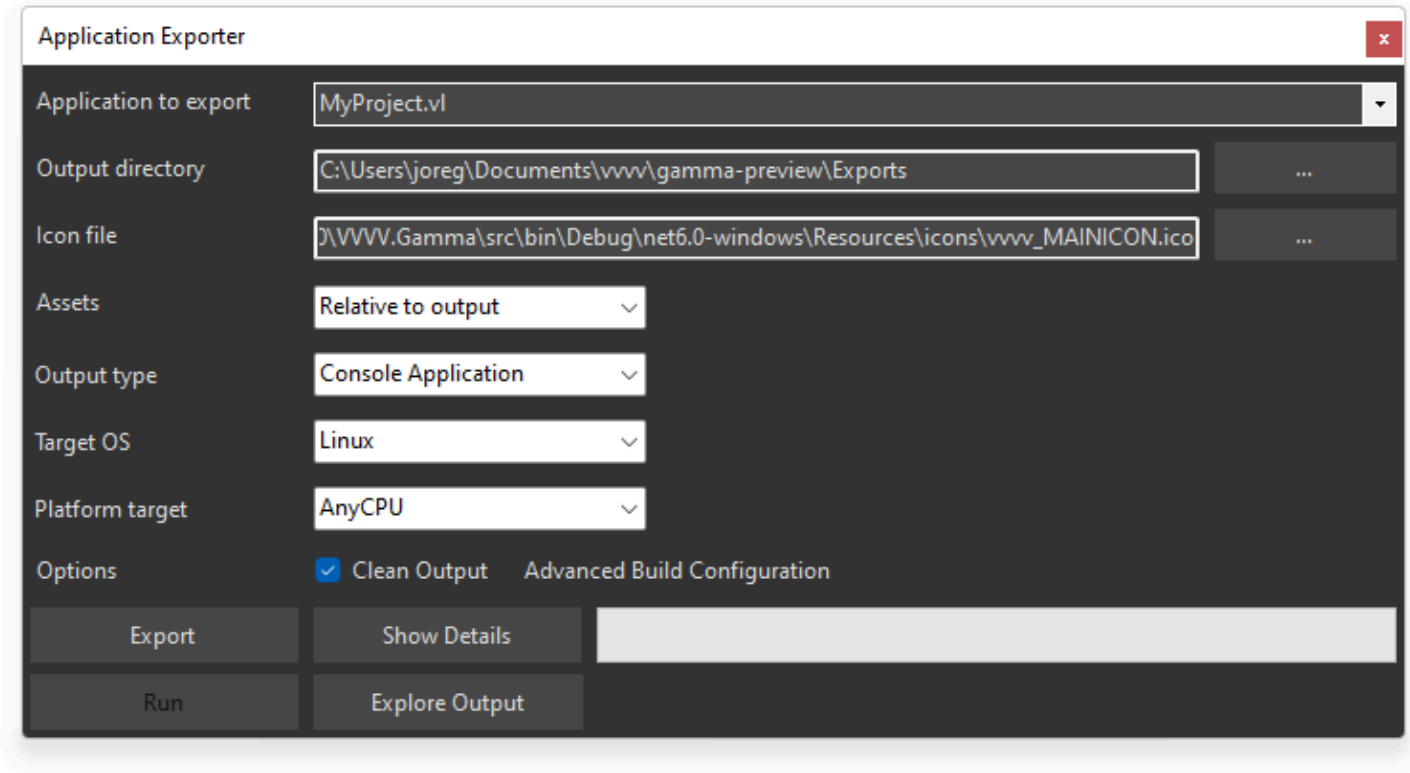
Fill out the form, make sure to enable the `Write` scope and press `Submit`. This will prompt you with an access token that you copy-paste into the account configuration.

What the Pixelfed and Mastodon?

Pixelfed and Mastodon are part of the larger Fediverse, a federation of decentralized social networks. In contrast to the more well-known legacy walled-gardens, those work more like e-mail: Anyone can run their own instance of a Pixelfed or Mastodon server and still communicate with anyone else, because the fediverse shares a common communications protocol: ActivityPub. No ads, no tracking, no blockchain, no crypto. Just a simple way to share images and text over the internet.

Exporting Applications

vvvv allows you to export a patch into an executable, standalone program. In order to do so, open the Application Exporter via `Quad > Export...` or shortcut `F10`.



The Application Exporter

Application to export

Choose which application to export (in case you have multiple projects open at the same time).

Output directory

Choose where the exported program and files will be created.

After a successful export, the output directory will contain a directory with the name of your application. Inside this directory you find the executable. To run the program on another PC you need to copy the whole content of this directory.

Icon file

Choose an .ico file to be associated with the generated executable.

Assets

Choose how assets will be referenced in the exported application:

- **Relative to document**: This option is most useful during development, to quickly test exports without having to worry about moving assets around. Assets will be referenced from where they are.
- **Relative to output**: Use this option for final exports: It requires you to manually place your assets relative to the generated executable as they were relative to your root document during development. Like this the whole output can then be moved around and deployed to other PCs.

Output type

Choose between Windows (GUI) or Console application. A Console app will open a Windows Console and run the Update operation for only one frame, then immediately Dispose itself.

Use a **KeepAppAlive** node to prevent this default behavior.

Target OS

Choose the OS for which to create output for. If you choose *Any*, export will create executables for all available targets, otherwise only for the one selected OS.

Platform target

Choose between CPU architectures x64 (64bit), x86 (32bit) or any.

Options

Console App

Choose between Windows or Console app. A Console app will open a windows Console and run the Update operation for only one frame, then immediately Dispose itself.

Use a **KeepAppAlive** node to prevent this default behavior.

Clean Output

If active, removes artefacts of previous exports (ie. deletes the \src folder) before exporting. This will cause exports to take longer but also makes sure previous artefacts don't interfere with the new export.

Export

- Press the **Export** button and wait until the green progressbar is full and the **Run** button becomes available
- Press **Run** to test/run your program
- **Explore Output**: opens a file explorer at the specified output directory

Source Directory

Next to the application directory you'll also find a `\src` directory. This is an artefact that vvvv creates during export and can be safely deleted.

Note

.NET developers may find this interesting though, as it contains a completely valid c# solution of the exported project that can be opened, viewed and modified with Visual Studio.

Dependencies

If your application is referencing VL.Stride, make sure the target PC also has the following dependencies installed:

- Microsoft Visual C++ Redistributables: 64bit or 32bit
- .NET8 SDK (For FileTexture and FileModel nodes to work)

For versions prior to 5.0

For applications exported with this older version of vvvv, you'll also have to install:

- MSBuild Tools

Advanced build configuration

The build process can be customized in many details. Next to your main .vl file, place a .props file with the same name. This is actually an .xml file which you can configure to your needs using MSBuild syntax.

Useful nodes

- Args [System] to access commandline arguments the app was called with
- Nodes from the advanced [System.Console] category

Useful libraries

- Terminal.Gui for creating console applications with a text based UI

Configuring a renderers appearance

Referencing the nuget VL.CoreLib.Windows adds the following nodes:

- SetWindowState and WindowState
- SetWindowMode

These allow you to configure the renderers caption, controlbox, framing and more.

Code Signing

In order to have your executables to run without a warning on other PCs, you need to sign them with a certificate using SignTool.

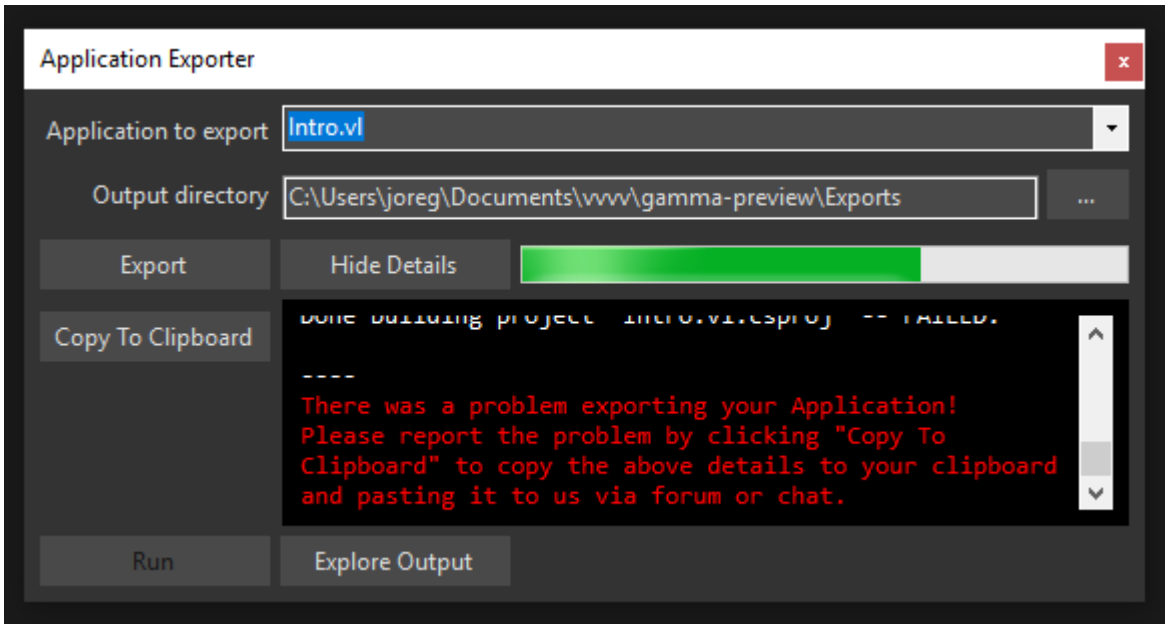
Troubleshooting

Exported app doesn't run on target PC

Chances are that you're missing a dependency on the target PC. See Dependencies above.

Export fails

In case the export fails, the console will be opened to show there was an error.



The Application Exporter reporting a problem

Export fails with error MSB3073

This error originates from Stride. If you're facing this issue, please try this workaround:

- From within vvvv open the NuGet command line `Quad menu -> Manage NuGets -> Commandline`
- From the commandline execute:
  - `nuget install System.Security.Cryptography.Pkcs -version 6.0.4`
  - `copy System.Security.Cryptography.Pkcs.6.0.4\lib\net6.0\System.Security.Cryptography.Pkcs.dll %userprofile%\.nuget\packages\stride.core.assets.compilerapp\4.2.0.2121\lib\net8.0`
- Now try the export again

Export fails with "..Found multiple publish output files with the same relative path.."

If the file in question is `ijwhost.dll` it might work to specify:

```
<ErrorOnDuplicatePublishOutputFiles>false</ErrorOnDuplicatePublishOutputFiles>
```

in the .props file you can edit via `Advanced build configuration` as explained here.

Export fails with NuGet dependency issues

Read the red error message carefully. There will be a reason given for the problem you're facing. If that reason hints at incompatible packages, you may have accumulated packages in your NuGet folder over time, which prevent the export from succeeding. In such a case try starting from a clean NuGet folder. Here are the steps to do so:

- Open your NuGet folder: `Quad menu -> Manage NuGets -> Show Installed` opens your `C:\Users\..\AppData\Local\vvvv\gamma\nugets` folder
- Close vvvv
- Go one level up from your NuGet folder and rename it so you have: `C:\Users\..\AppData\Local\vvvv\gamma\nugets_backup`
- Open vvvv and your patch
- The document menu will be red, meaning it's missing referenced dependencies
- Click the `Document menu -> Dependencies` and rightclick to select all missing dependencies, then choose: "Install exact referenced version"
- Wait for all packages to be installed again, then try the export again

Export fails with "..could not copy file.."

This may happen when you have packages referenced as source repository. To still get a successful export in such cases you can try:

- In the export log find the line starting with "dotnet publish -c Release" and copy the whole commandline
- close vvvv
- open a commandline (cmd.exe)
- paste the command and run it

Export fails With vvvv gamma 2021.4.x

There is a known incompatibility with newer versions of MSBuild tools than what vvvv expects. So in order to make sure the right version of MSBuild tools is installed do as follows:

- Uninstall all versions of Visual Studio and Build Tools you can find on your machine
- Then run the vvvv 2021.4.x installer again with having the “Build Tools” checkbox enabled

None of the above

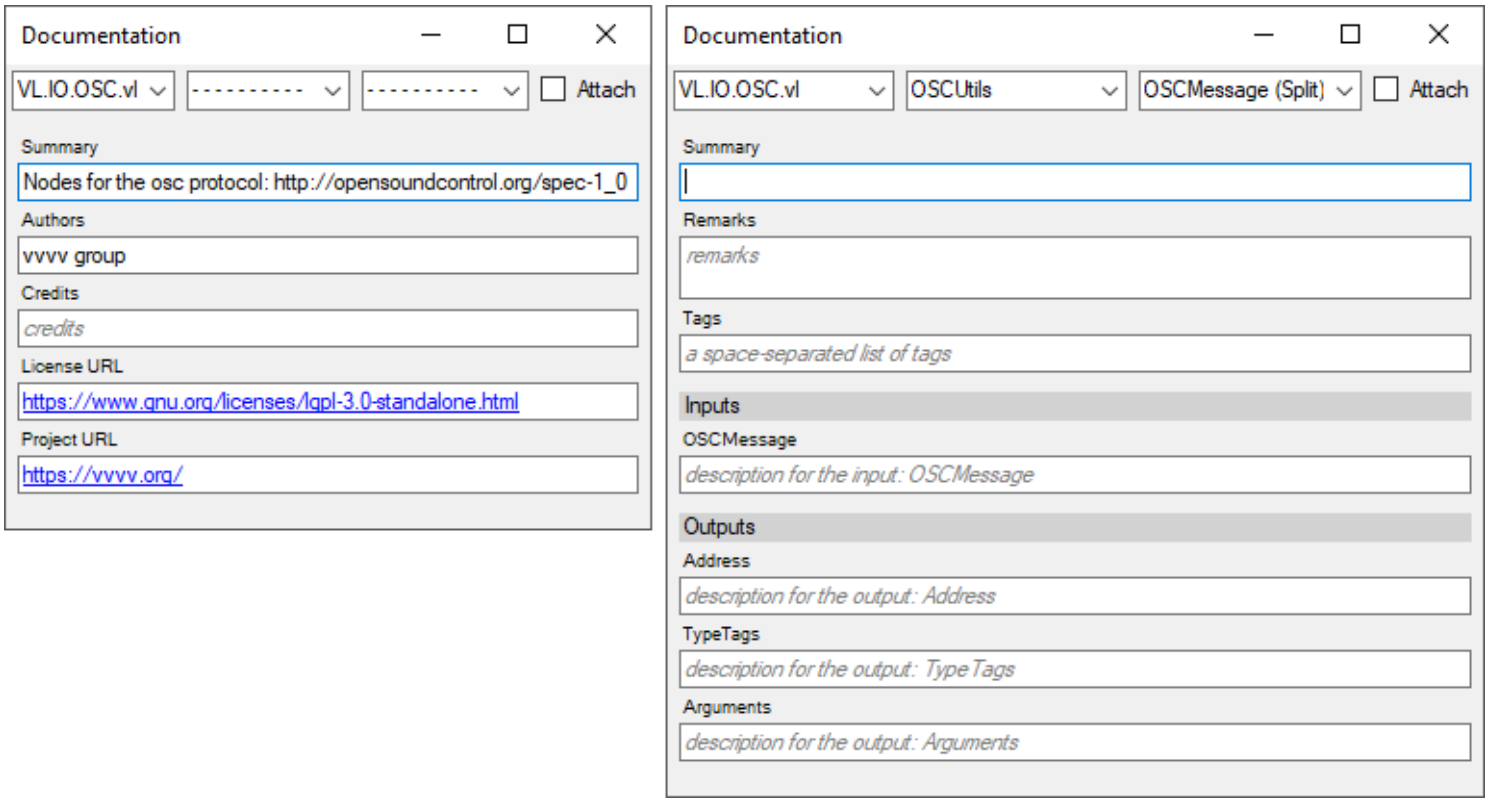
Please send us the console output by pressing "Copy To Clipboard" and pasting it to us via the forum.

Documentation Window

Using the documentation window you can add documentation to all your elements. Press `Ctrl` `M` to open it and then use the 3 pulldowns on top to navigate:

- Documents
- Patches in the selected document
- Operations in the selected patch

As long as the *Attach* checkbox is not activated navigating between the patchs in the editor lets the documentation window follow the active patch.



The Documentation Window

Specifying documentation is mostly useful when preparing libraries that you want to share with others. When working on documents for your own projects you'll hardly ever use it.

Document

- Summary: One sentence description of the content of the document
- Authors
- Credits
- License URL
- Project URL

Datatype

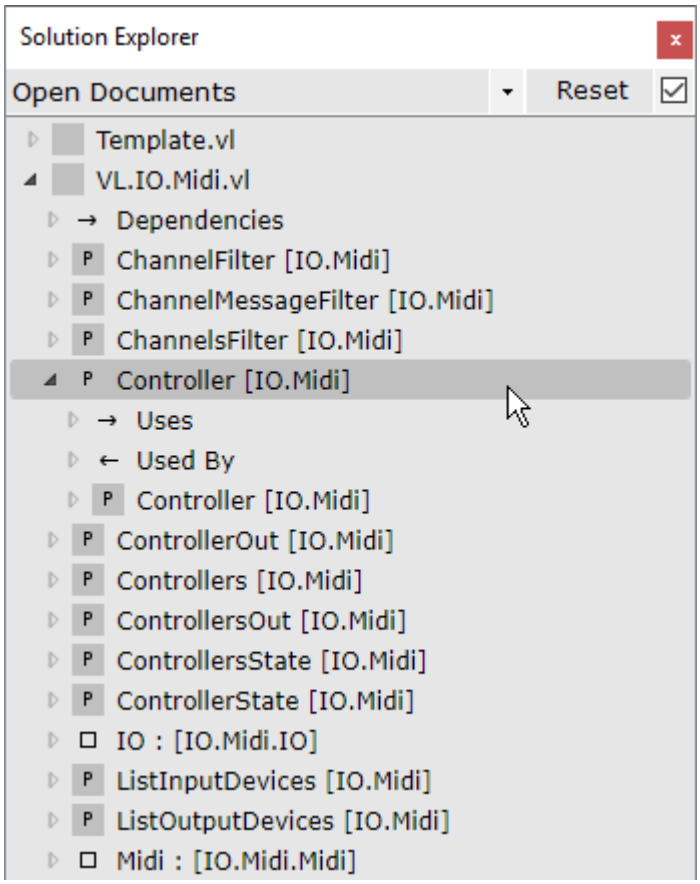
- Summary: One sentence description of the datatype
- Remarks: Additional usage info, warnings, bugs,...
- Tags: a comma-separated list of tags to find the datatype in the NodeBrowser
- Properties: description of each individual property of the datatype

Operation

- Summary: One sentence description of the operation
- Remarks: Additional usage info, warnings, bugs,...
- Tags: a comma-separated list of tags to find the operation in the NodeBrowser
- Inputs: description of each individual input of the operation
- Outputs: description of each individual output of the operation

Solution Explorer

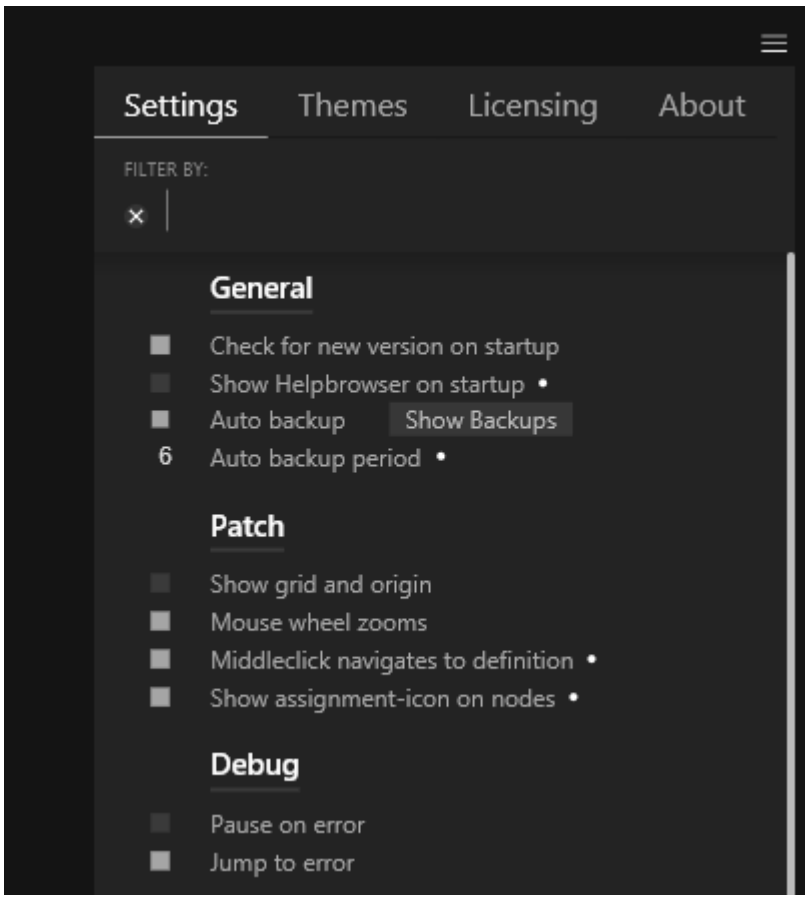
The Solution Explorer can be opened by pressing `Ctrl` `J` and allows you to get an alternative overview view and navigation for your project.



The Solution Explorer

Settings

The settings can be found in the Hamburger menu in the top right corner of the editor window:



In versions prior to 2021.4, the settings had to be edited manually in a text file, see: `Quad > Settings`.

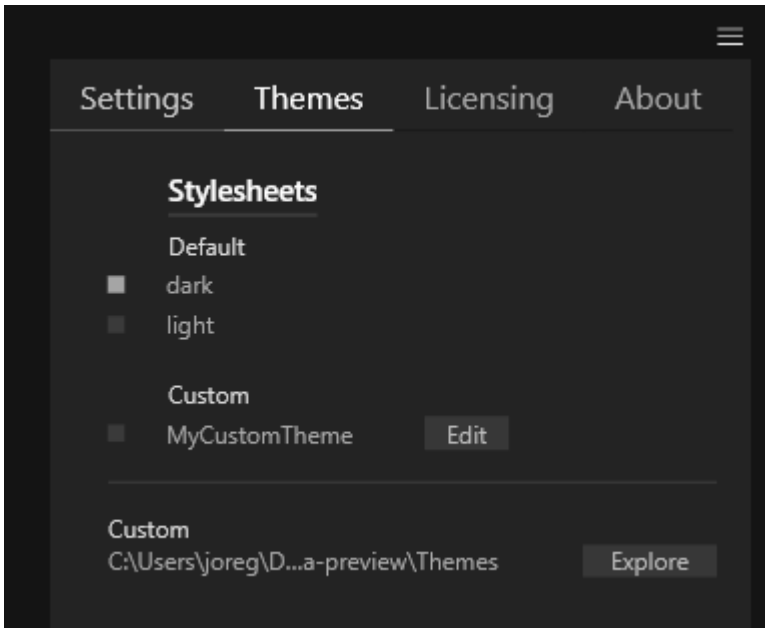
Local Settings

You can also provide a local settings file via commandline options when you start up vvvv.exe:

```
--settings C:\path\to\setings.xml
```

Themes

The vvvv editor supports theming. You can quickly switch between a dark and light theme via the Hamburger menu in the top right corner:



Custom Themes

Custom themes can be created using (a limited set of) CSS:

- Copy one of the provided themes from `C:\Program Files\vvvv\vvvv_gamma...\Stylesheets`
- Modify it
- Place it in your user document folder `...\Documents\vvvv\gamma\Themes` to have them available in the Themes tab

Commandline Arguments

The following is a listing of commandline arguments that can be used when starting vvvv.exe:

Description	Argument
Allow to run multiple instances of vvvv in parallel	`--allowmultiple` or`-m`
Do not start the patch runtime on startup	`--stoppedonstartup`
Open a VL document on startup	`--open` or `-o` + path to file
Override loading the default settings	`--settings` + path to file
Do not load editor extension packages	`--noextensions`
Replace the default global location for NuGets	`--nuget-path` + path to directory
List of package repositories used by VL to lookup packages. Most useful, when working on libraries.	`--package-repositories` + a semi-colon separated list of package repository directories
Tell the exporter to add a path to the generated NuGet.config file, which in turn will be used by dotnet to build the generated project	`--export-package-sources` + path to directory with .nupkg files
As of 5.0: Opt-out of the read-only default for libraries in order to work on them. For details, see Editable Packages.	`--editable-packages` + a semi-colon separated list of packages. Glob patterns are allowed, e.g. VL.IO.*
As of 5.3: Log issues during startup to %UserProfile%\Documents\vvvv\gamma\vvvv.log Example:	`--log`

```
vvvv.exe -o "c:\myproject\foo.vl"
```

Args.txt

A quick way to apply commandline arguments to vvvv is specifying them in an `args.txt` file in your Sketches folder:

```
C:\Users\..\Documents\vvvv\gamma\Sketches\args.txt
```

The language VL

VL is the name of the language used in vvvv. Here is a quick overview:

Language Components

- The most important components of VL are Nodes and Links
- Nodes and links live on a canvas called Patch
- One or multiple patches are contained in a *VL Document*. VL Documents are stored as files on disk
- Every patch can define one or more Operations. Operations can be created as nodes in other patches
- There are two kinds of patches, one is merely a container for independent operations the other is a data type. If a patch is a data type its operations can store and/or share data via Properties. Properties can be accessed and/or modified in operations via Pads
- One *VL Document A* can reference another *VL Document B* to use its data types and operations as nodes. File B is then called a *Dependency* of A
- Patches can also contain *Regions*. A region defines a new computational context. There are different kinds of regions
- Nodes and regions can have inputs and outputs called *Pins*
- Static data can be entered into an operation using IOBoxes. IOBoxes are little editors for basic types like numbers, text, color... They can also be used to display the current value of anything connected upstream.
- Links transport data from outputs to inputs. Therefore they define the data flow and execution order of the nodes in an operation.

Multi-Paradigm

- VL combines metaphors known from dataflow, functional and object-oriented programming
- It is strictly evaluated
- It comes with regions aka visual code blocks (loops, if, delegates, ...)
- It features Process nodes aka simple lifetime management
- It has Adaptive nodes aka adhoc polymorphism

The Type System

- VL is statically typed
- It has automagic type inference
- It has first class support for mutable and immutable datatypes
- It supports generics aka parametric polymorphism (with bounded quantification)
- And interfaces aka subtype polymorphism

Patches

A "Patch" is a canvas that holds Nodes, Links and other VL language elements. A VL document can comprise of many patches. There are two main types of patches:

- Datatype Patches
- Definition Patches

Every VL document has two main patches that can be reached via the Document Menu:

- The Application patch: A special form of a datatype patch
- The Definitions patch: The root of all definition patches

Application patch

The main entry point of a VL document. If any nodes are placed here, they will be executed as soon as the document is opened, either directly or as a dependency of an other document.

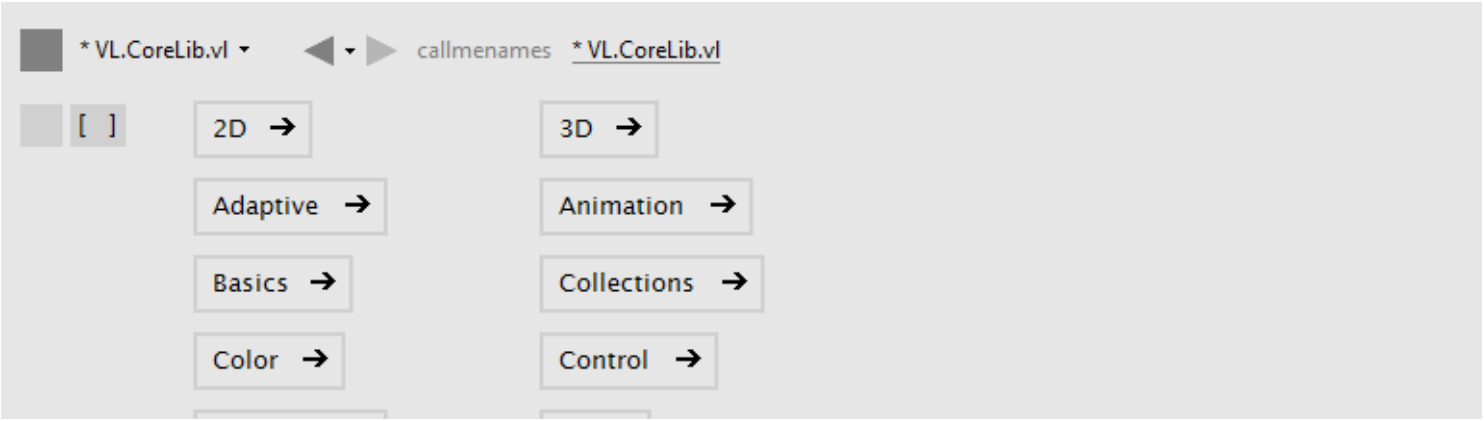
This is typically the place where you start creating your program. You can reach this patch via the shortcut `Alt` `A`.

Application patches can also hold definitions, but this is not considered particularly good practice.

In case a documents application patch is empty, the document is only used as a library, ie. only providing node definitions to any document referencing it.

Definitions patch

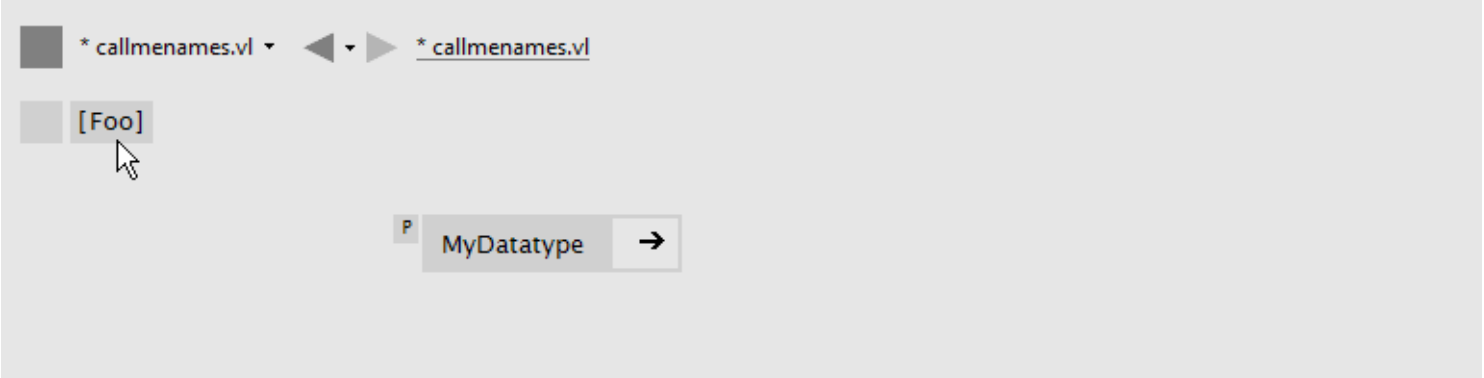
This is where all the node definitions of a VL document are placed. Here you can use Categories and Groups to build a hierarchy and organize your definitions. You can reach this patch via the shortcut `Alt` `Shift` `A`.



Section of the definitions patch of VL.CoreLib.vl

Here we typically see a range of type-definitions and categories, though a document patch can also directly hold static operations.

The document patch can set or omit a base category.



Document base category set to "Foo"

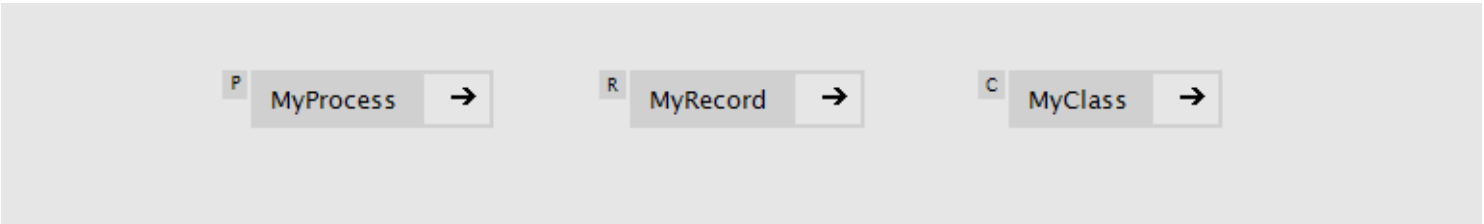
Datatype Patches

There are different types of datatype patches that can be switched between in the Patch Explorer:

- Process
- Record
- Class
- Interface
- Forward

Process, Record and Class patches can have Properties and Member Operations. Interface and Forward are a bit more special, see below.

Every datatype patch has a corresponding type-definition element in a definition patch.



Datatypes: Process, Record and Class

There are different ways to create a new datatype patch:

- In the NodeBrowser type the name of the node you want to create and then choose `Node` to create a process node
- Press `Ctrl` `P` to create a process node at the cursor and open the new patch
- Press `Ctrl` `Shift` `P` to open the new patch without creating a node application

In either case, a corresponding type-definition is automatically placed in the definitions patch of the active document.

Process

The most common type of datatype patch is the "Process". It holds the definition for a Process Node, ie. its life-time is bound to the existence of a node.

A processes' member operations can either be directly part of the process or not. The Patch Explorer can be used to decide about this for every operation. Also the order of execution of multiple operations in a process can be configured there by dragging the operations up or down.

The Application patch of a document is a special Process patch:

- It has a Create and an Update operation but doesn't allow you to add additional operations
- It cannot be instantiated as a node, but an instance of it is running as soon as its document is opened directly or as a dependency for another document

Record

Defines an immutable datatype. As opposed to a Process, its life-time is not defined by the existence of a node. Instead, any number of instances of a Record can be created, update and disposed at any time.

The typical life-time of a record goes like this:

- An instance is created using a call to its `Create` operation
- The instance is stored in a collection
- Operations like `Update` (or others that were defined) are called on it repeatedly or only from time to time and return a new instance (replacing the previous one) that is stored in the collection again
- For this, activate the Process toggle in the Patch Explorer
- The instance is removed from the collection in order to kill it. In case the record holds unmanaged resources it is also necessary to call its `Dispose` operation before removing it from the collection.

Every node that modifies a record type, essentially makes a copy of it (with changes applied) and returns a new instance. Thus, modified Records always need to be written back into a Pad, for their changes to survive to the next frame!

The fact that a record is at anytime a fixed, immutable snapshot of data, makes it specifically suitable for use in a dataflow programming language like VL.

A record can optionally also define a Process. For this, activate the Process toggle in the Patch Explorer.

Class

Defines a mutable datatype. Basically similar to the Record with one main difference:

Every node that modifies a class type really modifies the original instance! No matter how far down the line a node is that operates on a class type, it is always the original instance that is being modified. So what's passed on over links, from pin to pin, is not data, but only a reference to the original instance.

A class can optionally also define a Process. For this, activate the Process toggle in the Patch Explorer.

Interface

Not officially supported yet.

Forward

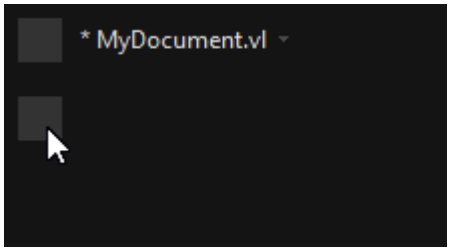
See Forwarding.

The Patch Explorer

Gives a quick overview of elements in a patch and allows to configure the patch name and type and further properties depending on the type.

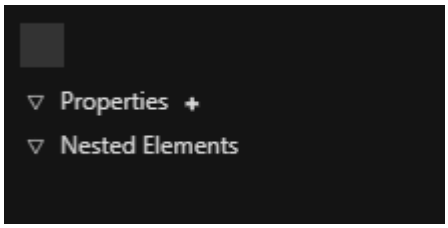
Showing and Hiding the Explorer

By default the patch explorer is not showing. It's visibilty can be toggled by clicking the lower of the two Quad icons in the top left corner of the editor:



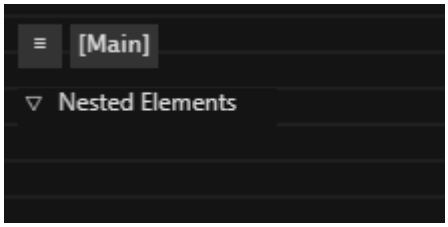
Depending on the type of patch, the explorer shows the relevant information:

Application Patch Explorer



- Does not allow to specify a name
- Lists all Properties of the patch and allows to add/remove them
- Lists additional nested elements, like Datatype Patch definitions and Static Operation definitions

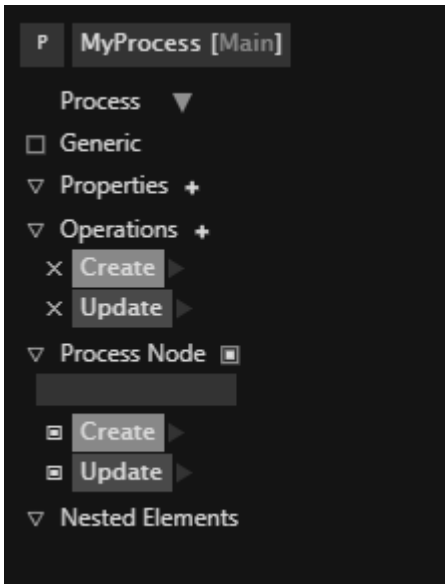
Definition Patch Explorer



- Allows to specify a Category which is applied to all elements in the document
- Lists nested elements, like Datatype Patch definitions and Static Operation definitions and Categories

Datatype Patch Explorer

Process/Record/Class



- Allows to specify the datatypes name
- Allows to set the type of datatype patch
- For types Record/Class only: Allows to specify an Aspect
- Allows to specify whether the datatype can have generic inputs/outputs
- For types Record/Class only: Lists all Interfaces and allows to add/remove them
- Lists all Properties of the type and allows to add/rename/remove them
- Lists all Member Operations of the type and allows to add/rename/remove them
  - On each operation the Signature can be shown and manipulated
- Allows to configure the Process Definition
  - Enable/Disable the Process
  - Set an Aspect
  - Enable/Disable the State Output
  - Define which operations are part of the Process by toggling their checkbox
  - Define the order of operations in the Process by dragging the operations up/down
- Lists nested elements, like Datatype Patch definitions and Static Operation definitions

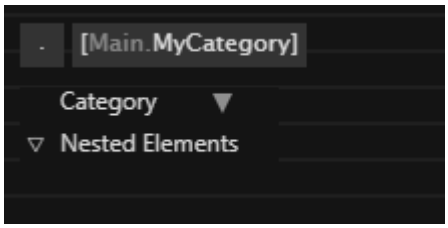
Interface

Not officially supported yet.

Forward

See Forwarding .NET Libraries.

Category Patch Explorer



- Allows to specify a name for the Category or Group
- Allows to change the type of Category
- Lists nested elements, like Datatype Patch definitions and Static Operation definitions and Categories

Nodes

Nodes are the main building blocks of a patch. They have input Pins at the top and output Pins at the bottom. Pins are the hubs that allow Nodes to be connected via Links.

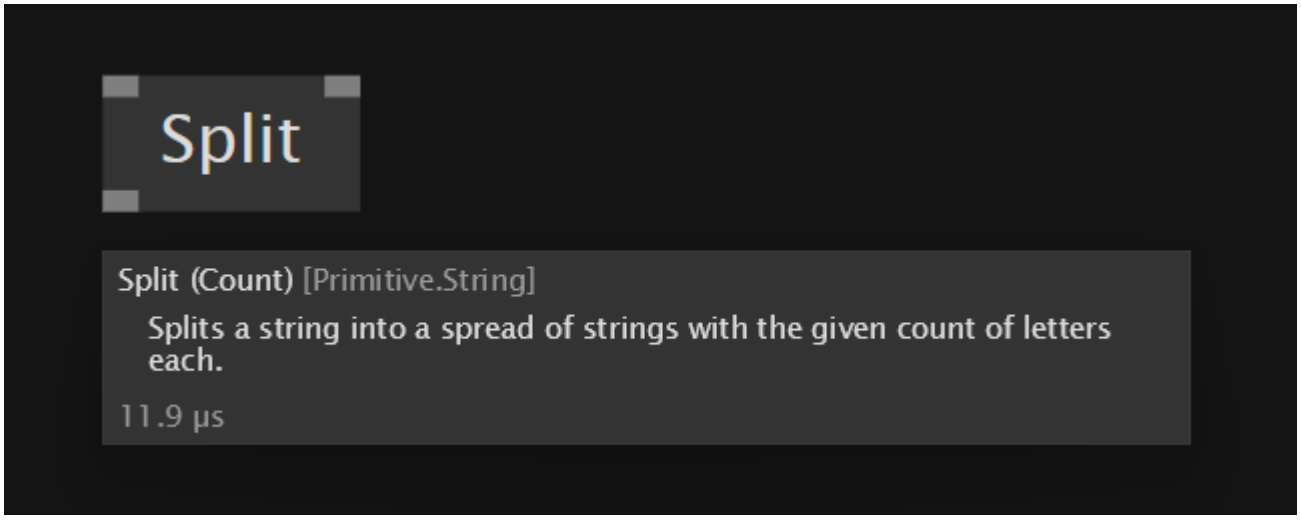
Nodes are also referred to as the "application" of a node definition.

Node Name

A nodes name consists of the following components:

- Its display name
- An optional version
- Its category (think namespace)

Hovering a node to view its tooltip shows its full name:



The node "Split" of version "Count" in the category "Primitive.String"

Types of Nodes

There are different types of nodes:

Image: Process node, Static Operation node, record operation node, class operation node

Process Nodes

A process node represents a single instance of a patch.

The name *Process* comes from the fact that it can be understood like a little machine that does some initialization routine (the `Create` operation) once it is first executed and then continues to execute one or more of its operations in a loop, maintaining an inner state from one frame to the next. Typically, a Process node has at least an `Update` operation, but it is not limited to that.

Process nodes have a distinguished look with the bars behind their pins being darker. This makes them visually heavier, hinting at the fact that they are holding state, ie. storing data between consecutive executions.

For more on defining Process Nodes, see Datatype Patches.

Static Operation Nodes

Operation nodes are nodes representing a single operation.

They have a lighter look than Process Nodes, with the bars behind their pins not being visible. This indicates that they are not operating on a state, ie. they do not store any data between consecutive executions.

Apply input

If the first input and first output of a static operation node share the same datatype, they can have an *Apply* pin added via `context-menu > Configure`.

The Apply input defaults to true. When disabled, the operation is bypassed and the input returned unchanged as the output.

This is essentially a shortcut to surrounding the node with an If Region.

Record Operation Nodes

Record operation nodes are part of a record, on which they operate. They are visually higher, as they also display the name of the datatype they belong to in smaller type, below the nodes name.

They have an optional "State Output" pin that is visually not connected to the corresponding "State Input" pin, indicating that the object going out is always a completely new object, cloned and modified from the incoming object.

Class Operation Nodes

Class operation nodes are part of a class, on which they operate. They are visually higher, as they also display the name of the datatype they belong to in smaller type, below the nodes name.

They always have a "State Output" pin that is visually connected to the corresponding "State Input" pin, indicating that the object coming in is the same as the object going out, only modified.

Optional Pins on Nodes

Nodes can have Pins that are not visible by default. Rightclick a Node and press Configure to show a little inspector that allows you to show/hide optional Pins.

Pin groups

Some Nodes have Pin groups, which allow you to change their number of pins.

Examples of nodes with Pin groups: Group, Cons, +

Typically a Node has either a single input or output Pin group, in which case it pins can be added/removed by pressing `CTRL` `+` or `CTRL` `-` respectively.

For keyboard shortcuts in case of multiple Pin groups on a node, see: Pin group shortcuts.

Navigating to a Nodes definition

If a node is defined by a patch, you can navigate to its definition via pressing `RightClick -> Open` on the node. Any node that spots an arrow icon has a patch behind in the same document or a document that is directly referenced as a file dependency. This patch can quickly be opened via middle-clicking the Node.

Image:Node with a patch behind it

See also the setting "Middleclick navigates to definition" to enable the middleclick to navigate to any patch even if it is not in the same or a referenced document.

If a node is defined by SDSL shader code, the corresponding code editor will open. See Editing Shaders.

Nodes that are defined by C# code cannot be inspected.

Links

Links are the connections between pins on which data flows from one node to another. There are 3 different kinds of links:

Image:Normal Link, Reference Link, Delegate Link

In datatype patches, links can have colors, which tells you which member operation they belong to.

A "yellow sock" on a link is a warning that the source of the link is mutable and it connected to more than one downstream node. Please read the full explanation in the tooltip to learn what this means and how you can deal with it.

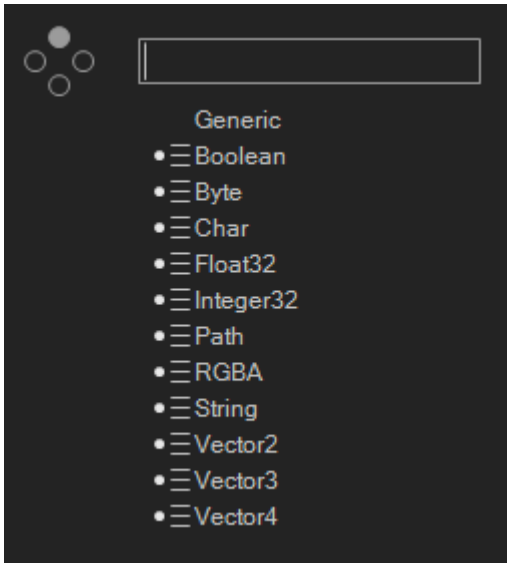
For many different actions you can do while creating a link, or on existing links, see Link Shortcuts.

IOBoxes

Short for "Input/Output boxes", they allow you to *input* constant data into your program or *output* it for debugging or display purposes.

Some IOBoxes of different types

You typically create an IOBox by starting a link from an input or output pin and then using a Middleclick (or `ALT` + leftclick) to create the according IOBox. Alternatively you can create an IOBox by selecting one from the nodebrowser popping up when you right doubleclick in the patch.



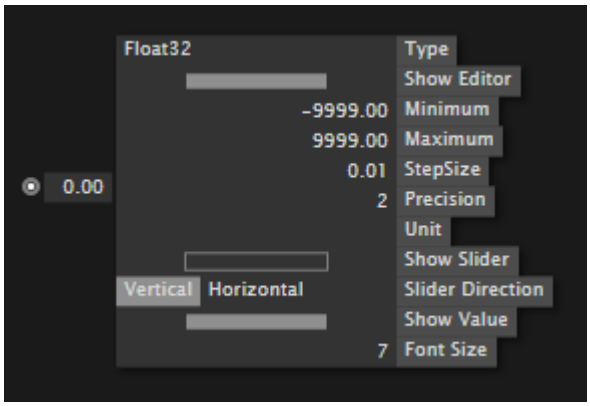
Choose an IOBox from the nodebrowser after double rightclick

An IOBox exists for each primitive datatype and they all have some special features which you can learn about here:

Configuring IOBoxes

Configuring IOBoxes via their inspector works the same for all of them:

- Middleclick the IOBox
- or Rightclick its label and select `Configure`.



Configuring a number IOBox

Numbers

Number IOBoxes work the same for whole (integer32, byte, ...) and real (float32, ...) numbers:

- Doubleclick to enter a value via keyboard

Note

You can also enter math formulas like, e.g.: "1/3" that will be immediately be evaluated and fill the IOBox with the result

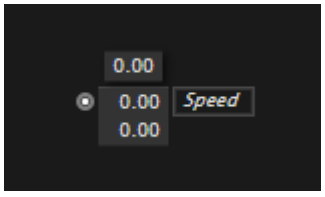
- Right-drag up and down to change the value gradually ** hold `SHIFT` while dragging, to divide the step-size by 10 ** hold `CTRL` while dragging, to divide the step-size by (another) 10 ** hold `ALT` in combination with the above to multiply instead of divide the stepsize
- `ALT` + Rightclick to reset the value to its default

Via the inspector you can configure the IOBox:

- Choose minimum and maximum and stepsize that will be taken into account when right-draging the value
- Choose a display precision
- Choose to display a unit for the value. The unit has no effect on the value and is only good for display purposes
- Choose to show a horizontal or vertical slider

Vectors

For Vector IOBoxes you can foremost configure whether you want to view their individual components in a vertical or horizontal stack.

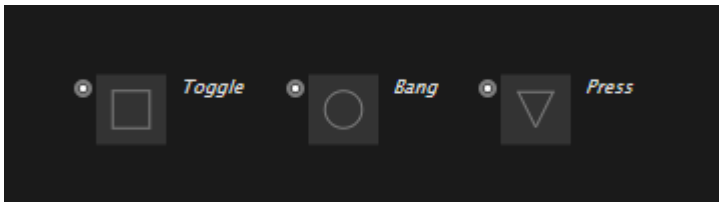


A 2d vector IOBox with the all-components editor visible

Also you can change all components at once by changing the value popping up above the IOBox when hovering it.

Booleans

A boolean IOBox has three different button modes:



The three modes of a boolean IOBox

- Toggle: a rightclick toggles between TRUE and FALSE
- Bang: a rightclick makes it return TRUE for one frame, otherwise returns FALSE
- Press: returns TRUE as long as it is right-pressed, otherwise false

Strings

Changing values in string IOBoxes works as follows:

- Doubleclick to enter text via the keyboard
  - While entering text press `CTRL` `ENTER` to add a new line
- `CTRL` + Rightclick to open the file chooser dialog
- `SHIFT` + Rightclick to open the directory chooser dialog

Via the inspector you can configure the IOBox:

- Choose one of three string types:
  - *String*: the default
  - *Comment*: to put a comment in a patch
  - *Link*: to put a link as comment in a patch that open in the browser on rightclick
- Choose to visualize non-printable characters (ie. those with an ascii code < 32)

The three different types of string IOBoxes

Colors

Color IOBoxes let you enter colors in many different ways:

- Doubleclick to enter the name of a color via the keyboard
- Doubleclick to enter the values of a color in different formats:
  - a string of type: "H:0.00 S:0.00 V:1.00 A:1.00" where each component (hue, saturation, value and alpha of the HSVA color model) is a value between 0 and 1
  - a string of type: "R:0 G:255 B:0 A:255" where each component (red, green, blue and alpha of the RGBA color model) is a value between 0 and 255
  - a string of type: "RRGGBBAA" where each of RR, GG, BB and AA are pairs of hexadecimal values from 0 to 255, eg for red specify: "FF0000FF"

Change colors via mouse interaction:

Description	Action
Change brightness	Rightdrag up/down
Change hue	Rightdrag left/right
Change saturation	`Ctrl` + Rightdrag up/down
Change the alpha channel	`Shift` + Rightdrag up/down

Paths

Path IOBoxes can be used to enter filenames or directories. By default they always assume you want to choose a filename!

Note

Path IOBoxes always store relative paths if possible but actually hide this fact from you! This will lead to confusion in the rare case that you actually want to specify an absolut path: While IOBox and tooltip will show the absolut path you entered, internally a relative path is stored. So if you really need to specify an absolut path, use a string IOBox followed by a ToPath [IO] node.

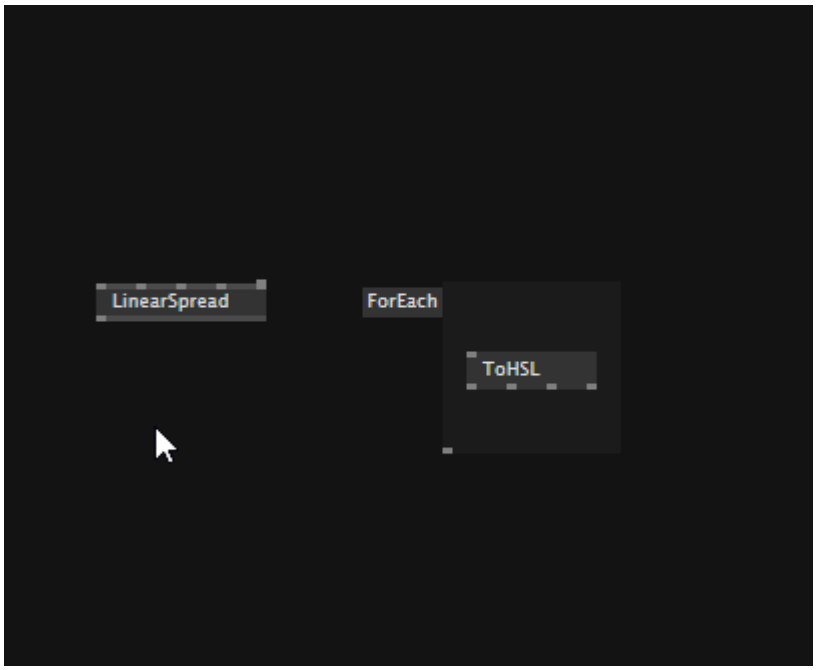
- Rightclick to open open the file chooser dialog
- `SHIFT` + Rightclick to open the directory chooser dialog
- Click the [O] icon to open the currently selected file with its associated application
- `ALT` + click the [O] icon to view the file/directory in windows explorer

Via the inspector you can configure the IOBox:

- Choose between *File* or *Directory* as path type which simply determines which dialog a rightclick on the IOBox will pop up

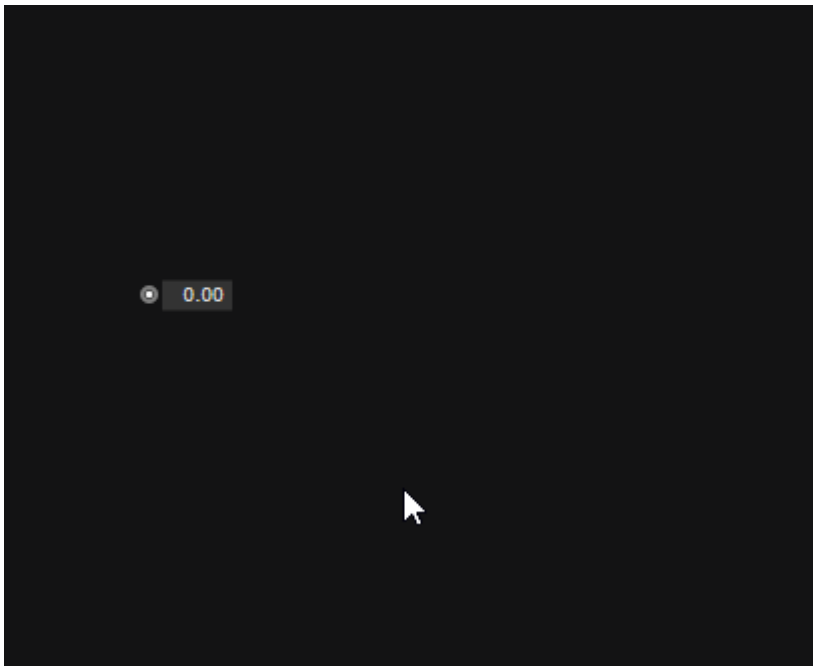
Collections

Collection IOBoxes work with all the above datatypes. Most often you'd create them automatically by starting a link from a pin that has a collection type (e.g. Spread, Sequence,...) and then middle-click to create the according IOBox automatically.



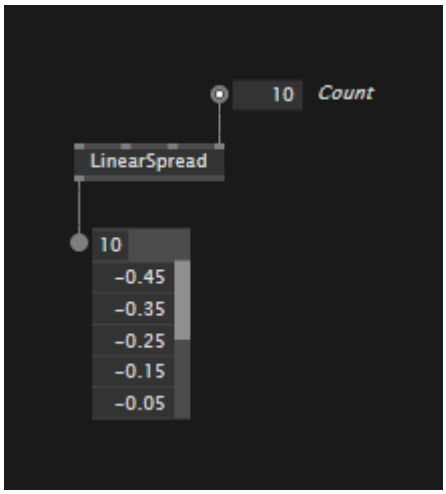
Middleclick to create an IOBox

If you want to manually create a collection IOBox, first create a normal IOBox and then configure its type to be of eg. `Spread<Float32>`.



Annotating the type of an IOBox

The number you see topleft in the IOBox specifies the number of elements in the collection and can be changed. By default a collection IOBox will display up to 5 elements. When the collection contains more items, a scrollbar will be shown.



A spread of floats inspected via a collection IOBox

Via the inspector you can configure the IOBox:

- The number of maximum visible entries
- Show/Hide the entry index
- Define whether the entries will be displayed as a vertical or horizontal stack
- Add/Remove entries

Operations

Operations define a simple functionality. They take input, do something with it and return a result. Operations cannot hold state, meaning they cannot store any data between consecutive calls. Data instead is stored in Properties.

Definition vs. Application

Using the term "operation" alone can be ambiguous if not from the context it will be clear whether we actually mean an "operation definition" or an "application of the operation definition" which again is synonymous with "node". In this chapter, we're using the term "operation" as shortcut for "operation definition".

Types of Operations

There are two different types of operations in vl:

- Member operations
- Static operations

Member Operations

The term *member* refers to the fact that those operations belong to and operate on the data of a datatype.

Datatypes can have many operations, most often they have at least a `Create` and an `Update` operation. To distinguish multiple member operations in a patch, VL uses colors for Pins and Links. There are three reserved colors:

- White: for the Create operation
- Gray: for the Update operation
- Dark red: for the Dispose operation

All other colors are applied randomly from a color pallette and have no meaning whatsoever. They are only there to indicate the belonging of colored elements to a certain operation. To check which color refers to which operation, use the Patch Explorer or hover the pin and find the operation mentioned in the tooltip.

Image:A member operation definition and its application as a node

Creating a Member Operation

Member operations are either created via the Patch Explorer, or during the assignment workflow, where you can choose to assign to a new operation and then specify the name of the operation to be created and assigned to at the same time.

Assigning Nodes, Inputs/Outputs and Links to operations

Use the elements context menu to assign it to one of the available operations or create a new one.

Often it makes sense to start assignments on Input or Output pins. Note that assignments auto-propagate through the whole patch. They only stop at Pads or Process Nodes, which act kind of like bridges between the operations in that they store values written by one operation and have them available for retrieval by an other operation.

There are cases though where no Input or Output pin is part of an operation. In that case consider setting an assignment onto a link or Operation Node.

Note

Process Nodes cannot be assigned to an operation. Instead you'll see that their Pins can assign to different operations, meaning that different parts (operations) of a Process Node can be executed on different operations in the containing patch.

Clearing Operation assignments

To remove the assignment from an element, also use the context menu -> Clear assignment.

By default, elements with no assignment will "fall back" to being executed on Update, if it is available.

The Dispose Operation

In cases where you deal with an unmanaged object it is necessary to add an operation named `Dispose` and use it to in turn execute a Dispose node for the object. This will make sure that when the parent patch is disposed, that the managed resource will be disposed as well.

Since there is no way to know whether an object is unmanaged, a simple test you can do, is try to connect Dispose [IDisposable] to an instance of it. If this connection is allowed, you know that the resource is unmanaged and needs you to manually call Dispose on it.

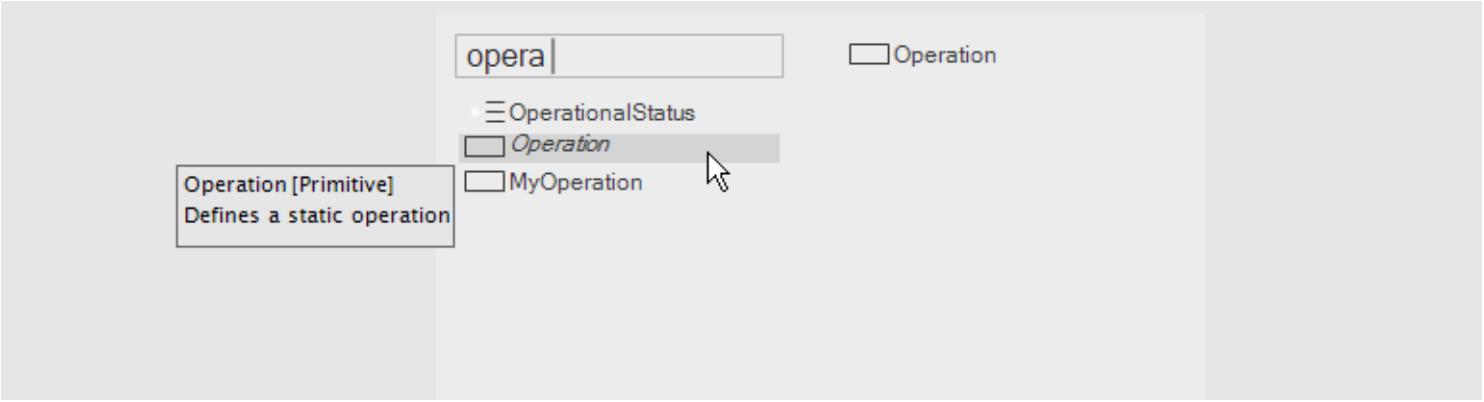
Static Operations

Static operations are on their own, operating only on data they are being fed with.

Image:A static operation definition and its application as a node

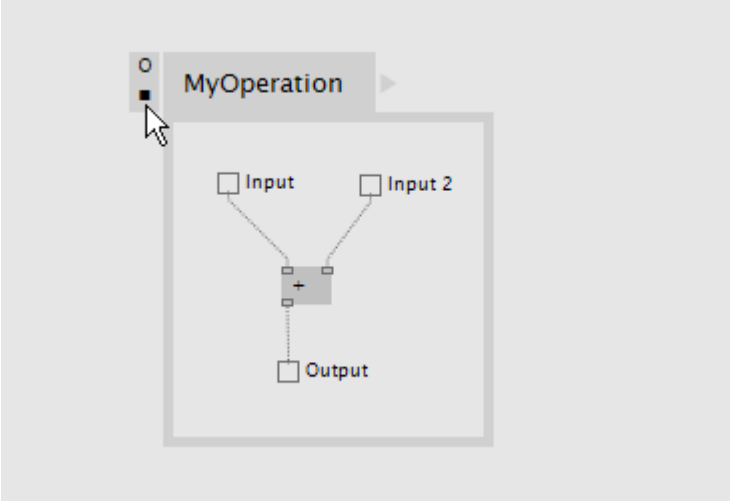
Creating a Static Operation

Static operation definitions can be created via the NodeBrowser.



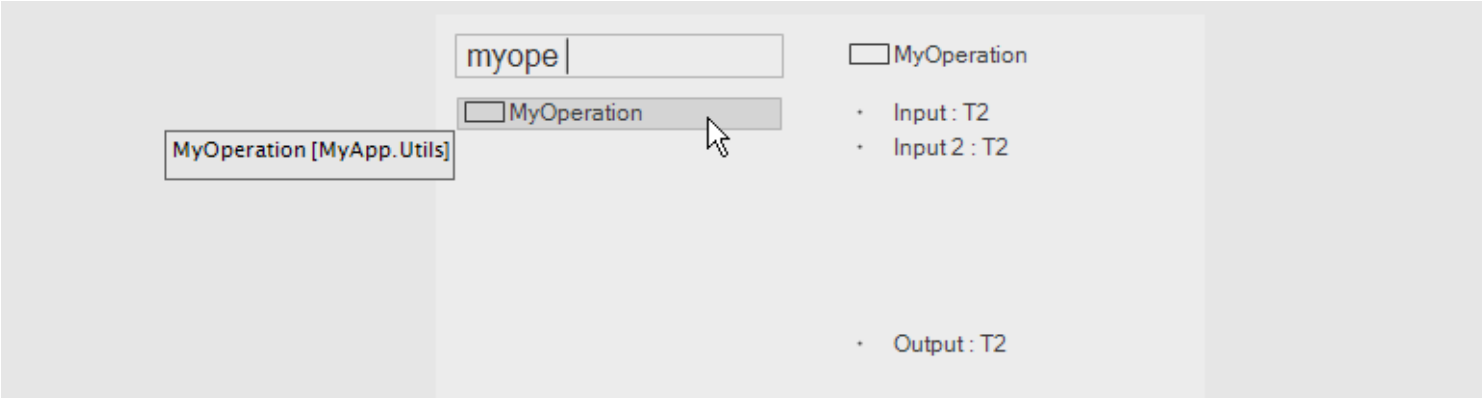
Choose to create an operation definition in the NodeBrowser

By default, static operations have their *Is Generic* property set to false. Errors will be shown for all inputs and outputs of the operation whose datatype is not specified or cannot be infered. To allow generic inputs, enable this toggle.



The "Is Generic" toggle of an operation definition is off by default

Once created, the operation definition shows up in the NodeBrowser and can now be created as a node.



The newly created operation can now be selected via the NodeBrowser

Input and Output Pins

Inputs and Outputs in operation definitions show up as Pins on the corresponding Node.

There are two ways of creating pins:

- With a link at hand, hold `CTRL` while left-clicking
- Doubleclick to bring up the NodeBrowser, then type the name, you want the pin to have, then choose either `Input` or `Output`

Configuring Input and Output Pins

Use a pins configuration menu to to configure it. You can reach the menu either way:

- Middleclick the pin
- Rightclick the pin and choose `Configure`

Annotating Inputs and Outputs

"Annotating" means to manually specify a datatype for a pin. In the configuration menu, the topmost entry allows you to specify a Type. Doubleclick the entry to edit it.

Note

Type names are case-sensitive, ie it is important that you're using correct spelling when setting a type.

Defaults for Inputs

When an input is annotated with a type you can also specify a default for it in the configuration menu.

Visibility for Inputs and Outputs

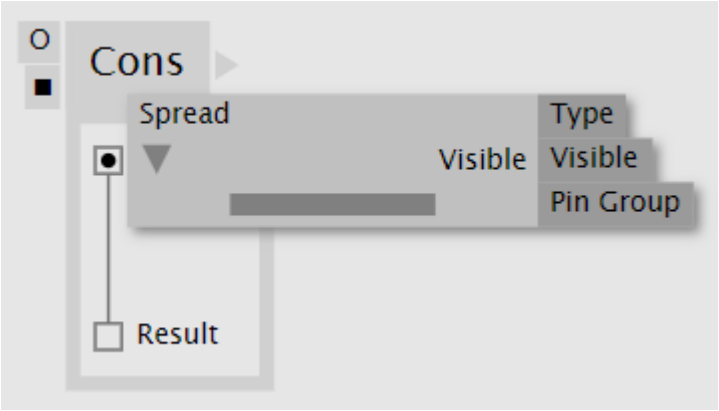
As the creator of a node you can also decide if certain pins should not be visible by default. A reason to do so would be that the pin is of rather special interest and default usage of the node doesn't require it.

When setting a pins visibility to `Optional` it can be shown by a user of the node, using the nodes configuration menu. If a pin is set to `Hidden` it cannot be used by a user of the node.

Pin groups

Pins of type `Spread<T>`, `Array<T>`, `MutableArray<T>`, `Dictionary<string, T>` and `MutableDictionary<string, T>` can also be changed to a so called *Pin Group*. Pin Groups allow you to dynamically add/remove pins to a node. For the keyboard shortcut to do so, see Pin Group Shortcuts.

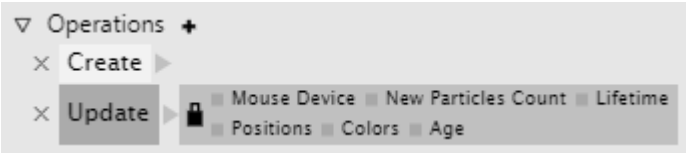
In order to make a pin into a pin group, it has to be annotated to one of the above types. Only then you can set the Pin Group flag in the configuration menu to TRUE.



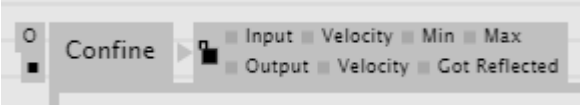
Operation Signature

The signature of an operation allows you to define the order in which its Inputs and Outputs show up on corresponding nodes.

For static operations the signature can be opened directly on the operation definition region. The signature of member operations can be accessed via the PatchExplorer.



Locked signature of member operation "Update" in the Patch Explorer



Unlocked signature of static operation "Confine"

By default signatures are *locked*, meaning the order of pins is defined by their left-to-right placement in the patch. In order to manually manage an operations signature you have to *unlock* it by pressing the Lock icon toggle. In an unlocked signature you can arrange pins via drag'n'drop.

The Connect To Signature feature only works on locked signatures, where the system has full control over managing existence and order of pins.

Doubleclicking a pins name allows you to rename it. A middleclick on a pin allows you to annotate it with a type.

Properties

Datatypes can use properties to store data. You can get an overview of the properties of a datatype via the Patch Explorer.

Image:Properties listed in the PatchExplorer

You can add and remove properties via the Patch Explorer, but specifically for adding properties, you'd often simply create Pads.

If you're coming from textual programming, you may also think of properties as "variables" with the caveat that they can only be written to once per operation!

Pads

In a patch, Pads are used to get (read data from) or set (write data to) properties. Pads refer to properties via name, meaning all Pads with the same name refer to one and the same property. Names are case-sensitive!

In every operation you can assume that first always all Pads are read from. Then all its operations are executed and only as the last step all Pads are written to.

If a link goes into a Pad (from above), data is written into this Pad. If a link leaves a Pad (at the bottom), data is read from this Pad.

Pads can have multiple links coming in and/or going out. Note though, that while multiple links can go out on the same operation, all incoming links need to be on different operations! Think about it this way: Pads cannot be used to store intermediate values during the execution of an operation. They can only be used to store data between the execution of different operations.

A little triangle above a Pad is a hint that there is a Pad with the same name in the patch that is also written to.

Image:Different operations writing to the same Pad

Adding Pads

You can add Pads via Nodebrowser in three different ways:

1. Enter the name of the Pad you want to create and then choose the entry `Pad`.
2. Choose the entry `Pad` and then enter a name
3. Choose from the list of existing properties that are listed in the Nodebrowser

Renaming Pads

Doubleclick a Pad's name to change it. When renaming a Pad, only this one instance is renamed and eventually referring to a different property. If a property with the new name of a Pad did not exist so far, a new property is automatically added at this point!

To rename all Pads that share a name at the same time, rename the property via the Patch Explorer instead.

Anonymous Pads

Pads without a name are called "anonymous Pads". They don't refer to a property but still allow you to store data between the call of multiple operations.

You can quickly insert an anonymous Pad into a link, by pressing `Shift` while doubleclicking the link.

You can also use anonymous Pads simply as a hub to join many links into one.

Image:Anonymous Pad use as a hub for multiple links

The datatype of a property

A property's datatype can either be:

1. Generic
2. Inferred
3. Annotated

ad 1) Generic By default properties are generic, meaning they don't have a datatype assigned. On Pads this is visible when they are only showing the outline of a circle.

Properties are generic as long as none of their associated Pads are have a datatype either inferred or annotated.

ad 2) Inferred If the compiler has inferred a type for a Pad from the links that are connected to it, it is showing as a filled circle. You can see the inferred datatype in the Pad's tooltip by hovering it.

Image:Generic Pad vs. Pad with a datatype inferred

ad 3) Annotated To set the type for a property manually, you can annotate one of its Pads. Middleclick a Pad to open a little inspector where you can edit its type. As an alternative to the middleclick you can rightclick the Pad's label and choose -> Configure.

Image:Annotating a Pad

You can recognize Pads that are annotated manually as they have a dot in their circle.

Image:Annotated Pad

Pads vs. IOBoxes

A Pad and an IOBox are essentially the same thing: While the IOBox has a value editor and a comment (on its right side), a Pad has a name (on its left side).

You can convert between the two via Rightclick -> Replace...

You can also enable the value editor for any Pad or hide it for any IOBox.

Execution Order

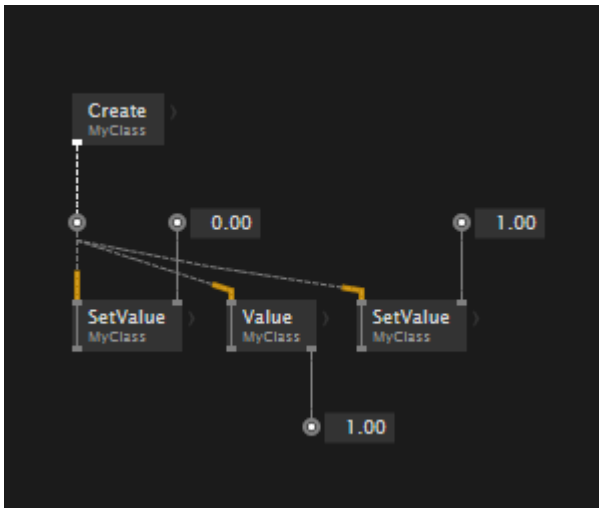
In most cases the order of execution of nodes is obvious: From top to bottom, in the order they are connected with each other through links.

No link dependency

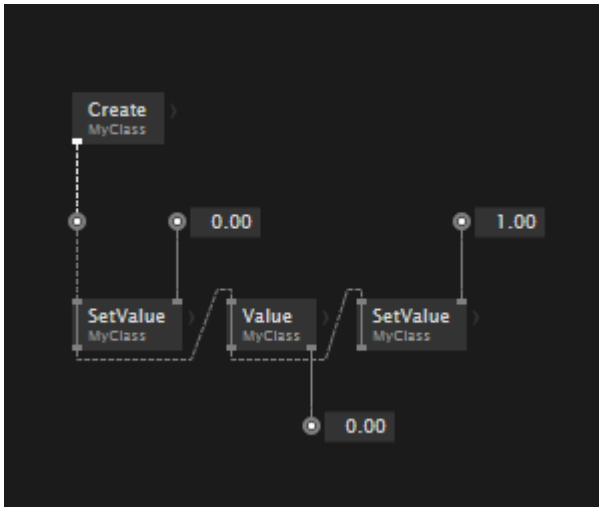
There are cases though, where there is no link dependency between nodes that would express an order of execution. This may be fine or not, depending on your use case. Here are some cases where this may matter and how to solve them:

Multiple writes to mutable data

When writing to (ie modifying) a mutable datatype several times in one frame, execution order typically matters. In this example the Value is read but the fact that it is reading "1.00" for this frame is not defined, it could be "0.00" as well, since there is no order defined. This is also what the "yellow socks" warning on the links is about.

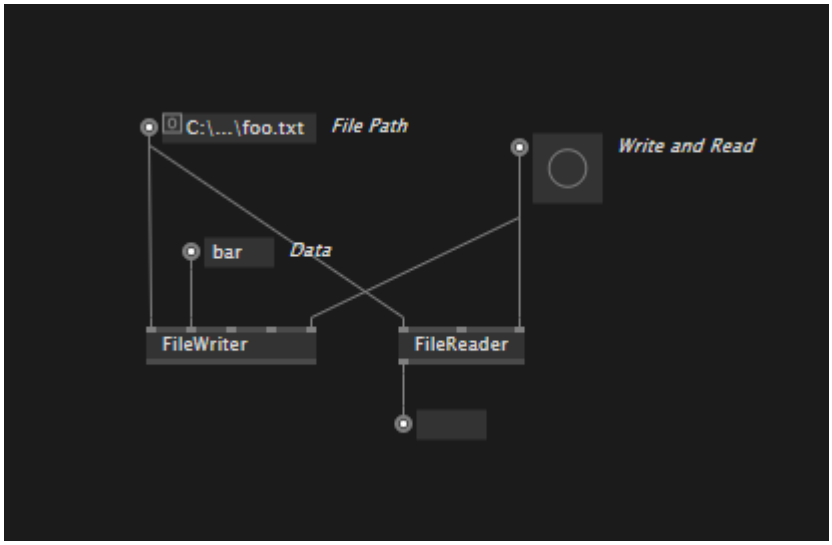


Instead of connecting the operations to the pad "in parallel", connect them "in series", and thus specifying a well defined order of execution.

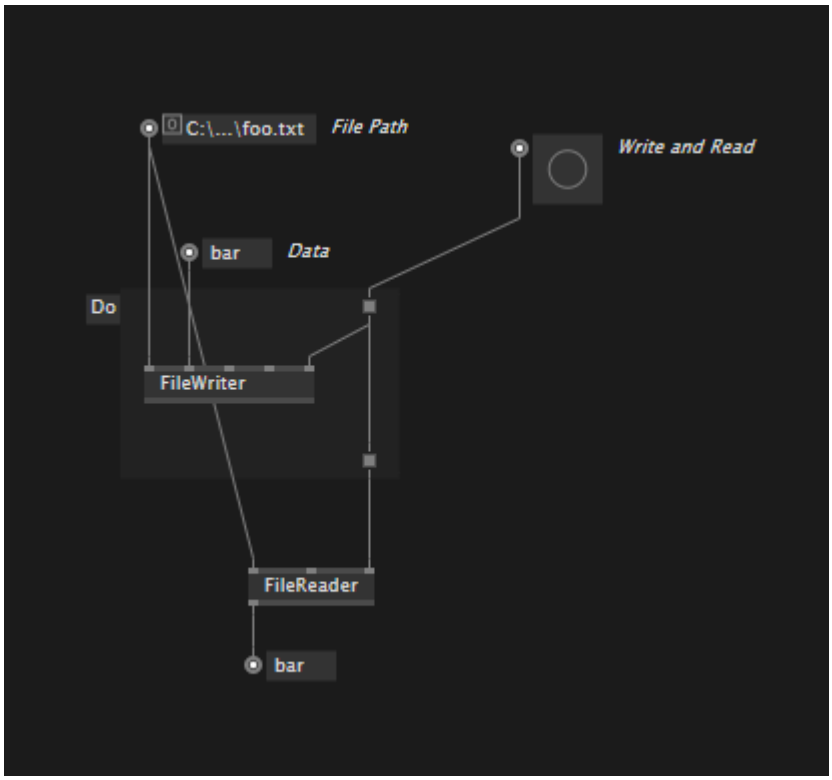


Nodes with no connection in the patch

If you want to write data to a file and read it in the same frame, you need to make sure to write before you read. A naive patch like this, wouldn't make sure of this and thus might randomly work, or not:



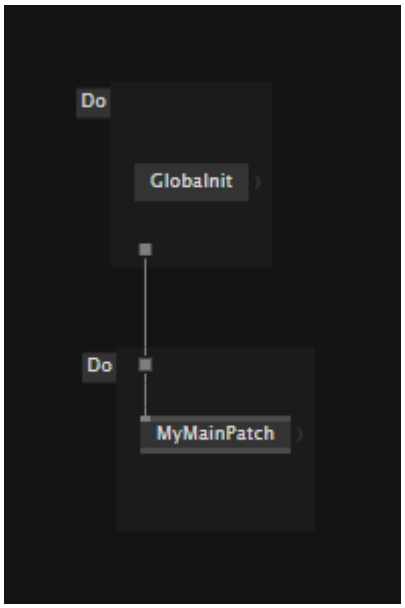
In order to create links between nodes in such situations, you can use the `Do` region. The region itself does nothing but letting you create input and outputs for it so you can use those to define an order of execution.



Nodes without any pins

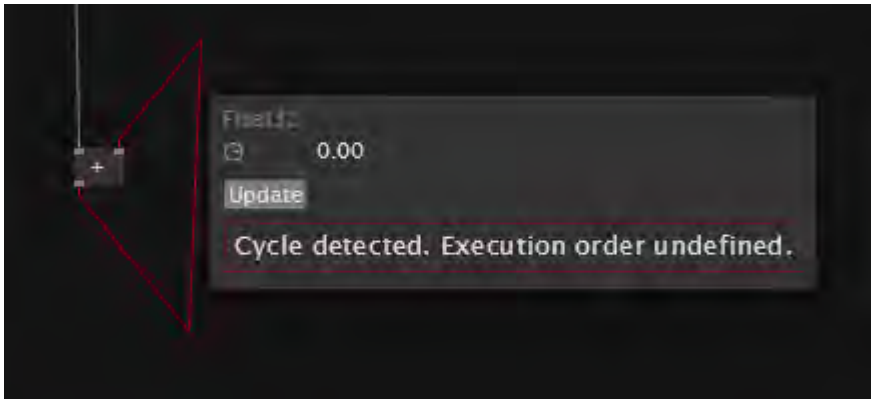
There are cases of nodes that don't have any pins at all. Often those are operations to globally initialize the state of a library, in which case it is important to have them execute before anything else. In those

situations a `Do` region can help you build an order of execution, like so:



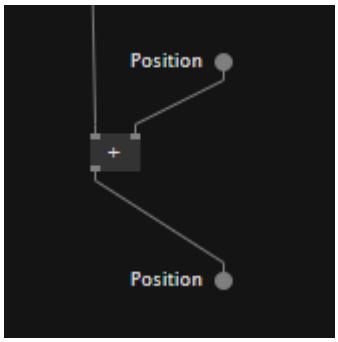
Circular graphs

When trying to make a circular link connection, VL will prevent you from doing so. If you force the link by pressing `Space` while making the connection, you'll see an error like this:



Think about it: If VL would allow you to do this, it would never know where to start the execution. Therefore in such situations you need to find out where to best store the value of a computation from a previous frame and use it in the next frame.

To solve this, introduce a Property and use Pads to write a value in one frame and read it in the next frame:

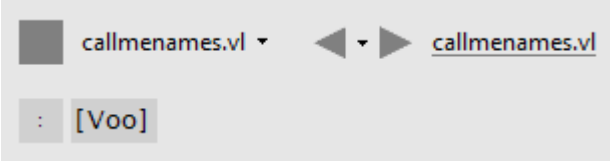

Position
+
Position

Categories

Categories in VL are synonymous to "namespaces" in other programming languages. They allow you to structure your libraries of nodes.

A documents category

Every VL document starts a category which can be defined in its Definitions Patch.



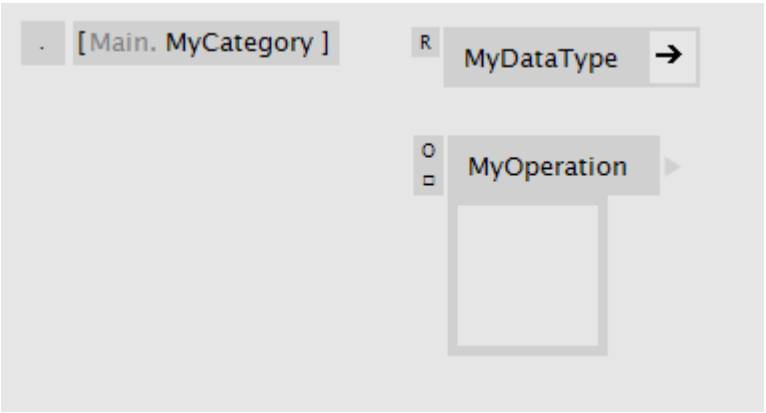
"Voo" specified as a documents category

Category elements

Category elements can be added to the Definitions Patch via the NodeBrowser by choosing "Category", to build a category structure that holds different parts of a library.

A categories name appends itself to the category of its parent patch. That way you can build up any category hierarchy, that you then see in the NodeBrowser. Multiple category levels are allowed with dot notation. e.g. *MyCat1.MyCat2* etc.

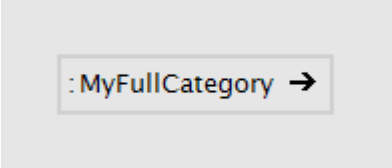
Category patch from the outside



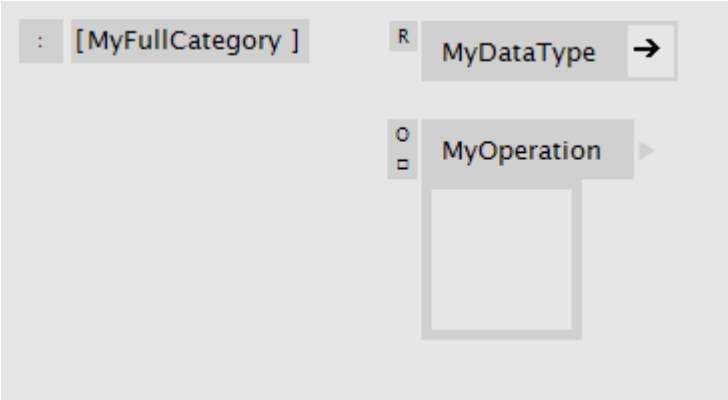
Inside a category patch

Full Category

A Full Category is similar to a normal Category, only that it doesn't add its category to the parent but starts a new root category.



Category patch from the outside



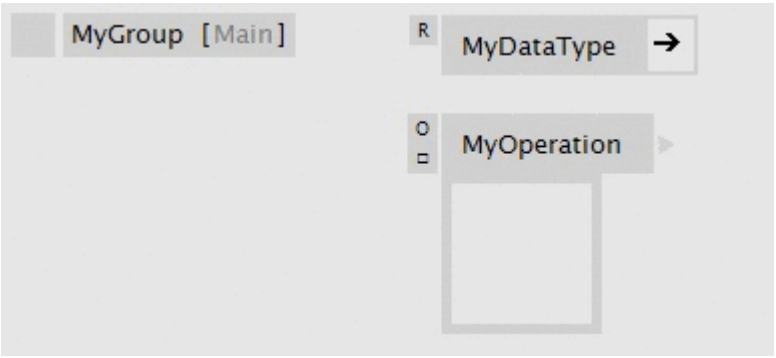
Inside a category patch

Note

Empty categories are not showing-up in the NodeBrowser.

Changing the Patch Type

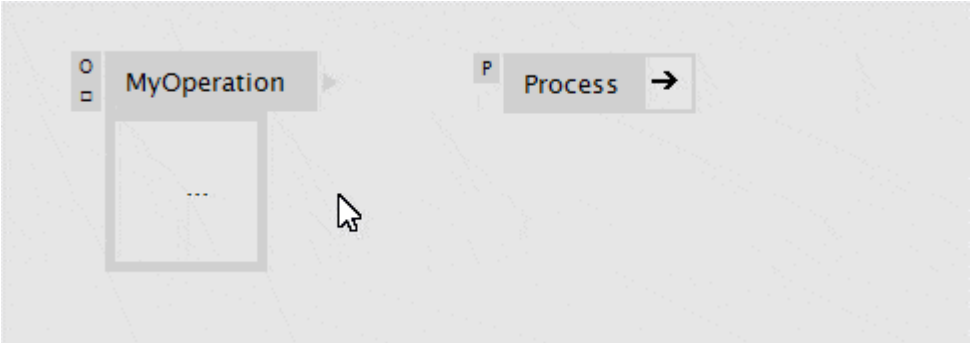
You can easily convert a category into a Group patch and vice versa using the patch type enum. Note how the label changes and represents the actual category structure:



Converting a group into a category

Setting Categories on Definitions

As if the above didn't offer enough options already there is one more way to specify a category for an operation or a datatype definition:



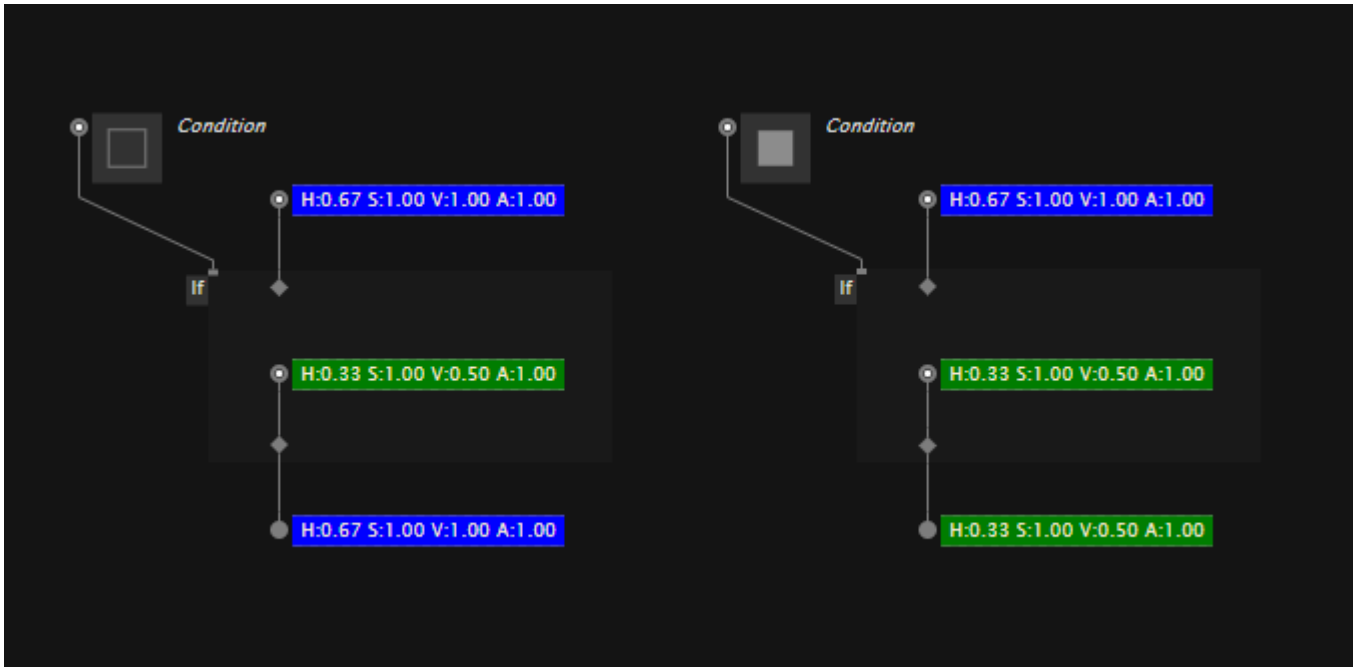
Setting a Category on an datatype or operation definition

Conditions

At this point, the only conditional language primitive in VL is the `If` region.

The If region

The If region can be used to conditionally execute parts of a patch. If the `Condition` input is set to true, then the patch inside the region is executed, otherwise the values on its border control inputs are passed through to their corresponding outputs.

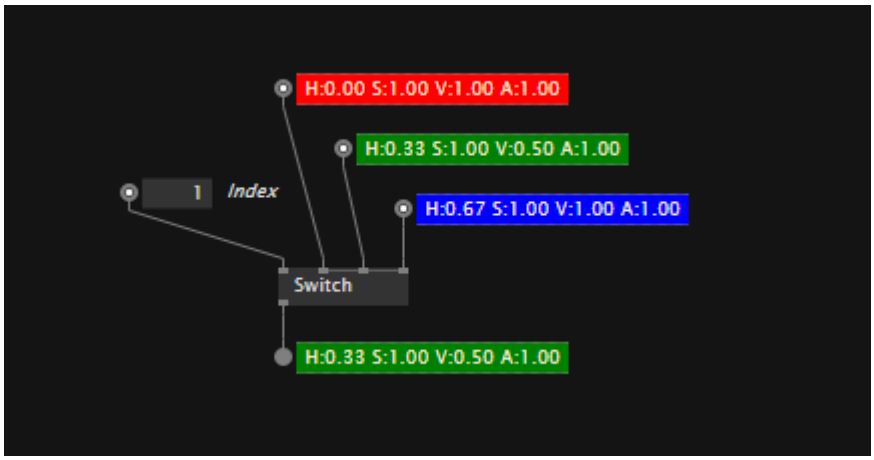


You can quickly surround a bunch of nodes with an If region, by selecting them and then choosing `Surround -> If` from the rightclick context menu.

Moving nodes into and out of the region works by pressing `Space` while dragging them.

Switch

While there is no dedicated Switch region, there is a Switch node that at least lets you route multiple potential inputs to one output, depending on a condition or index.



The Switch node has a pingroup for its inputs. Select it and press `Ctrl` `+` or `Ctrl` `-` to add/remove inputs.

Loops

There are two different Loops in VL:

- Repeat: a classic for-loop with an *Iteration Count* input to specify the number of iterations the loop executes
- ForEach: executes for each slice of a spread entering the loop via a splicer.

In the NodeBrowser you'll find different nodes named Repeat and ForEach. To get these primitive ones, choose the versions written in *italic*.

Image:Choosing Repeat or ForEach from the NodeBrowser

Getting data into a loop

There are 3 different ways of getting data into a loop. All of them work for both the Repeat and the ForEach loop:

Direct Connection

Data can be linked directly into a loop which results in each of the loops iterations receiving the same data.

Image:A direct connection into a loop region

Splicer

Splicers allow you to access consecutive slices of a spread in consecutive iterations of a loop. Entering a loop with a link via the splicer-bar that shows up when starting a link, automatically leads to each iteration of the loop receiving one slice of the incoming spread.

Image:A spread connects into a loop using a splicer

Multiple spreads can go into the same loop via splicers. In case of a ForEach loop the number of its iterations is then determined by the lowest of the slice-counts of all incoming spreads!

Image:A foreach loop receives a spread with 20 slices and another one with 15 slices via splicers causing it to execute 15 times.

In case of a Repeat Loop the iteration count determines the number of iterations ignoring the slice-counts of the spreads coming in via splicers. When the iteration count is higher than a spreads slice-count, slices of the spread are being repeatedly accessed with the loops index taken modulo the spreads slice-count.

Image:A Repeat loop has its Iteration Count set to 5 and receives a spread with 2 slices and another one with 3 slices via splicers causing it to execute 5 times.

By default splicers have no names. Sometimes it may help to label a splicer for clarity in which case you simply doubleclick the area right of it to enter a name.

Accumulator

Accumulators allow you to hand data over between iterations of a loop. Once initialized from outside of the region, the accumulator can be accessed and modified in each iteration and is then passed on to the next iteration of a loop. The final value is then available via the accumulator output.

An accumulator can thus be understood as a variable declared outside and then modified in each iteration of a loop.

Image:An accumulator is modified in each iteration.

By default accumulators don't have a name. Only to distinguish multiple accumulators in the same loop from each other, they are automatically labeled using roman numbers. If you want to specify your own you can do so by doubleclicking the area right of an accumulator to change its name.

Getting data out of a loop

Since you can never link directly out of any region, there are only two different ways of getting data out of a loop. They work for both the Repeat and the ForEach loop:

Use an outgoing splicer to collect the results of all iterations and return them as one spread.

Use an outgoing accumulator to receive the final value as modified by all iterations of a loop.

Special Pins

There are three special Pins which you can create only inside loops via the NodeBrowser:

Image:The Index, Break and Keep pins in a loop

Index

Returns the current loop iteration number.

Break

Set it to true to break out of the loop at any time before its actual iteration count is reached. Note that the breaking iteration is still fully executed, resulting in output splicers to include the result and accumulators to be modified by this iteration.

To find out if a loop executed to the end or it was interrupted by a break, the *Break* output can be tested.

Keep

Set it to true or false in each iteration to specify whether results of this iteration will be included in a spread returned by an outgoing splicer.

Note that the keep has no influence on accumulators, meaning that accumulators will still be changed for iterations that are not 'kept'.

Other Loops

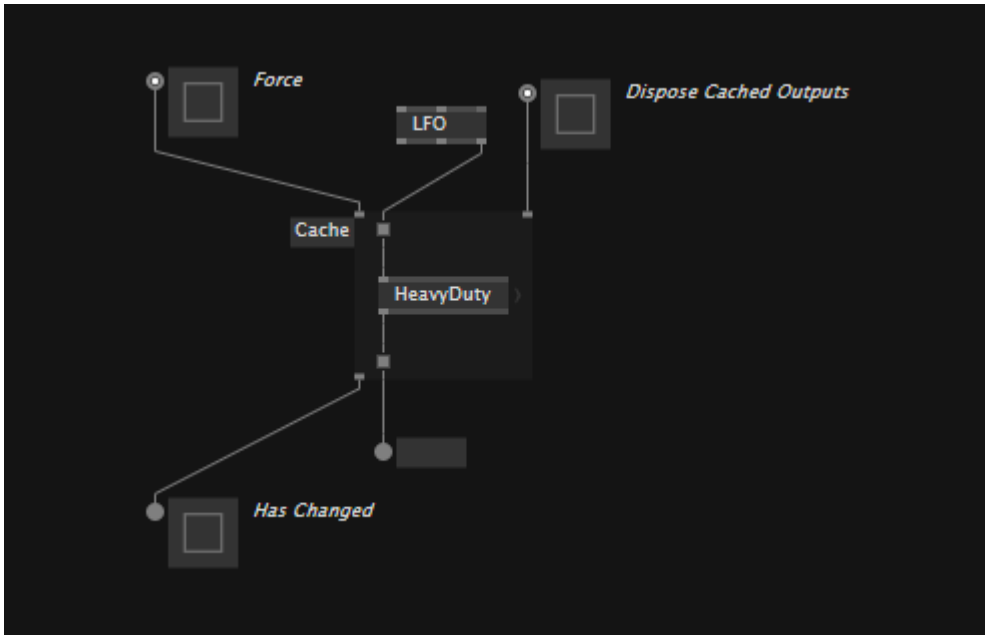
While

Use a Repeat loop with its *Iteration Count* set to a very high number which in this case you can consider as the maximum iteration count you specify in order to make sure your patch can never hang indefinitely. Inside the loop you specify the condition that needs to be met for the loop to continue to execute and connect its negation to the loops *Break* output.

Image:Simulating a while loop using a Repeat loop

The Cache region

To prevent parts of a patch being executed every frame, you can use a `Cache` region. The number one use-case for Cache regions is optimizing performance by making sure things only get executed when they really need to, thus saving precious CPU cycles.



All nodes inside a Cache region are only executed if one of its border control inputs is changed or its `Force` input is set to true.

Once executed, the regions output border control points hold (ie. cache) the results until the region is executed again.

The `Dispose Cached Outputs` input defines whether objects, cached in one of the regions output border control points, will be disposed, before a new result is being cached. As a rule of thumb: If the objects class has a Dispose() method, you'll most likely want to activate this input, except you're intentionally dealing with its disposal in a different way.

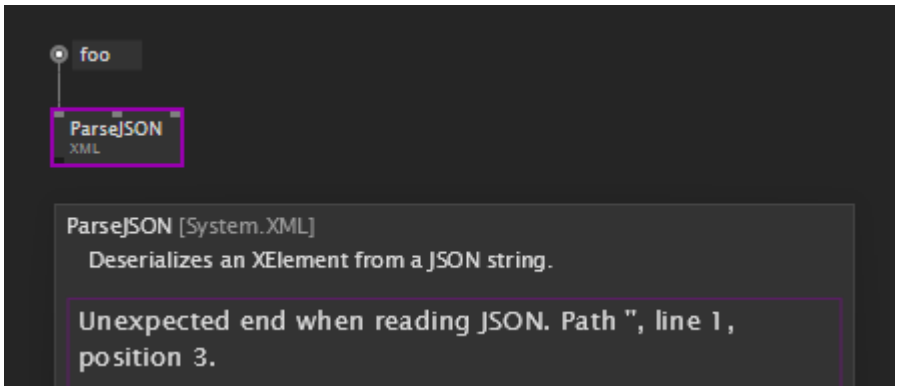
The `Has Changed` output returns true for every frame the the region was executed.

You can quickly surround a bunch of nodes with a Cache region, by selecting them and then choosing `Surround -> Cache` from the rightclick context menu.

Moving nodes into and out of the region works by pressing SPACE while dragging them.

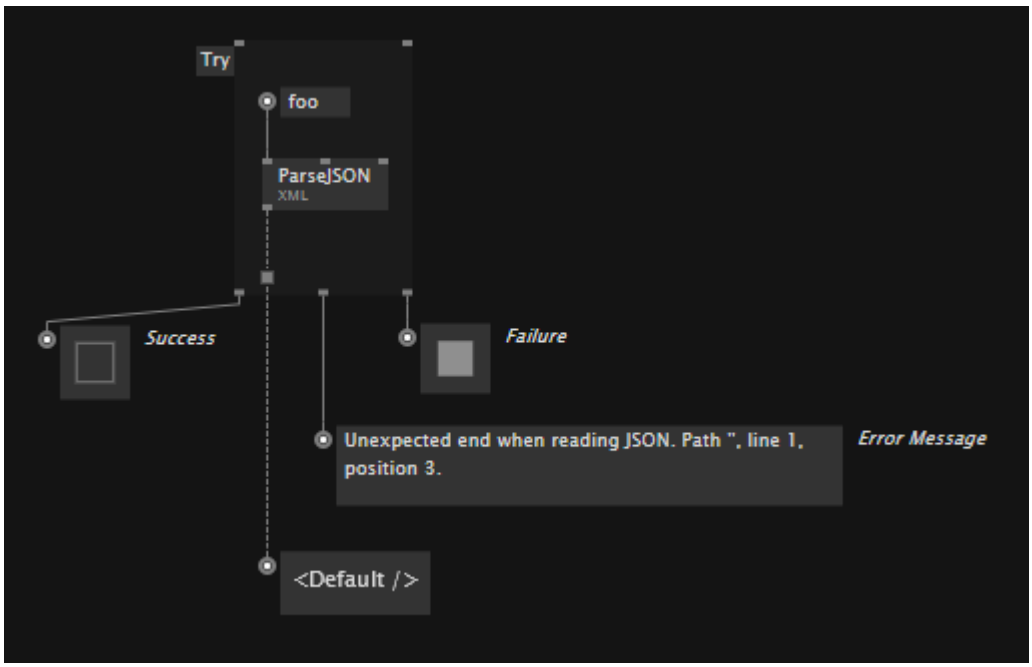
Exception Handling

Nodes sometimes get a pink border, which means that they are throwing a runtime error. Hovering the node with the tooltip will show you more information about the error:



Depending on your Setting for "Pause on Error" the execution of your patch will either pause or continue. But even if the execution continues it may fail again at the same spot at a later time. Therefore it is good practise, to "handle" exceptions.

In order to handle runtime errors programmatically, you can surround the culprit with a Try region:



This will allow you to react to problems in your patch gracefully without interrupting the execution of your program.

Enumerations

An enumeration type (or enum type) is a value type defined by a set of named constants. VL has two different types of enumerations:

- Static Enums
- Dynamic Enums

Note

As of now, enums can only be used but not created in VL. For defining a new enum, for now you'd have to resort back to writing C# and loading a compiled .dll that includes the enum type.

Static Enums

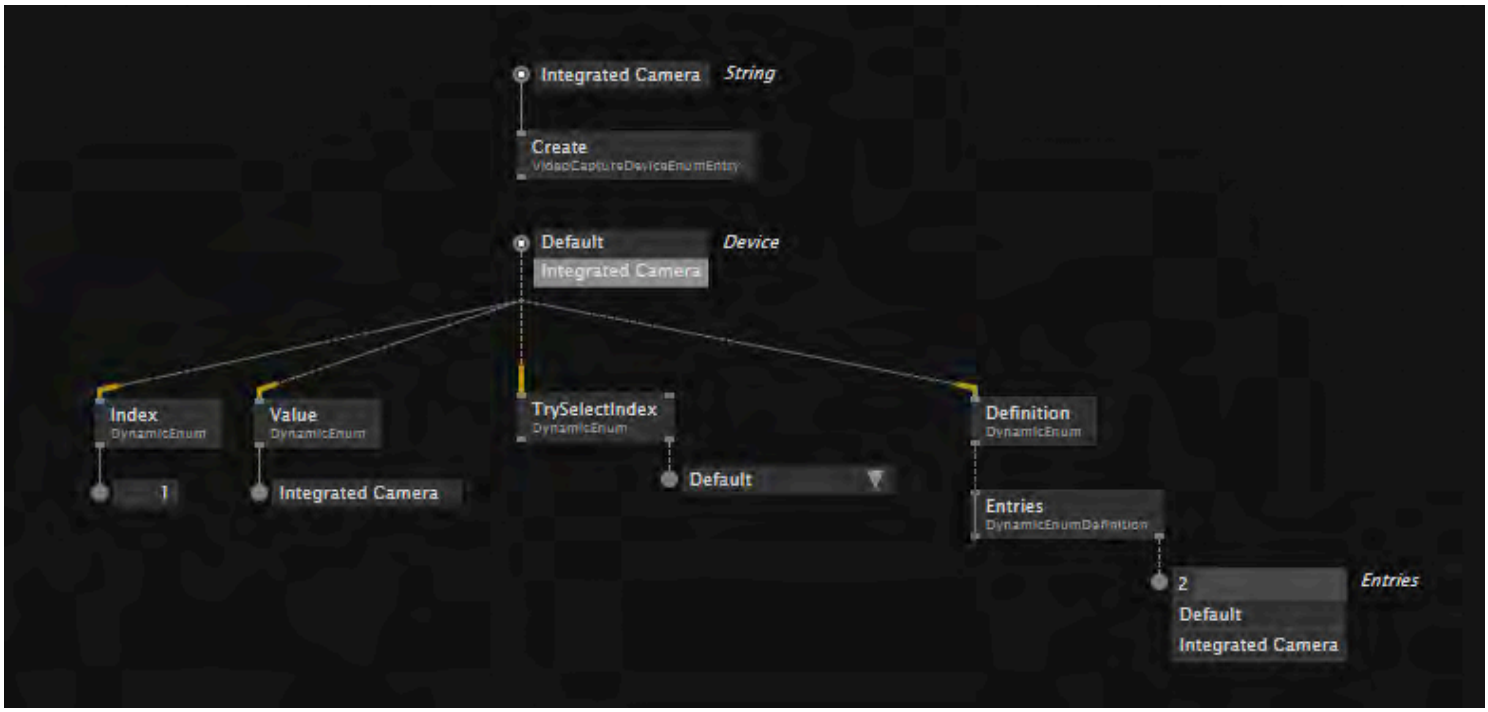
Entries of a static enum are fixed and cannot change at runtime. An example would be the type `LinearSpreadAlignment`.

For working with static enums, use nodes from the `Primitive.Enum` category.

Dynamic Enums

A dynamic enum can have entries added, removed or changed during runtime. They are used for example for device enumerations.

For working with dynamic enums, use nodes from the (advanced) `Primitive.DynamicEnum` and `Primitive.DynamicEnumDefinition` category.



For defining custom dynamic enums in C# see: Defining Dynamic Enums.

Delegates

Making use of delegates in a patch consists of two parts:

- Defining a delegate
- Invoking a delegate

Defining a delegate

Invoking a delegate

Generics

Frames

Frames help you structure your patches visually. You can put a frame behind parts of your patch and give it a title and color.

For an overview of all keyboard shortcuts related to frames, see Frame Shortcuts.

When selected, the frame can be tinted using one of the predefined colors:

You can move a frame around without its content, by dragging the gray bar (when selected). To move a frame and its content, drag it on its title.

Screenshots

Besides being structural elements, frames also allow you to take screenshots easily and repeatably:

- Press the Printer button to make a screenshot, then rightclick it to see the captured file in explorer
- Alternatively press `Ctrl` `2` to take a shot of the selected frame
- Press `Ctrl` `5` to take screenshots of all frames in a document at once

To create a quick screenshot of an area without even creating a frame, simply press `S` while making a selection. This will copy the screenshot to the clipboard (so you can simply paste it into the chat or a

forum reply) and also place a .png next to the current .vl document.

Recordings

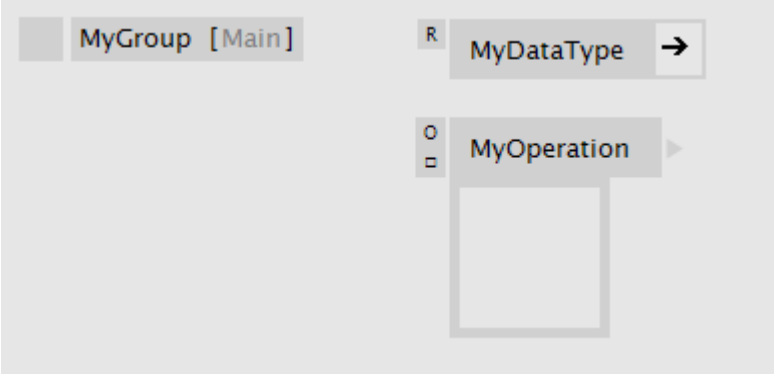
Apart from single screenshots you can also record an animated gif of the area of a frame:

- Press the Record button to start a recording, the same button again or `Esc` to stop it, then rightclick it to see the recorded file in explorer
- Alternatively toggle `Ctrl` `4` to start/stop recording the selected frame

Groups

Groups in VL help you structure elements visually but they don't have any meaning to the language, like Categories do. So you can use Groups to make extra space on a patch by hiding elements away in a new patch without adding to the category structure.

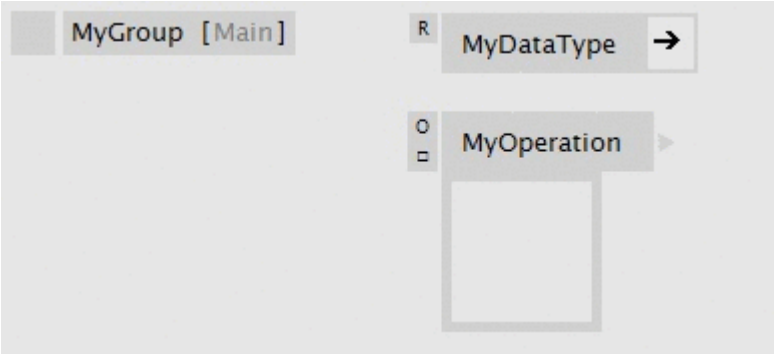
Group elements can be added to the Definitions Patch via the NodeBrowser, by choosing "Group". Just like Categories, Groups can also be nested.



Inside a group patch

Group patch from the outside

A group can easily be converted to a Category:



Converting a group into a category

Compilation

Everytime you make a change in a patch, vvvv compiles it on-the-fly and updates your running programm accordingly. We call this "Hotswap" and yes, this is similar to what you may know as .NET Hot Reload. Only that in vvvv it happens automatically, always, and you'll most often not notice it.

When the compiler is active, you'll see a little indicator in the top left corner of the editor, right below the Quad menu:

Color of Indicator	Meaning
Gray	Building Symbols
Orange	Emitting C# code

On-the-fly compilation, while often not noticable at all, can cause severe lag, when working on large projects or libraries if all .vl files have to be considered for changes all the time. Therefore vvvv gamma 5.0 introduced the idea of read-only packages.

Read-only packages

Patches in read-only packages are excluded from on-the-fly compilation. Like this, they run optimized in the same way as when you export them. Apart from faster execution, the fact that the compiler doesn't have to worry about them saves CPU cycles while working and also leads to a smaller overall memory footprint of vvvv which in turn removes stress from the garbage collector.

Restrictions in read-only packages

Patches part of a read-only package can be recognized by this banner:

In read-only patches, beware of the following restrictions:

- Tooltips will not show any data flowing in the patch
- Any modification you make is not being taken into account

If you do make changes and save the patch, those changes will only be detected on next startup of vvvv, which will trigger a one-time re-compilation of the patch.

What makes a package read-only?

By default all packages are read-only. Like this the startup time and memory usage of vvvv is significantly improved.

This includes:

- All packages shipping with vvvv
- All packages you install in addition and reference in your project. This makes sense since you're never supposed to change those other than by getting a different version of a NuGet
- All packages you reference from a source package-repository

Editable packages

The most likely reason you'd want to opt out of the read-only default for certain packages, is when you have referenced them via a source package-repository to actually work on them.

In this case you need to use the commandline argument `editable-packages` when starting vvvv. Here is an example to opt out of precompilation for all packages starting with "VL.Devices" and the package "VL.Audio":

```
--editable-packages VL.Devices*;VL.Audio
```

Note

In addition this makes any package editable that depends on those you specify!

Identifier Naming Conventions

VL uses Pascal Case as casing convention. Examples:

Data types:

```
Particle
ParticleSystem
AlignedBox
```

Operations:

```
Update
GetPosition
SplitCurve
```

In general the following characters are allowed, not starting with a number or space:

a-z A-Z 0-9 + - * / = ~ < >

Pads and pins should contain spaces to make them visually more pleasing and distinct from operations:

```
Velocity
Map Mode
Word List
```

Categories can include periods:

```
Math
Collections.Spread
Animation.FrameBased
```

Libraries

vvvv's functionality is structured into individual libraries, also known as NuGet packages. Not all of them are shipping with vvvv, but they can easily be installed. Most of them will be open-source and many of them are provided and maintained by your fellow vvvv users.

To learn how to use nugets in vvvv please refer to the documentation on Managing NuGets or watch the HowTo Use NuGets video.

Besides the VL.CoreLib, which is accessible by default, here is an overview of what's available:

Category	Content
2d Graphics	Skia, paths, svg, pdf, ...
3d Graphics	Stride, models, materials, shaders, textures, ...
Animation	Particle Systems, Timelines, ...
Audio	Audio analysis, playback, recording. Sound synthesis, ...
Augmented Reality	Aruco markers, ...
Computer Vision	OpenCV, Dlib, ...
Databases	
Devices	Depth Cameras, Arduino, Lights, Lasers, SICK, ...
IO	Networking Protocols, ...
Machine Learning	Wekinator, RunwayML, Lobe, ONNX, ...
Projection Mapping	Warping, Blending, Softedge, VIOSO,...
Video	Video playback, capturing, ...
Misc	

Missing anything?

- Check the work-in-progress section of the vvvv forum
- Browse the github topic VL
- Scroll the nugets taged VL
- Apart from the above libraries specifically made for vvvv, you can also use almost any .NET library
- We do offer custom development, don't hesitate to get in touch!

You can also make your own libraries for personal use (for sharing nodes and types among different projects) or for sharing with others. See Extending vvvv.

Referencing Libraries

VL Documents can reference 3 different types of dependencies:

- VL Nugets
- .NET Nugets
- Files

When a document references a dependency, it means that all public nodes in that dependency will be available to it via the NodeBrowser.

Nugets

NuGet is the package managing system for .NET. Nugets are packages that can contain many .dll and/or .vl files that expose nodes to the referencing document.

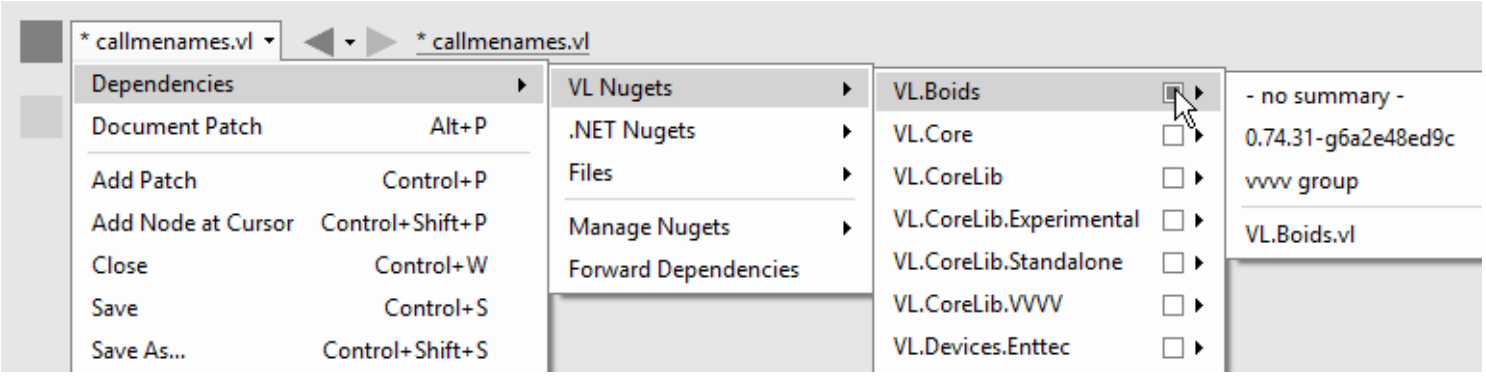
For installing nugets, see Managing Nugets.

VL vs. .NET Nugets

A VL Nuget is a nuget specifically created for use with vl that won't work for any other .NET language. It is still a valid nuget in the original terms of NuGet but since it contains .vl documents it will not be useable outside of vl.

A .NET Nuget on the other hand is more generally targetting any .NET language.

You can reference either VL or .NET nugets via the menu by navigating to it and pressing the right mousebutton to toggle its selection:



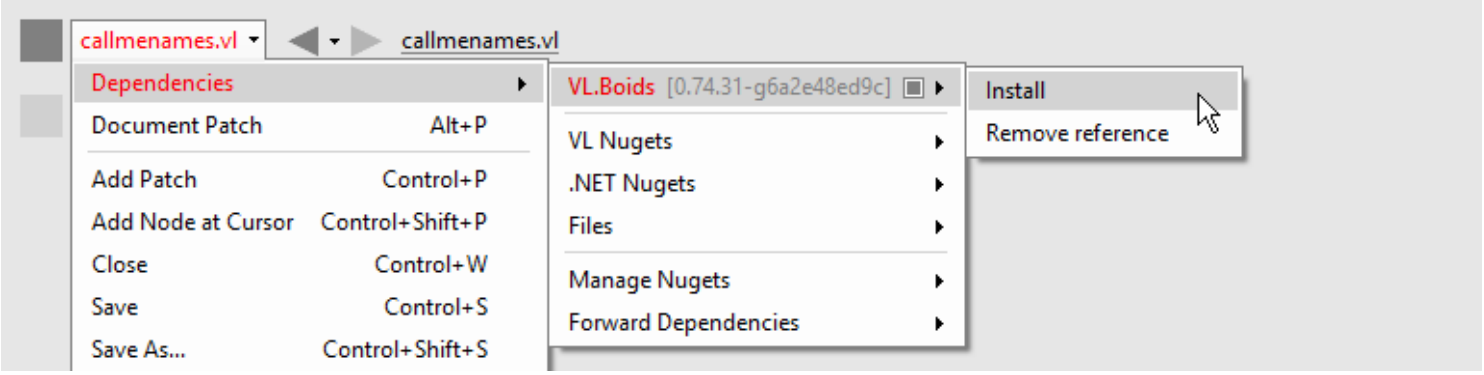
Rightclick toggles adding/removing a nuget reference

Missing Nugets

If a nuget that is referenced by a document cannot be found, it will be listed in red in the Dependencies menu. In such a case a rightclick on a red entry allows you to:

- *Install*: attempt to install from nuget.org. This will obviously only work if the nuget can be found online
- *Remove Reference*: remove this nuget as a dependency of this document

Note that you can rightlclick to select multiple red entries and then choose to apply either install or remove to all of them at once.



Missing nuget options

Unmanaged/Native dependencies

Some nugets are shipping with or depending on unmanaged/native .dlls which cannot be picked up by vl automatically since there isn't a pattern in the nuget specification as to how those should be handled. So in order to get such unmanaged dependencies of a nuget picked up, for now you'll have to add a search-path for vl via a batch file, like so:

```
SET PATH=%PATH%;c:\path\to\nugets\nativelibs;
vvvv.exe
```

Files

A vl document can reference other .vl documents and managed .dll files.

From Disk

Here are 3 ways to reference local files:

- Drop a .vl or .dll file onto a patch
- Press `Alt` `Ctrl` `E` to select files via a file browser
- Via menu:Document[Dependencies > Files > Add Existing...]

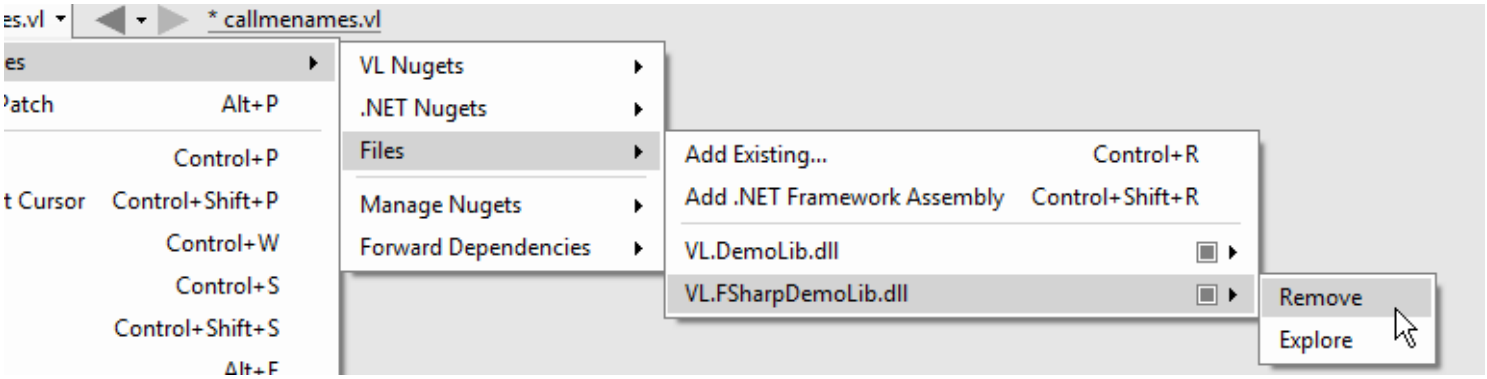
Add existing File as dependency

Missing Files

Files that are showing up in red cannot be found on disk. You can rightclick to remove ore replace their reference.

Removing or Replacing Files

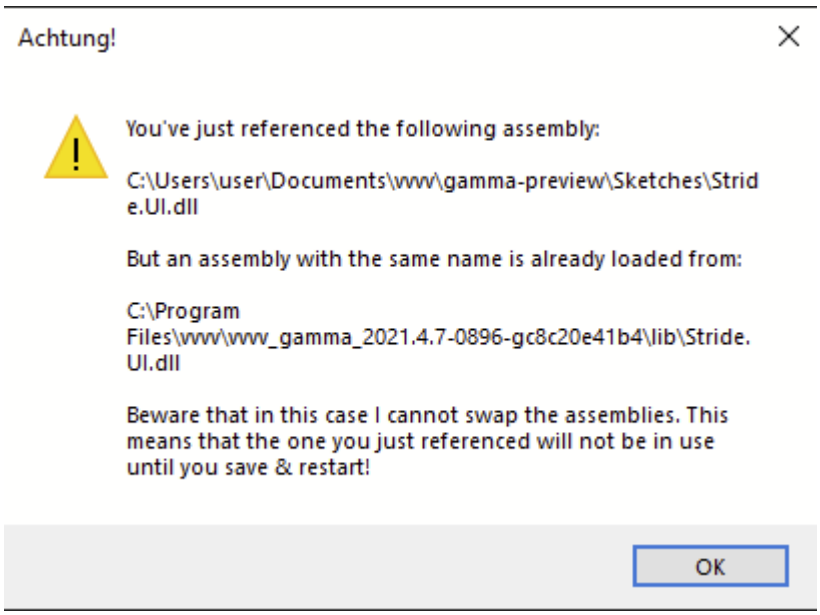
You can rightclick a file reference to remove or replace it. Note that you can also rightclick to select multiple files in a row and then apply "Remove" to all of them at once.



Remove files

Duplicate reference warning

When referencing a .dll you may encounter a warning similar to the following warning:



The warning pops up because a file with the same name is already loaded by vvvv.

There are two situations in which this may occur:

1. Changing the location from which to load a .dll:
   You've set a reference to the .dll before but have since moved it to a different location on disk and now want to fix the reference to that new position.

2. Referencing a .dll that vvvv itself has already loaded:
   You're setting a reference to a .dll in one location but a .dll with the same name has already been loaded from another location, most likely by vvvv itself.

If you are certain that those are actually the same files, only in a different location, then you can ignore the warning.

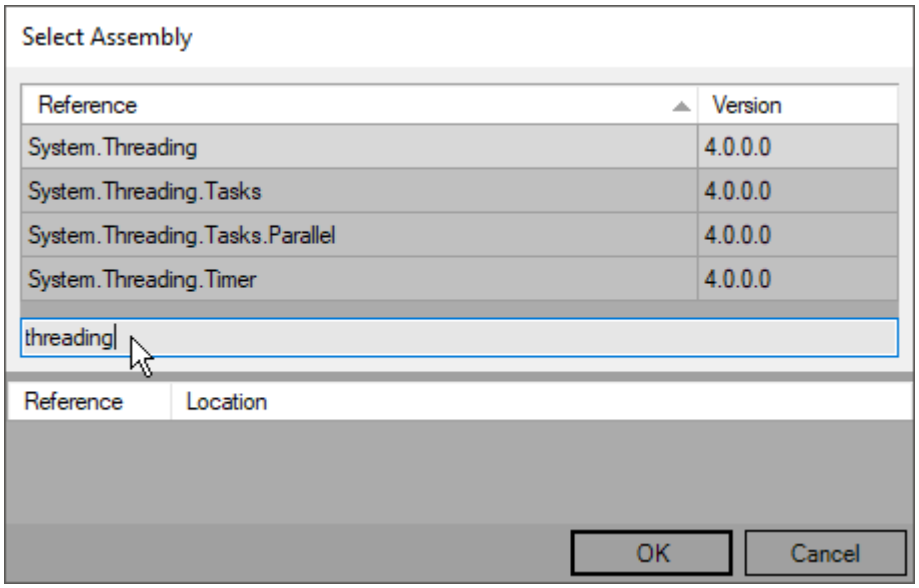
Otherwise in case 1) restarting vvvv should help, but in case 2) you'll actually not have a chance to get this solved. .dlls loaded by vvvv cannot be changed at all. If for some reason you need to use a newer version of a .dll than the one vvvv is currently using, please start a thread on the forum about this and we'll see what we can do.

Libraries from the GAC (Global Assembly Cache)

By default .NET comes with a large number of assemblies that can be referenced. They are stored in the GAC on all machines that have .NET installed and can be referenced from there via:

- Press `Ctrl` `Shift` `E`
- via `Document > Dependencies > Files > Add .NET Framework Assembly...`

In the dialog you need to double-click entries that you want to add as references.



Use `Ctrl` `F` in this window to find libraries in the GAC

The VL.CoreLib

The default library of VL that provides nodes and types for the most basic patching needs. Here is an overview of the Categories it adds to a document that references it.

Category	Content
2D	2d primitives like Vector2, Rectangle, Circle,... and 2d transformation and collision nodes. Further any 2d related math nodes.
3D	3d primitives like Vector3, Box, Sphere,... and 3d transformation and collision nodes. Further any 3d related math nodes.
Adaptive	Nodes that can operate on different datatypes, like a + [Adaptive] that can operate on numbers, strings, colors ... or a Length [Adaptive] that works for 2D and 3D vectors.
Animation	Timebased nodes like time-generators (LFO, Stopwatch, ...) and filters (Damper, Oscillator, ...). Also has a subcategory *FrameBased* that contains similar nodes that operate framebased instead.
Collections	Contains most notably the Spread, but also other simple collections like the Sequence, Dictionary and HashSet.
Color	Contains the RGBA color type and operations to convert to/from different color spaces.
Control	Nodes to patch control flow, like FlipFlop, MonoFlop,...
IO	Mouse, Keyobard and Touch nodes as well as nodes for file IO, Path (directory, filename) and Networking
Math	General math, algorithms,...
Primitive	Contains the primitive datatypes, like Bool, Byte, Integer32/64 Float32/64, Char, String
Reactive	Nodes for reactive programming
System	XML, JSON, DateTime, Serialization, ...
Text	TypeWriter

Collections

A large number of different collection types is shipping with the VL.CoreLib:

Sequence

The base type for collections in VL is the *Sequence*. It corresponds to what is known as *IEnumerable* in .NET world. We just gave it a more human-readable name.

Spread

The Spread is a specialized sequence. The elements in a spread are called slices. When asking a spread with 4 slices for the slice with index 6, instead of complaining, it takes the index modulo its count, ie 4 mod 6 = 2 and returns the slice with index 2.

SpreadBuilder

The SpreadBuilder is the mutable variant of the Spread for use in scenarios where performance counts. A typical scenario would be when slices need to be added/removed to/from a spread in a loop. In such a case use a SpreadBuilder to modify the spread and use ToSpread in the end to go back into the save immutable world.

Dictionary

The Dictionary is a key-value collection. Items (values) are added to the dictionary given a label (key). Keys have to be unique and can therefore be used to retrieve individual values from the dictionary.

The key is often a string, but can really be any other datatype.

HashSet

Represents a set of values.

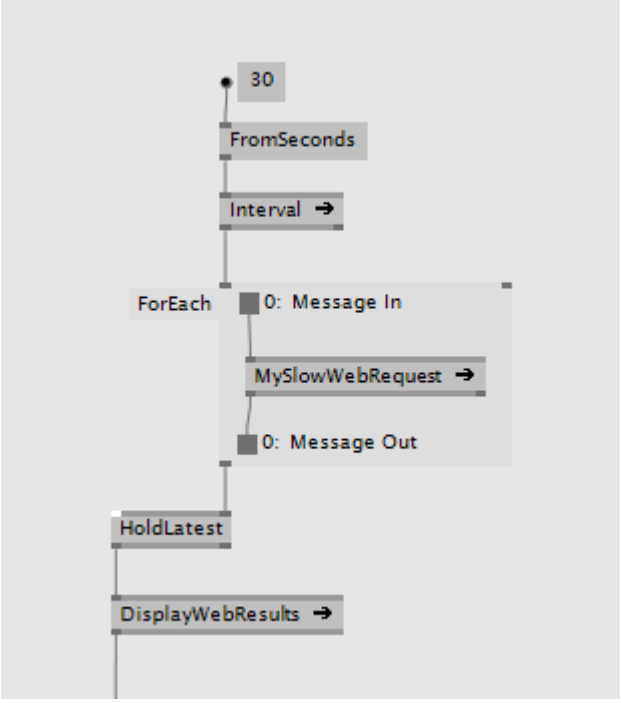
Enable the `Advanced` aspect in the nodebrowser to get access to many more collection types.

Reactive

The Reactive category gives you tools to handle asynchronous events, background calculations and even enables you to build your own mainloop that runs on a different CPU core.

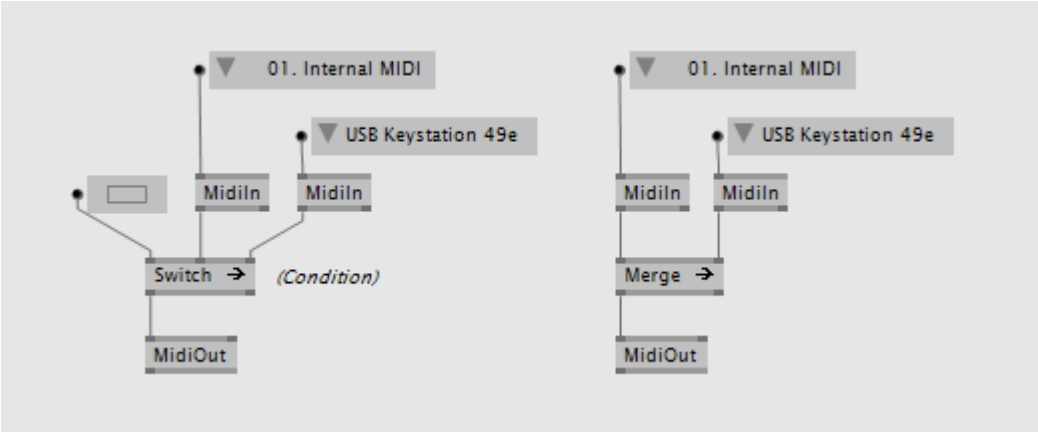
Processing Events

The go to node for handling events is the *ForEach Region* in the category *Reactive*. This region allows you to place any node inside and can also remember any data between two events. There is also one with version *Keep* that can filter out events using a boolean output. This region is very similar to the ForEach region for spreads, only that its input and output is event values in time instead of slices of a spread.



Refresh web data every 30 seconds in the background and pass the result on to the mainloop

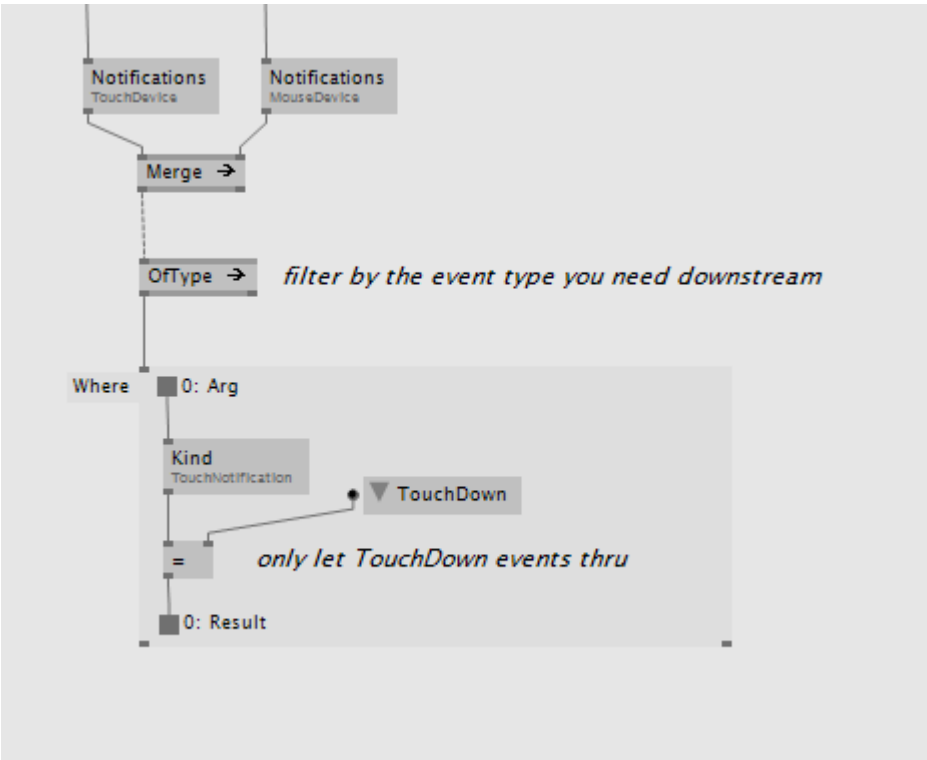
Switching or merging event sources



Switching or merging midi events

Filtering

There are also filtering options with *OfType* or *Where*:



Only get TouchDown events from a combined event stream

Other nodes include

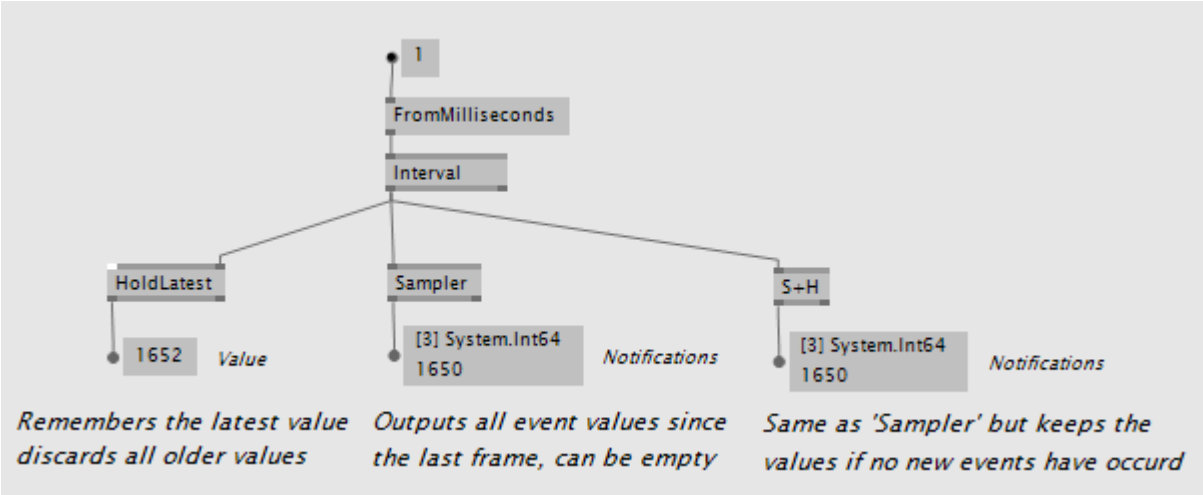
- Skip,
- Delay,
- Delay (Selector),
- Scan,
- Switch, ...

Receiving Events

If you want to leave the observable world and pass event values to the mainloop use one of the 3 nodes

- HoldLatest: Returns always the latest value
- Sampler: Returns all event values since the last frame, can be empty
- S+H: Same as *Sampler* but returns the same values until the next event occurs

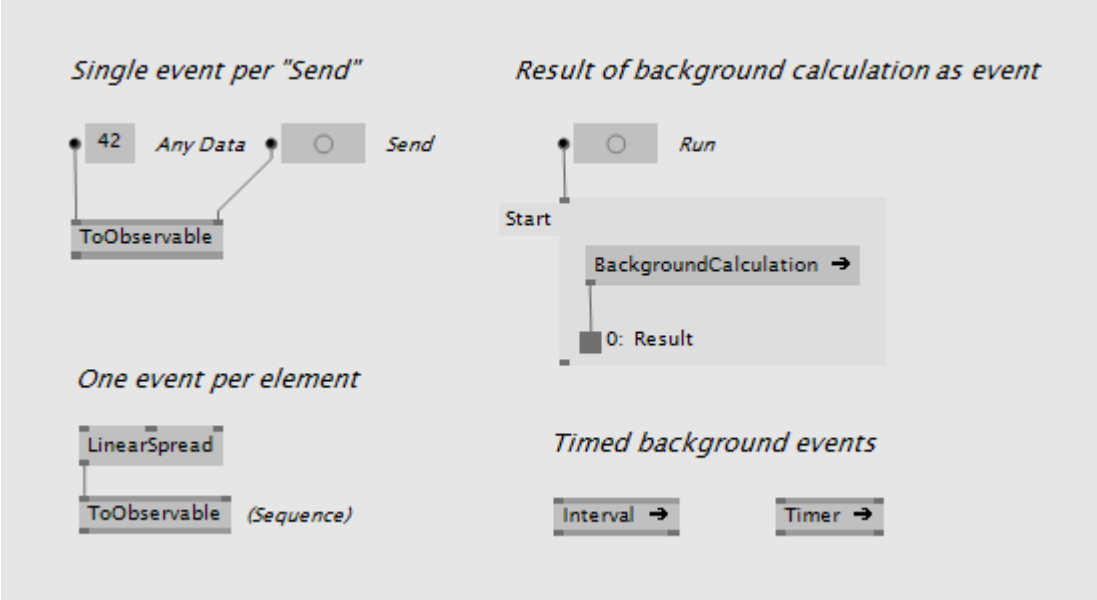
which all behave a little bit different. Depends on what you need:



Three ways to get event values to the mainloop

Creating Events

It's also easy to generate event sources of your own:



Different ways to create observable event sources

Note

Only send values of type Record as event data because they are thread safe. If you send values of any Class type be sure that you know exactly what you are doing!

Serialization and Deserialization

Serialization is the process of translating data structures into a format (text or series of bytes) that can be stored (for example, in a file) or transmitted (for example, across a network) and reconstructed later. The opposite operation, extracting a data structure from a serialized format, is called deserialization.

via Wikipedia

Common Formats

There are many ways this can be done but there are three commonly used text formats for serialization, called XML, JSON and CSV. While binary formats are typically smaller, which results in faster read/write times, the advantage of text formats is human readability which helps with debugging and version-control. When interacting with the web it is common to choose JSON as this can be easily parsed and created from java-script.

Automatic

When you need to send data-structures from one instance of your application to another over a network and the data does not have to be saved to disk, chances are that you don't care about the actual format.

In this case you can use one of the runtime serializers provided that can serialize most datatypes directly without the need for building an extra data-structure for serialization. While this is quick and easy, it comes at the cost of not having any control over the format and thus may have some overhead in the formats size, as the serialization process may include data that you'd not even need to serialize.

For serialization to XElement (ie. XML) best use the nodes from the `System.Serialization` category:

- Serialize -> XElement -> Deserialize
- Serialize (Log Errors) [Advanced] -> XElement -> Deserialize (Log Errors)

Further, the following nodes from the package `VL.Serialization.FSPickler` can be used:

- Serialize (XML) -> String -> Deserialize (XML)
- Serialize (JSON) -> String -> Deserialize (JSON)
- Serialize (Binary) -> MutableArray of Byte -> Deserialize (Binary)

Note

The serialized format these nodes generate is "volatile" in a sense that it may not be compatible between different versions of VL.

Also: Deserializing JSON or XML that was not generated with the corresponding Serialize nodes will fail, if the individual attributes are not ordered alphabetically! This is a peculiarity of FsPickler and we'll

therefore probably have to replace it at some point. Meanwhile you may want to have a look at using nodes from the Json.NET library for such cases.
In case you want to learn more about the inner workings of those nodes, check the documentation of the FSPickler library on which they are based on.

image: example MyType automatically serialized and deserialized

Manual

When you're saving the state of a program to disk, you may want think about different versions of your file-format because the data-structure you're saving may evolve over time and you may still want your application to be able to load files saved with earlier versions.

In this case you'll want to define the format yourself as it is then in your hand to change the format whenever needed, adapt the serialization and deserialization process accordingly and have the option to retain backwards-compatibility by providing different de/serializers for different versions of your format.

Custom Serialization

The following nodes allow you to build data-structures

- XElement (Join) [XML]
- XAttribute (Join) [XML]

that can then be serialized to a string format using:

- ToJSON [XML]
- ToString [XML]

or can be directly saved to disk using:

- FileWriter (JSON) [IO]
- FileWriter (XML) [IO]

Building an extra data-structure that is only used for serialization is an overhead but also has the advantage that you can leave things out that you don't need to serialize and define exactly how the resulting format will look like.

image: example data-structure and its serialization as XML and JSON strings

Custom Deserialization

The following nodes allow you to read JSON or XML files from disk:

- FileReader (JSON) [IO]
- FileReader (XML) [IO]

or parse JSON or XML strings into a an XElement structure:

- ParseJson [XML]
- Parse [XML]

Then use the following nodes to access individual fields of the data-structure:

- XElement (Split) [XML]
- XAttribute (Split) [XML]
- XElementsByName [XML]
- XPathSelectElement [XML]
- XPathSelectElements [XML]
- XPathEvaluate [XML]

image: deserializing a given example json/xml

JSON

Instead of operating on a JSON object directly, in VL by default the JSON is converted to an XElement which can be easily inspected and modified. Vice versa, an XElement can always be converted to JSON.

Note

If in an advanced use case you find yourself in a situation where this conversion from JSON to XElement is not feasible, you can still operate on JSON directly by referencing a library like JSON.NET, as described in Using .NET Libraries.

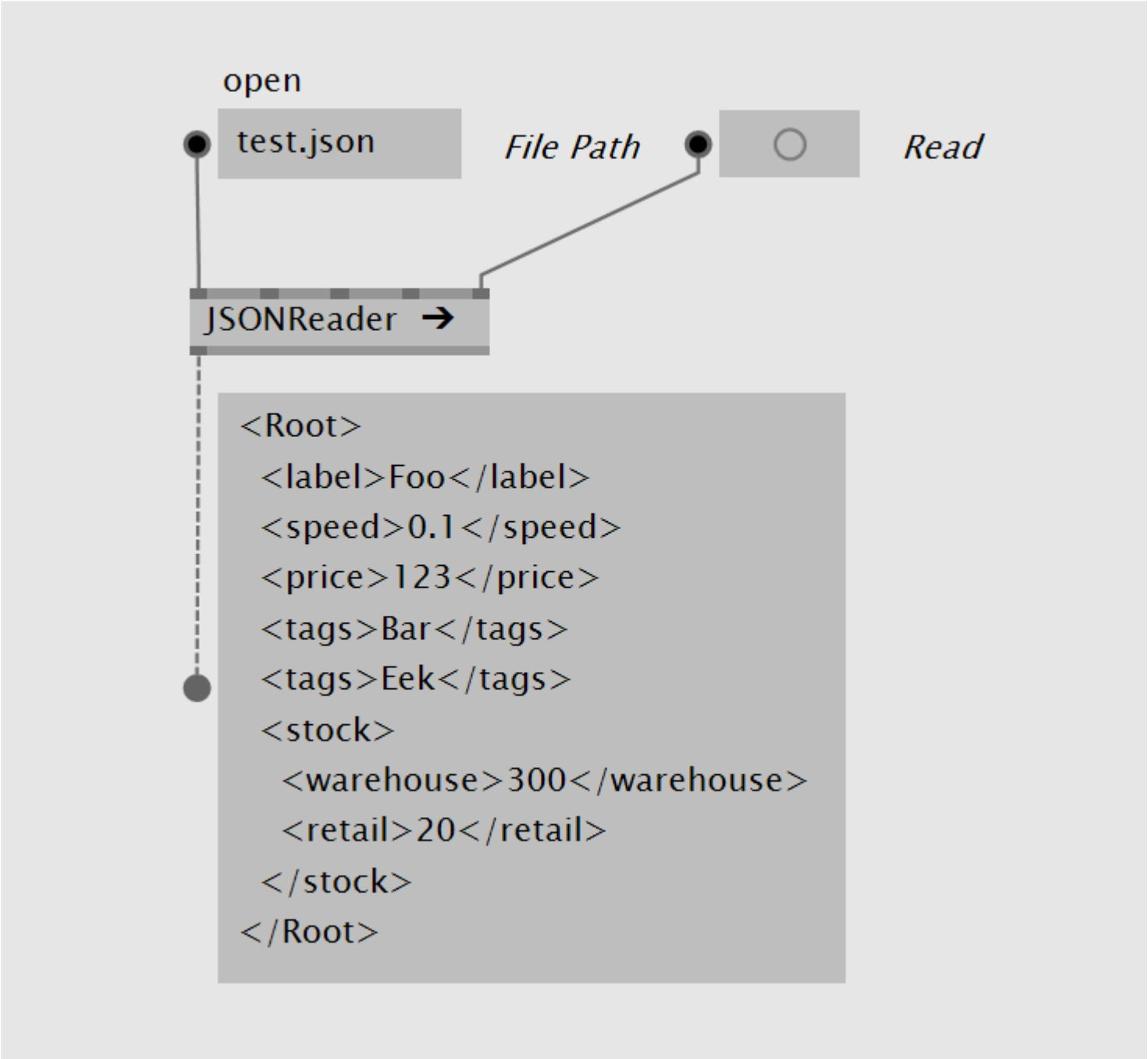
Loading a JSON file

Loading a file is potentially a time-consuming process and thus can interrupt your otherwise smooth framerate. VL therefore provides two options for file readers:

- a simple to use, but blocking option
- a non-blocking option that requires a few more clicks to set it up

Blocking

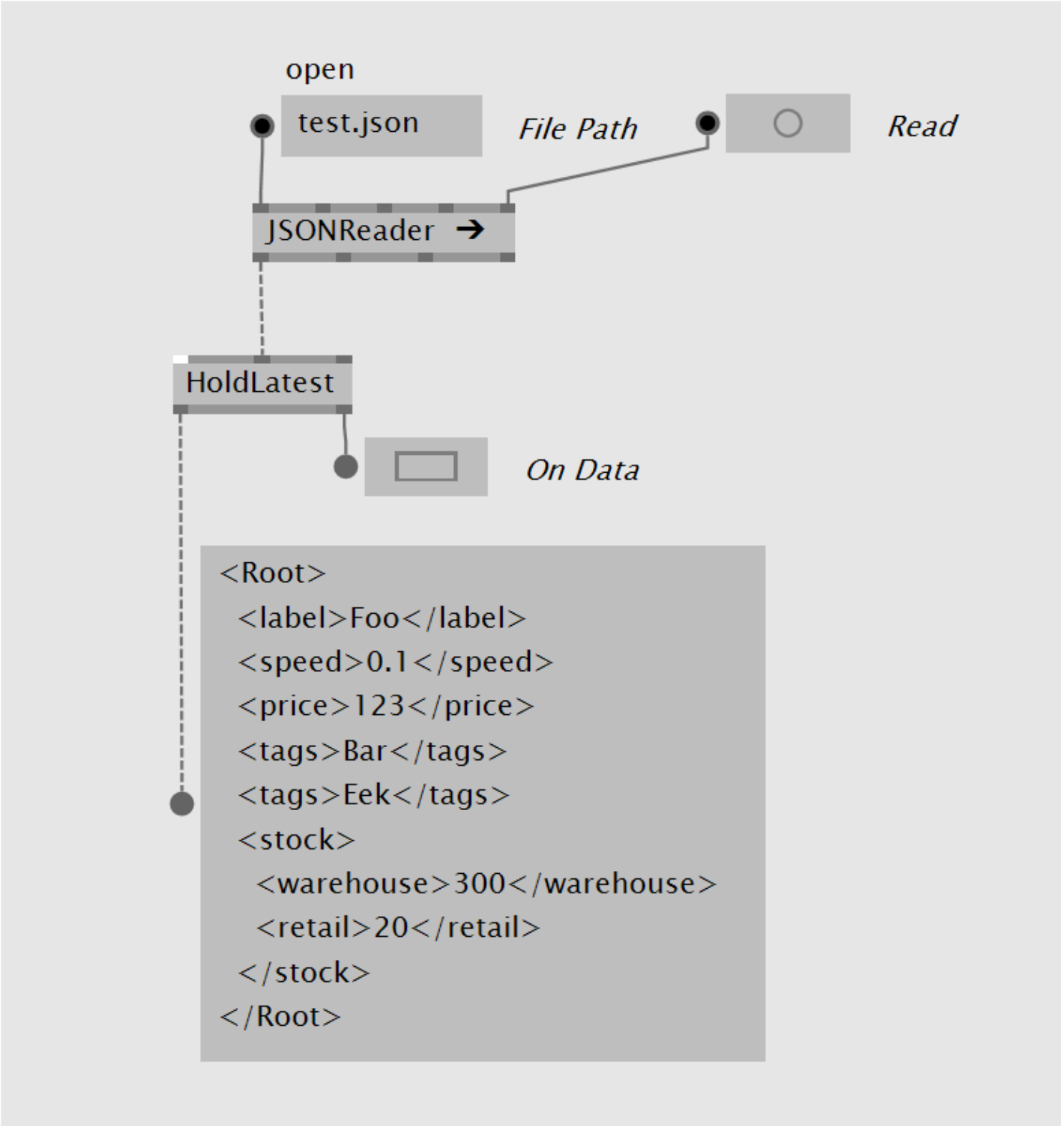
Use the *JSONReader [System.XML]* node to read a .json file and get the result in the form of an XElement:



The JSONReader loads a .json file and returns it as XElement instantly, potentially blocking the execution of the rest of the program

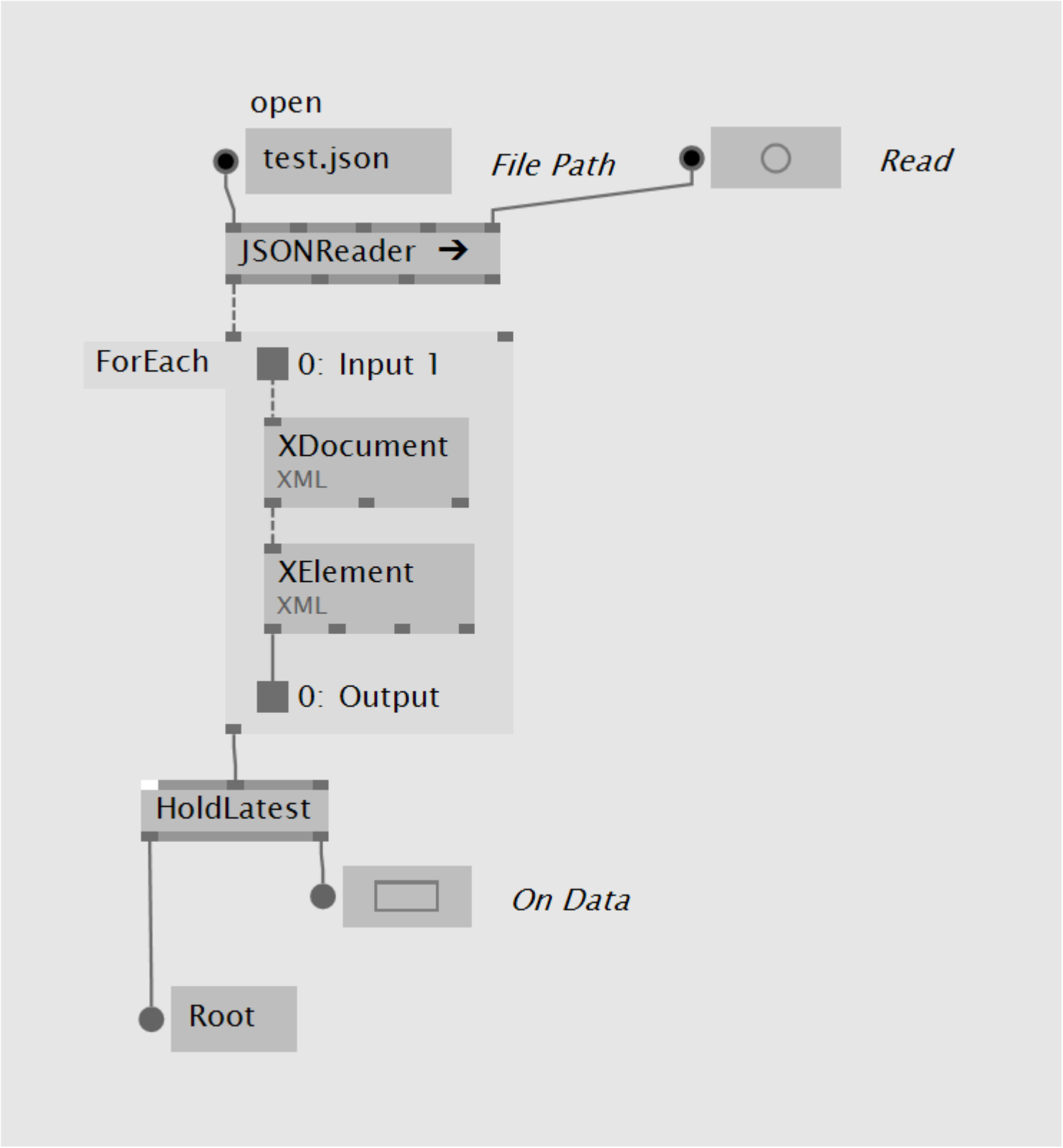
Non-Blocking (Reactive)

The simplest way to load a .json file asynchronously is to use the *JSONReader (Reactive) [System.XML]* in connection with a *HoldLatest [Reactive]*. Once the file is loaded, the HoldLatest will bang its **On Data** output and return the files content as an XElement:



The JSONReader (Reactive) loads a .json file and returns it as XElement in a later frame, so that the rest of the program is not interrupted

But while you're in the reactive/asynchronous world, you can also do some further parsing to the file, by e.g. using the *ForEach [Reactive]*:

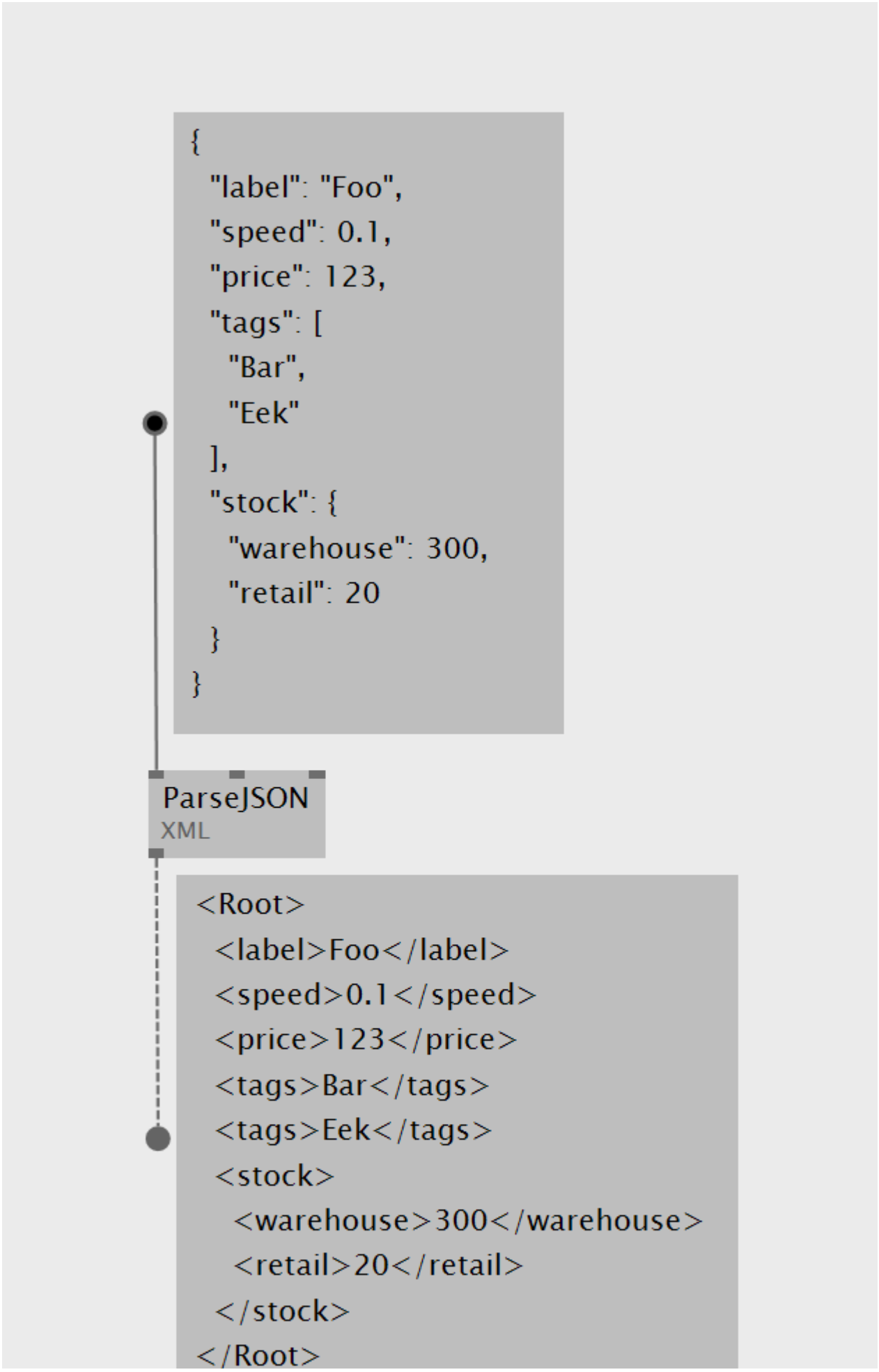


In addition to loading the file asynchronously, in this example the name of the root element is being extracted as a simple example. But obviously here you can do more expensive operations that would still not interrupt your framerate

Like this loading and parsing is done asynchronously and only when both is done, you get access to the result for further processing.

Parsing a JSON string

If you have a string in JSON format then simply use the *ParseJSON [System.XML]* node to convert it into an XElement for further processing:


{
"label": "Foo",
"speed": 0.1,
"price": 123,
"tags": [
"Bar",
"Eek"
],
"stock": {
"warehouse": 300,
"retail": 20
}
}
ParseJSON
XML
<Root>
<label>Foo</label>
<speed>0.1</speed>
<price>123</price>
<tags>Bar</tags>
<tags>Eek</tags>
<stock>
<warehouse>300</warehouse>
<retail>20</retail>
</stock>
</Root>

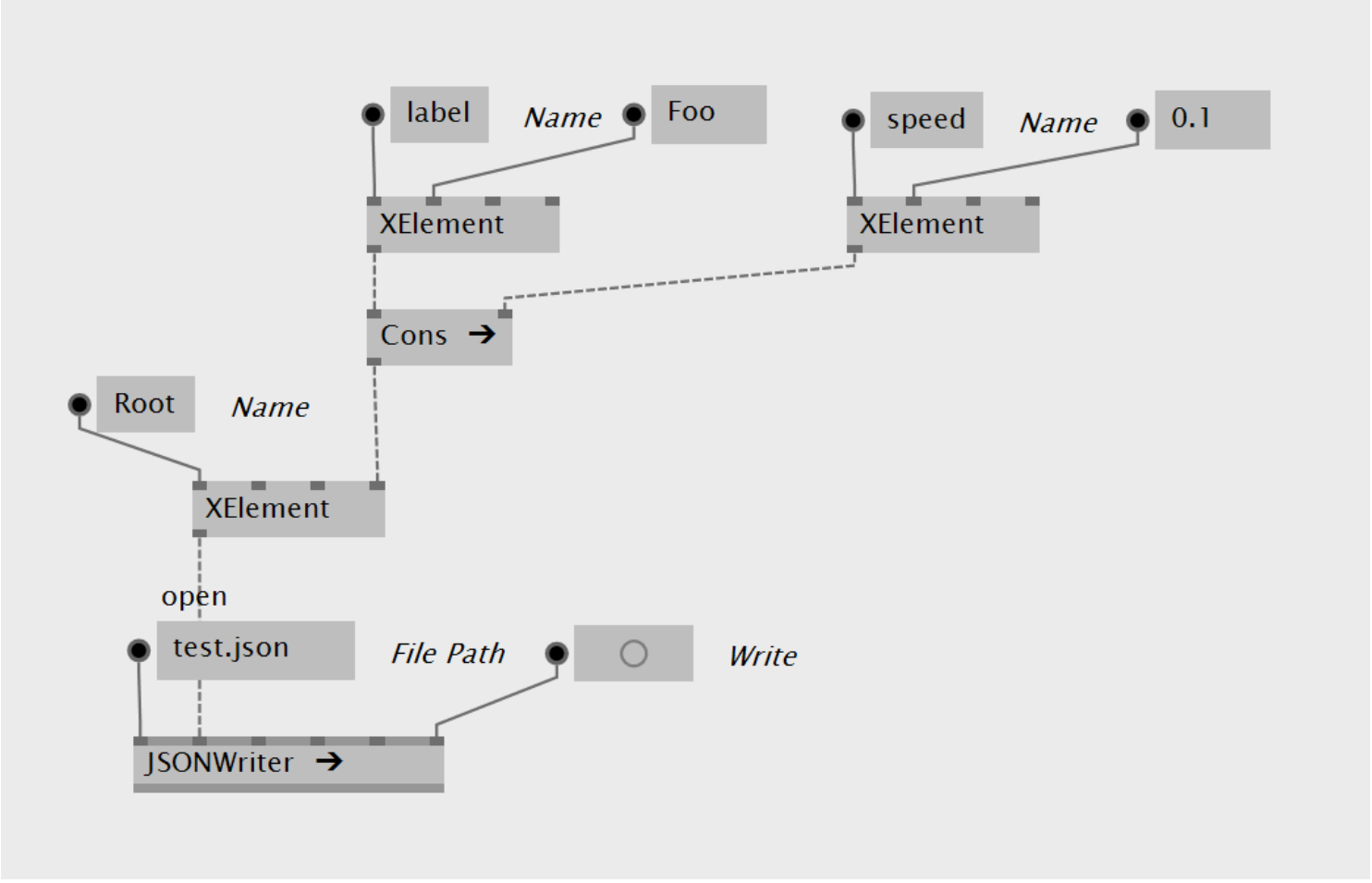
Saving a JSON file

Saving a file is a potentially time-consuming process and thus can interrupt your otherwise smooth framerate. VL therefore provides two options for file writers:

- a simple to use, but blocking option
- a non-blocking option that requires a few more clicks to set it up

Blocking

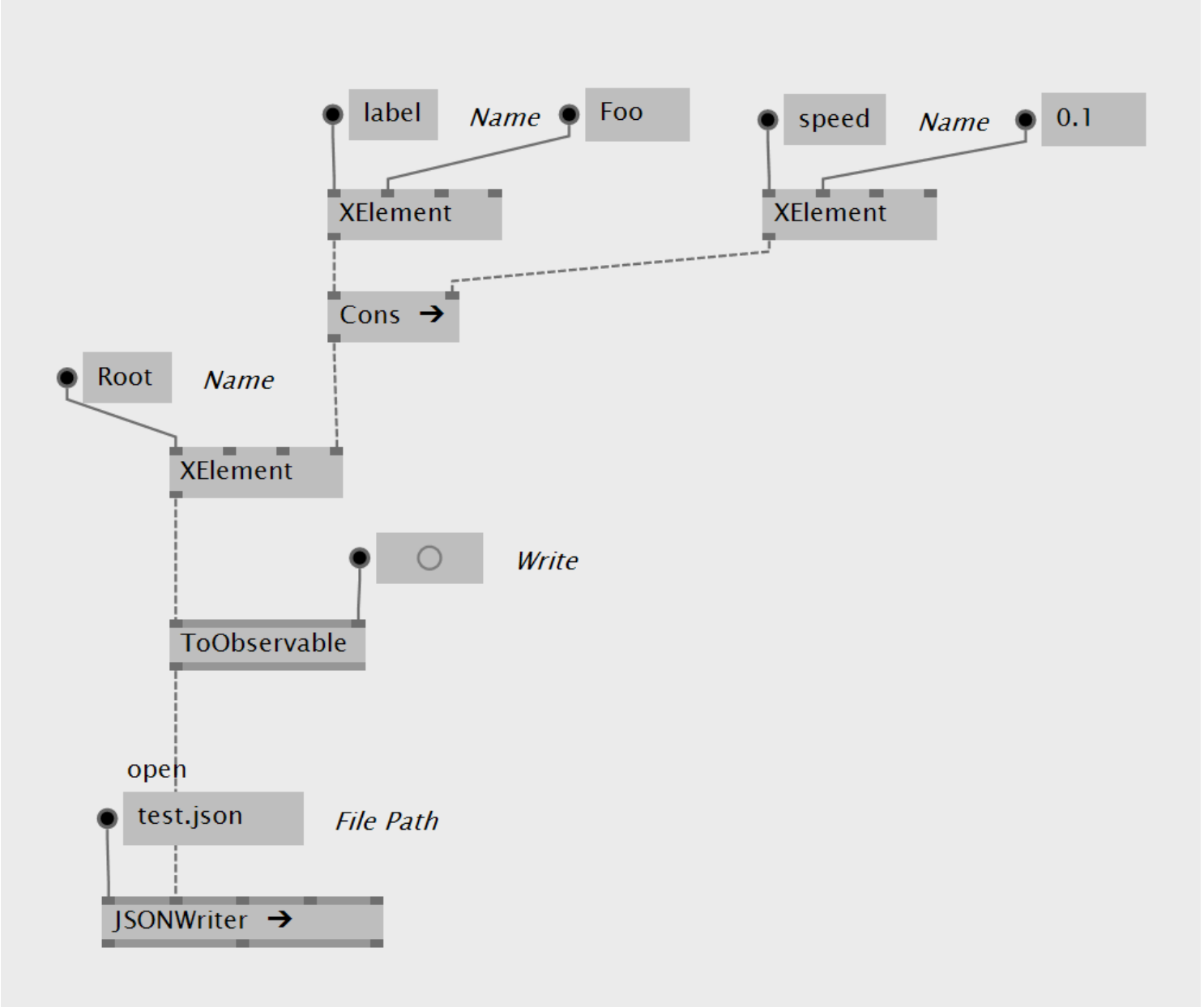
Use the *JSONWriter [System.XML]* node to write a given XElement into a .json file:



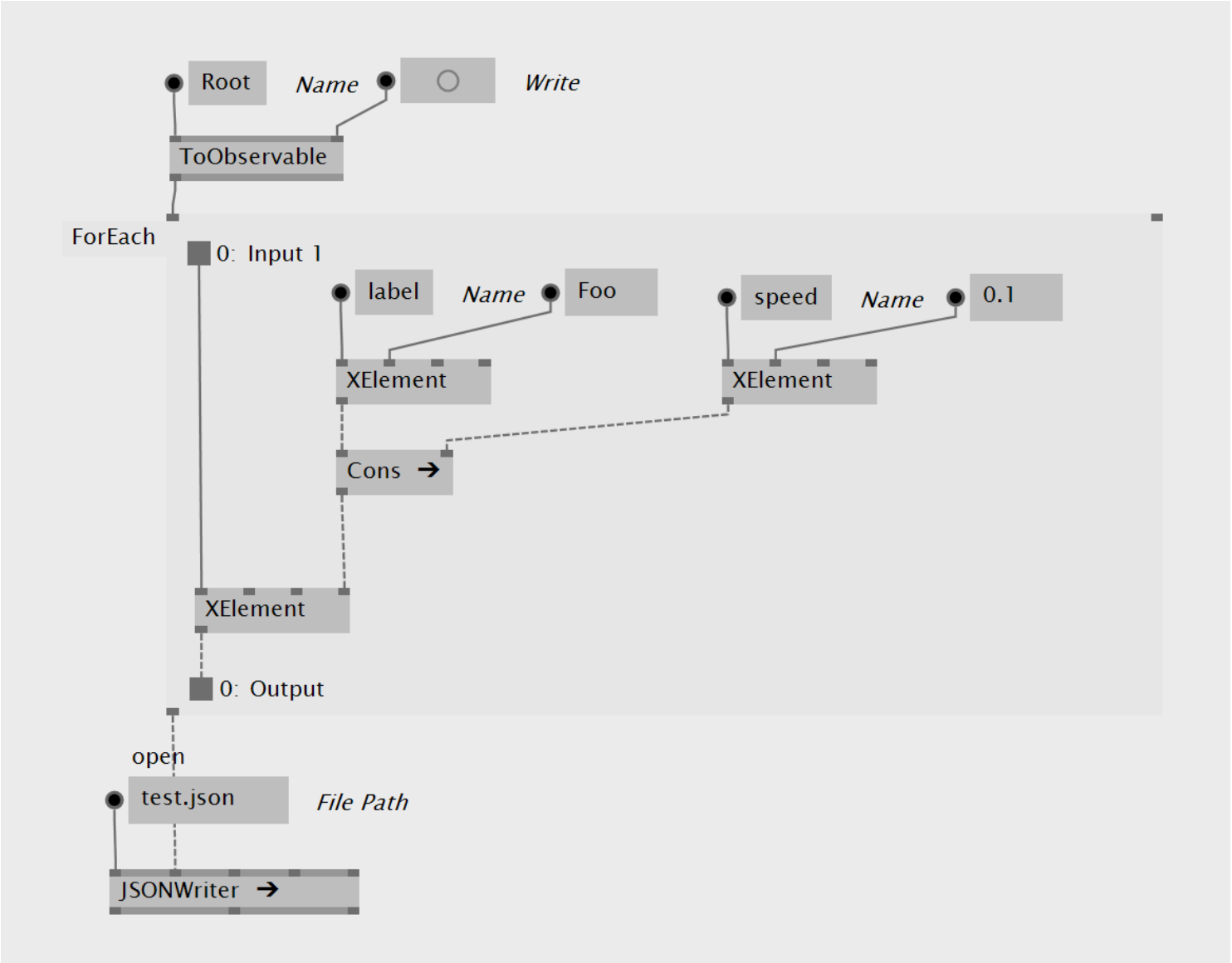
Non-Blocking (Reactive)

The simplest way to save a .json file asynchronously is to use the *JSONWriter (Reactive) [System.XML]* in connection with a *ToObservable [Reactive]*. Connect the XElement to the **Message** input of the

ToObservable node and bang its **Send** input to start the operation. Once saving is done, the **On Completed** output of the JSONWriter will bang:



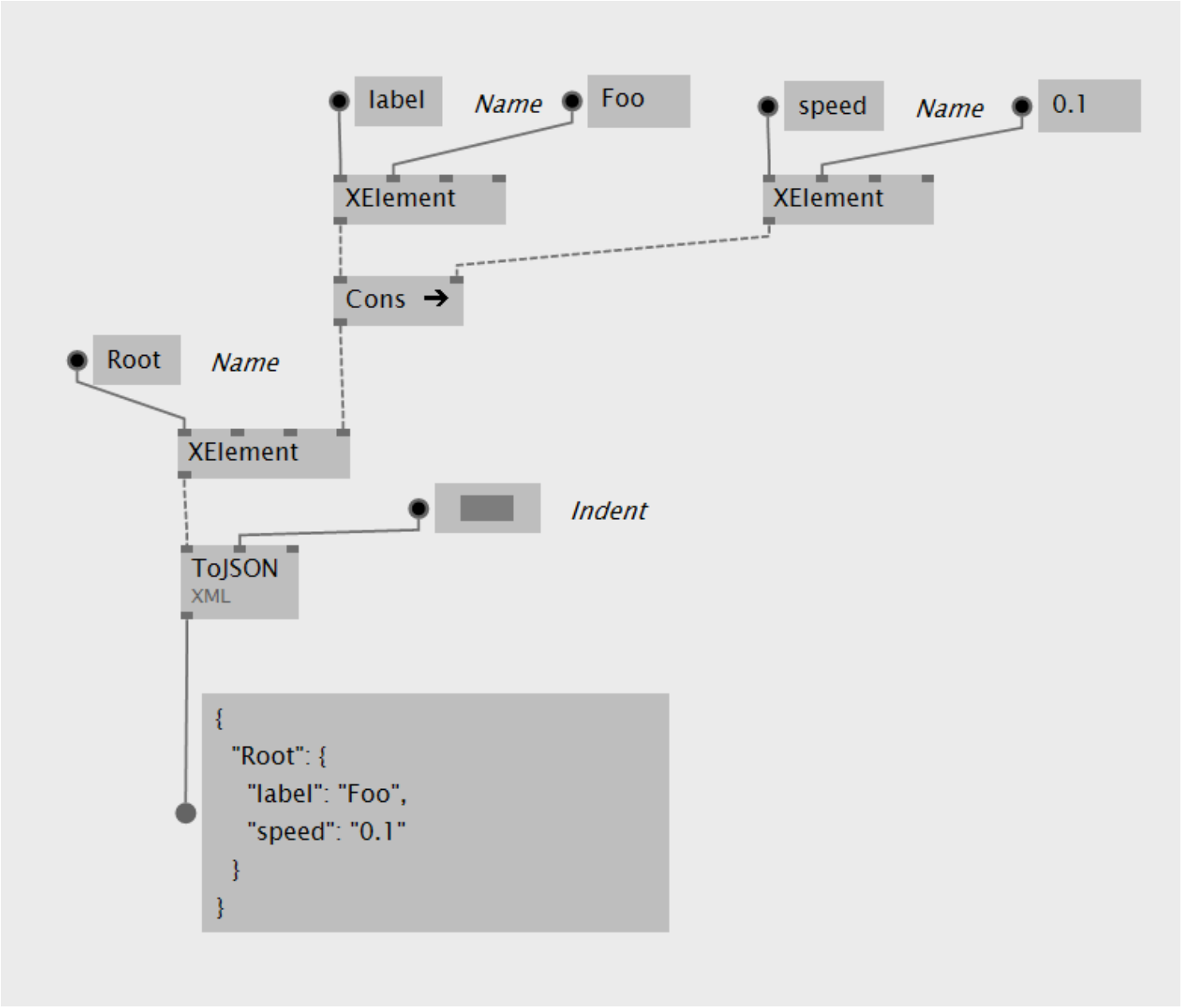
But even creating the XElement structure could already be time-consuming so you could also off-load that part of your patch to the reactive world and do it only just before writing the file by e.g. using the *ForEach [Reactive]*:



Like this creating the XElement and saving the file is done asynchronously and does not interrupt your framerate.

Converting an XElement to a string in JSON format

If you have an XElement that you want to simply convert to a string in JSON format, use the *ToJSON [System.XML]* node:


label
Name
Foo
speed
Name
0.1
XElement
XElement
Cons
Root
Name
XElement
Indent
ToJSON
XML
{
"Root": {
"label": "Foo",
"speed": "0.1"
}
}

XML

The datatype for handling XML data in VL is called *XElement*.

In the nodebrowser explore the category *System.XML* to get an overview of all the nodes available for operating on XElements.

The examples on this page refer to the following XML structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<entries>
  <entry visible="true" >
    <id>1</id>
    <label>Foo</label>
    <description>A Thing</description>
    <speed>2.4</speed>
  </entry>
  <entry visible="false">
    <id>2</id>
    <label>Bar</label>
    <description>Another Thing</description>
    <speed>4.2</speed>
  </entry>
</entries>
```

Extracting data from an XElement using XPath queries

General information about XPath can be found at W3CSchools: XML and XPath

Each XElement can have:

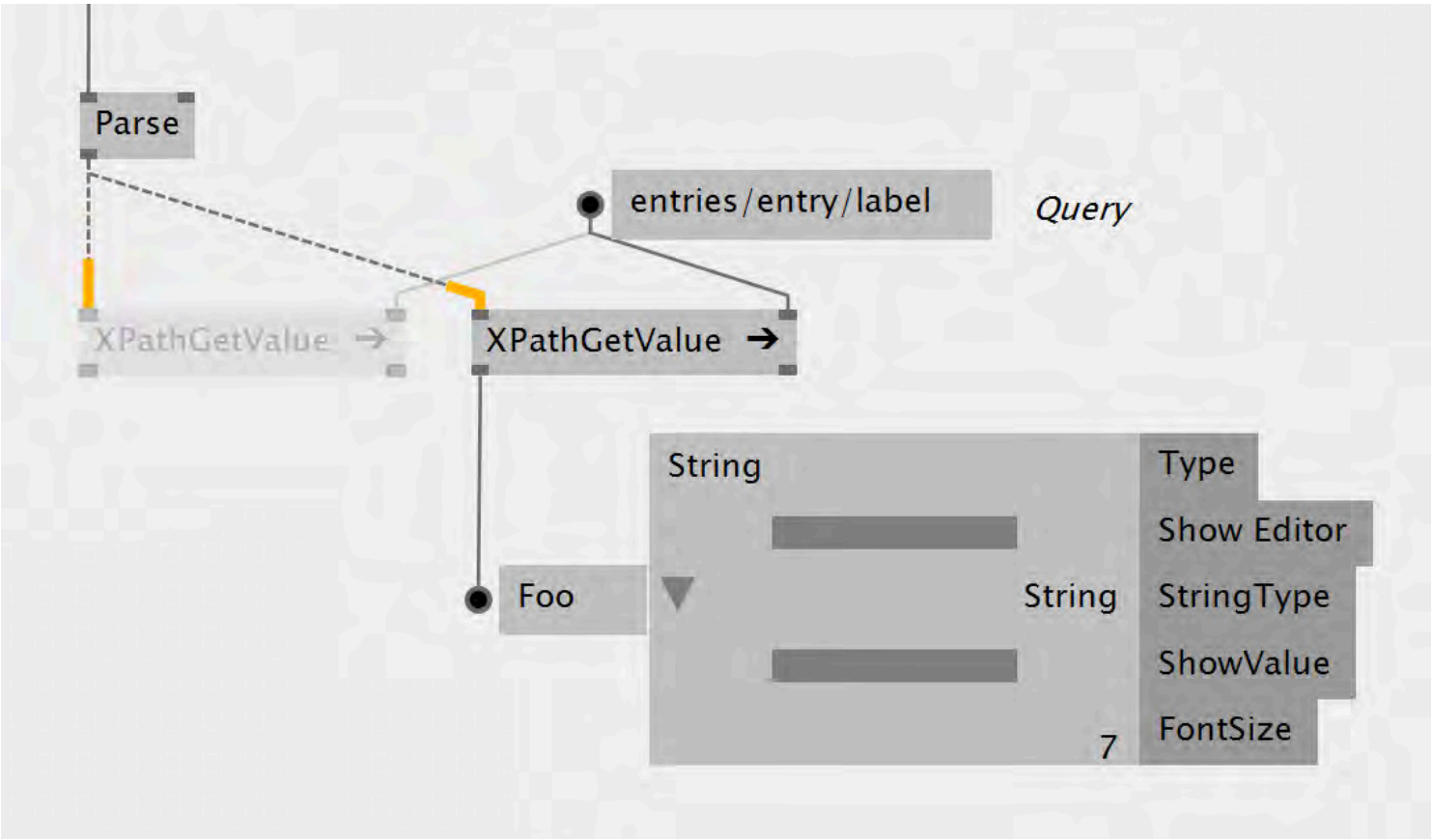
- a value
- a list of XAttributes
- a list of child XElements

Referring to the example above:

- the value of the first "label" element is "Foo"
- the element "entry" has one attribute with the name "visible". the value of the first entry's "visible" attribute is "true"
- the child elements of the "entry" element are: "id", "label", "description" and "speed"

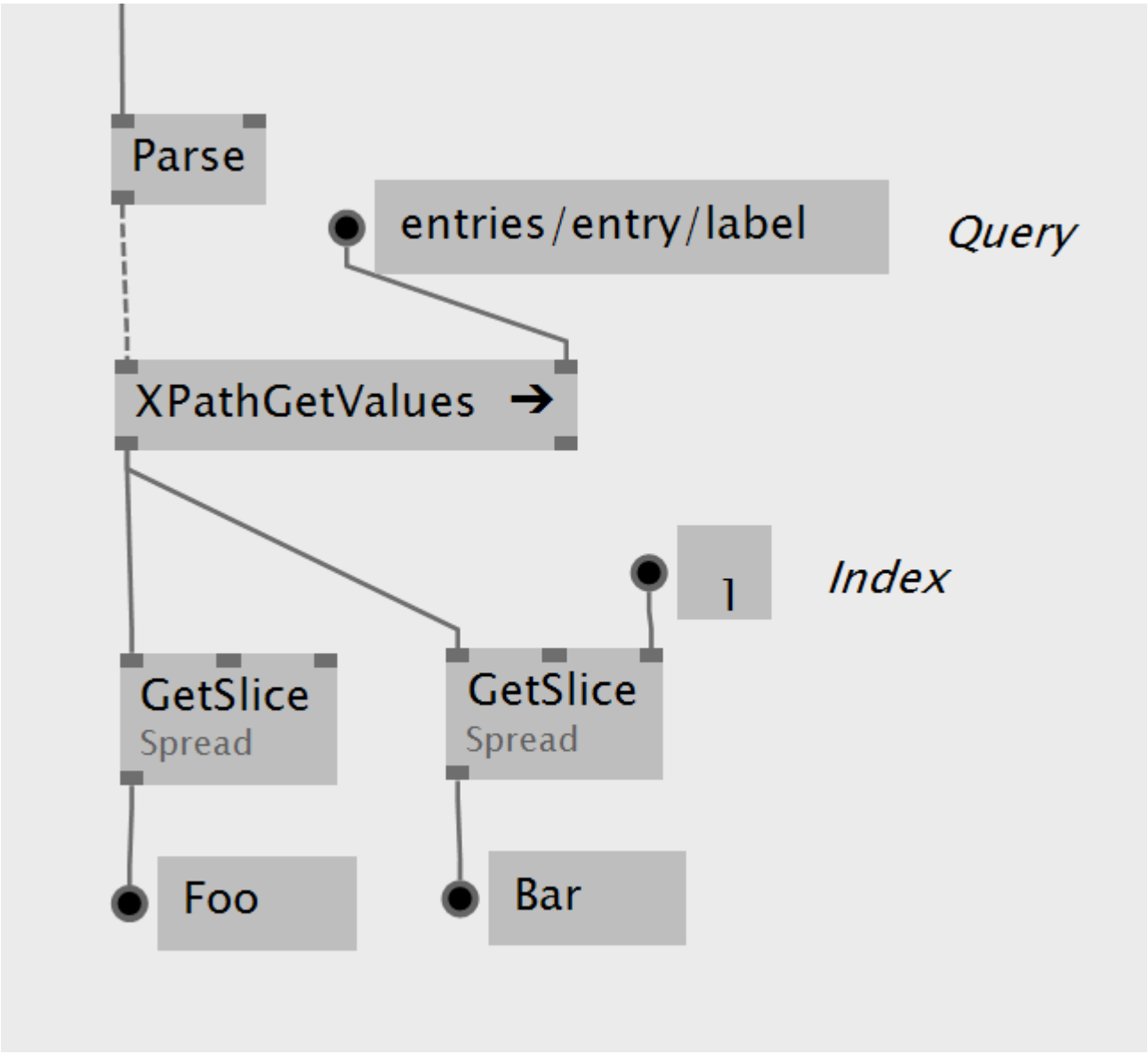
Accessing an element's value

To access the value of only the first occurrence of an element, use the *XPathGetValue [System.XML]* node:



The first XPathGetValue node is grayed out (ie. not in use), because it does not yet have anything connected to its output. The second node has an IOBox connected that is configured to type 'String' which tells the XPathGetValue node to interpret the XElements value as a string

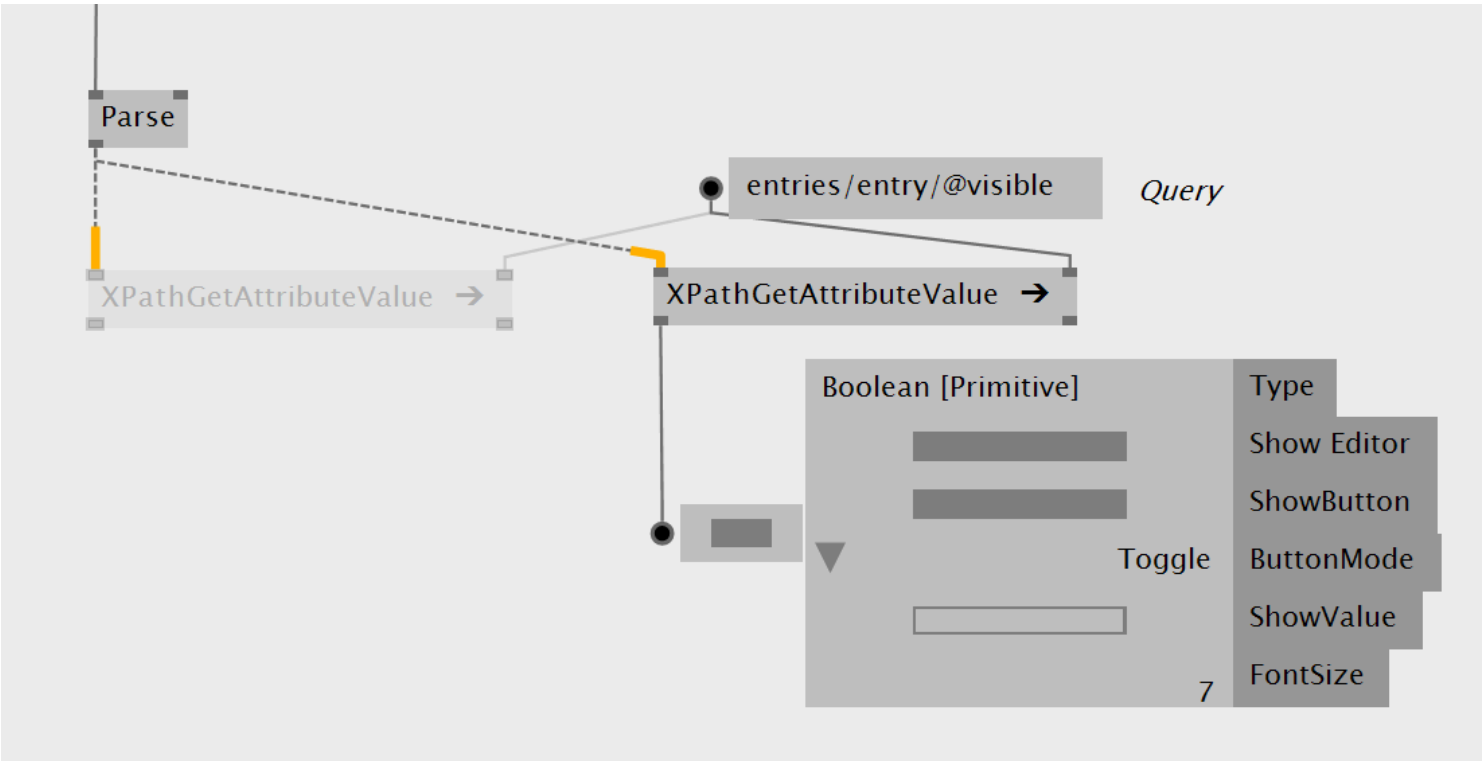
To get the values of all occurrences of an element, use the *XPathGetValues [System.XML]* node:



XPathGetValues returns the values of all queried elements as spread of whatever type is connected

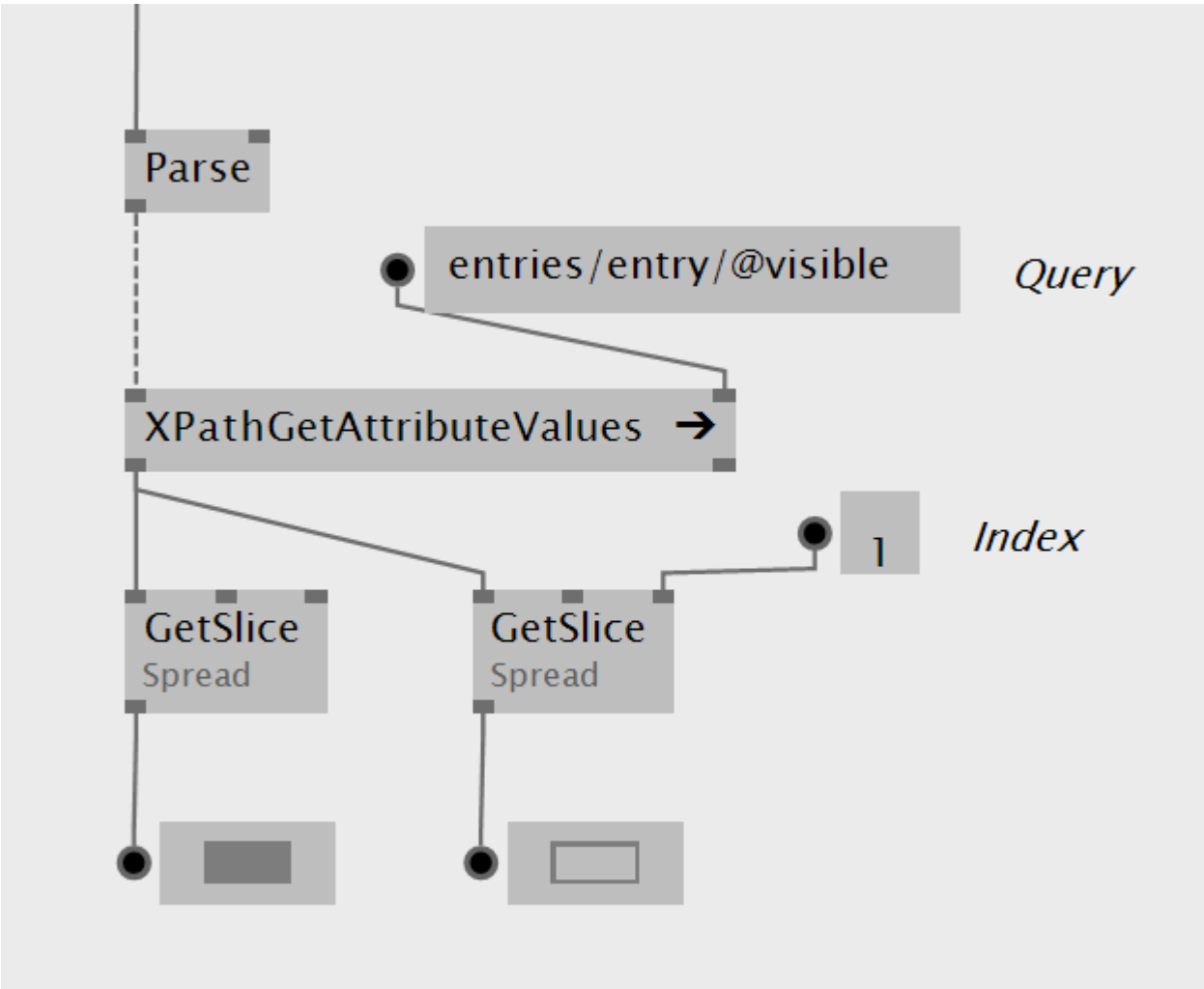
Accessing an element's attributes

To access an attribute of only the first occurrence of an element, use the *XPathGetAttributeValue [System.XML]* node:



The first XPathGetAttributeValue node is grayed out (ie. not in use), because it does not yet have anything connected to its output. The second node has an IOBox connected that is configured to type 'Boolean' which tells the XPathGetAttributeValue node to interpret the XElements value as a boolean

To get the attributes of all occurrences of an element, use the *XPathGetAttributeValues [System.XML]* node:



XPathGetAttributeValues returns the values of all queried attributes as spread of whatever type is connected

2d Graphics

vvvv's 2d rendering engine VL.Skia is based on SkiaSharp and ships with the vvvv installation.

> Skia is an open source 2D graphics library which provides common APIs that work across a variety of hardware and software platforms. It serves as the graphics engine for Google Chrome and Chrome OS, Android, Flutter, Mozilla Firefox and Firefox OS, and many other products.

It comes with an extensive collection of help patches documenting the different aspects of the library. Browse them via the HelpBrowser.

Skia Features

- Drawing 2d primitives and paths
- Drawing images and SVGs
- Drawing text
- Rendering of Lottie animations
- Export as image, SVG or PDF

Additional Libraries

- NuGet: VL.Elementa is an advanced widget and layouting library for VL.Skia
- NuGet: VL.CEF.Skia is a HTML renderer
- NuGet: VL.PolyTools extends VL.Skia Paths with high level polygon and polypath objects

3d Graphics

vvvv's 3d rendering engine VL.Stride is based on the Stride 3d Engine and shipping with the installation. It allows for 2 distinct workflows:

- A **high-level**, easy to use SceneGraph approach, where you build 3d scenes by simply adding models, and lights to a scene. Models can be given materials to define their look
- A **low-level** approach, where you work with the graphics API directly

Both workflows can be easily combined, see Rendering for more details.

You you can write shaders (vertex, pixel, geometry, compute) using the Stride Shading Language (an extension to HLSL) to customize your rendering in both workflows.

A range of Post FX, like ambient occlusion, depth of field, bloom, etc. are available too. VL.Stride also allows you to output content to VR Devices.

In general the Stride Documentation is useful for understanding key concepts of the engine.

Topics

- Rendering
- Models and Meshes
- Geometry
- Text rendering
- Transparency
- Shaders
- All about TextureFX shaders
- Editing shaders
- Projection Mapping
- Graphics cards

Additional libraries:

- NuGet: VL.Fuse is a collection of GPU tools and libraries to use with VL.Stride. Think: Distance Fields & Raymarching, Particles, Procedural Geometry, Textures and Materials, GPGPU. For details, see The FUSE Lab
- NuGet: VL.CEF.Stride is a HTML texture renderer
- NuGet: VL.Addons for a wide range of additional TextureFX
- NuGet: VL.Alembic for loading Alembic files
- NuGet: VL.Stride.Text3d for rendering extruded 3d text
- NuGet: VL.Stride.SDFToMesh for converting an SDF function to a mesh
- NuGet: VL.IO.PLY for loading PLY pointcloud files

- NuGet: VL.IO.Teximp for texture IO and processing
- NuGet: VL.Assimp is an alternative 3d model loader for Stride
- NuGet: VL.OpenEXR for loading OpenEXR image files
- NuGet: VL.Boids-GPU Boids algorithm implemented Stride
- NuGet: VL.Radiosity 2D Radiosity Shader

Useful tools

- Stride Shader Explorer to browse available shaders to inherit from (requires also Stride to be installed)
- List of Material Editors
- ALVR to stream VR content to headsets via Wi-Fi

For an alternative, very primitive wireframe 3d engine, see VL.Skia3d: NuGet: VL.Skia3d

Rendering

VL.Stride offers two workflows for rendering:

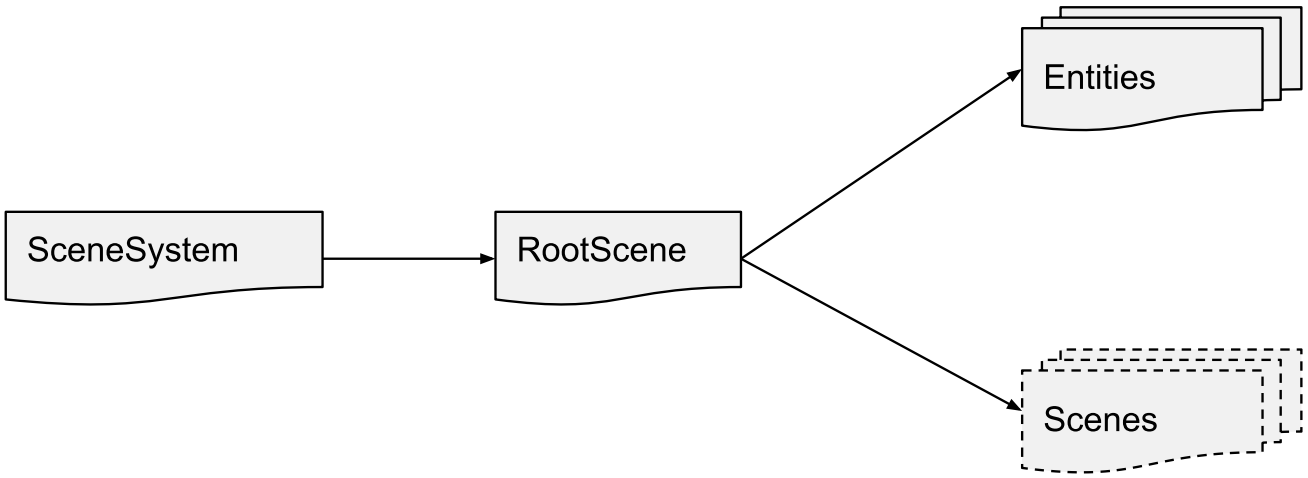
- **High-level**: Work with models, lights, materials, textures (Entity-Component-System)
- **Low-level**: Work with draw calls, pipeline states, and GPU resources directly

If you worked with a game engine before, then you've used the high-level approach. If you're coming from vvvv beta and you worked with DX9/DX11, then you've been using the low-level approach.

Both workflows can be combined without any drawbacks, and both can render into a texture or an output window. You can also write shaders for both.

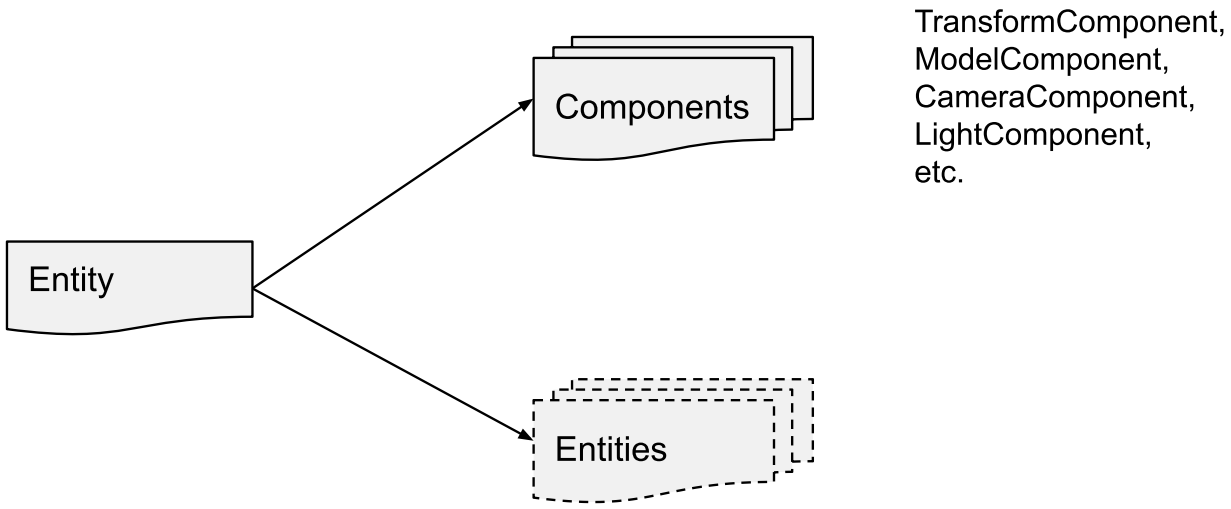
High-level (scene graph)

Commonly known as entity-component-system (ECS). The scene graph consists of a tree of scenes that have entities.



Scene graph data structure

Each entity has a list of components that define the behavior and functionality of the entity. An entity can also have a list of child entities.



Entity data structure

Every entity has a `TransformComponent`. Child entities will multiply their transformation with the parent transformation.

To build the scene graph you can use the `Group` or `Group (Spectral)` nodes in the category `[Stride]`. A group node is technically just an entity that sets the input entities as its child entities.

Root nodes

`SceneWindow` and `SceneTexture` both set up the scene system. Connect a `RootScene` to either of them and start building the scene graph from there.

See the help patches: `Overview Scene Graph Basics`, `Overview Scene Graph Advanced`, and `Work with Children`.

Camera

The `SceneWindow` node comes with a built-in default camera that can be used with the mouse to look around in the scene. The default camera can be overwritten by connecting a camera to the *Camera* input pin.

To build your own camera, you can use the Entity node and connect a `CameraComponent` to it, or use the `Camera` node that combines these two. The help browser has a section for cameras with several help patches.

Models

See Models and Meshes.

Lights

A light component can be attached to any entity and it will use the transformation of the entity as the light transformation. The help browser has a dedicated section for lights with many help patches.

See also: Stride Lights and Shadows doc

Post Effects

The Stride render pipeline has a set of post-processing effects that can be applied to the rendered 3d scene. Such as ambient occlusion, bloom, and other screen space or image base effects.

The help browser has a `PostFX` section with many help patches.

See also: Stride Post Effects doc

Low-level (custom rendering)

This workflow allows you to manage your own draw calls with the graphics API. It takes more effort to use because you need to know about shaders, buffers, pipeline states, and other graphics API features.

The main data type is `IRenderer`. This interface can be implemented to take part in the rendering by connecting it to a render sink. The `MeshRenderer` or `QuadRenderer`, for example, are implementations of this interface.

You can order draw calls with the `Group` and `Group (Spectral)` nodes in the category `[Stride.Rendering]`. These group nodes are implementations of `IRenderer` that pass the draw call to the renderers connected to the input.

Renderer sinks

There are several sinks to which you can connect an `IRenderer`. Depending on the use case and the moment in which you want to render.

`RenderEntity`

To participate in the scene rendering, this node can be placed in the scene graph and will pass on the draw call of the `SceneWindow` or `SceneTexture` to the connected `IRenderer`. It also has a setting to specify the render stage of the scene:

- `BeforeScene`: non-graphical, useful to prepare buffers or textures for the scene
- `Opaque`: the normal 3d render stage

- `Transparent`: the transparent stage, after Opaque
- `AfterScene`: after the scene, this can be used to draw into the final render target
- `ShadowCaster*`: these stages can be used to render into the shadow maps

`RenderTexture`

Render something into a texture with a specified size and format. Useful for rendering helper textures, such as masks, text, or other basic graphics that will be used in the scene.

`RenderWindow`

Render something into a window directly, without the high-level scene setup. Useful to display a fullscreen texture or to compose the final output of an application.

`RendererScheduler`

A very low-level node that schedules a draw call without a sink. It is used, for example, by the TextureFX nodes to render into a texture.

If there is more than one `RendererScheduler`, then the order in which they are called in the update loop, is the order in which they will be called during the rendering.

For more details, see also: Stride Low-Level API doc and Programming guide for Direct3D 11

Models and Meshes

You can load a model from file with the `FileModel` node. The following file types are supported:

- `.fbx;.dae;.3ds;.gltf;.glb;.obj;.blend;.x;.md2;.md3;.dxf;.ply;.stl;.stp`

The loaded model can be connectd to a `ModelEntity` to render it with a material.

At the moment we don't support automatic loading of materials, textures, animations or skeletons. This will be added in a later release. See the help patch `Load Assets from File` for an example, that also shows how to assign multiple materials to the model.

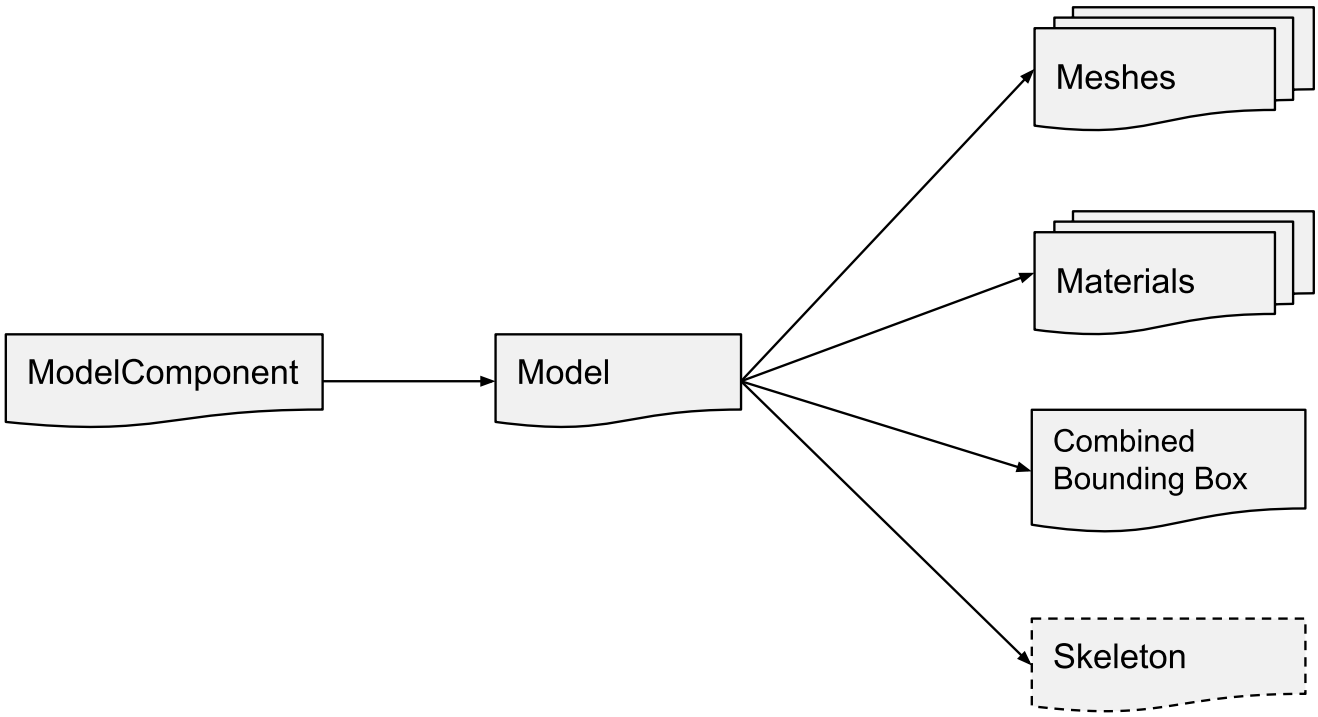
Models can also be loaded from a Stride game project. This has the advantage that you can set up the model, including materials in the Stride editor. See Assets and Animation int the Stride manual. See the help patch `Load Stride Project` and `Modify Entities from a Stride Project`.

Difference between Model and Mesh

Model

A `Model` is a high-level class that combines geometry (meshes) with appearance (materials) and optionally a skeleton for animation. This makes it possible to represent a simple model with just one mesh and one material or a complex 3d object, such as an animated character.

In the scene graph, a model has to be assigned to a `ModelComponent` that is part of an `Entity`. See Rendering.

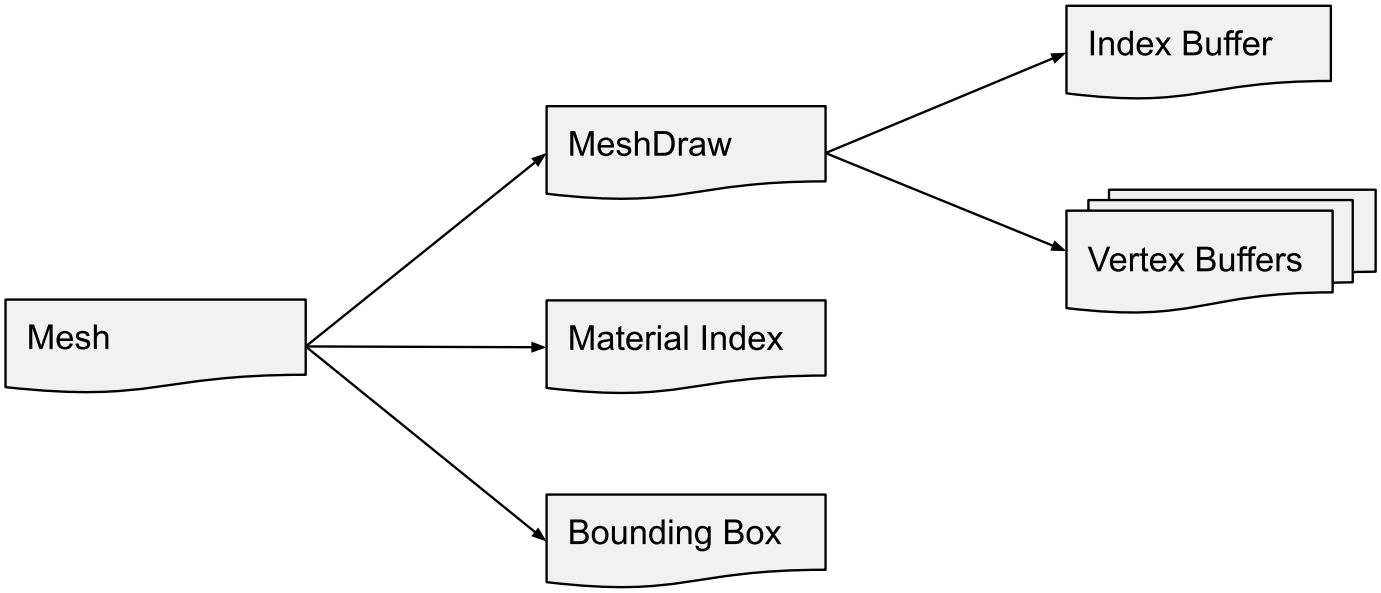


Model data structure

To join the data of a model you can use the node `MeshModel` for the simple case of one mesh and one material or `MeshesModel` for multiple meshes and materials. To connect a single model to the scene graph, you can use the node `ModelEntity` that internally does all the entity and component setup for you.

Mesh

A `Mesh` is a part of the model that contains the geometry information and an index that points to a material in the material list of the model.



Mesh data structure

The actual geometry data is stored in a class called `MeshDraw`, it holds the GPU resources for the index and vertex buffers that will be used to draw the geometry. In detail, the `MeshDraw` has `IndexBufferBinding` and `VertexBufferBinding` properties that hold the respective buffer plus some information for the graphics pipeline. So the full path from model to first vertex buffer is: `Model.Meshes[0].Draw.VertexBuffers[0].Buffer`.

The vertices stored in a vertex buffer can have different fields, such as normals, texture coordinates, etc. To make the workflow of getting the vertex data easier, you can use the `MeshSplit` nodes, see the help patch `Split a mesh into its components`.

Dynamic Mesh

The nodes `DynamicMesh` or `DynamicMesh (Indexed)` create a mesh from vertex and/or index data.

To connect a single mesh to the scene graph, you can use the node `MeshEntity` that internally does all the entity and component setup for you. A mesh can also be rendered with the low-level workflow using a `MeshRenderer`. See Rendering for more details on that. Also, see the help patch `Dynamic Mesh` for an example of setting up a mesh and rendering it.

Geometry

Rhino.3dm

Library to access Rhino *.3dm Files.
NuGet: VL.Rhino.3dm

See also

Relevant libraries that have not yet been tailored for VL. Refer to Using .NET libraries for learning how to explore them.

- geometry3Sharp
- GShark

Text rendering

2d Graphics

For VL.Skia you have the following options:

- Skia itself comes with an extensive set of nodes for high-quality simple text rendering
- Try VL.RichtextKit for rendering richttext
- Using VL.CEF.Skia to render html content allows for complex formatted text to be rendered

3d Graphics

For VL.Stride you have the following options:

- For quick, simple Text rendering use Text [Stride.Models] (experimental)
- Use any of the above (2d Graphics) options via a SkiaRenderer or SkiaTexture node in Stride
- Using VL.CEF.Stride to render html content allows for complex formatted text to be rendered
- Use VL.Stride.Text3d for rendering extruded 3d text
- Try VL.BMFont
- Try FontStashSharp (Text rendering library addon for Stride)
- For the best available option go with the Slug library. Requires a separate license from them. If you need help with the implementation, get in touch.

Transparency

Transparency and 3d rendering with depthbuffer is a tricky topic that isn't completely solved for the general case. The only thing you can do is to find the right rendering technique that suits your specific setup.

Materials

The Stride render system has a render stage for transparent objects. This render stage is called directly after the normal opaque stage. In the transparent stage objects that have a transparency feature connected to the material get rendered. These objects get automaticllay sorted from back to front, only read from the depth buffer and blend themselves over the scene with additive or alpha blending.

There are 3 different transparency fetures: **Blend**, **Additive**, or **Cutoff**. See the Stride Material doc for a detailed description of these modes and their parameters. You can find the nodes for these material features in the node browser in the category `[Materials.MiscAttributes.Transparency]`.

Note

If an object gets moved into the transparent render stage by the render system, it will not write into the depth buffer anymore. This means that it will not occlude other objects.

Custom rendering

If you render your own custom shaders, you can control the blending and the interaction with the depth buffer via the nodes `BlendStateDescription` + `BlendStateRenderTargetDescription` and `DepthStencilStateDescription`. The node RenderEntity also offers the possibility to specify the render stage, see Rendering.

There are also preconfigured blend states like `Additive`, `AlphaBlend` and `AlphaBlendPremultiplied` in the category `[BlendStateDescription]`.

For further reading on this topic and the related problem, you can have a look at the vvvv beta documentation.

Shaders

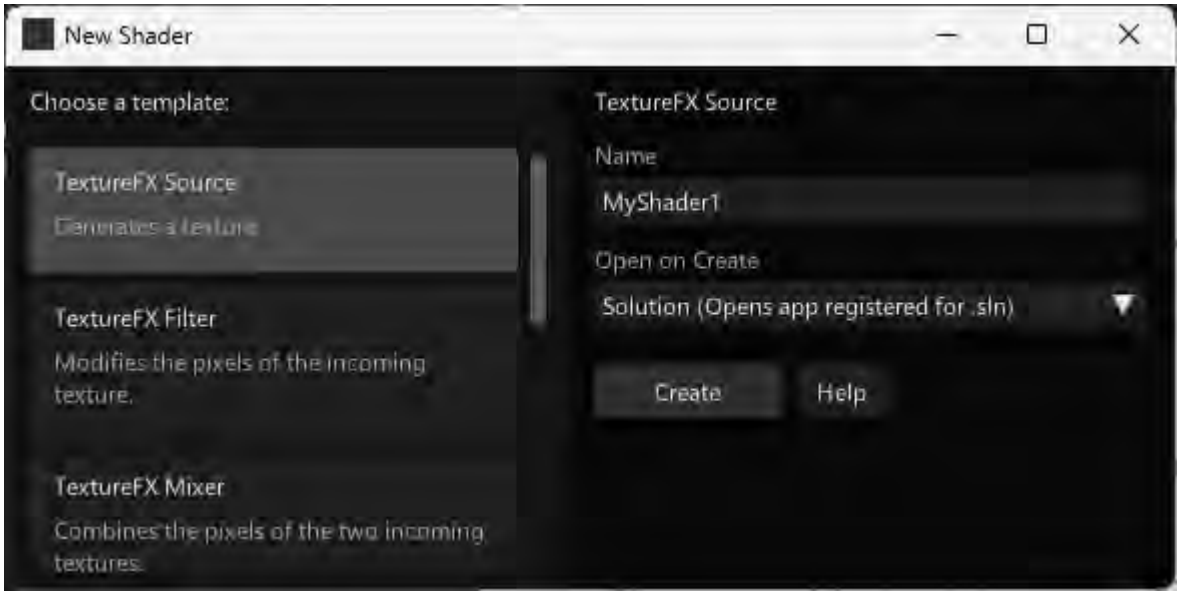
Shaders are written in SDSL, an advanced high-level shader language that supports OOP concepts and multiple inheritance. This allows to write short and nice looking shader code.

Here is a step by step guide to get you started:

Prepare an editor

vvvv does not come with a build-in shader editor. See Editing Shaders for different options.

Start from a Template



Use the built-in Sharder Wizard (as of version 5.0)

- `Quad` -> `New` -> `Shader File`
- Choose one of the templates
- Specify a name for the new shader
- In the `Open on Create` pulldown you can choose:
  - Solution: This is the best option, assuming you have Visual Studio installed with the Stride Extension as explained in Editing Shaders
  - Open the .sdsl file: If you don't have Visual Studio installed, you can also simply edit the .sdsl files with any text editor
  - Open Folder: In case you don't want to edit the file at this point, you can also just see where it is located by having the explorer opened, pointing to it
- Press `Create`
  - This will create the new shader file on disk, reference the VL.Stride package with your current document (if it isn't already) and open the shader

Create the shader node

Open the NodeBrowser and find the shader by the name you gave it.

Now anytime you save a change in your shader file, the node will be updated accordingly.

Further details

Scope

Any .vl document that has VL.Stride set as a dependency will pick up shader files that are placed next to it in special folder called "shaders". Multiple .vl documents can share the same shaders folder.

Note

Shader files share a global scope, two files with the same name are not allowed, even if they are referenced by two different .vl documents.

Special Suffixes

If a shader file ends with one of the pre-defined suffixes, the shader will be converted into a VL node.

_ShaderFX

A node that just represents "a piece of code" that can be used to compose larger shaders. This is the most flexible type of node, it can work together with all other shader node types.

_DrawFX

A node that can be used to draw geometry.

_ComputeFX

A node that represents a compute shader to work with arbitrary data on the GPU.

_TextureFX

Specialized nodes to process textures. See more in the chapter TextureFX.

Core Concepts

Includes and Static Calls

You can use the #include directive just as you would in HLSL. But often you'll not need it because you can call a static function of any shader that is in scope (e.g in the same directory, or both in the /shaders folder next to any .vl document that is loaded). Static functions are functions that don't use any stream variables or class variables, like shader inputs. See also Static Calls in the Stride documentation.

If you have a file `MyUtils.sdsl` like this:

```
shader MyUtils
{
    float4 Invert(float4 col)
    {
        col.rgb = 1 - col.rgb;
        return col;
    }
};
```

You can call its static functions in another file like so:

```
shader MyFx_TextureFX : FilterBase
{
    float4 Filter(float4 tex0col)
    {
        return MyUtils.Invert(tex0col);
    }
};
```

Inheritance

The main purpose of inherance is re-using existing shader code. You can think of it like importing or including the code of another shader into your own shader.

For examples, see Inheritance in the Stride documentation.

To understand the shader inheritance hierachy better, you can use the Stride.ShaderExplorer to get an overview and browse the shaders.

Composition

Compostion allows a shader A to use other shader B like a variable and call functions of it. The main feature is, that the other shaders C or D can be used as composition if they inhertit from the shader class B, that is expected as composition variable in shader A. The fact that you can use different implementations (shaders that inherit from B) as the composition, allows for polymorphism known as **interfaces** in OOP languages.

For examples, see Composition in the Stride documentation.

Streams

SDSL has a convenient way to pass parameters across the different stages of your shader. Simply declare a variable as stream variable and write and read to it in any shader stage. The SDSL compiler will generate the input and output structs for each shader stage.

For examples, see Automatic shader stage input/output in the Stride documentation.

TextureFX

TextureFX is a specification for nodes that do GPU/shader based texture operations. Shaders are written in SDSL which is a superset of HLSL. We distinguish Sources, Mixers, Filters and Utils.

Here is what you need to know to write your own:

Creating a new TextureFX

First, see Editing Shaders on setting up an external shader editing application.

Then follow Start from a Template for the quickest way to write a shader. If instead you want to create a shader file manually, here is what you need to take care of in order for the node factory to pick up a file and interpret it as a TextureFX shader:

- The file must be placed in a subfolder called `shaders` next to your .vl file
- The shader name must be unique among the shaders shipping with vvvv and your own
- The shader name must end in **_TextureFX**
- The filename must be: **[shader-name]_TextureFX.sdsl**

Category and Aspects

By default, every TextureFX node will show up in the `Stride\Textures` category. In order to move a node to a subcategory, use a node attribute.

Aspects, like "Experimental", "Internal", "Obsolete" and "Advanced" can be specified in two different ways:

- Either as part of the shaders filename, in which case you must not forget that the shader name itself must be identical to the filename
- Or as part of the category node attribute.

Base Shaders to inherit from

There are a bunch of shaders you can inherit useful functionality from. Multiple Inheritance is allowed!

- Shipping with Stride: Use the Shader Explorer to browse available shaders to inherit from (requires also Stride to be installed)
- Shipping with VL.Stride: Explore the .sdsl files in: C:\Program Files\vvvv\vvvv_gamma_...\lib\packs\VL.Stride.Runtime...\stride\Assets\Effects

Recommended base shaders

TextureFX

TextureFX derives from ImageEffectShader, SpriteBase, ShaderBase, Texturing and ShaderUtils.

FilterBase

Derives from TextureFX. Allows to you implement the Filter() function, which comes with the color of the input texture as parameter:

```
shader MyFx_TextureFX : FilterBase
{
    float4 Filter(float4 tex0col)
    {
        tex0col.rgb = 1 - tex0col.rgb;
        return tex0col;
    }
};
```

Note

Using the `tex0col` input is not mandatory and you can still add other texture inputs to sample from.

MixerBase

Derives from TextureFX. Allows you to implement the Mix() function, which comes with the colors of the two input textures and a fader parameter:

```
shader Mix_TextureFX : MixerBase
{
    float4 Mix(float4 tex0col, float4 tex1col, float fader)
    {
        return lerp(tex0col, tex1col, fader);
    }
};
```

Note

Using the `tex0col` and `tex1col` inputs is not mandatory and you can still add other texture inputs to sample from.

ShaderUtils

ShaderUtils defines constants like PI and gives access to many commonly used shader snippets.

Include Files

See Includes.

Node Attributes

Attributes allow you to configure your TextureFX node. Here is an example of some attributes applied to a shader:

```
[Category("Filter")]
[Summary("Description for what the filter does")]
[Remarks("Any special notes")]
[Tags("Space-separated list of tags")]
[OutputFormat("R8G8B8A8_UNorm_SRgb")]
shader MyFX_TextureFX : TextureFX
{
    stage override float4 Shading()
    {
        return ColorUtilityTemp.LinearToSRgb(InTex0());
    }
};
```

Attribute	Description
Category	If not specified, the node will show up under `Stride\Textures`. Specifying a category allows you to put the node in a subcategory from there. Use `:` to define a new root category (e.g. `:My.Category`). Also aspects can be added among the category here, like e.g. Filter.Experimental
Summary	A short info that shows up as tooltip on the node in the NodeBrowser and when hovered in a patch.
Remarks	Additional info regarding the node visible on the tooltip in the patch.
Tags	A list of search terms (separated by space, not comma!) the node should be found with, when typed in the NodeBrowser.
OutputFormat	Allows to specify the outputs texture format. Valid Values: PixelFormats. If not specified, defaults to R8G8B8A8_UNorm_SRgb.
WantsMips	Requests mipmaps for a specific texture input. See Mipmaps below.
DontConvertToLinearOnRead	You'll most likely not need this flag! If set, disables the automatic sRGB-to-linear conversion that happens when reading (sampling) from an sRGB input texture. Only relevant if the input texture format has the `_SRgb` suffix and the pipeline is set to linear color space, which is the default. See sRGB and Linear Color Space below.
DontConvertToSRgbOnWrite	You'll most likely not need this flag! If set, this flag disables the automatic linear-to-sRGB conversion that happens when writing the shader result into an sRGB texture. Only relevant if OutputFormat has the `_SRgb` suffix and the pipeline is set to linear color space, both of which is the default. See sRGB and Linear Color Space below.

Source Node Attributes

The following attributes are specifically for use with Source TextureFX:

```
[TextureSource]
shader Foo_TextureFX : TextureFX
```

Attribute	Description
TextureSource	Specifies a shader to behave as a TextureFX Source. Also: Any Texture input pin will keep its name as declared (For Filters and Mixers this is not the case. There the pins are renamed to have concise namings across all nodes)

Pin Attributes

Every pin definition can have the following Attributes:

Attribute	Description
Summary	A short info that shows up as tooltip on the pin
Remarks	Additional info that shows up as tooltip on the pin
Optional	Pins marked as optional don't show up on the node by default. They need to be activiated via the nodes configuration menu.
Color	To have a float4 input show up as a color pin
EnumType	To have an int input show up as an enum. **NOTE**: This also requires you to define the specified enum in C# and have it referenced by the .vl documents you're using the TextureFX in.
Default	Only for Compute inputs to specify their default. For primitive inputs you can simply set the default with the variable definition.

Examples

```
[Color]
[Summary("The color to do this and that")]
float4 MyColor;

[EnumType("VL.Stride.Effects.TextureFX.NoiseType")]
int Type;

[Default(1, 1, 1, 1)]
compose ComputeFloat4 Control;
```

Inputs

Every TextureFX node has exactly one texture output and a couple of inputs by default:

Sources

Name	Type	Optional	Description
Output Format	PixelFormat Enum	x	The format of the output texture, defaults to R8G8B8A8_UNorm_SRgb
Output Size	Int2		The size of the output texture

Name	Type	Optional	Description
Enabled	Boolean		Whether or not the output is updated

To make a TextureFX a "Source", specify the "TextureSource" attribute.

Filter, Mixer and Utils

Name	Type	Optional	Description
Input	Texture		
Sampler	SamplerState	x	Allows to override the default sampler
Control	GPU		Allows to blend between the input and the result of the operation
Output Format	PixelFormat Enum	x	Allows to override the format of the output texture, defaults to `None`, meaning the format of the input texture is used
Output Size	Int2	x	Allows to override the size of the output texture
Output Texture	Texture	x	Allows to render the output in a given texture, rather than using nodes own texture
Apply	Boolean		Whether the effect is applied to the input texuture, or the effect is bypassed and the input is returned unchanged

Multiple passes

At this point there is no support for multiple passes in shader code. That said, you can still create multipass TextureFX by preparing the passes as individual TextureFX and then plugging them together in a patch. For an example, see how the Glow filter is done.

Note that for such cases it makes sense to mark the individual passes with the "Internal" aspect, because they probably are not meant to be used on their own and therefore should not show up in the NodeBrowser.

Mipmaps

Some effects need mipmaps for the input texture. This can be indicated via the `[WantsMips("")]` attribute. It takes a comma separated list of texture variable names that need mipmaps. The TextureFX wrapper will then take care of generating the mipmaps, if the texture doesn't have them already. To save performance, an additional input pin is created that controls whether the mipmaps should be generated in every frame or only when the texture instance changes, the default is `true`.

```
[WantsMips("Texture0, MyTexture, ...")]
shader Foo_TextureFX : TextureFX
```

sRGB and Linear Color Space

By default the rendering pipeline is set to linear color space. This is the correct color space for doing color math, such as blending and lighting. But almost all images are stored in non-linear sRGB color

space because it allows for lower bit depths, hence smaller file sizes. To solve this, graphics APIs have low bit depth pixel formats with the `_SRgb` suffix to indicate that the pixel values are in sRGB color space.

The (linear) graphics pipeline will automatically convert from sRGB to linear when a pixel is sampled from an sRGB texture and it will automatically convert from linear to sRGB when a pixel is written into an sRGB texture set as render target.

However, if you copy shader code that was written for an legacy sRGB/non-linear pipeline (as DX9/DX11 in vvvv beta were), you might want to indicate that the input and output colors are in sRGB space.

To do this, you can use two attributes that declare the read and write intent.

- `[DontConvertToLinearOnRead]`, input should stay as sRGB. This can involve an internal copy of the texture if the resource is not typeless (i.e. is strongly typed).
- `[DontConvertToSRgbOnWrite]`, output is already sRGB.

```
[DontConvertToLinearOnRead] //could involve a copy for each input texture
[DontConvertToSRgbOnWrite] //almost cost free
shader MySRgbFX_TextureFX : FilterBase
{
    float4 Filter(float4 tex0col)
    {
        tex0col.rgb = tex0col.rgb;
        return tex0col;
    }
};
```

These attributes will only do something if the input textures or render target have the `_SRgb` suffix.

Because there can be more than one input texture, it is also possible to specify a comma separated list of variable names of input textures to set the input attribute only for specific ones:

```
[DontConvertToLinearOnRead("Texture0, MyTexture")]
[DontConvertToSRgbOnWrite]
shader MySRgbFX_TextureFX : FilterBase
```

System Values and Shader Semantics

If needed, HLSL shader semantics can be used.

Many of those are already available in more human-readable terms inherited via the ShaderBase.

Render Target Size

A common requirement is the size of the render target, this is provided via the `ViewSize` variable. It describes the size of the current viewport, which is the full size of the render target for TextureFX:

```
float2 targetSize = ViewSize;
```

Time

The current time and the frame time diffrence can be obtained by inheriting from the Global shader and using the `Time` and `TimeStep` variables. The values are automatically set by the runtime.

```
shader MyBlinker_TextureFX : FilterBase, Global
{
    float4 Filter(float4 tex0col)
    {
        var blink = frac(Time) > 0.5;
        tex0col.rgb = tex0col.rgb * blink;
        return tex0col;
    }
};
```

Editing shaders

Shaders are written in SDSL which is a superset of HLSL.

vvvv does not come with a built-in shader editor. Instead you can use any text editor of your choice. Simply associate the file-ending `.sdsl` with it. If you now Rightclick -> Open on a shader node, the code will open in the specified editor. Whenever you save the file, the shader node will be updated.

Syntax Highlighting

For syntax highlighting you have to use an editor that supports HLSL syntax highlighting. Try one of these:

- Visual Studio Code is a light version of Visual Studio. Add the Shader languages support for VS Code extension, assign `*.sdsl` files to HLSL syntax, and you get syntax highlighting and basic code completion.
- You can also use the Sublime editor with the "HLSL Syntax" package.

Error Reporting

For serious shader coding you'll not want to miss error reporting. This requires

- Visual Studio 2022
- The Stride extension for Visual Studio, which comes with the Stride installer
- Stride itself must also be installed for this to work. To see which exact version of Stride is required for your vvvv version, check the "About" dialog in vvvv

Additional recommendations

- Stride Shader Explorer: A tool that lets you explore the built-in shaders and their inheritance hierarchy
- Enable the scroll bar code map in Visual Studio
- Productivity Power Tools for highlighting the selected word

This setup will give you the best shader editing experience, including syntax highlighting, code completion, code navigation and error reporting.

Projection Mapping

Projection mapping is a big term that can mean many things. Depending on the goal, different technical solutions are needed. Here is an overview of the growing list of options vvvv provides:

BadMapper

NuGet: VL.BadMapper

This packet is a work-in-progress collection of tools for convenient manual blending (softedge) and warping (homography, bezier) of content.

Also includes a simple 6-point camera based ProjectorCalibration tool.

Mapper

Projection mapping tool.
NuGet: VL.Mapper

Domemaster

Download: WIP
Rendering Scenes in Domemaster format for use in fulldome projections.

MPCDI

Need support for the VESA MPCDI standard? Please get in touch, we can provide this on demand.

ScalableDisplay

NuGet: VL.ScalableDisplay

Support for the auto projector-alignment technology by Scalable Display Solutions.
Let's you apply scalable mesh files created using the Scalable Display Manager (Trial version).

Scalable Tutorial:

- Calibrating Multiple Clients to Produce One Image

VIOSOWarpBlend

NuGet: VL.VIOSOWarpBlend

Support for the auto projector-alignment technology by VIOSO.
Let's you apply calibrations created using VIOSO6 or domeprojection's ProjectionTools.

VIOSO documentation:

- 2d calibration workflow for projecting flat textures or videos
- 3d calibration workflow for projecting 3d scenes with possibly dynamic spectator eye-point (think: head-tracking)

Virtual Reality (VR)

OpenXR

The default VR backend in VL.Stride shipping with vvvv.

Supported extensions:

- Passthrough
- HandTracking via VL.Stride.OpenXRExtensions

OpenVR

Supported as alternative VR backend via VL.Stride shipping with vvvv.

Vive Trackers

Use Vive Trackers without HMD.
NuGet: VL.IO.OpenVR

Graphics Cards

vvvv is always only using one GPU! This means that you cannot simply add a second GPU to a PC, move a render window to it and assume that this window is then running on that second GPU!

If your system has multiple GPUs, you can decide for each program (including vvvv.exe or any program you exported), which GPU it is running on. This is done by assigning a graphics performance preference for it.

An exception to the single-GPU rule is, when using multiple GPUs in SLI mode.

Animation

2d Particle System

A CPU based particle system library.
NuGet: VL.Animation.ParticleSystem

TrackObject

Follows and identifies objects over time and assigns them a stable and unique ID.
NuGet: VL.TrackObjects

LoopTool

A collection of useful nodes designed to simplify the creation of repeating loop animations.
NuGet: VL.LoopTool

Timeliners

Kairos

A value animation system consisting of a vvvv nodeset and custom UI editors.
NuGet: VL.Kairos

Tilda

A sequencing timeline that contains clips of various types.
GitHub: Tilda

State Machine

AutomataUI

A finite Statemachine Editor.
NuGet: VL.AutomataUI

See also

- Chataigne
- ossia score
- IANNIX
- Duration
- VEZÉR
- TWO

Audio

Playback, recording, analysis, synthesis

For audio playback, recording, analysis and sound synthesis vvvv ships with the VL.Audio package. For more advanced features like beat tracking and pitch analysis, install the optional VL.Audio.GPL package, which as the name implies, stands under the GPL license. For encoding/decoding LTC timecode, see VL.Audio.LTC.

For an alternative player, see also:
NuGet: VL.GameAudioPlayer

Audio drivers

VL.Audio supports both WASAPI and ASIO drivers. By default the engine will try to use the system WASAPI drivers which should work out of the box with default settings.

To choose a different driver or different settings, use either the DriverSettings node or the Configuration GUI, which is available via `Quad -> Extensions -> VL.Audio.Configuration` or `Alt` `C` as soon as VL.Audio is referenced in a document.

In case you want to use an ASIO driver with your soundcard that doesn't come with dedicated ASIO drivers, here are some options:

- FlexASIO and FlexASIO GUI
- ASIO4All
- FL Studio ASIO as included with the free download of FL Studio

Useful tools

- Dante Via Versatile Audio Networking
- Virtual Audio Cable

MIDI

Midi nodes are shipping with vvvv via the VL.IO.Midi package.

Additional utilities: VL.MiDi.Music.Utils

Useful tools:

- MIDI Monitor
- Virtual Audio Cable
- loopMIDI
- virtualMIDI

FMOD

Wrapper for the FMOD Studio API allowing you to emit FMOD events and control their parameters.
NuGet: VL.FMODStudio

Augmented Reality (AR)

Aruco Markers

A set of nodes to detect Aruco markers is included with VL.OpenCV.
NuGet: VL.OpenCV

See also

Relevant libraries that have not yet been tailored for VL. Refer to Using .NET libraries for learning how to explore them.

- EasyAR

Color

ColorBlender

For color matching and palette design. Based on ColorBlender.
NuGet: VL.ColorBlender

ColorLovers

Search for color palettes. Using ColorLovers.
NuGet: VL.ColorLovers

ColorThief

Grab the color palette from an image. Based on ColorThief.
NuGet: VL.ColorThief

UniColor

A wrapper for Unicolour, color conversion, interpolation and comparison.
NuGet: VL.Unicolour

Computer Vision (CV)

Augmenta

Support for the Augmenta people tracking system.
NuGet: VL.Augmenta

MediaPipe

A set of nodes to get results out of MediaPipe models to do:

- Face detection and tracking
- Face landmarks
- Hand tracking and gesture recognition
- Object detection and tracking
- Image classification and segmentation

Using mediapipe-touchdesigner under the hood.
NuGet: VL.MediaPipe

OpenCV

Most of the OpenCV library can be used in vvvv. This includes:

- Image filters
- Camera Calibration
- SolvePnP
- Blob-, Contours-, Features- Circles-, Lines-, Marker-, Object (HAARCascade), QRCode detection
- Object Tracking
- FaceRecognition
- Background substraction

NuGet: VL.OpenCV

DlibDotNet

GitHub: VL.DlibDotNet

See also

Relevant libraries that have not yet been tailored for VL. Refer to Using .NET libraries for learning how to explore them.

- Accord.NET

Databases

Redis

The VL.IO.Redis NuGet is shipping with vvvv.

InfluxDB

NuGet: VL.InfluxDB

See also

Relevant libraries that have not yet been tailored for VL. Refer to Using .NET libraries for learning how to explore them.

- Database Wrapper for SQL Server, MySQL, PostgreSQL and Sqlite

Devices

Category	Content
Depth Cameras	Kinect, RealSense, Astra, LeapMotion, ...
Lights & Lasers	DMX, Art-Net, Ilda, ...
Robots	
Other Devices	

Depth Cameras

Devices that provide depth pointclouds and/or skeleton tracking.

Stereolabs ZED

For using Stereolabs ZED cameras.
NuGet: VL.Devices.ZED

Intel RealSense

For using Intel RealSense cameras.
NuGet: VL.Devices.RealSense

Orbbec Astra

For using Orbbec Astra cameras.
NuGet: VL.Devices.Astra

Microsoft Azure Kinect

For using the Microsoft Azure Kinect. NuGet: VL.Devices.AzureKinect
Body tracking NuGet: VL.Devices.AzureKinect.Body

Microsoft Kinect2

For using the Microsoft Kinect2.
NuGet: VL.Devices.Kinect2
Also: NuGet: VL.Kinect2FuseUtils

Microsoft Kinect

For using the Microsoft Kinect v1/XBOX360.
NuGet: VL.Devices.Kinect

Nuitrack

For using the Nuitrack API with different depth cameras.
NuGet: VL.Devices.Nuitrack

Ultraleap Motion Controller

For using the Ultraleap Motion Controller for handtracking.
NuGet: VL.Devices.LeapOrion

Lights & Lasers

Art-Net

Nodes for sending and receiving Art-Net are shipping with vvvv.
NuGet: VL.IO.ArtNet

DMX

An Enttec DMXUsbPro sender node.
NuGet: VL.Devices.ENTTEC

Laser control

ILDA laser control
NuGet: VL.ILDA

If you need support for the below or any other, please get in touch, we can most certainly offer custom development for enabling your preferred devices.

- Laser Animation Sollinger
- ShowNET

Robots

See also

Relevant libraries that have not yet been tailored for VL. Refer to Using .NET libraries for learning how to explore them.

- KUKA prc
- RoboDK

Other Devices

StreamDeck

A package for using the Elgato StreamDeck button display.
NuGet: VL.Devices.StreamDeck

SpaceMouse

https://3dconnexion.com/de/
NuGet: VL.Devices.SpaceMouse
NuGet 2: VL.Devices.SpaceMouseHID

Eye Tracker

https://en.wikipedia.org/wiki/The_Eye_Tribe NuGet: VL.Devices.TheEyeTribe

WinTab

For using WinTab Tablets (e.g. Wacom)
NuGet: VL.Devices.WinTab

SICK

For using Lidar devices by SICK
NuGet: VL.Devices.SICK

GameController

WIP: VL.GameController

XBOX 360 Controller

For using the XBOX360 Controller
NuGet: VL.IO.Xbox360Controller

PJLink

For using the PJLink protocol
NuGet: VL.PJLink

M8Display

Dirtywave M8 Remote Display
NuGet: VL.M8Display

PTZ

Set of nodes to work with PTZ cameras
NuGet: VL.PTZ

See also

Relevant libraries that have not yet been tailored for VL. Refer to Using .NET libraries for learning how to explore them.

- Nintendo WiiMote
- BITalino

IO

Category	Content
Networking	UDP, TCP, ZMQ, MQTT, WebSocket, OSC, TUIO, RCP, HTTP, ...
Others	SerialPort, LINK, Mouse, Keyboard

Networking

UDP

UDP server and client nodes are shipping with vvvv.

TCP

TCP server and client nodes. For decoding received byte streams, use the Tokenizer nodes.
NuGet: VL.IO.TCP

WebSocket

Websocket server and client nodes. For decoding received byte streams, use the Tokenizer nodes.
http://websocket.org
NuGet: VL.IO.WebSocket

OSC

The VL.IO.OSC NuGet, including OSCServer and OSCClient nodes, is shipping with vvvv.
http://opensoundcontrol.org
HowTo Videos:

- Send OSC Messages
- Receive OSC Messages

Useful tools:

- TouchOSC
- Open Stage Control
- OSC/Pilot
- Jockey
- oscHook
- poseHook
- Sensors2
- SoundCool

OSCQuery

The VL.IO.OSCQuery NuGet, implementing the OSCQueryProposal, is shipping with vvvv.

TUIO

The VL.IO.TUIO nuget, Tuio tracker and client nodes, is shipping with vvvv. http://tuio.org
NuGet: VL.IO.TUIO.HDE for a TUIO Simulator and Monitor Extension

MQTT

https://mqtt.org
NuGet: VL.IO.M2MQTT
Or: VL.IO.MQTTnet

Brokers

- For a broker cloud service try shiftr.io - An IoT Platform for Interconnected Projects
- For a local broker, try Mosquitto

Clients

- Search mobile app stores for "mqtt"

ZMQ

https://zeromq.org
NuGet: VL.IO.NETMQ

RCP

An easy way to remote control parameters in your patches. http://rabbitcontrol.cc
Client: http://client.rabbitcontrol.cc
NuGet: VL.IO.RCP

HTTP Rest

Get and Post nodes are shipping with vvvv.

Also: NuGet: VL.SimpleHTTP

WebServer

NuGet: VL.IO.HTTP.WebServer

NDI

Video Streaming https://ndi.tv
NuGet: VL.IO.NDI

SPNet

Receive data via the Stage Precision SPNet protocol in VL.
NuGet: VL.IO.SPNet

IO

SerialPort, UART, rs232

Nodes for communicating via the serial port are included with vvvv. For decoding received byte streams, use the Tokenizer nodes.

Ableton Link

Synchronizes musical beat, tempo, and phase across multiple applications running on one or more devices.
NuGet: VL.IO.AbletonLink

Global Mouse and Keyboard

Global mouse and keyboard hook.
NuGet: VL.IO.MouseKeyGlobal

Blueiot

Support for the BlueIOT open api. Blueiot offer an RTLS (real time location system) or indoor positional tracking system, based on bluetooth. They claim to achieve an accuracy of 0.1 meters and are compatible with all Bluetooth 4.0 or above tags and mobile phones.
NuGet: VL.BlueIOT

Machine Learning (ML)

Wekinator

A set of nodes to easily interact with The Wekinator
NuGet: VL.Wekinator

OpenAI

Talk to the OpenAI GPT-3 API (requires API key!)
NuGet: VL.OpenAI

VLML

NuGet: VLML

VL.OpenCV

Comes with nodes to run PoseNet and Yolo3. CPU only.
NuGet: VL.OpenCV

ONNX

Run ONNX models inside vvvv.
NuGet: ONNX

VL.Dlib

Allows to run certain machine-learning models.
GitHub: VL.DlibDotNet

See also

Relevant libraries that have not yet been tailored for VL. Refer to Using .NET libraries for learning how to explore them.

- ML.NET
- Accord.NET
- SciSharp

Related tools:

- PerceptiLabs A GUI for TensorFlow
- Teachable Machine

Legacy

Both Runway and Lobe unfortunately no longer exist:

- Runway: Sunset Notification: ML Lab
- Lobe: End of life announcement

Physical Computing

For tasks like controlling motors or reading sensors, use a microcontroller.

Microcontrollers

Arduino/Firmata

Nodes for communicating with Arduinos (or other microcontrollers) running the Firmata Protocol are shipping with vvvv. Reference the VL.IO.Firmata package via the Node Browser and see its helppatches in the Help Browser.

Phidgets

For using Phidgets: Products for USB Sensing and Control
NuGet: VL.Devices.Phidgets

nanoFramework

Want to use plain C# to write code for your microcontrollers? Try the nanoFramework which allows you to write C# with a reduced set of .NET libraries that run on hardware like ESPs and more.

Meadow

Want to use full C#/.NET on a microcontroller? Try Meadow.

Projection Mapping

Projection mapping is a big term that can mean many things. Depending on the goal, different technical solutions are needed. Here is an overview of the growing list of options vvvv provides:

BadMapper

NuGet: VL.BadMapper

This packet is a work-in-progress collection of tools for convenient manual blending (softedge) and warping (homography, bezier) of content.

Also includes a simple 6-point camera based ProjectorCalibration tool.

Mapper

Projection mapping tool.
NuGet: VL.Mapper

Domemaster

Download: WIP
Rendering Scenes in Domemaster format for use in fulldome projections.

MPCDI

Need support for the VESA MPCDI standard? Please get in touch, we can provide this on demand.

ScalableDisplay

NuGet: VL.ScalableDisplay

Support for the auto projector-alignment technology by Scalable Display Solutions.
Let's you apply scalable mesh files created using the Scalable Display Manager (Trial version).

Scalable Tutorial:

- Calibrating Multiple Clients to Produce One Image

VIOSOWarpBlend

NuGet: VL.VIOSOWarpBlend

Support for the auto projector-alignment technology by VIOSO.
Let's you apply calibrations created using VIOSO6 or domeprojection's ProjectionTools.

VIOSO documentation:

- 2d calibration workflow for projecting flat textures or videos
- 3d calibration workflow for projecting 3d scenes with possibly dynamic spectator eye-point (think: head-tracking)

Video

Playback and Video Input

- NuGet: VL.Video - shipping with vvvv, see Playing back video
- NuGet: VL.HapPlayer - Hap and HapR playback
- NuGet: VL.OpenCV
- NuGet: VL.GStreamer (experimental)

Image Sequences

ImagePlayer nodes are shipping with the VL.Skia and VL.Stride packs, see Playing back video.

Cameras

See also Depth Cameras.

Device	Nuget
uEye by IDS Imaging	VL.Devices.uEye
uEye & uEye+ by IDS Imaging	VL.Devices.IDS (using IDS peak)
Industrial Cameras by TheImagingSource	VL.Devices.TheImagingSource
DSLR Cameras	VL.Devices.DigiCamControl

Capture Cards

Device	Nuget
DeckLink by Blackmagic	VL.Devices.DeckLink

Spout

Spout allows you to send realtime video between Windows applications with near-zero latency or overhead. Shipping with vvvv via VL.Stride.

NDI®

Network Device Interface (NDI) is a high performance standard that allows anyone to use real time, ultra low latency video on existing IP video networks.
NuGet: VL.IO.NDI

Misc Libraries

FuzzySearch

Wrapper for the text similarity metric library SimMetricsCore and several nodes covering Fuzzy Search tasks. Included with vvvv: VL.FuzzySearch

SRT

Wrapper for ong.andrew's Universal.Common.Subtitles library.
NuGet: VL.SRT

GraphLayout

GraphLayout Implemetaion.
NuGet: VL.GraphLayout

ScreenRecorder

Records fullscreen or a given area as mp4 movie. Including sound.
NuGet: VL.ScreenRecorder

$Q Super-Quick 2d Gesture Recognizer

The $Q Super-Quick 2d Gesture Recognizer for VL.
NuGet: VL.2D.DollarQ

QRCode/Barcode

QRCode encoder and decoder.
NuGet: VL.QRCode

LerpX

Collection of operators to manipulate data.
NuGet: VL.LerpX

Voronoi and Delaunay

NuGet: VL.2d.Voronoi

2d Simplify

Polyline simplification.
NuGet: VL.2d.Simplify

2DUtils

A collection of tools for 2D purposes in VL..
NuGet: VL.2DUtils

SmallestCircle

Calculate the smallest enclosing circle for a set of 2D points.
NuGet: VL.SmallestCircle

Curve Fitting

Fitting bezier curves to a given set of points.
NuGet: VL.2D.CurveFitting

Noise

Noise Generator in many flavours.
NuGet: VL.FastNoise
NuGet: VL.FastNoiseLite
NuGet: VL.SharpNoise

SharedMemory

A shared memory pipeline for vvvv.
NuGet: VL.SharedMemory

DBSCAN

Implementation of viceroypenguin's DBSCAN .NET library.
NuGet: VL.DBSCAN

Markov Chain

A Markov chain generator.
NuGet: VL.Markov

GlassWindow

Make transparent render windows.
GitHub: VL.GlassWindow

WinForm Utils

To set useful properties on Form windows.
NuGet: VL.WinformsUtils

String Manipulation

Provides some useful string manipulation nodes.
NuGet: VL.StringExtensions

SuperMontior

This library wraps allanrodriguez' MonitorDetailsReader and builds small useful utils on top of it.
NuGet: VL.SuperMonitor

Syslog

Set of nodes to communicate with a Syslog server.
NuGet: VL.Syslog

PsTools

Set of nodes to execute PsTools.
NuGet: VL.PsTools

HardwareMonitor

Read sensor data provided by OpenHardwareMonitor.
NuGet: VL.HardwareMonitor

OpenWeather

Retrieves weather data from OpenWeather's OneCall API.
NuGet VL.OpenWeather

SunCalc

Provides Sun and Moon related calculations in VL.
NuGet VL.SunCalc

FsNotify

Convenience node to retrieve file system events as observables.
NuGet VL.FsNotify

FileTypeAssociation

Query Win API to retrieve information about associated Filetypes.
NuGet VL.FileTypeAssociation

Spotify

A VL wrapper for JohnnyCrazy's SpotifyAPI.Web.
NuGet VL.Spotify

RogueSharp

RogueSharp has functions useful for reasoning about 2D grids where some cells are walkable or transparent. For example pathfinding and field of view. Also includes a dice rolling function.
NuGet: VL.RogueSharp

Kafka

Kafka for VL
NuGet: VL.Kafka

Harmony

Library for working with musical concepts
NuGet: VL.Harmony

Interpolator

Binary search set of nodes, to interpolate any kind of data
NuGet: VL.Interpolator

See also

Relevant libraries that have not yet been tailored for VL. Refer to Using .NET libraries for learning how to explore them.

- For PDF, XLS, Word, Powerpoint, Barcode, QRCode see: FreeSpire Components
- PDF document generation: QuestPDF

On Demand

If you're missing support for a specific device or library, it can most certainly be added on demand.

vvvv's libraries are all open-source and for programmers it is trivial to extend them with custom sets of nodes.

We do offer priority support for your development requirements, please **get in touch** to see if/how we can work things out together.

Here some things you might be looking for:

- Support for specific camera devices
- Support for specific capture cards
- Support for specific protocols
- Lasers by Laser Animation Sollinger
- Support for ShowNET Hardware
- Support for specific OpenXR Extensions
- Planar Image Tracking and more via EasyAR
- VESA MPCDI
- Notch LC video Codec
- VST3 audio plugins
- Python scripting
- NVIDIA Rivermax
- Integrations
  - Disguise (RenderStream)
  - 7th Sense
  - Green Hippo
  - TouchDesigner (TouchEngine)
  - Unreal Engine
  - Ventuz
  - Brainsalt

Extending vvvv

vvvv makes it easy for you to extend it with your own nodes and libraries. Here is how:

Custom nodes

- Using .NET libraries
- Writing Nodes using C#
- Custom Regions

Developing libraries

- Contributing to an existing library
- Creating a new library
- Forwarding .NET libraries
- Node Factories
- Aspects
- Design Guidlines
- Providing Help
- Publishing a NuGet

Extending the vvvv editor

- Creating Editor Extensions

Related Webinars

- Using .NET NuGets
- Turning a .NET library into a VL library

Using .NET Libraries

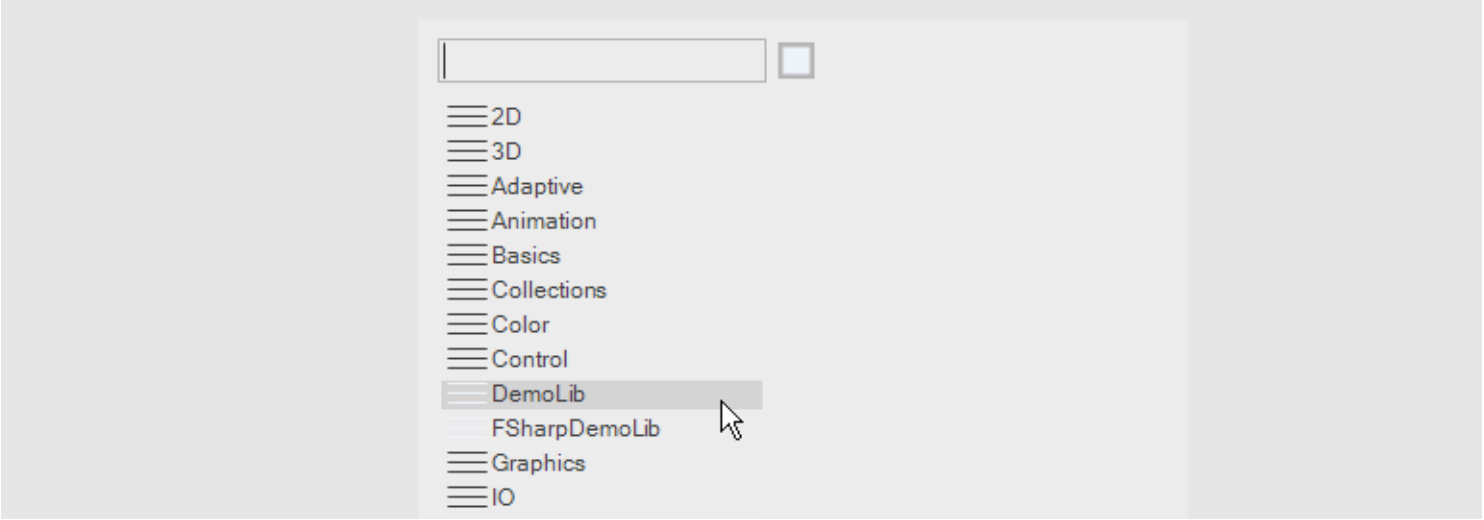
Any static or member method of a public class defined in a .NET .dll can be used as a node in VL.

Referencing Libraries

To get access to the content of a library in a patch you only need to reference it as a dependency in the .vl document.

Accessing nodes

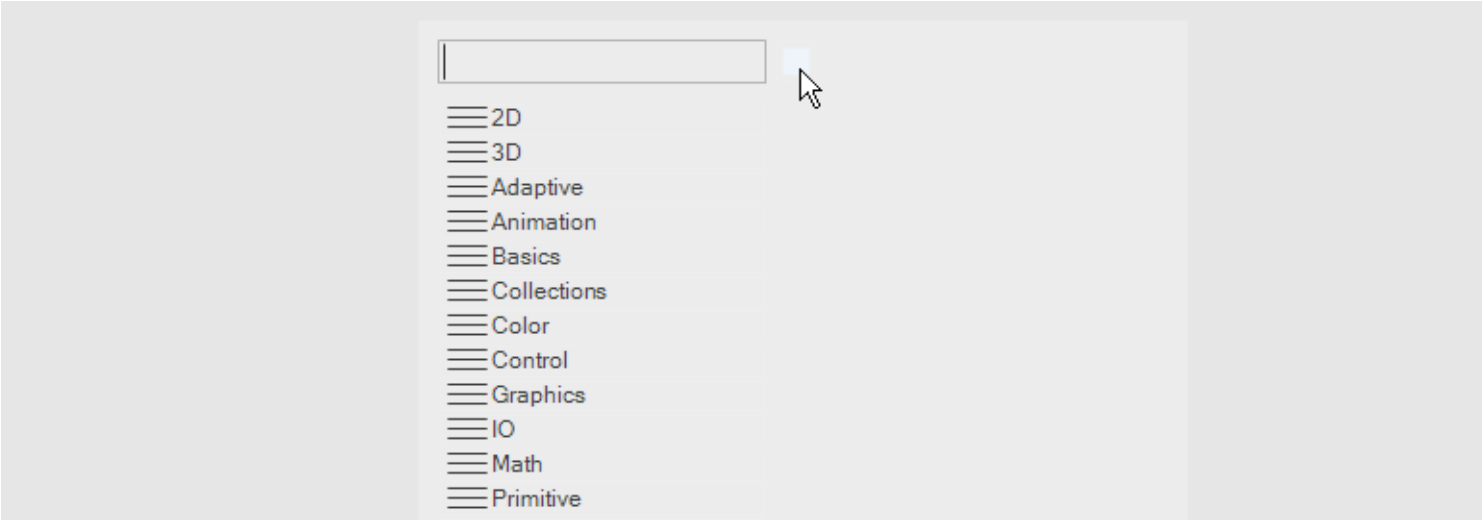
Each referenced .dll shows up as a top-level category in the NodeBrowser. Enter it to explore the libraries' namespaces, types and operations. Hover the individual operations to read their associated xml-documentation. Click any operation to use it as a node in the patch.



NodeBrowser with the namespace 'DemoLib' from the VL.DemoLib.dll showing up as a toplevel category
Like this you can use practically any thirdparty .NET library and start patching with it immediately.

Hiding dependencies from the Nodebrowser

At times it may be overwhelming to see all nodes from all dependencies in the nodebrowser. Use the toggle next to the edit-field to include or hide nodes from all dependencies in the nodebrowser.



Toggle to show/hide nodes of dependencies

For your Consideration

Having access to just about any .NET library directly in VL is indeed quite convenient and powerful. With many simple libraries you'll have instant success. But obviously most libraries you'll find in the wild are not designed to be used in a dataflow context like VL. Patching with those is still possible but will require a better understanding of things than we'd normally want to ask from our users. Therefore we consider this feature of "*using .NET libraries directly*" to be for the more advanced audience.

Below is a list of typical issues you'll encounter with third-party libraries which are often reasons for wrapping them into a more VL friendly form. See Forwarding .NET Libraries for a few features we built into VL to make it easy to create such wrappers.

Incompatible types

Libraries often use their own types for Vectors, Matrices,.. which will not be compatible directly with the corresponding VL types. You'll need to find a way to convert between those. Sometimes it may only be value ranges, like: angles or color components in VL go from 0..1 while other libraries often use different ranges.

Wrappers can hide those conversions from the user.

Mutability

In a dataflow context it is most usable to deal with immutable datatypes. However most types from .NET libraries are mutable.

Events

Many .NET libraries expose events that do not conform to the .NET Core Event Pattern, which means that they cannot be automatically translated to Observables in VL. For those cases you'll have to write a wrapper in C# that converts the event to an observable manually, see Observable.FromEvent.

Error Handling

There are different ways libraries can deal with errors but typically as users we want to have one consistent way.

Building a wrapper around a library allows us to adopt the VL way of handling errors and expose that to the end-user.

Too Low-level

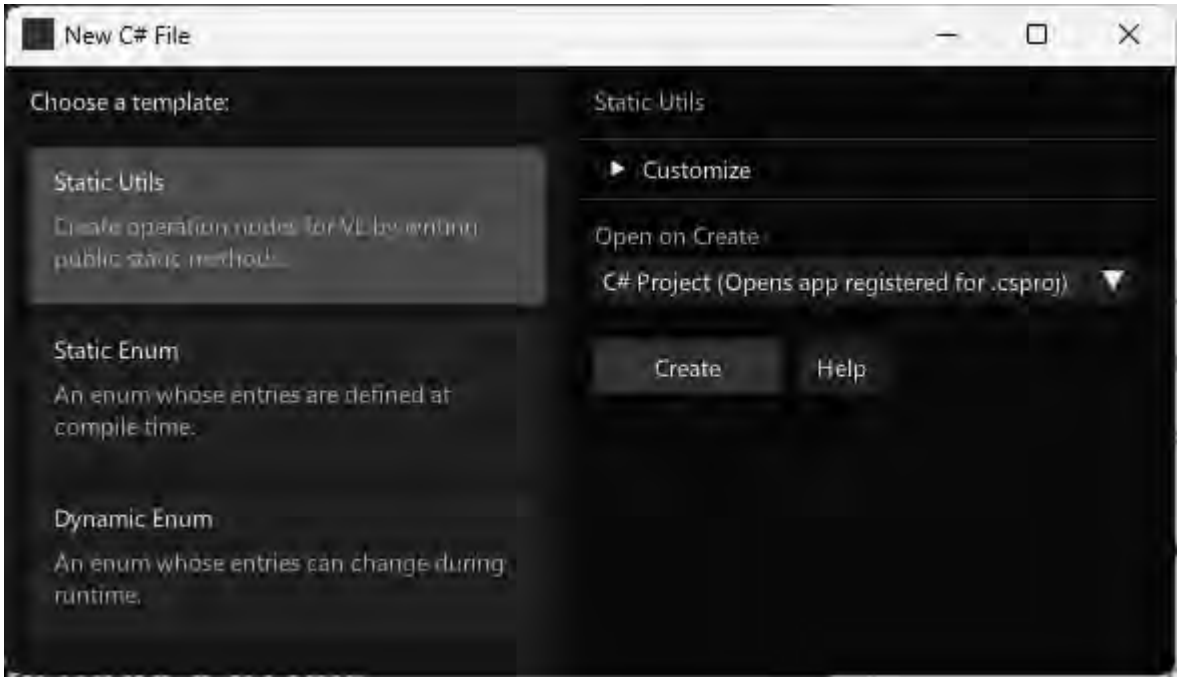
Third-party libraries are often rather low-level in the functionality they provide. In a visual language like VL we prefer to use more high-level nodes that already combine some features of a library to reusable building blocks.

Wrappers can be useful to combine often used low-level functionality into convenient high-level nodes.

Writing nodes using C#

Writing your own nodes for VL using C# requires no VL specific knowledge or preparation. Essentially you're writing plain C# code that VL then turns into nodes. Here is a step by step guide to get you started:

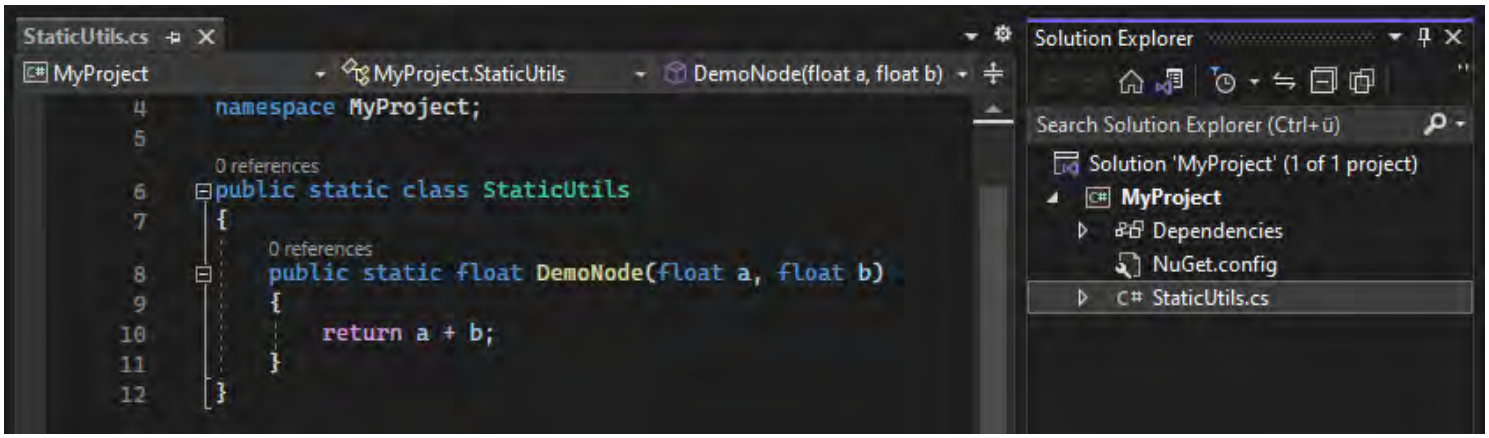
Start from a Template



Use the built-in C# Wizard (as of version 5.0)

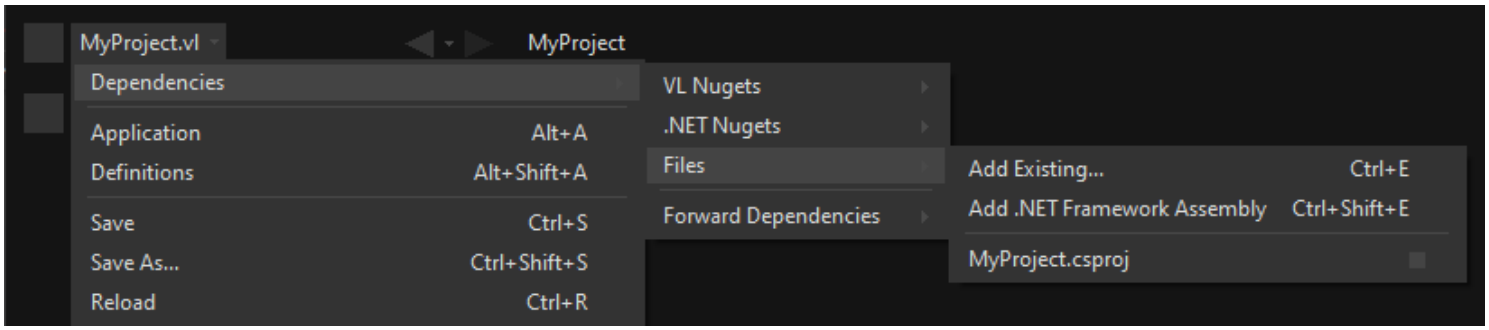
- `Quad` -> `New` -> `C# File`
- Choose one of the templates
  - By default a .csproj file with the name of your current main document will be created. If such a .csproj already exists, it will add the C# file to it. This is assuming the typical scenario will be one .csproj file with possibly many .cs files for your project
  - Optional: To override this default behavior, you can open the `Customize` dropdown:
    - Manually specify a name for the .cs file
    - Choose to which among possibly multiple .csproj files you want the file added to
    - Uncheck `Use Existing` to create a new .csproj file
- In the `Open on Create` pulldown you can choose:
  - Open the .csproj: Ideally you have an IDE like Visual Studio 2022 installed and open the .csproj file
  - Open the .cs file: If you don't have a full IDE installed, you can also simply edit the .cs files with any text editor
  - Open Folder: In case you don't want to edit the file at this point, you can also just see where it is located by having the explorer opened, pointing to it
- Press `Create`

- This will create the file(s) on disk and reference the .csproj file to you current main document



The Static Utils template opened in Visual Studio 2022

The first time a new .csproj file is created, you will see it is automatically referenced to your active document, like so:



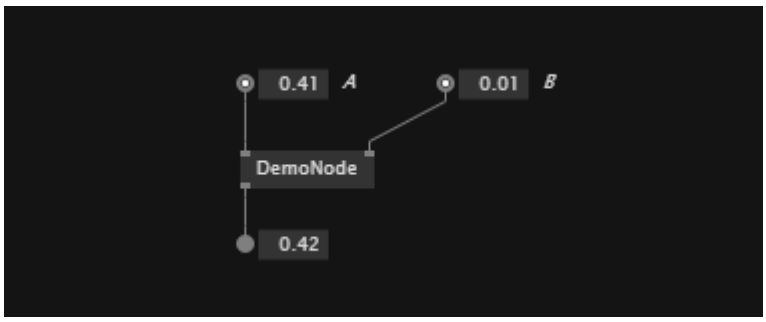
A .csproj file referenced in a .vl document

> NOTE Don't use referenced .csproj files when you're working on a library you're going to ship as a NuGet! It would force the whole package and all packages that depend on it editable, meaning you'd lose the benefit of a read-only package.

Create the node

Open the NodeBrowser and find the methods and classes of your c# file by their names.

The Utils templates' code for example will then translate to the following node in VL:



Resulting node in VL

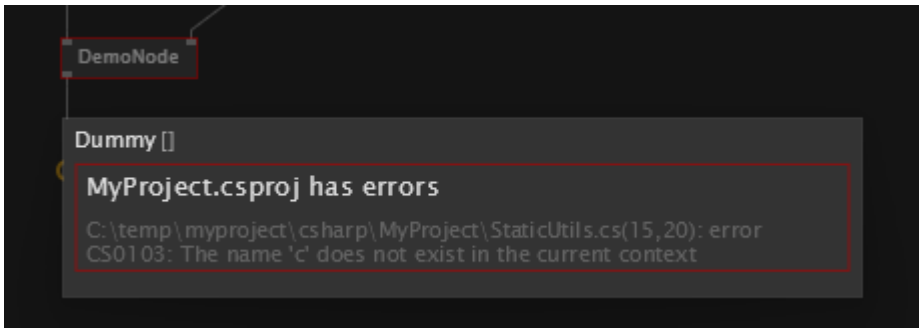
Compilation and Hotswap

Everytime you make a change in a .cs file and save it, code compilation will be triggered and the running code immediately "hotswapped".

Static methods

This works flawlessly as long as you're only working with static menthods since those can be replaced on-the-fly without any side-effects.

If there is an error in your C# code, all nodes stemming from the same project will turn red, with the tooltip indicating an error with the project, pointing you to the exact .cs file and line of the first error found.



Classes

If you're dealing with stateful code, it gets a bit more tricky. Here are two typical scenarios:

Process node

Assuming you want to treat your C# class like a process node in VL, ie one instance per node, not dynamically spawning/killing instances, then attach the `ProcessNode` attribute to it.

This allows vvvv to properly create/dispose instances of your class as needed, whenever you make a change to your C# code.

Dynamic instances

If your C# class is going to be used more like a "Particle", ie you'll be dynamically spawning/killing instances, a Forward as mentioned above will not help you with dispose issues that you may run into. So here is what you have to know:

Every time you save your .cs file, you will loose all running state of instances that are defined in C# code!

As long as your C# code is fully managed, this will not be too big an issue. You'll see pink nodes with Nullpointer exceptions ("Object reference not set an instance of an object") in your patches where those instances were and restarting the patch with F9 will get you back into a running state.

It gets more tricky as soon as your C# code depends on unmanaged code (e.g. WinForms, device libraries,...) which requires manual disposal of resources. vvvv does not know about those resources and can therefore not clean those up properly! In such cases you'll end up with unfreed resources whenever saving your .cs file which often leads to undefined behavior (eg. devices that cannot be accessed anymore). Only a complete restart of vvvv will help to get into a working state in such situations!

Debugging

When editing your code with Visual Studio, you can set break-points in your C# code. Then attach to vvvv.exe and see the break-points hit.

Examples

Every public member of a public class you write in C# will turn into a VL node. You can also attach the `ProcessNode` attribute to a public class to make it available as a node.

Here are some simple examples and a few more details that will help you create your own nodes. Those are also available via: https://github.com/vvvv/VL.DemoLib

For more general considerations also see: Design Guidelines

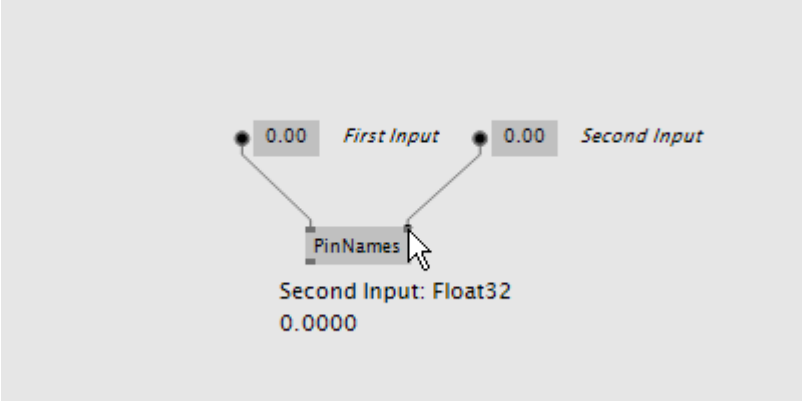
Namespaces

The Namespace you specify in C# will be used as the category in VL. Nested namespaces (using dot syntax) will be translated to nested categories accordingly. The `ImportAsIs` attribute allows to import only a certain namespace, thereby stripping it from the resulting VL category.

Pin Names

For better readability in VL, an operation's arguments are separated at camelCasing. So "firstInput" in C# turns into "First Input" in VL. The default “return” value is called "Output" in VL.

```
public static float PinNames(float firstInput, float secondInput)
{
    return firstInput + secondInput;
}
```

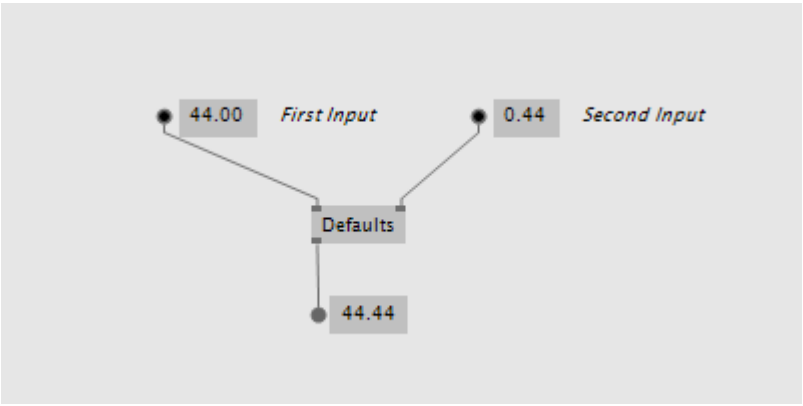


Tooltip shows pin name

Default Values

Simply use the C# notation for defaults to define defaults for inputs in VL.

```
public static float Defaults(float firstInput = 44f, float secondInput = 0.44f)
{
    return firstInput + secondInput;
}
```

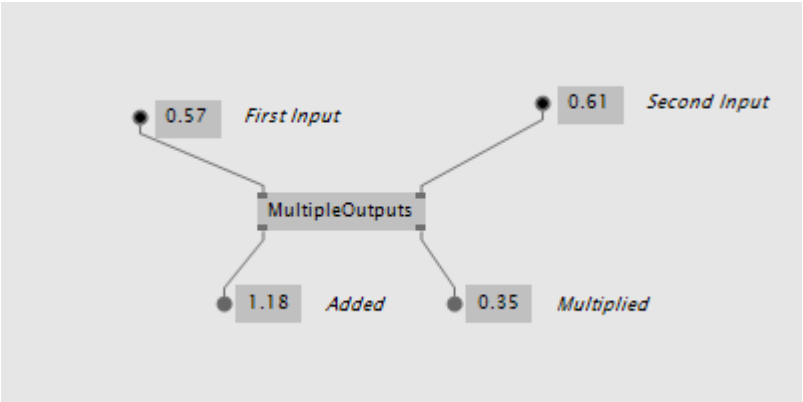


Default on Input

Multiple Outputs

Instead of returning a single value you can also use one or even multiple out parameters that will show up as output pins on the VL node:

```
public static void MultipleOutputs(float firstInput, float secondInput, out float added, out floa
{
    added = firstInput + secondInput;
    multiplied = firstInput * secondInput;
}
```



A node with multiple outputs

Function Overloading

You can write multiple operations with the same name that only differ in the number of input parameters:

```
public static float MyAddition(float input, float input2)
{
    return input + input2;
}

public static float MyAddition(float input, float input2, float input3)
{
    return input + input2 + input3;
}
```

Choosing the respective node in the NodeBrowser will then ask you for a further choice to specify the which version you want to use.

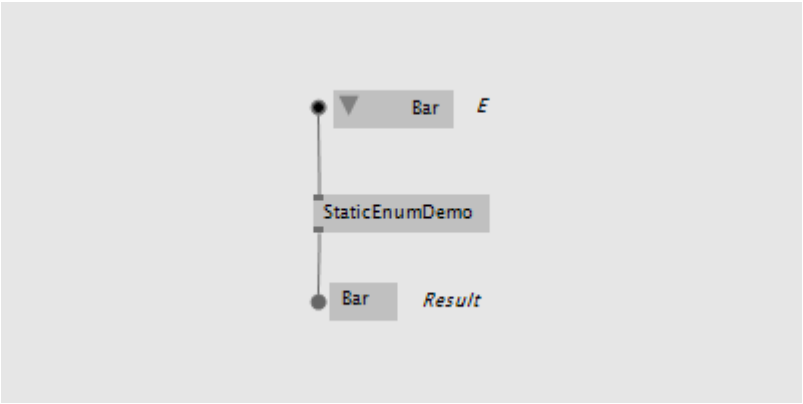
Image:NodeBrowser shows two nodes

Using Enums

You can use custom C# enums as input or output types to operations:

```
public enum DemoEnum { Foo, Bar };

public static string StaticEnumDemo(DemoEnum e)
{
    return e.ToString();
}
```



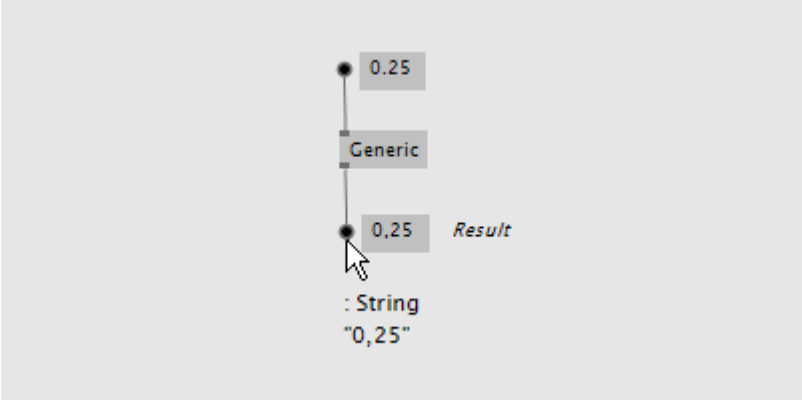
Enum IOBox in VL patch

For an example of a dynamic enum (ie, one whose entries change during runtime), see below.

Using Generics

VL embraces generics, so of course you can write generic nodes easily:

```
public static string Generic<T>(T input)
{
    return input.ToString();
}
```

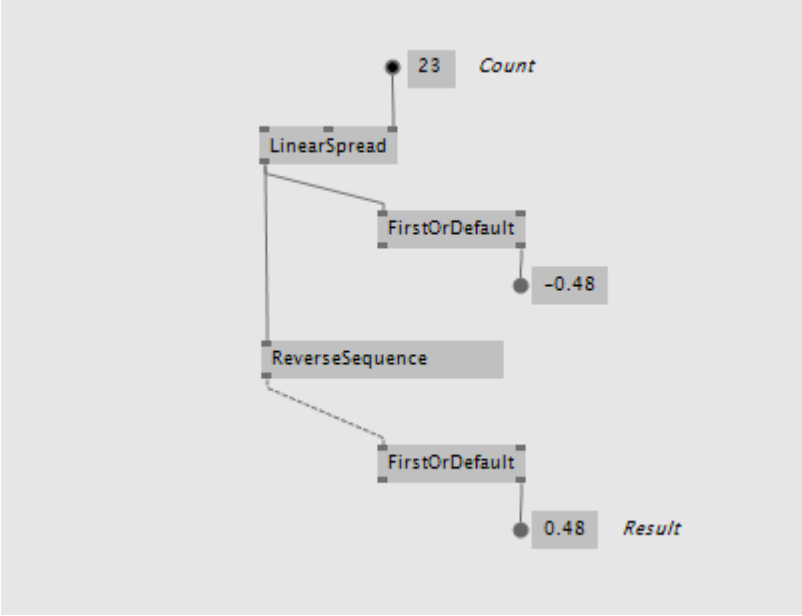


Generic pin out of node

Operating on Spreads

The C# IEnumerable<> appears as Sequence<> in VL:

```
public static IEnumerable<float> ReverseSequence(IEnumerable<float> input)
{
    return input.Reverse();
}
```



Spread node

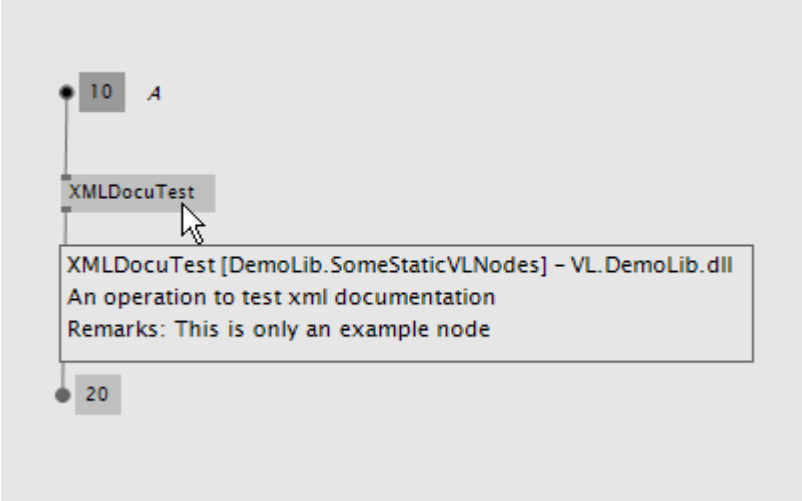
Documentation

Use XML documentation in C# to provide some information about your nodes:

- Summary: A one-liner info about the node

- Remarks: Some additional remarks, like usage instructions, warnings,.. can be multi-line
- Param name: Short info for each Input
- Returns: Short info about the result of the node

```
///<summary>Multiplies input by two</summary>
///<remarks>Some additional remarks</remarks>
///<param name="a">The A Parameter</param>
///<returns>Returns 2 times a</returns>
public static int HTMLDocuTest(int a)
{
    return a*2;
}
```



Documentation shows up in NodeBrowser and Tooltip

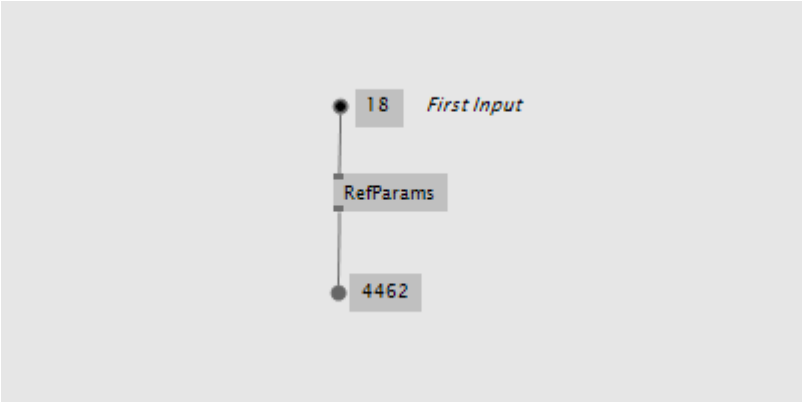
Note

Don't forget to enable "XML Documentation File" in the C# projects properties to make sure the .xml file holding the documentation is generated. This file will then always need to be next to the .dll, therfore always move those two files together!

C# Ref Paramters

You can use C# *ref* parameters, but beware: Assigning the parameter leads to undefined behavior in VL (for now), so never write to but only read from *ref* parameters!

```
public static int RefParams(ref int firstInput)
{
    return firstInput + 4444;
}
```



A node with a _ref_ parameter as an Input

Datatypes

Any datatype that you define as class or struct in C# can be used in VL:

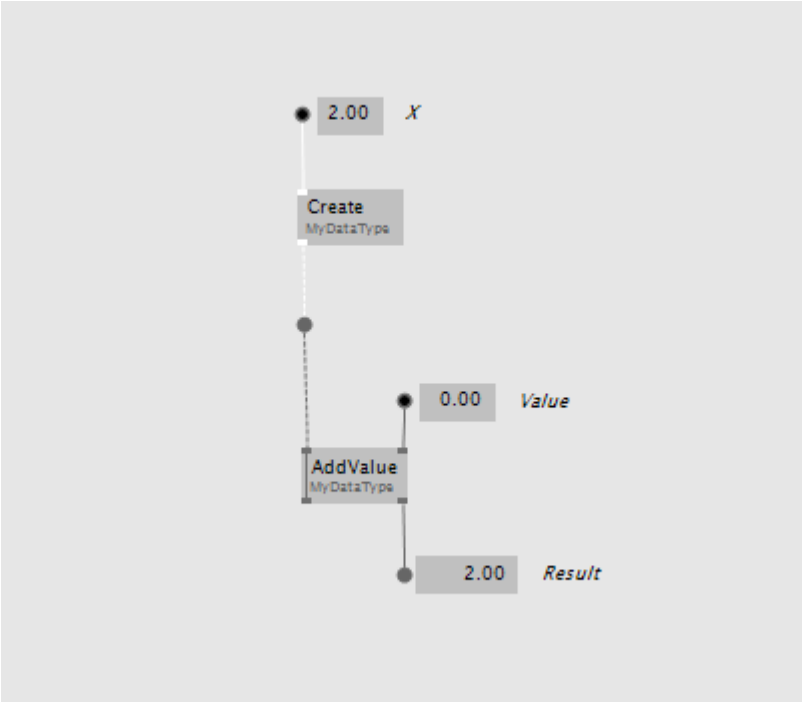
- Any constructor will be available as a `Create` node
- Any public member will be available as a node in VL
  - A property will lead up to two nodes, one for the getter and one for the setter.
  - An event will be translated to a node of the same name returning an `IObservable<EventPattern<>>`. See below for details.

```
public class MyDataType
{
    private float FX;

    public MyDataType(float x)
    {
        FX = x;
    }

    public float AddValue(float value)
    {
        var lastFX = FX;
        FX += value;

        return FX;
    }
}
```



Corresponding nodes

Process nodes

Any class can be turned into a process node by attaching the `ProcessNode` attribute to it. By default all its public members will be used as its fragments. The attribute provides various ways to tweak this behavior.

Note

The attribute only works if the assembly has the `[assembly:ImportAsIs]` attribute set. Newly created C# projects will have this attribute set, for existing ones you'll have to add it on your own.

```
[ProcessNode]
public class Counter
{
    public MyProcessNode(int initialValue)
    {
        Value = initialValue;
    }

    public void Increment() => _counter++;

    public void Decrement() => _counter--;

    public int Value { get; set; }
}
```

Events/Observables

VL translates .net events that conform to the .NET Core Event Pattern to Observables automatically. So you can simply use events in your code and then access them in VL via the Observable pattern.

Here is an example of C# events without and with event arguments:

```
public class MyDataType
{
    public event EventHandler OnValueChanged;
    public event EventHandler<MyGenericEventArgs<float>> OnValueExceeded;
    ...
}

public class MyGenericEventArgs<T> : EventArgs
{
    public readonly T Value;

    public MyGenericEventArgs(T value)
    {
        Value = value;
    }
}
```

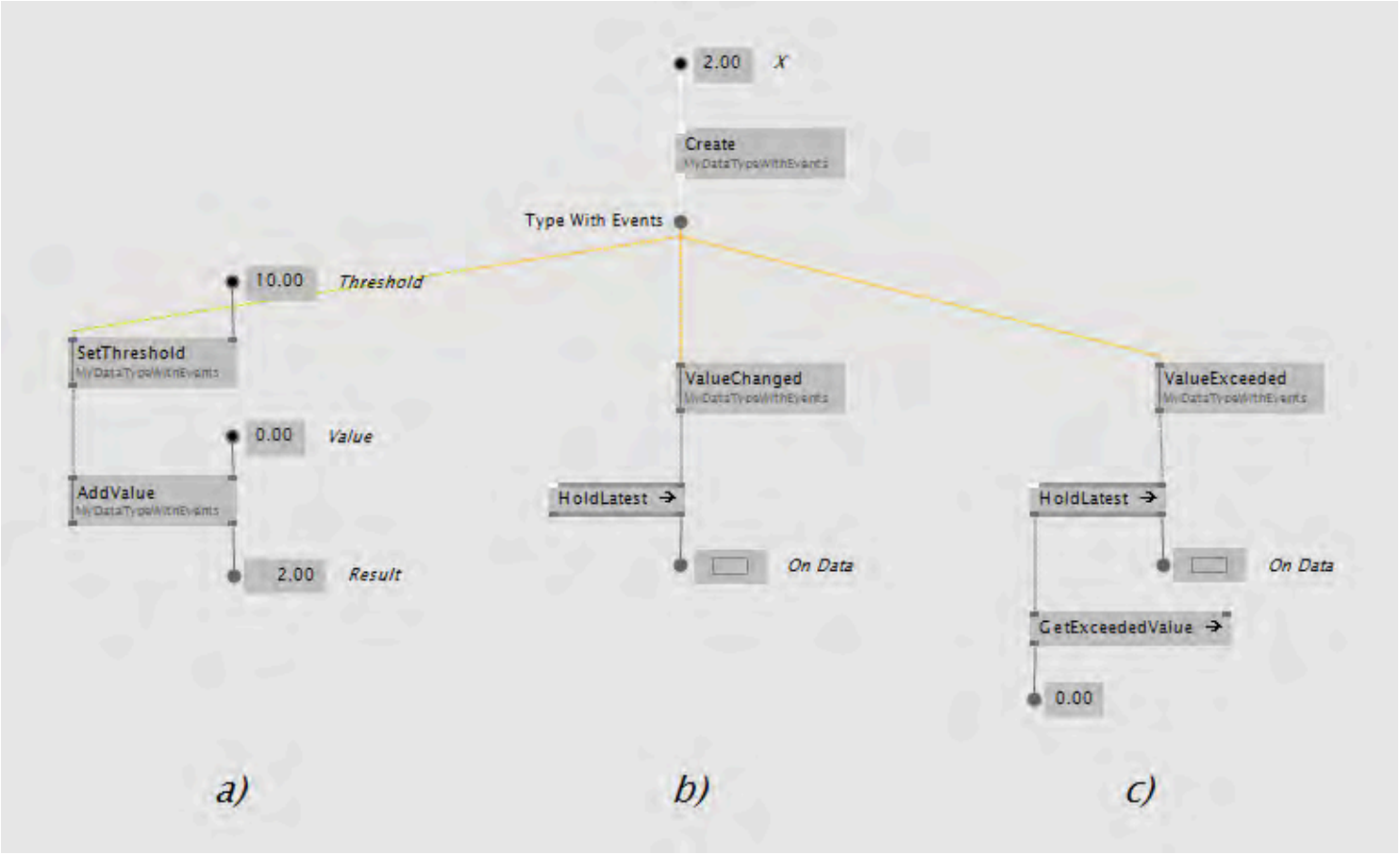
In your code those could be called like this:

```
public float AddValue(float value)
{
    if (value != 0)
    {
        FX += value;
        OnValueChanged?.Invoke(this, EventArgs.Empty);
    }

    if (FX > FThreshold)
        OnValueExceeded?.Invoke(this, new MyGenericEventArgs<float>(FX));

    return FX;
}
```

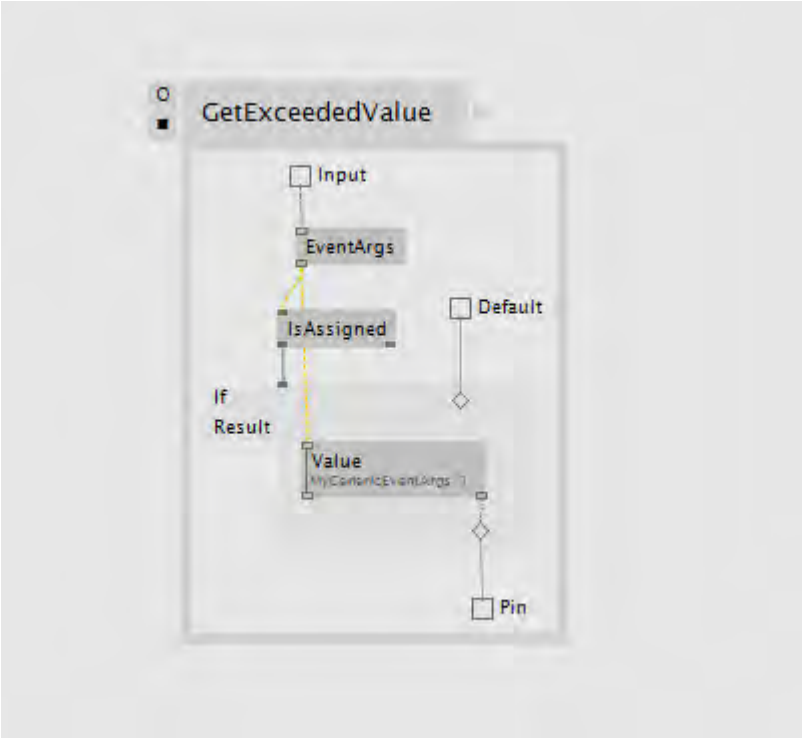
In VL those events are available as nodes of the same name that return an `Observable<EventPattern<>>`:



How this looks in VL: a) Member Operations, b) ValueChanged event without any arguments, c) ValueExceeded event with an argument

- In case your event does not have any arguments *(section 'b' on the image above)*, but simply sends a bang when something happens, use the `On Data` output of the HoldLatest [Reactive] node to be informed of the event.

- If your event does have arguments *(section 'c' on the image above)* you'll receive an `Observable<EventPattern<MyGenericEventArgs<>>>` which you'll have to unpack using the EventArgs [Reactive.EventPattern] node, which is available via the VL.DevLib package. The node then gives you access to the Sender and Value of the EventArgs.



Unpacking using EventArgs

For general information on workig with Observables see the chapter about Reactive Programming.

Dynamic Enums

Dynamic enums are useful in cases where you want to offer users a list of items to choose from, where the entries of that list may change during runtime. A typical example are nodes that give access to hardware devices that can be plugged in and removed anytime.

Consider a normal enum in C#:

```
enum MyEnum = { Foo, Bar }
```

Here `MyEnum` is what we call the type and `{ Foo, Bar }` makes its definition.

And the way we want to use such an enum in our code is to have it as the type of an input parameter to one of our operations, like this:

```
public static string EnumDemo(MyEnum e)
{
    return e.ToString();
}
```

Implementing dynamic Enums for VL

Now in order to create a dynamic enum for VL we also need those two elements, the type and the definition. Both need to be implemented as classes in C#:

- The type needs to implement `IDynamicEnum`

- The definition needs to implement `IDynamicEnumDefinition`

both of which come with the VL.Core nuget.

To make their use easier there are also baseclass implementations available:

- `VL.Lib.Collections.DynamicEnumBase<T, U>`
- `VL.Lib.Collections.DynamicEnumDefinitionBase<U>`
- `VL.Lib.Collections.ManualDynamicEnumDefinitionBase<U>`

Note that the definition base classes are Singleton, meaning that its implementation takes care that always only one instance exists of it globally. We want this because it is important that any node that is referring to a specific enum definition always gets exactly the same entries!

Using the above two baseclasses, an implementation of your own dynamic enum could look like this:

1. Create an enum type

First derive from the `DynamicEnumBase` to create your own enum type.

```
[Serializable]
public class MyEnum: DynamicEnumBase<MyEnum, MyEnumDefinition>
{
    public MyEnum(string value) : base(value)
    {
    }

    [CreateDefault]
    public static MyEnum CreateDefault()
    {
        //use method of base class if nothing special required
        return CreateDefaultBase();
    }
}
```

The code above most likely doesn't need many changes for your own implementation except:

- Give it a proper name instead of "MyEnum", something like e.g. "MidiInputDevice". Note the singular in the naming: This type represents one entry in the enumeration.
- Note the second type parameter `MyEnumDefinition` which connects your enum to its definition and should similarly be called "MidiInputDeviceDefinition"

2. Provide available entries

Derive from `DynamicEnumDefinitionBase` to implement the class that provides the available entries of your enum to the system. Here you only have to override two functions: One that can return a list of

current enum-entries as strings and another one that tells the system when your enum-entries have changed.

```
public class MyEnumDefinition : DynamicEnumDefinitionBase<MyEnumDefinition>
{
    //return the current enum entries
    protected override IReadOnlyDictionary<string, object> GetEntries()
    {
    }

    //inform the system that the enum has changed
    protected override IObservable<object> GetEntriesChangedObservable()
    {
    }

    //optionally disable alphabetic sorting
    protected override bool AutoSortAlphabetically => false; //true is the default
}
```

Implementations here will vary depending on your usecase. A simple example could look like this:

```
public class ComPortDefinition : DynamicEnumDefinitionBase<ComPortDefinition>
{
    protected override IObservable<object> GetEntriesChangedObservable()
    {
        return HardwareChangedEvents.HardwareChanged;
    }

    protected override IReadOnlyDictionary<string, object> GetEntries()
    {
        Dictionary<string, object> portNames = new Dictionary<string, object>();

        foreach(var portName in NetSerialPort.GetPortNames()
            .Where(n => n.StartsWith("com", StringComparison.InvariantCultureIgnoreCase)))
        {
            //the return dictionary holds the names of the entries as key with an optional "tag"
            //here the tag is null but you can provide any object that you want to associate wit
            portNames[portName] = null;
        }

        return portNames;
    }
}
```

For using dynamic enums in VL, see: Enumerations.

Custom Regions

Regions

Regions can be descibed as *node-like building blocks that have a hole inside*: They do something specific - this is the part where they are similar to nodes. But they are somehow "unsure about the details", so they let the end-user step in and ask for those details - this is what makes them regions.

In general we can describe regions as nodes with a *callback mechanism*: A way to call back that small patch inside the region, patched by the end-user of the region.

Region flavors

VL offers several of those callback mechanisms for region developers.

- Delegate-based Callbacks
  - short lived (stateless)
    - `Func<>` or `Action<>` based
    - based on a custom delegate type (typically declared in C#)
  - long running, process-like (stateful)
    - based on two delegates (one for creating a state, one for updating the state)
- Regions built with the `CustomRegion` API
  - long running with support for border control points

Delegate-based Regions

Delegate-based Regions regions allow a region designer to feed any data into the inside of the region and request back any other data. All you need to do is to

- `Invoke` a delegate of any type - e.g. `(Vector2, Rectangle) -> Boolean`
  - feed the data into the inside of the region by feeding data to the `Invoke` call
  - use the returning value somehow meaningful

Now what's left is to actually delegate the task of filling in the details to the user of your region - to the application side. You do this by asking for a specific "delegate implementation":

- Create an Input Pin and connect it to the first pin of the Invoke node. VL now knows that the node you are currently designing can be either
  - a node with a delegate Input pin or
  - a region!

By that you basically let the "delegate implementation" of the user / the inside of the users' region flow into your algorithm.

You now can reason about when to call back that patch. For example you could call back the patch several times or just when some condition is true. You have complete freedom regarding when to call beack the users application of your region.

Custom delegate types

These allow you to define Regions that come with nicely named Pins. If you are not afraid of C# then please give them a try!

- They enhance the readability of the Region as you can name your pins
- They allow for several Outputs

For details see: https://github.com/vvvv/VL-Language/issues/5

Stateful - delegate-based

The basic idea here is that the region is built in a way that it allows for process nodes to be placed inside.

Regions of that flavor can

- instanciate the user patch and
- update the patch.

Often enough those regions only manage one instance of the users' patch, but in theory such a region can also manage the whole lifetime of many instances those patches.

Stateful Get your hands dirty

This workflow still needs some more love:

Search for `UserPatch`. You'll find a helper patch that allowed to declare some stateful region. Take this as an example for now. Note that in your specific case you might need to change the delegate type in order to fit your needs. Copy over that UserPatch and refine it on your side.

`CustomRegion` API based

Since 2021.4 VL offers a way to build regions that have `Input Border Controlpoints` (BCP) and `Output BCPs`. And again: you can patch them.

This is a powerful feature as it allows the end user stay in the flow. Getting data into or outof the region suddenly is effortless.

Where Delegate-based Regions can get arbitrarily complex, only asking for a small detail via a delegate, Regions based on this CustomRegion API are typically focussing only on small tweaks.

Here are some regions that showcase the CustomRegion API:

- Comment
- Do
- Try
- ManageProcess

Usage

In order to patch a new region, all you need to do is to

- define a new `Process`
- have an input pin typed `ICustomRegion` on `Update`
- use the configuration menu on the input pin to configure certain aspects of the region.

The workflow up to now is very similar to how you had an input pin of type delegate earlier on, just this time it is of type `ICustomRegion`.

When you now instanciate your new region via the node browser in your help patch, you already will get a region. No questions asked. *If you are unhappy that you didn't get asked for Node or Region by the node browser: Check the configuration menu of that Region input pin*

What you are telling the VL system is: I want to define a region, but actually let me just patch that. Dear VL System, you just need to hand me over the specific usage of the region by the user. Tell me everything about the application/user side, let me reflect over that, and I'll take care of the rest by just patching the logic of the region.

So, now you can use that custom region instance and ask for the specifics and how the user actually used your region.

The `ICustomRegion` type:

- `Input`, `Outputs` are describing the input and output BCPs. You can see this as a reflection-like API that allows you to reason about the static nature of how your region got used.
- `InputValues` allows to inspect the values that flow into the regions' input BCPs.
- `SetOutputValues` allows to finally write into the output BCPs. These are the values that will flow downstream of the region.

Up to now we only focused on the outside of the region. Now let's tackle the inside:

- `CreateRegionPatch` allows you to instanciate one instance of the user patch, retieving a `ICustomRegionPatch`, a type that basically only allows you to update that users' region patch.

You normally only want to manage the lifetime of one instance, but are not limited to that - thus the seperation into the *region* (`ICustomRegion`) and the instanciation of the *inside* of the region (`ICustomRegionPatch`).

If you only want to manage one instance you might want to use the helper node `CustomRegionPatch`, that takes care of

- instanciating one instance in the beginning of the lifetime of the region application
- calling update of the region patch each frame
- disposing the disposable parts of the users' patch (e.g. some of those used process nodes)

Please see `Do [Control]` or `ManageProcess [Primitive]` for examples and more comments.

The `Do [Control]` is escpecially interesting as it is very basic. The regions purpose is a bit thin on the first glance: It only

- takes the incoming values
- feeds them to inside perspective input BCPs
- calls the patch
- takes the values that ended up on the inside of the output BCPs
- and finally makes these values accessible to the outside perspective of the region

So basically it does nothing. It just executes the inside. Why would I use such a region ever? As a user I could have just not used the region, and my nodes would have been executed as well, so what's the point? Well, subtelties only. It helps sometimes to structure your code. It is very very much comparable to a code block in a textual language.

Anyway. You want to create a region that does more? Just patch along!

- Take the input BCP values, do something with them and then feed them to the inside of the region
- Or take whatever the user patched as output BCPs, try to reason about them, do something with them and feed the transformed values to the outside of the region.

Now your imagination is needed...

Configuration options

- **Node Or Region** - whether the region can also be created as a node. Useful when composing regions.
- **Supported Control Points** - choose what kind of control points your region supports: ** `None` - no control points are allowed ** `Border` - rectangle shape, the data shouldn't be modified by the region when crossing the border ** `Accumulator` - diamond shape, the control points come as a pair, the output should be same as the input if the region doesn't execute ** `Splicer` - triangle shape, when crossing the border the input data should get split apart and the ouput data should get put back together. The splitting and joining needs to be handled by the region. However the system will help out with the typing of the inner and outer parts of the control points (if a type constraint is specified).

- **Control Point Type Constraint** - define the type constraint the system will put on each control point. For example if you specify `Spread`, then the user will only be able to connect spreads to the region. For splicers the system will try to align the inner type argument with the inner part of the control point.

User Expectations

When you design your region you might focus on BCPs with a certain data type. Note however: the user might still want some standard behavior for when the data type is different. Consider to implement a fallback mechanism for such a BCP that just channels the untouched data from the outside to inside or the other way around, very much like seen in the `Do [Control]` region.

Current Limitiations

Note that there are still some constraints for your ideas :(

- It's not possible to define multiple control point kinds (like `Accumulator` and `Splicer`)
- Pins inside the region patch are not supported. So you currently always need to BCPs. A workaround for this limitation is to check for a BCP with a certain name or type and treat this differently.

Please let us know of your needs: https://github.com/vvvv/VL-Language/issues/51

Contributing to existing libraries

Collaboration on existing opensource libraries works via git. To contribute to an existing library, you'd make a fork, make your changes locally and finally file a pull-request with the original repository. See Contributing to a Project in the official git book to get a general idea of the workflow.

Note that one of the main steps in this process is to be able to see the difference between the original and your changed document. While this can be done rather easily with textual programming languages, this is one of the main problems in visual programming, at least with VL. As of now there is no simple way to provide a quick comparison between two versions of a document. So keep this in mind when making a pull-request. The recipient may have a hard time figuring out what your changes actually were. So try to do the following:

- Try to apply additions/removals/changes in separate pull-requests
- If your change is only additions, try to keep them in a separate .vl document
- Make sure to clearly describe your changes in words
- Consider adding screenshots of before/after to your pull-request

Source package-repositories

To work on local copies of repositories it is recommended that you create a directory like

```
vl-libs\
```

that you clone all repositories into. Say you want to contribute to the VL.Audio and VL.Devices.Kinect2 packages, your directory structure would look like this:

```
vl-libs\VL.Audio\
vl-libs\VL.Devices.Kinect2\
```

Such a directory holding one or more sub-directories that are the sources of packages for vvvv is called a *package-repository*. To make vvvv aware of a package-repository, specify the path to this package-repository via commandline argument:

```
--package-repositories C:\Users\foo\Documents\repos\vl-libs
```

You can of course also set multiple paths if you want to maintain different package-repositories in separate directories:

```
--package-repositories "C:\Users\foo\Documents\repos\vl-libs;C:\Users\foo\Documents\repos\vl-lib
```

Packages found in a package-repository path will show up among `Dependencies > VL Nugets` like any other nuget you installed. So you can simply reference them in any .vl document and work on them. Note though that some packages may have a C# solution that would need to be built before the package works!

By default source-packages are read-only, meaning you cannot edit them in vvvv. In order to be able to edit individual packages you have to specify them as editable packages via commandline argument, like:

```
--editable-packages VL.Audio;VL.Devices*"
```

Switching between source and binary packages

Say you first installed the VL.Audio package as nuget and later decided to fork it and clone its sources to work on them as a source-package. By putting the sources in your package-repository path and telling vvvv about that path, vvvv will prefer source-packages to binary nugets of the same name! Like this you can conveniently switch between working directly with your versioned sources or binaries of the same name by simply setting (or not) the package-repository path.

Creating a new Library/Package/Nuget

The most consistent way to contribute a set of nodes for vvvv is by creating a library and shipping it in form of a NuGet Package.

Source and binary nugets

NuGet packages are typically installed in binary form. vvvv adds the idea of "source nugets" in that it allows you to directly reference the sources of a package just as if it was already a binary nuget. For vvvv to recognize a directory as a source-nuget it has to be found in a directory specified as a package-repository and follow these conventions:

Innards of a nuget

For vvvv, a source-nuget in a directory is defined by two files:

- <packName><packName>.nuspec
- <packName><packName>.vl

e.g.:

```
\VL.Devices.Leap\VL.Devices.Leap.nuspec
\VL.Devices.Leap\VL.Devices.Leap.vl
```

The .nuspec file describes a nuget in a simple text format as defined by the Nuspec Reference.

The .vl file acts as the central entry point when using the nuget. It defines all actual nodes that you get by using the nuget: everything has to be either patched in there or marked as a 'forwarded dependency'.

> NOTE Make sure non of the .vl files in a package has a referenced .csproj file! It would force the whole package and all packages that depend on it editable, meaning you'd lose the benefit of a read-only package.

In order for nugets to work with vvvv you have to provide the following structure where of course all the directories are optional and only needed when actually used by a specific nuget:

```
\lib              //for managed .dlls
\runtimes         //for native/un-managed .dlls *
\src              //for c# sources
<packName>.nuspec
<packName>.vl
```

* See Architecture Specific Folders

Creating and Publishing a NuGet

Once you're ready to create a NuGet .nupkg from your library, there are different options:

- Use a build-service like GitHub Actions
- nuget.exe commandline
- NuGet Package Explorer UI

Forwarding .NET Libraries

By Using .NET Libraries we have direct access to a vast range of nodes for patching. Many of those libraries though will not be very convenient to use in the dataflow context of VL.

To make those libraries accessible to more casual users we often want to curate exactly what nodes and types of the original library are seen by them. Forwarding allows us to insert a very thin wrapper layer to conveniently provide such curation.

Reasons to Forward

- Selectively forward types and operations of .NET .dlls
- Adjust VL relevant meta information on the types (like mutability and known type structure)
- Choose a relevant category for the nodes and types in VL
- Do simple unit or type conversions (e.g. from angles in radians to angles in cycles)
- Rename pins, operations, types
- Set default values for input pins
- Provide convenience process nodes that wrap some low-level functionality into more high-level nodes
- Design lifetime management of disposable objects

Note

An important aspect when forwarding types from a library is that we don't want to introduce new wrapper-types. Therefore using Forwarding does not introduce new types!

Forwarded types are compatible to the original library in order to allow users to fall back to low-level functionality of the original library and combine that with the use of the higher-level wrapper.

Also, the wrapper can act as a useful layer to shield the end-user of a vl-library from changes in the original library. Instead of confronting a vl-user directly with the (e.g. naming) changes of an original library, forwards allow us to implement ways to not break patches in such cases.

Forwarding Types

In a typical scenario you create *one .vl document* to forward types from one or more .NET .dlls or C# Projects (.csproj). This .vl document is then the only thing a user of your library will have to reference.

Create Type Forward

1. Set a reference to the .NET .dll or .csproj

In a blank .vl document set a reference to the .NET .dll(s) or .csproj files you want to forward types from. See Referencing Files.

2. Prepare a Category

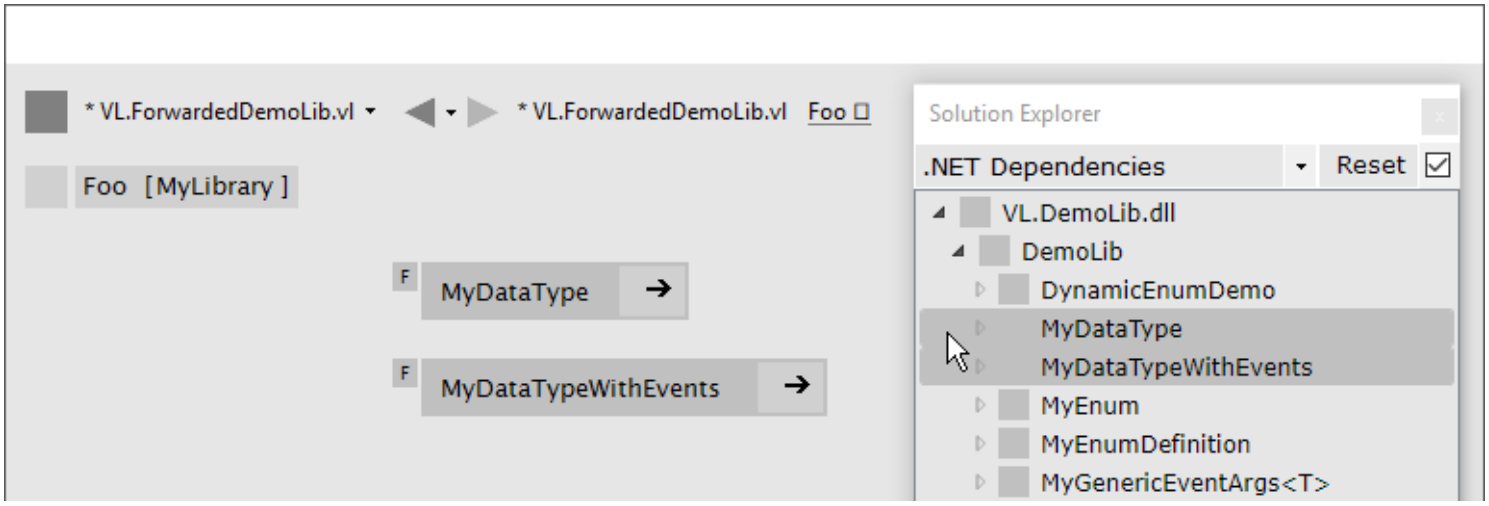
An imported type will show-up in the category you place it in. So, create the categories you need in your documents Definition Patch.

3. Create Type Forward

There are two ways to create a type forward:

3.1 Drag-Drop from the Solution Explorer

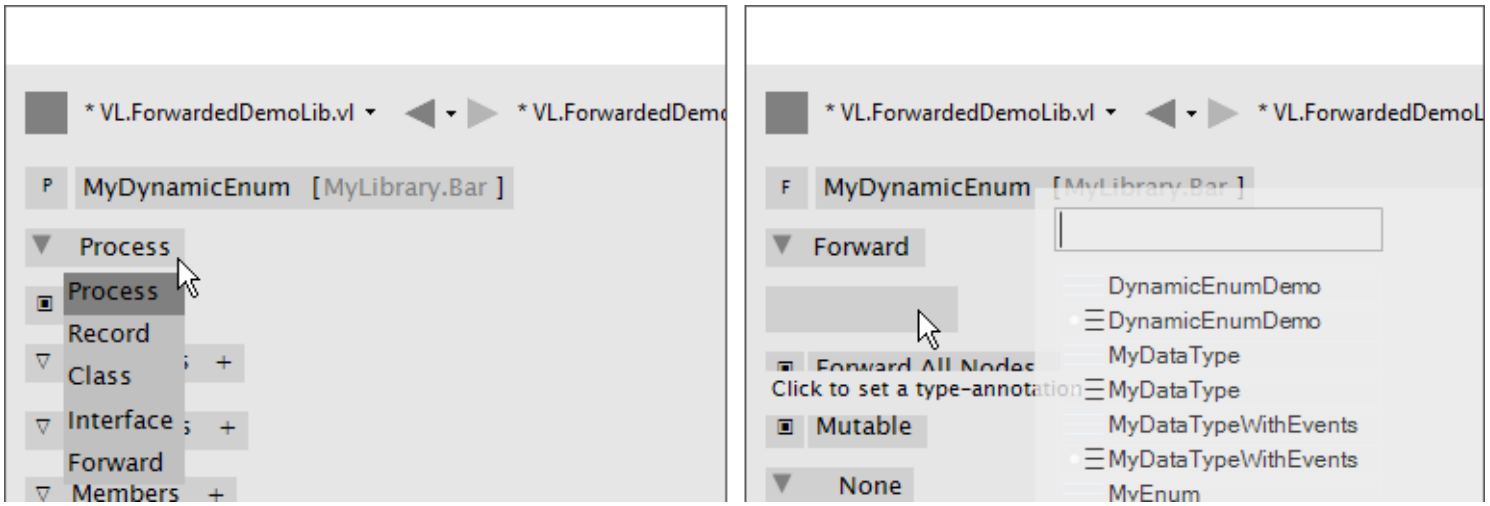
1. Open the group you want to drop the type into
2. Open the Solution Explorer
3. Choose '.NET Dependencies'
4. Find the type you want to forward
5. Drag-Drop the type into the group



Drag-Drop type from Solution Explorer into group

3.2 Manually

1. Open the group you want to drop the type into
2. Create a new *Process* patch
3. Set the `patch type` to 'Forward'
4. Set a `type-annotation` on the patch



Left: Click the `patch type` drop-down and set 'Forward'. Right: Click the `type-annotation` and choose a type from the NodeBrowser

Configure Type Forward

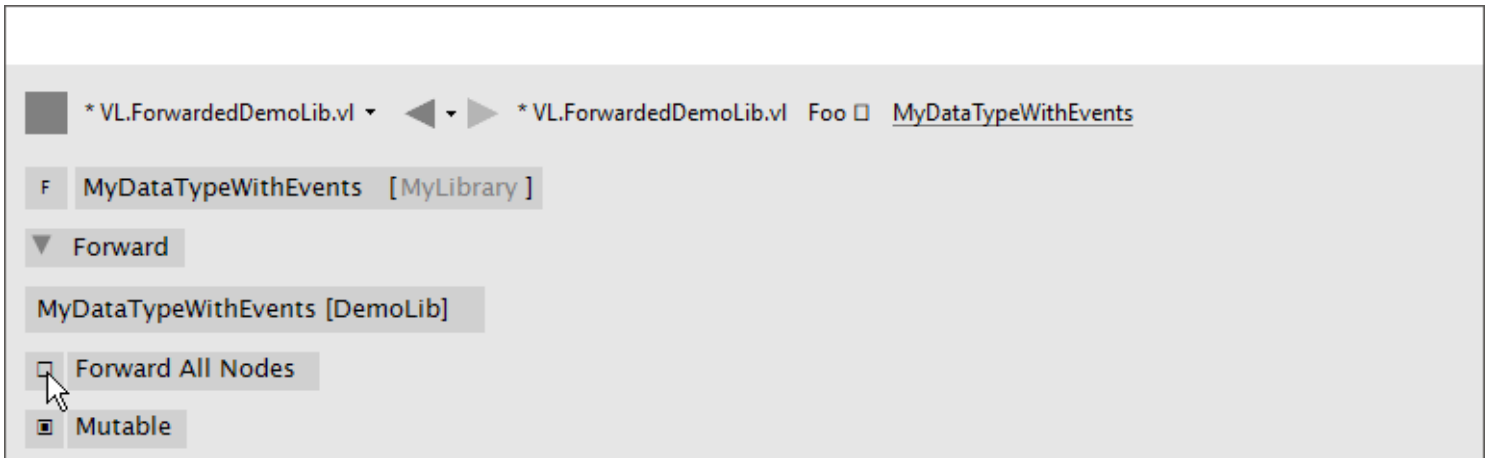
Rename Type

Typically you may want to keep the name of a type from the original library. If you have a good reason to rename it, simply do so.

Renaming a Type

Forward All Nodes

When you create a type forward, every operation of the type is forwarded as a node by default. If you prefer to selectively forward only a subset of a types' operations, uncheck the 'Forward All Nodes' option.



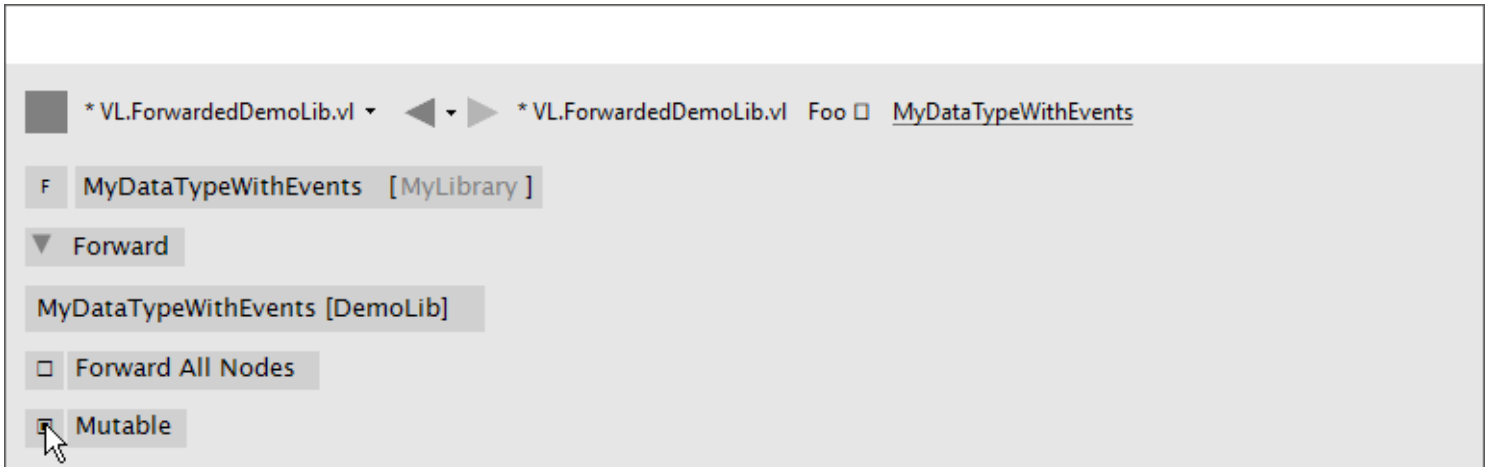
Forward All Nodes

Note

Even if the option is activated you can still adjust the forwarding of individual operations by simply creating an operation forward for them, see below.

Mutability

.NET Libraries don't come with meta information of whether a type is mutable or not. Because of this we need to tell VL manually by setting the mutable flag accordingly.



Mutable checkbox

Since most .NET types are mutable, this flag is activated by default. Here is how to detect whether a .NET type is immutable:

- It only has readonly fields
- Each of its fields is of an immutable type
- Optional: it has `WithFoo(TFoo newValue)` methods to get a new instance (= a new immutable snapshot) of that type, where all fields are set to the values as in this instance, but only the field `Foo` is set to the `newValue`.

In an upcoming version of C# watch out for 'Records'. They should ease the pain for writing immutable types.

Known Type Structure

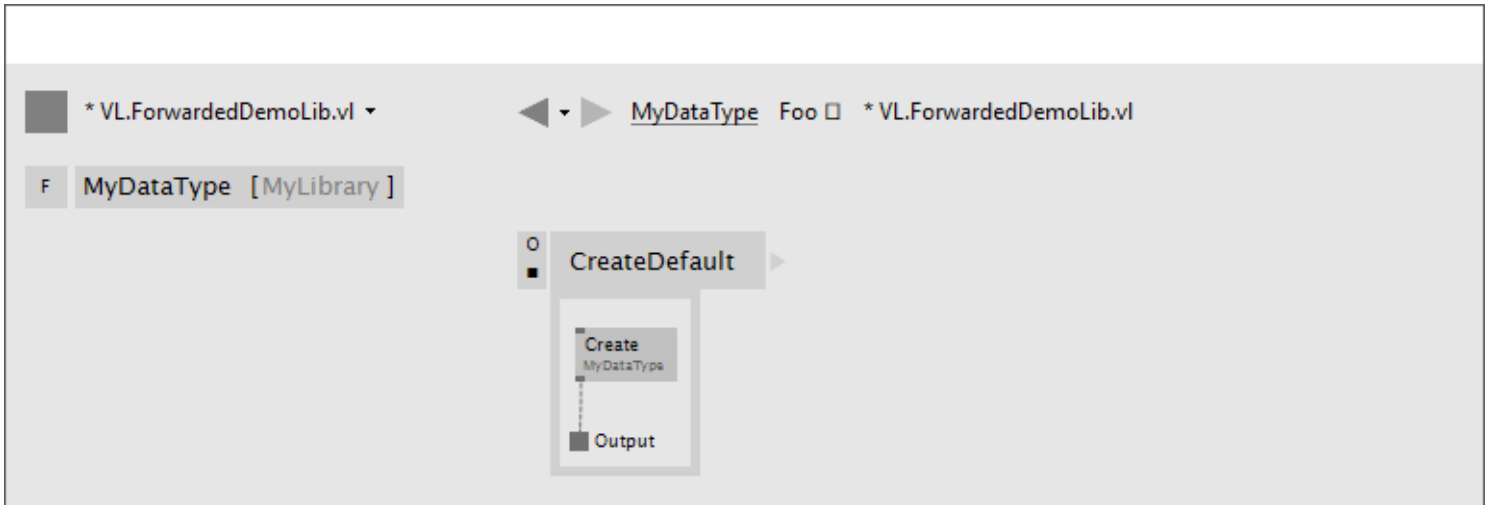
TODO

Image: Known Type Structure

Create Default

Member operation nodes often expect a type on their main input and throw a 'Null Pointer Exception' as long as nothing is connected to it. In order to prevent this, we need to tell VL how it can construct a default instance of a type whenever needed.

To do so, simply create an operation called `CreateDefault` in a type forward patch and implement it so that it returns an instance of the type. Often this requires nothing more than returning the result of a constructor of the type.



Creating a Default for a type

Process Node

Each type forward can also directly expose a process node. This is exactly the same as exposing a process node from an ordinary patch.

- In the forward, navigate to the Patch Explorer and activate the "Process Node" checkbox.
- Then manually forward a constructor of your C# type

This gives you a working process node of your C# type.

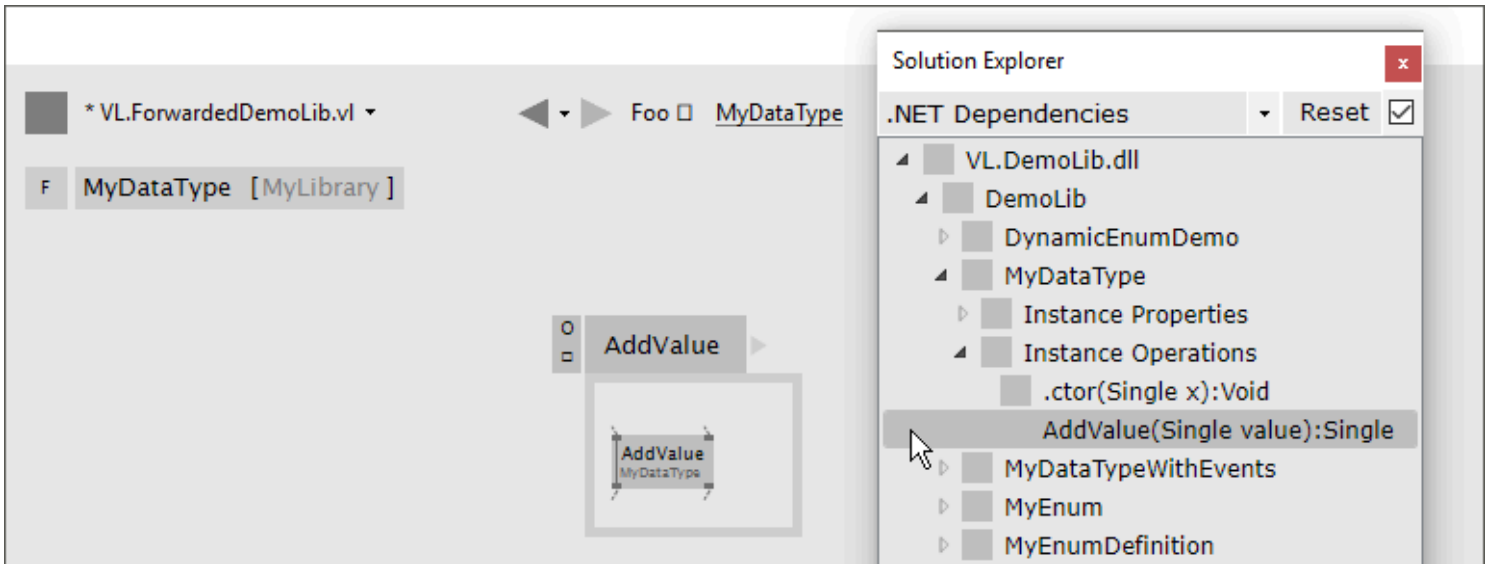
If you want to expose more than one process node from a single type forward, you have to create an extra process definition for each additional process node. Those will not forward the type but simply use the types operations to create the desired process.

Forwarding Operations

As shown above, a type forward can easily forward all of its operations automatically. Even with "Forward All Nodes" activated though, it can make sense to manually forward some operations to tweak their pins.

To create forwards for individual operations:

1. Open the type you want to drop the operation into
2. Open the Solution Explorer
3. Choose '.NET Dependencies'
4. Find the operation you want to import
5. Drag-Drop the operation into the type



Dropping the operation into type

Note

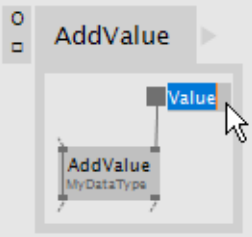
You can also select multiple operations and drop those into the patch at once.

You now have a forwarding operation definition wrapped around the node you want to forward. All pins of the forwarded node are automatically reflected in the signature of the forwarding definition. That also means that any change to the signature of the node (ie. pin added/renamed/removed in its underlying .NET code) will also be automatically reflected in the forwarding definitions signature. If for some reason this behavior is not desired, see "Manually managing the Signature" below.

Still you can apply the following modifications to a forward without manually managing its signature:

Renaming a Pin

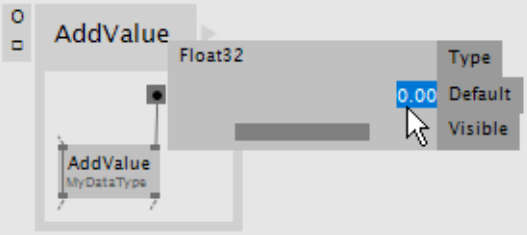
If you have a good reason to change the name of a pin, e.g. in order to have it conform to the VL naming conventions, then do so by manually creating an input or output for a particular pin and renaming it.



Renaming a Pin

Setting a Default

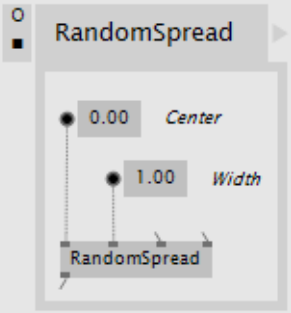
Parameters of operations rarely have meaningful defaults. In order to forward a pin with a proper default, manually create an input for a particular pin and set a default for it.



Setting a default on an input via Middleclick or `Rightclick > Configure`

Hiding a Pin

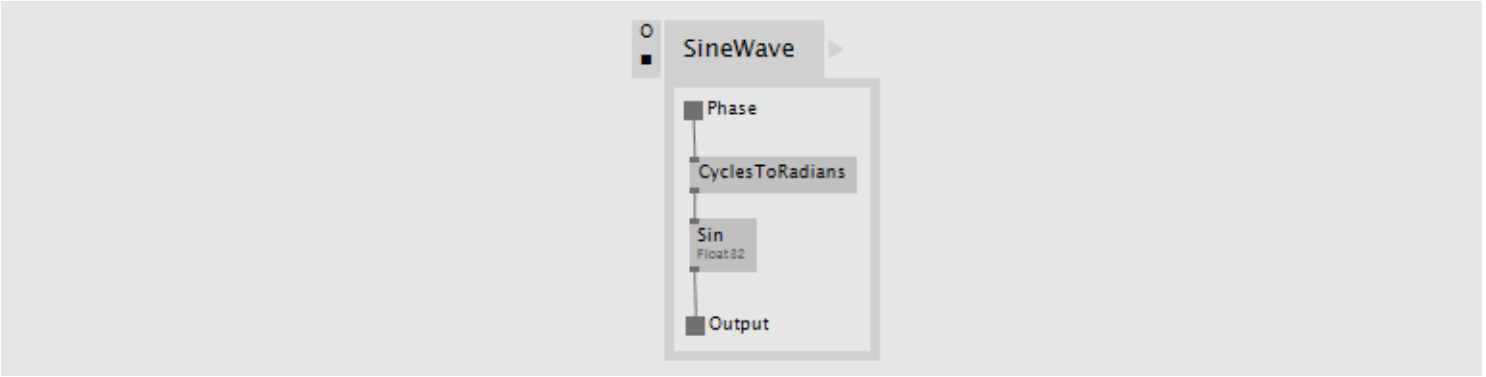
Even if the automatic forwarding of all pins is on, you can override forwarding of individual pins by simply connecting an IOBox to them.



Hiding a Pin

Type or Unit Conversions

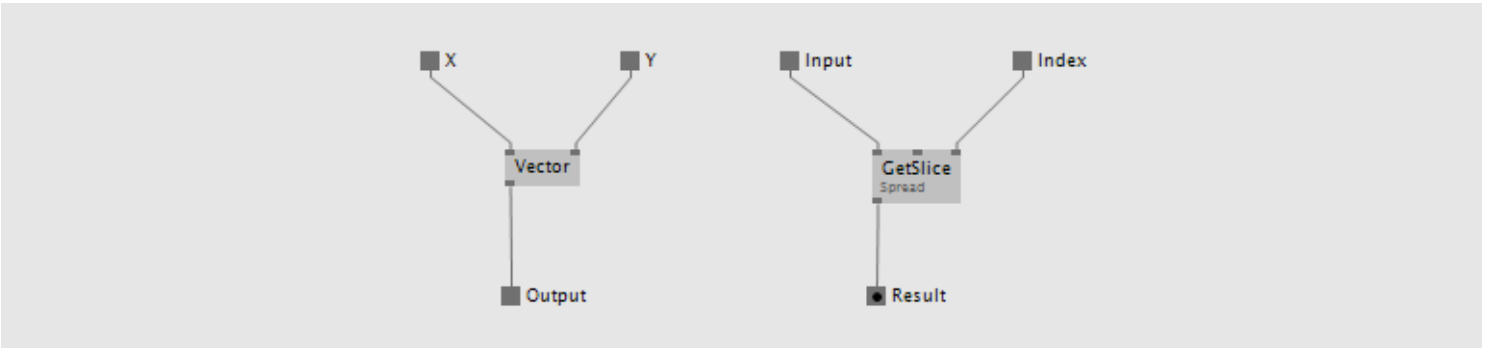
Forwards are a good place to do simple type or unit conversions. Consider an operation that takes angles in radians, but you want to use vl-conform cycles.



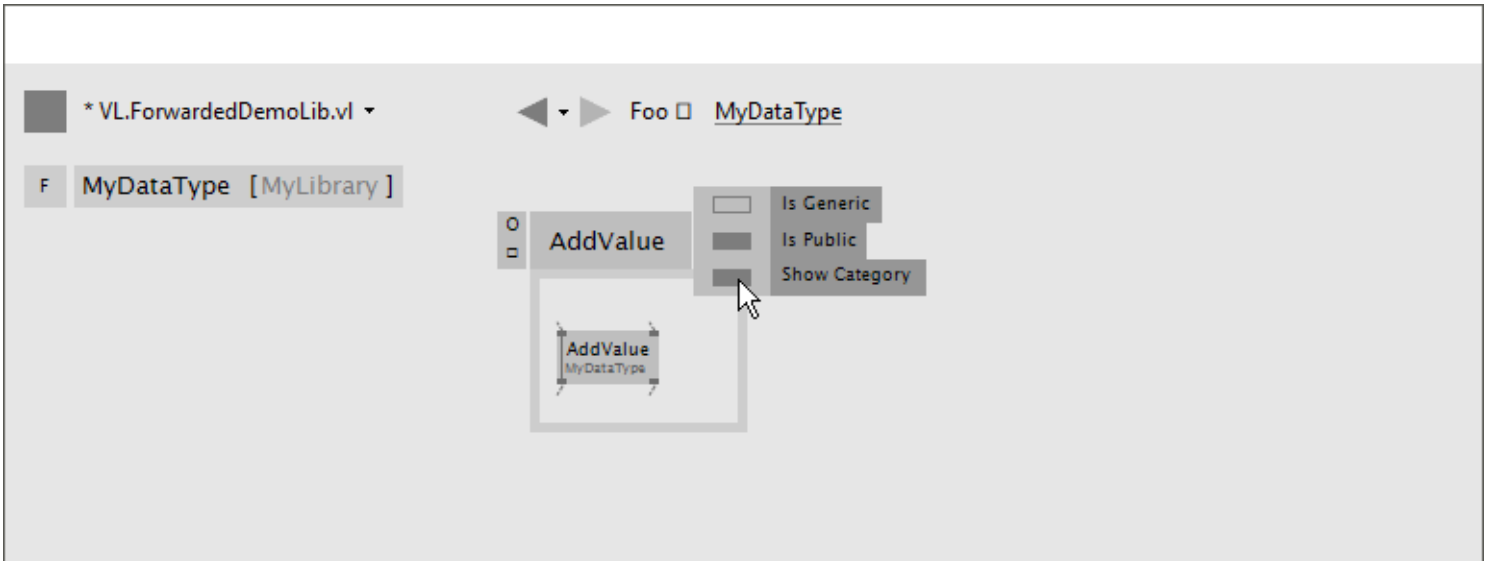
SineWave takes an angle in cycles

Show Category

By default member operations have this activated while static operations don't. The only reason to change this default should be nodes like the Vector (Join) where the fact that they are members is not relevant to the readability of a patch. Compare the following:



Vector (Join) [2D.Vector2] does not show its category, while GetSlice [Collections.Spreads] does Rightclick on the header of the operation you're forwarding and choose `Configure > Show Category` to specify whether or not a node shows its type category.



Show Category checkbox

Manually managing the Signature

When forwarding a node, you'll usually want to automatically sync its signature to the one of its surrounding definition. This is why by default the two options which manage this behavior are on:

- Locked Signature (ie. managaged by the system rather than manually by the user)
- Connect to Signature (only works with locked signatures)

A reason to disable these would be if you want to create a stable API for a vl library that you don't want to be automatically adapting to changes in the underlying .NET library. Since a change in the .NET library may cause an incompatibility for users of your vl library you'll want to have the chance to review such changes and decide how to forward them to your API.

Note

Both features "Locked Signature" and "Connect to Signature" are not limited to usage in forwarding definitions. There are other scenarios where they may be useful.

Locked Signature

Unchecking "Locked Signature" has two implications:

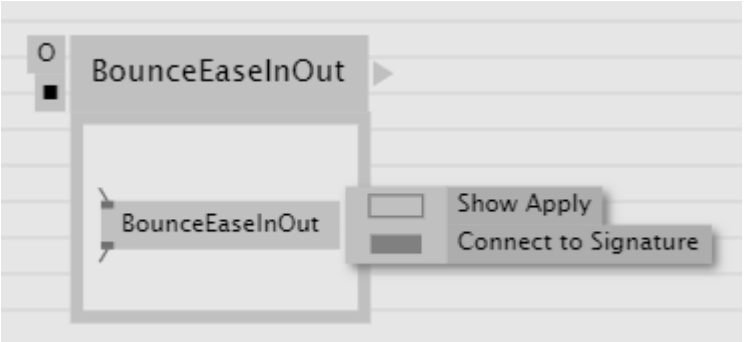
- Pins in the signature will now no longer automatically be sorted by the x-position of their representations in the patch
- Pins will not be automatically added to/removed from the signature for nodes that have "Connect to Signature" activated, if their signature changed. Instead, the signature will now show warnings which allow you to inspect those changes and react to them

See also Operation Signature.

Connect to Signature

Connect to Signature is enabled by default for nodes dropped in from the solution explorer for being forwarded. This helps saving some clicks in that it automatically connects the node to the surrounding signature, just as if for each pin you would have created a link to an own pin with the same name. If you want to have more manual control over which pins of a node are being forwarded you can disable the feature.

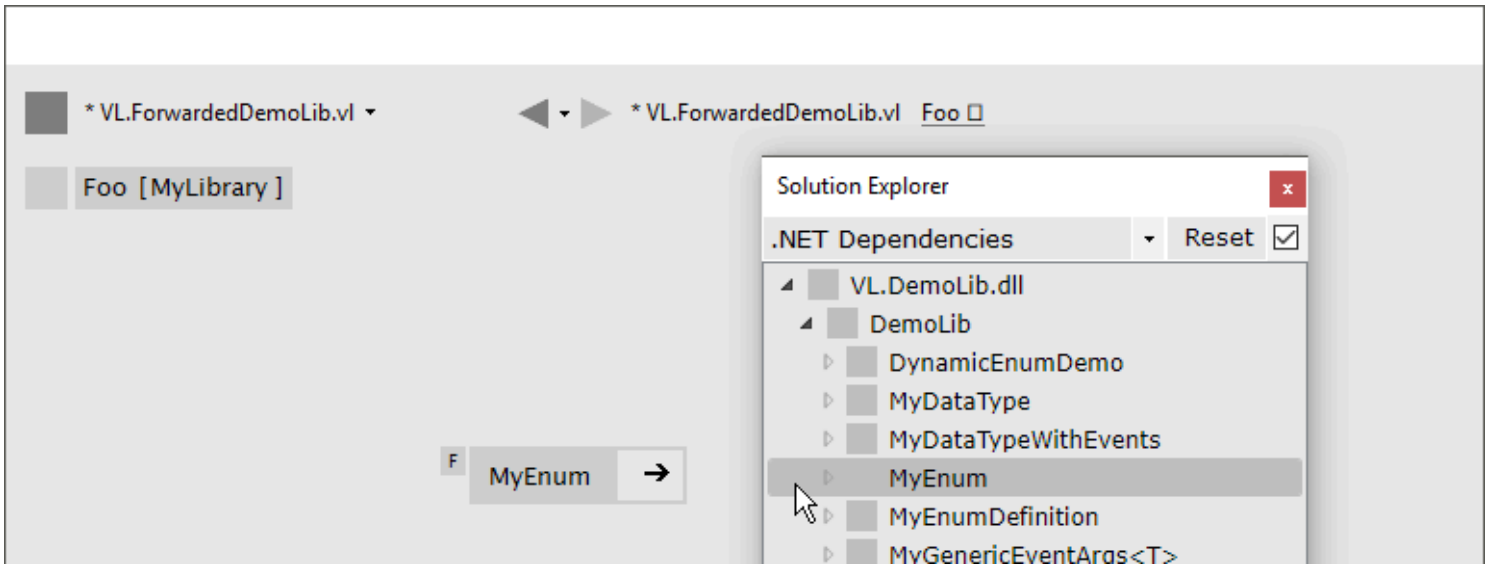
Rightclick on the node you're forwarding and choose `Configure > Connect to Signature`.



The "Connect to Signature" feature

Forwarding Enums

In order to forward an enum from a .dll to the user of a .vl document simply drag-drop the enum onto the patch.



Enum Forward

Wrapping Non-Standard Events or Delegates

Events or delegates in third-party libraries are often a reason for writing a little c# wrapper. While events that conform to the .NET Core Event Pattern are conveniently translated to observables in vl automatically, many libraries use non-standard events or delegates in which case you'll have to write a conversion to observable in c# manually using Observable.FromEvent that comes with the System.Reactive nuget.

Here is an example. Let's assume the library has a datatype `Tablet` that has an event defined like this:

```
public event PacketArrivalEventHandler (int x, int y, int z);
```

and you want to receive a notification when that event is fired, via the output of a node in VL.

First you need to create a class for the type of notification you want to receive in VL. It may look like this:

```
public class PackageArgs: EventArgs
{
    public readonly int X;
    public readonly int Y;
    public readonly int Z;

    public PackageArgs(int x, int y, int z)
    {
```

```
        X = x;
        Y = y;
        Z = z;
    }
}
```

Then you can create a static operation node that receives an instance of the `Tablet` in VL and returns an `Observable<PackageArgs>` on its output:

```
public static class TabletHelper
{
    public static IObservable<PackageArgs> PackageArrived(Tablet tablet)
    {
        return Observable.FromEvent<Tablet.PacketArrivalEventHandler, PackageArgs>(handler =>
                {
                    Tablet.PacketArrivalEventHandler paHandler = (x, y, z) =>
                    {
                        handler(new PackageArgs(x, y, z));
                    };

                    return paHandler;
                },
                paHandler => tablet.PacketArrival += paHandler,
                paHandler => tablet.PacketArrival -= paHandler);
    }
}
```

Image: how this looks in vl

Note how the node is placed on Create here and saved in a pad, instead of Update in order to have the Observable only created once, which is what we want. If for some reason you need to place the node on Update (e.g. because the Tablet on its input may change), here is a little trick you can add to cache the observable and only re-create it when the input changes:

```
public static class TabletHelper
{
    public static IObservable<PackageArgs> PackageArrived(Tablet tablet)
    {
        return CachedObservables.GetValue(tablet, x => PackageArrived_((Tablet)x))
    }

    static IObservable<PackageArgs> PackageArrived_(Tablet tablet)
    {
        return Observable.FromEvent<Tablet.PacketArrivalEventHandler, PackageArgs>(handler =>
                {
                    Tablet.PacketArrivalEventHandler paHandler = (x, y, z) =>
```

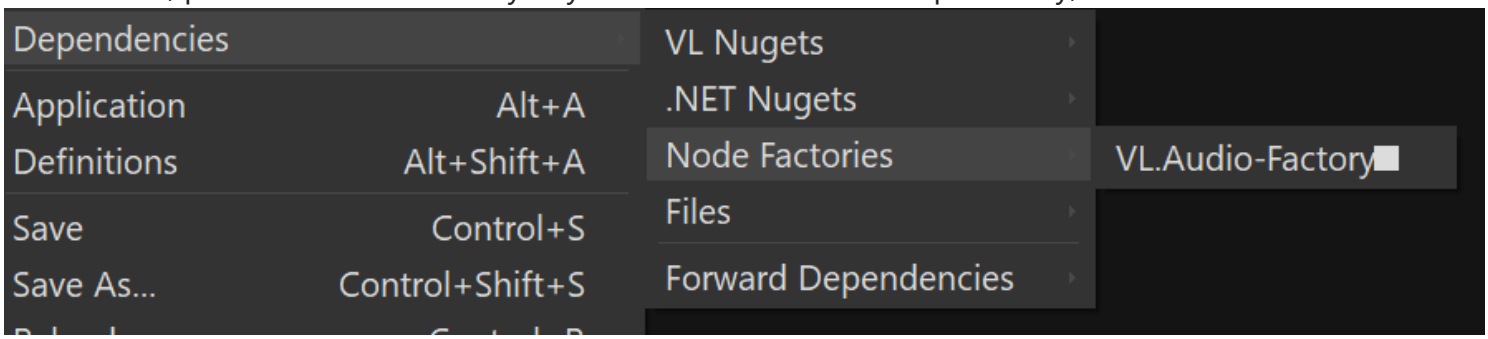
```
                    {
                        handler(new PackageArgs(x, y, z));
                    };

                    return paHandler;
                },
                paHandler => tablet.PacketArrival += paHandler,
                paHandler => tablet.PacketArrival -= paHandler);
    }
}
```

Node Factories

Node Factories allow you to create complete nodes in a language like C#, without the need to drag & drop types & methods onto a VL canvas. Factories allow you to describe nodes dynamically via an API. That way you also can create a bunch programatically.

When done, please select the factory in your VL document as a dependency, like so:



For examples, see:

- https://github.com/vvvv/VL.StandardLibs/blob/main/VL.Video/src/Initialization.cs
- https://github.com/vvvv/VL.Audio/blob/main/src/Initialization.cs
- https://github.com/vvvv/VL.RunwayML/blob/master/src/Initialization.cs
- https://github.com/vvvv/VL.StandardLibs/blob/main/VL.ImGui/src/Initialization.cs
- https://github.com/vvvv/VL.StandardLibs/blob/main/VL.Stride.Runtime/src/Initialization.cs

Aspects

Aspects are mostly used for library development. They allow to:

- Distinguish "standard" from "advanced" and "internal" nodes
- Mark nodes as "obsolete" or "experimental"

In the NodeBrowser aspects can be used to filter nodes.

VL has the following keywords for aspects:

- Advanced
- Internal
- Experimental
- Obsolete
- Adaptive

Advanced

Advanced is probably the most important aspect. The idea behind it is that any library developer provides a few super cool nodes and types that 90% of all users of a library should use. So the developer wants to have these easy to use nodes show up in the node browser by default. Only if a more advanced use case emerges for the user of the library all other nodes and types should be available as well.

In order to do that a library developer can structure their library as they please and then simply place the additional advanced nodes into the category Advanced which informs the NodeBrowser to hide them if the Advanced button is off, which is the default (as seen in the gif above).

The NodeBrowser also has a button to show/hide the 'standard' nodes (which is on by default) to make it easy to check your library for consistency.

Internal

With this aspect the library developer can fine grain the node visibility even further.

Internal makes nodes and types only available inside the .vl document that defines them. It can be used for all little utils and helpers that wouldn't make sense in a different context and should not be "exposed" by the document when it gets referenced by another document.

Obsolete and Experimental

These two should be used for past and future nodes.

Nodes that are not yet finished or are proof of concept for a new technology should be placed in the category *Experimental*. These nodes have to be used with care since they might be unstable or will have

breaking changes in the future.

Obsolete is quite self explanatory, nodes with this aspect are only there to maintain backwards compatibility and should not be used anymore because there is probably newer/better version available.

Adaptive

You can create an *adaptive* category anywhere and it will add all node signatures inside it to the adaptive system, so every type can implement such a node. e.g. the adaptive operators +, -, *, /, =, ... are placed in the *Math.Adaptive* category, which makes them show up under *Math* in the NodeBrowser.

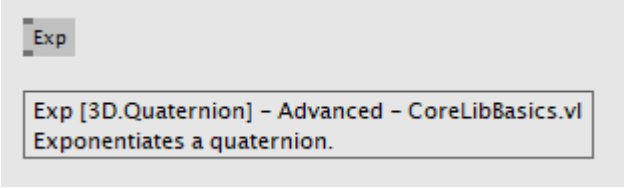
There are different ways to apply aspects to nodes:

Specifying aspects for groups of nodes

Using Categories with any of the above keywords as names, allows you to apply aspects to all nodes defined within the category.

Note that even though specified as categories, aspects do not contribute to a nodes actual full category. So for example categories *MyLibrary.Particles.Advanced* or *MyLibrary.Advanced.Particles* will simply become *MyLibrary.Value*.

In a tooltip aspects are clearly distinguished from categories:



Specifying an aspect for a single datatype definition

For a Process, Record or Class, aspects can be set via the Patch Explorer.

Aspects assigned to a type, also apply to all its nested elements!

Specifying an aspect for a single operation

To specify an aspect for a single operation, simply use the keyword in the Version part of an operation name, like:

```
GetBytes (Advanced) or Transform (Normal Advanced)
```

Design Guidelines

During the design of a library a lot of choices have to be made. Be it regarding namings or patterns being used or datatypes that are being exposed. Here is a collection of guidelines that we followed when we created the VL.CoreLib and other libraries we ship with VL. In order to provide users of VL with a consistent look/feel/behavior of libraries/nodes we recommend you follow these guidelines for the design of your own libraries.

Namings

Documents

The name of the main document of a library should hint at the category the library introduces. If a library introduces more than one categories (like, eg. the VL.CoreLib) choose a broader term that identifies the library.

```
VL.CoreLib.vl
VL.Devices.Leap.vl
VL.EditingFramework.vl
```

Document names

Node Names

A nodes name consists of 3 components:

- Name
- (Version)
- [Category]

Name

When choosing the name part of a node follow these rules:

- use CamelCasing, no spaces
- for process nodes use nouns: Sequencer, FlipFlop, Copier
- for operation nodes prefer verbs: Map, Copy, Sample
- avoid node names starting in "As.." like "AsString". Use "To.." or "From.." instead (see below for more rules on To/From nodes)

Version

As an alternative to "function overloading" VL uses Versions. They are optional and most nodes should not have a version. When versions are needed they can also consist of multiple space-separated words. Use the following rule to decide on a nodes version(s):

When you want to denote more specific versions of an already existing node with the same name the simpler one of two nodes with the same name should always have no version while the more specific one has a version that describes its speciality

Category

Just like a nodes name, the category is obligatory. Subcategories can be created by dot-separated terms. Further follow these rules when deciding about the category of a node:

- Prefer existing Categories over inventing your own
- Avoid excessive use of Subcategories

```
LFO [Animation]
Copier [IO]
Map (Range) [Math.Ranges]
RGBA (Join) [Color.RGBA]
RGBA (Join Vector4) [Color.RGBA]
```

Node names

Node Tags

Tags are additional search-terms you can equip nodes with that will help people to find them in the NodeBrowser. The following rules apply:

- a list of lower-case space-separated terms
- don't use any term that is already part of the nodes name, version or category.

Create vs. (Join) vs. From...

There is a special category of nodes that are typically called "constructors" as they create an instance of a datatype. Instead of naming all those nodes "Create" or calling them simply by their datatype name, eg. "Rectangle" we identified 3 different types of such constructor nodes that we each intuitifely want to name differently.

```
Create [Particle]
RGBA (Join) [Color.RGBA]
FromHSL [Color.RGBA]
```

Constructor names

Agreed, the boundaries between them are a bit blurry but here is how we go about deciding which naming scheme to use:

Create

When you patch your own datatype in VL it comes with a "Create" operation by default. We keep that default for complex datatypes, like eg. Particle, that are more than containers for a bunch of properties in

that they have some functionality. In those cases it feels right to see a node named Create [Particle] when we use it in a patch to understand that at this point a particle is being created.

Join and Split

If the datatype you create is more or less used as a container for a bunch of properties, it is often useful to have a pair of join/split nodes: The join-node has one input-pin for each property allowing you to create an instance of the datatype while the split-node is its inverse in that it has one output-pin for each property allowing you to split the datatype into its individual components. For those cases we don't use the default "Create" operation but instead rename it to "MyDatatype (Join)" and create a "MyDatatype (Split)" operation that each do nothing more than writing to/reading from the internal properties of the datatype.

```
Vector2 (Join) takes X and Y as inputs
Vector2 (Split) returns X and Y as outputs
RGBA (Join) takes Red, Green, Blue, Alpha as inputs
RGBA (Split) returns Red, Green, Blue, Alpha as outputs
```

Examples for Join/Split nodes

Note

For now you'll have to patch such join/split nodes manually. When you do so you'll notice an unwanted Input pin on the Join node and an unwanted Output pin on the Split node which you'll have to ignore for now. Later there'll be an option to get join/split nodes automatically for each datatype patch which will not have those unwanted pins.

From.. and To..

Nodes starting in "From.." or "To.." create and instance of a datatype by converting from a given one to the desired one. It could be argued that we should decide on naming all of those nodes either "From.." or "To.." for simplicity. But our rational for allowing both is that both variants make sense in terms of where the nodes are defined.

If you have a library called "FooStuff", that defines a datatype "Foo" it could make sense to have the following "constructor" operation for Foo:

- FromBar [Foo]

Also the following converter operation could make sense:

- ToBar [Foo]

If two nodes doing exactly the same thing as the two just mentioned would instead be defined in a library called "BarStuff" they'd be named:

- ToFoo [Bar]

- FromFoo [Bar]

..To.. Converters

If a converter merely converts between units, like from cycles to radians but the data-type of the input and output pins is the same the node name has to mention both units, like: CyclesToRadians. Since the data-type does not change here even hovering the pins wouldn't give sufficient information to understand what the node is doing.

Pins

- use spaces to separate words all starting with upper case
- avoid using generic names like "Do", "Update", ...

Order of Pins

Main input left, .. Reset typically right

Inputs

"Apply" is a reserved word for pin-names and therefore the compilier will complain when a user chooses this name manually for a pin. The reason for this is that there is a pattern where an "Apply" pin will automatically be created for operations. Like this, whenever we encounter an "Apply" pin we can be sure that this pattern is applied.

Operations

Any operation (both utility or member) that has

- either no output at all
- or one input named "Input" and one output named "Output" whose type is the same as the type of its "Input" and no further outputs

automatically gets an "Apply" input. The "Apply" pin is hidden by default and can be shown via the -> Configure menu on the node. It is set to "true" by default. Setting it to "false" will bypass the operation and simply pass the input value through to the output.

Process Nodes

Any operation of a process node that has

- no output

automatically gets an input that is named after the operation. This pin is set to "false" by default meaning the operation is not executed. Setting it to "true" will execute the operation.

Also see Pin Groups below.

Outputs

- Output vs. Result
- see below: Nodes that work async

Standard Datatypes

In order to keep the number of datatypes a user typically has to deal with at a managable level here is a list of datatypes that we use on inputs and outputs of nodes:

- Boolean
- Byte
- Integer32
- Float32
- Vector2/3/4
- Matrix
- Char
- String
- Path
- Spread

Note that in the implementation of a node you can of course use any datatype you want.

Standard Units

- Color Components (red, green, blue, alpha, hue, saturation, lightness) range from 0 to 1
- Angles are specified in cycles (a range from 0 to 1 counter clock-wise)

Patterns

Dynamic Pin Counts

Nodes like the "Cons" or the "+" can have their input count set on demand by the user. Pressing `Ctrl` `+` or `Ctrl` `-` with such a selected node will add/remove inputs accordingly.

Any operation that has exactly two inputs and one output whose type is the same as the first input gets this functionality automatically.

The other case, where you want to have a node to create pins on demand (think Timeliner, Automata) is not yet supported!

Adaptive Nodes

Adaptive nodes allow you to define the signature (ie. names and order of input and output pins) of a node and then provide concrete implementations for different datatypes.

In the NodeBrowser you'll only see one option instead of all the implementations and typically this choice will be fine because now the compiler will choose the correct implementation for you as soon as you connect any links to it.

Example: Think of a LinearInterpolation (Lerp) node that can have concrete implementations for different datatypes like Float32, Vector2,... one could even think of an implementation for strings but the signature of such a node would always be the same: Input 1, Input 2, Scalar, Output.

Adaptive Definition

Create an operation and make sure to put it in the toplevel ''Adaptive'' category. Add input and output pins and name them to your liking. You can even annotate individual pins but at least one of the pins should be left generic otherwise you cannot provide different implementations for this definition.

Adaptive Implementations

Create an operation in any other category using the same signature and implement it in a non-generic way, ie. this time all in- and outputs need to have a datatype infered or annotated.

When creating multiple implementations (for different datatypes) make sure you put the operations in different categories.

Replace an adaptive node with a specific implementation

There are cases where you'll want to make sure the compiler uses one specific implementation for an adaptive node. To choose a specific version for an adaptive node, first place the node via the NodeBrowser. Then doubleclick it to see all available implementations in the main panel from which you can simply choose one.

Process Nodes

Reset Inputs

Reset always takes precedence over other inputs (is lowest in process explorer)

- eg: FlipFlop

Nodes that operate async

- typical outputs
  - In Progress
  - On Completed
  - Success
  - Error

Exception Handling

Still to be defined (see internal issue #1511):

- simply throw errors as they occur
- test input ranges to prevent errors (e.g clamp or wrap incoming values to a save range,...). optionally report overflow via an Overflow (Bool) output
- return Default if operation fails and report Success
- use try/catch and report errors via a set of standard pins: Success (Bang), Error (Bang) and Error Message (String)

Caching outputs

When to do it and when not

Resolving relative paths from within a node

This is not possible yet. For now you need to use absolute filenames.

Saving Data

There is no way yet for a node to save data with a patch. For now you need to save anything in an extra file.

Events/Observables

If you are dealing with asynchronous datasources - async await, task, events - always hand them to your users as Observables. See Writing Nodes.

Resource Providers

Many thirdparty libraries we can use rely on unmanaged resources under the hood which requires the manual handling of their disposal when they are no longer needed. An example for such a resource type would be the Bitmap or usually any type that gives you access to a physical device. Forgetting to dispose such a resource usually quickly lead to errors.

Taking care of the disposal is not a big deal though as long as you only need access to such resources within on operation. Simply use Dispose [IDisposable] to free them when no longer needed.

Only when resouces need to be saved in fields for being accessed over time and thus they are leaving the scope of where they've been created things become more tricky.

For those scenarios VL comes with a category called [Resouces] which includes the following nodes: New, BindNew, Do, Execute, Using,...

Restore Methods

When importing types with generic type parameters, you need to write restore methods for them.

Default Values

Define default values for imported types by creating a Forward operation called "CreateDefault". Whenever the VL typesystem encounters that type it will look for a "CreateDefault" to avoid NUlL values in inputs of nodes.

This forward must not have a side-effect. This may not always be possible/make sense, then we'll still have to deal with null.

Immutability

Since in .NET it isn't possible (yet) to mark types as immutable you can do so when importing a type to VL.

Nugets

Don't reference your own nuget in any .vl documents that contribute to a nuget other than: help patches

Tests

Still to be defined: in what form to provide tests (patches, code,..) that can be run automated

Help Patches

See Providing Help

Providing Help

Library developers can provide patches demonstrating different aspects of the library.

We distinguish 5 types of patches:

- Explanation: Typically a single patch per library giving an overview of the whole set of nodes the library provides
- HowTo: A series of patches demonstrating how to achieve specific things using certain combination of nodes provided by the library
- Reference: A patch covering the functionality of one specific node
- Tutorial: Most often a link to a video tutorial
- Example: A patch more broadly showing a usecase of of a library, not necessarily explaining too much but more giving an idea of what's possible

These (except the "Examples") follow the idea of The Documentation System.

In order to be picked up by the helpbrowser, files need to be put in the right place and follow the naming convention:

```
\help\Explanation Overview of available nodes.vl
\help\HowTo Do something.vl
\help\Referece Nodename.vl
\help\Example Something Beautiful.vl
```

In case a library has a lot of help patches, you can also use up to two levels of subdirectories to structure them, like so:

```
\help\Topic\Subtopic\HowTo Do something.vl
```

By default the helpbrowser will display those patches in alphabetical order. If you want to change that, you can provide a Help.xml file in the \help directory that allows you to structure and order the content independent of their order in the filesystem. This also allows you to add additional help content in the form of links to online resources, like so:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Pack>
  <Topic title="Overview">
    <UriItem title="Explanation An Introduction to VL.OpenCV" link="https://youtu.be/4hPH5CokxwQ
    <UriItem title="Reference Finders" link="https://vvvv.gitbooks.io/the-gray-book/content/en/r
  </Topic>
  <Topic title="Topics">
    <Subtopic title="Images">
      <VLDocument link="Topics\Images\HowTo Draw images.vl" tags="picture render"/>
    </Subtopic>
```

```
  </Topic>
</Pack>
```

Note how for UrlItem elements you can provide a "mediaType" attribute that allows "text" or "video" values which will lead to little according symbols showing after the items title.

The search will work on all words in items titles. If you want to add more search terms that don't fit in an items title, use the "tags" attribute to add a list of space-separated search terms.

Help Flags

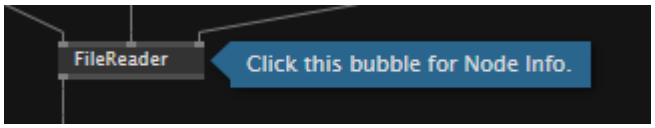
A help flag is used to specify which how-to patch should open when a user presses `F1` on a selected node.

In a how-to patch, select the node you want this patch to open for as a help patch and press `Ctrl` `H`. This sets a high-priority help flag. Press a second time to change it to low-priority, press a third time, to clear the help flag again.

FileReader with a high-priority help flag, SplitToLines with low-priority

High vs. Low Priority

When pressing `F1` on a node, the system will go through all help flags in all how-to patches of a library. If it encounters a high-priority help flag for the given node in a how-to patch, this patch will be displayed as the help patch. Therefore it only makes sense to specify a high-priority help flag once for each node!

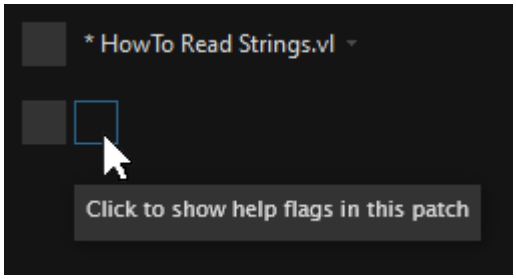


The bubble indicating which node the help patch was opened for

After pressing `F1`, when the user decides to also view the Node Info for the node, the help browser will display a list of how-to patches that include the node with a low-priority help flag set. Therefore it makes sense to specify multiple low-priority help flags for the same node in different how-to patches to indicate that those could also be interesting when looking for use-cases of the given node.

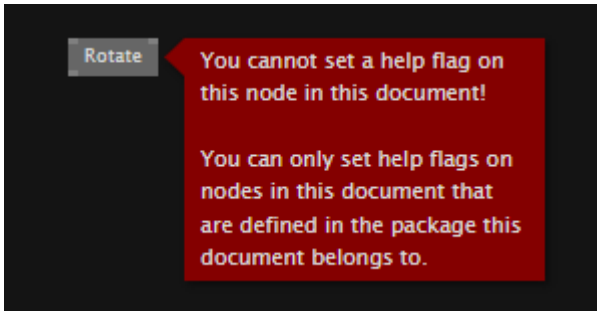
Help flag indicator

Since the end-user doesn't need to see the help flags, by default they are invisible when a help patch is opened. You can toggle the help flag indicator to show/hide help flags in this patch.



The help flag indicator indicating that this how-to patch has help flags set

Troubleshooting



Warning: Help flag cannot be set on this node

If you get this warning, check the following:

- Make sure the pack you're adding a help patch to is referenced as a source repository
- Make sure the node you want to set the help flag on is defined in the same package as the help patch you're currently preparing
- If you're still seeing the warning, save your help patch, restart vvvv, open the patch again and try to set the flag again

Publishing a NuGet

The following guide covers the necessary steps to setup a workflow on your Github repository that will publish your plugin to nuget.org or any feed you like using the PublishVLNuget Github Action.

The guide assumes you already have created an account for GitHub and nuget.org and that you have access to an existing VL GitHub repository.

References

The explained configuration is currently being used for libraries such as:

- VL.IO.OSC
- VL.Devices.AzureKinect
- VL.Devices.Nuitrack
- VL.Devices.RealSense
- VL.OpenCV

Feel free to use them as starting points for your own library.

A brief introduction to Github Actions

Github actions are small scripts with a specific purpose, allowing you to automate tasks on your repos. They are actually the building blocks of what's called a *workflow* : you chain several actions one after the other in your own small script, and decide under which condition the workflow is triggered (a new commit on `master`, a new tag, etc).

Our action will do the following tasks for you :

- Build your Visual Studio solution, if your plugin has one
- Download a package icon from an external url if you don't want to commit it to your repos every time
- Pack your nuget either using a `nuspec` or a `csproj` file
- Publish it to nuget.org (or any other feed)

An action takes input parameters, listed as key-value pairs. In a workflow script, our action could look like this :

```
- name: Publish VL Nuget
    uses: vvvv/PublishVLNuget@1.0.42
    with:
    csproj: src\VL.MyLib.csproj
    nuspec: deployment\VL.MyLib.nuspec
    icon-src: https://foo.bar/icon.png
```

```
    icon-dst: ./deployment/nugeticon.png
    nuget-key: ${{ secrets.NUGET_KEY }}
```

For a list of all the input parameters the action can handle, head over to its Github repo.

For more information on Github Actions, check the official documentation.

Preliminary notes

nuspec file

The nuspec file contains your nuget's metadata, such as version, author and dependencies. It also specifies which files should be included in your final package. We recommand putting it in a `deployment` folder at the root of your repo, but it can be anywhere you like.

For more information on the `nuspec` file format, check the documentation from Microsoft.

Dependencies

In the nuspec file, make sure you list the nugets needed by your library/project under the `dependencies` section.

Assets, binaries, help files, etc.

In the nuspec file, make sure you list any assets, dlls, help patches, etc. under the `files` section.

Version

Your package version should follow the semver specification.

A nuget package can come in two versions, a release version or a pre-release version.

A pre-release package implies that the package is currently under development and things can change drastically from one version to the next. Functionality can also be expected to break or become unstable from time to time.

A release package implies that the package has been properly tested and polished for production. No major breaking changes are expected to happen and stability within the package should be reliable.

If you want to publish a pre-release version of your package you need to instruct nuget.org that this is indeed a pre-release version. To do this you must add the `-alpha` suffix at the end of your package's version.

`csproj` file

If your plugin has a `csproj` file, it can also be used to pack your nuget in place of a `nuspec` file. For more information, please refer to this section of the Nuget documentation.

If you plan to use it for that purpose, just omit the `nuspec` input of the Github Action.

Package icon

Our Github Action allows you to specify a package icon from an external source with the `icon-src` and `icon-dst` input parameters. That way, you don't have to commit your icon file to your repo. The file will be downloaded and put inside your package each time your workflow runs.

Please note that you have to set the `icon-dst` input parameter to an already existing folder in your repo. We suggest you simply download it to the root of the repository, like so:

```
(...)
- name: Publish VL Nuget
    uses: vvvv/PublishVLNuget@1.0.42
    with:
    (...)
    icon-src: https://wwww.url.to/nugeticon.png
    icon-dst: ./nugeticon.png
```

Using a `nuspec` file

In your action, set the icon destination to the root of the repo :

```
(...)
- name: Publish VL Nuget
    uses: vvvv/PublishVLNuget@1.0.42
    with:
    (...)
    icon-src: https://wwww.url.to/nugeticon.png
    icon-dst: ./nugeticon.png
```

> Note : paths in the workflow file are relative to the root of the repo.

Then, in the `file` section, your nuspec file must reference it from where the action will download it (`src` attribute) and place it wherever you like (`target` attribute), making sure `target` matches where the `metadata` section expects it.

```
(...)
    <metadata>
        (...)
        <icon>icon\nugeticon.png</icon>
    </metadata>
    <files>
        (...)
        <file src="..\nugeticon.png" target="icon\">
```

```
    </files>
(...)
```

> Note : paths in the nuspec file are relative to where the file itself is placed.

Using a `csproj` file

You can setup an icon for your project inside Visual Studio. Beware you'll have to specify a path to a file that does not exist yet, since the Action will take care of downloading it later on. This can feel weird since Visual Studio's UI gives your a `Browse` button for you to pick a file. simply fill the path manually to match the `icon-src` property of your workflow file.

For instance, your worflow file would look like this:

```
(...)
- name: Publish VL Nuget
    uses: vvvv/PublishVLNuget@1.0.28
    with:
    csproj: src\Whatever\Whatever.csproj
    icon-src: https://wwww.url.to/nugeticon.png
    icon-dst: ./deployment/nugeticon.png
    nuget-key: ${{ secrets.NUGET_KEY }}
```

And your Visual Studio configuration like this :

Using the Action

Getting a nuget.org API key

The following steps will guide you through the nuget.org configuration, before proceeding please make sure you have a working account and are logged-in in nuget.org.

1. Click on your user name at the top right
2. Click on `API Keys` in the menu that pops up
3. Click on `+ Create`
4. Under `Key Name` type the name of your repository or project, this is going to be the official package name for the world when they type `nuget install <YourPackageName>`
5. Under `Package owner` make sure you select the appropriate option depending on your case, if the package should belong to an organization you are part of rather than to you, choose said organization now
6. Under `Glob Pattern` type: *
7. Click `Create`

NOTE: At this point you should see your newly created package listed with a yellow warning message reminding you to copy your key, ***this is a crucial step as this is the only time you will be able to copy this value***.

Click on `Copy` below your package's description and add it to your repository's secrets. To do so, please refer to this page of the Github documentation. Remember your secret's name, we will use it in the next step when we'll create our workflow file. We suggest you simply call it `NUGET_KEY`.

Creating the workflow file

Create a new `main.yml` file in your repository in a `.github/workflows` directory. Your repo structure should look like this :

```
├── .github
│   └── workflows
│       └── main.yml
├── deployment
│   ├── VL.MyLib.nuspec
├── help
│   └── Basics
│       ├── HowTo Foo.vl
│       └── HowTo Bar.vl
├── src
│   └── MyLib
│       ├── Baz.cs
│       ├── MyLib.csproj
│       └── MyLib.sln
├── README.md
└── VL.Whatever.vl
```

Before using `PublishVLNuget`, you need to add a few pre-existing other actions that are needed by it. So in your `main.yml` file, paste the following :

```
name: push_nuget

# on push on master
on:
  push:
    branches:
      - main
    paths-ignore:
      - README.md

jobs:
  build:
    runs-on: windows-latest
```

```
    steps:
    - name: Git Checkout
      uses: actions/checkout@master

    - name: Setup MSBuild.exe
      uses: microsoft/setup-msbuild@v1.0.0

    - name: Setup Nuget.exe
      uses: nuget/setup-nuget@v1
```

The `on` section describes under which condition the workflow is triggered. Here, we specify that when there's a new commit on `master` *except* if it's on `README.md`, we trigger the workflow.

Then, in our job, we add three actions :

- `actions/checkout` make sure our repo is checked-out on the `master` branch
- `microsoft/setup-msbuild` makes sure our action can use `msbuild.exe` to build your solution
    - As a consequence, if your plugin does not have a Visual Studio solution, you can omit this
- `nuget/setup-nuget` installs `nuget.exe`. We need it in our action to pack and push your plugin to nuget.org.

Now that everything is setup, we can add our action and fill its parameters accordingly.

```
- name: Publish VL Nuget
    uses: vvvv/PublishVLNuget@1.0.42
    with:
    csproj: src\VL.MyLib.csproj
    nuspec: deployment\VL.MyLib.nuspec
    icon-src: https://foo.bar/nugeticon.png
    icon-dst: ./nugeticon.png
    nuget-key: ${{ secrets.NUGET_KEY }}
```

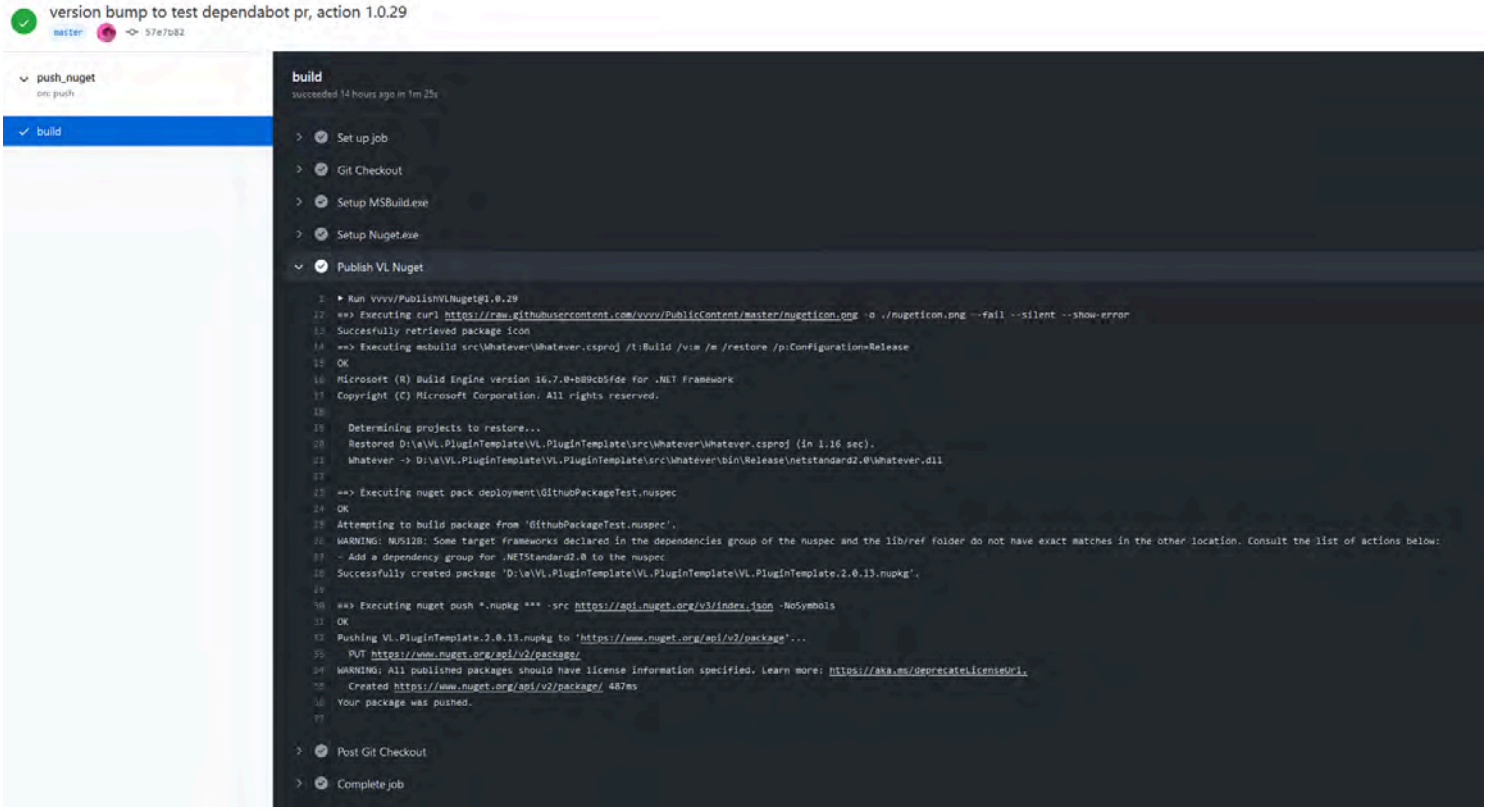
> NOTE : paths in the workflow file are relative to the root of your repo!

Wonder what is that `{{ secrets.NUGET_KEY }}`? Check Getting a Nuget API Key.

Push!

You can now push to your master branch and trigger a new deployment of your plugin. Make sure that you bump your plugin's version in your `nuspec` or `csproj` file, otherwise nuget.org (or any feed you're using) will refuse your plugin.

Head over to the *Action* section of your repo to monitor your worflow run in real time. If errors occur during the workflow run, they'll show up here.


version bump to test dependabot pr, action 1.0.29
master
57e7b82
push_nuget
on: push
build
build
succeeded 14 hours ago in 1m 25s
Set up job
Git Checkout
Setup MSBuild.exe
Setup Nuget.exe
Publish VL Nuget
1 ▶ Run vvvv/PublishVLNuget@1.0.29
12 ==> Executing curl https://raw.githubusercontent.com/vvvv/PublicContent/master/nugeticon.png -o ./nugeticon.png --fail --silent --show-error
13 Succesfully retrieved package icon
14 ==> Executing msbuild src\Whatever\Whatever.csproj /t:Build /v:m /m /restore /p:Configuration=Release
15 OK
16 Microsoft (R) Build Engine version 16.7.0+b89cb5fde for .NET Framework
17 Copyright (C) Microsoft Corporation. All rights reserved.
18
19 Determining projects to restore...
20 Restored D:\a\VL.PluginTemplate\VL.PluginTemplate\src\Whatever\Whatever.csproj (in 1.16 sec).
21 Whatever -> D:\a\VL.PluginTemplate\VL.PluginTemplate\src\Whatever\bin\Release\netstandard2.0\Whatever.dll
22
23 ==> Executing nuget pack deployment\GithubPackageTest.nuspec
24 OK
25 Attempting to build package from 'GithubPackageTest.nuspec'.
26 WARNING: NU5128: Some target frameworks declared in the dependencies group of the nuspec and the lib/ref folder do not have exact matches in the other location. Consult the list of actions below:
27 - Add a dependency group for .NETStandard2.0 to the nuspec
28 Successfully created package 'D:\a\VL.PluginTemplate\VL.PluginTemplate\VL.PluginTemplate.2.0.13.nupkg'.
29
30 ==> Executing nuget push *.nupkg *** -src https://api.nuget.org/v3/index.json -NoSymbols
31 OK
32 Pushing VL.PluginTemplate.2.0.13.nupkg to 'https://www.nuget.org/api/v2/package'...
33 PUT https://www.nuget.org/api/v2/package/
34 WARNING: All published packages should have license information specified. Learn more: https://aka.ms/deprecateLicenseUrl.
35 Created https://www.nuget.org/api/v2/package/ 487ms
36 Your package was pushed.
37
Post Git Checkout
Complete job

Editor Extensions

Editor extensions allow you to extend the vvvv editor with your own tools. Examples of such extensions are:

- The Key & Mouse display (shipping with vvvv)
- TUIO Simulator & Monitor (install via VL.TUIO.HDE nuget)
- Spout Monitor (install via VL.SpoutMonitor.HDE nuget)
- Desktop Pipette (install via VL.Pipette.HDE nuget)

Extensions can be entirely patched in vvvv and assigned a shortcut so they can be called by the user at anytime. You can find all currently loaded extensions in the main menu under:

`Quad > Extensions`

Creating an extension

Extensions are ordinary VL patches, with the one distinctive feature, that they are saved in a file ending in `.HDE.vl`, like:

`VL.MyExtension.HDE.vl`

Any such file that you have open in vvvv, is already running as an editor extension. You'll now most likely want to be able to invoke your extension by a shortcut or menu-entry. For this you'll have to register a command which you can do via the `Command` node that is available via referencing the `VL.HDE` nuget.

Or simply start from the template:

Creating an extension from the template

From the main menu choose:

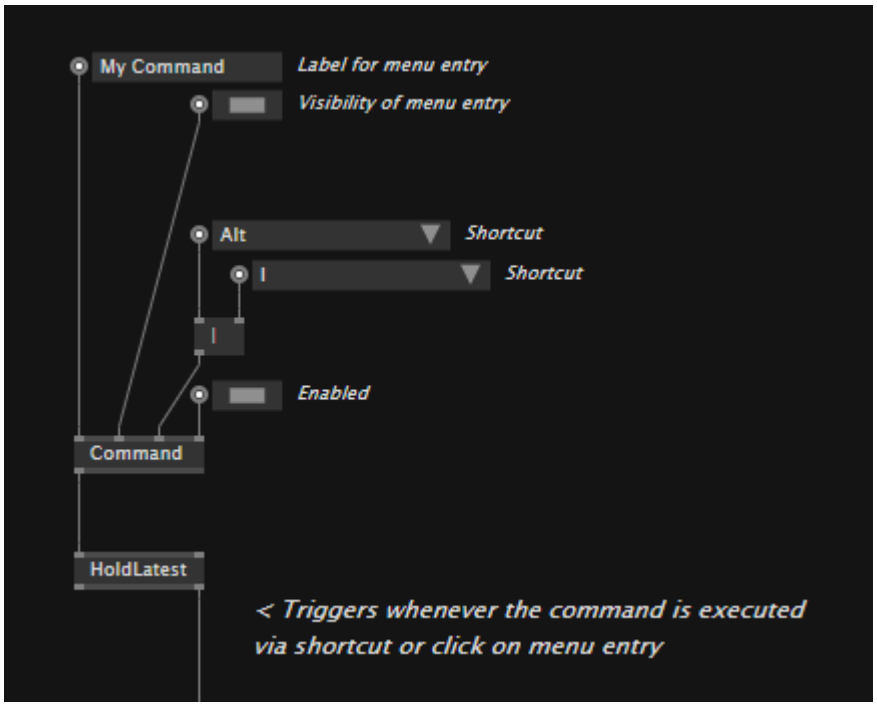
`Quad > Extensions > New Extension...`

and specify the destination and filename for your new extension .vl document in the file save dialog.

Note

The filename must end in `.HDE.vl` in order for the file to be a valid editor extension.

This copies the template to the given destination and opens the document for you, which should look like this:



Note the `Command` node here that allows you to register a command with the editor, by specifying:

- A label for the menu entry
- Whether the menu entry is visible
- A shortcut

Now check the output of the `Command` that will trigger whenever the command is executed via shortcut or click on menu entry.

You take it from here. A classic usecase will be showing/hiding an editor extensions window on repeated executions of the command. But as within any other patch, you're free to run even `format c:` here, so as always, handle with care...

Multiple extensions per .HDE.vl file

If that is what you need, then yes, simply register multiple commands in your document that trigger individual extensions.

Note though, that should one of your extensions have a runtime error, all the others running in the same document, might be affected as well.

Windows

An extension doesn't necessarily have to have a window at all (e.g. it could just run an operation on a bunch of selected nodes...). It can also have its very own idea of a window. But in many cases you'll just want to use a ready window that comes with `VL.HDE`. For now these two are available:

- SkiaWindow: A slimmed down version of a Renderer [Skia]

- SkiaWindowTopMost: Same as above, only doesn't get the focus and is always topmost (e.g. used in the Key/Mouse Display extension)

Window Bounds

As of now there is no way to tab or dock multiple extension windows. Setting a windows bounds is the best we can do for now. Yes, with many extensions open, this will get messy. Let's see...

In order to set the initial window bounds when an extension window opens, set the `Bounds` input on the SkiaWindow nodes. For convenience checkout out:

- WindowBounds: Positions the window right next to the editor
- WindowBounds (EditorHeight): Same as above but also sets its height to be the same as the main window (e.g. used in the HelpBrowser)

Interfacing with vvvv

The vvvv API gives you access to hovered and selected nodes and allows you to read and write pins.

- In the HelpBrowser check the `API` section for some examples.
- All the API nodes are available via the `VL.Lang` repository in the `Session` category.

Settings

For some extensions it will be interesting to have them configurable via the Settings. This is not yet possible though.

Packaging extensions in a NuGet

Extension-only NuGet

If you want to release a nuget that only contains an extension, its ID has to end in `.HDE`, e.g.:

`VL.MyExtension.HDE`

which in turn requires the .vl document inside that nuget to be following the same naming and be called:

`VL.MyExtension.HDE.vl`

Extension as part of a NuGet

Extensions can also be included with any NuGet that primarily offers other functionality. In such a case, the name of the extension document has to be identical to the package ID but with the `.HDE` suffix added. For example a NuGet named `VL.MyPackage` would contain at least these two documents:

```
VL.MyPackage.vl // main doc
VL.MyPackage.HDE.vl // extension doc
```

Restarting all extensions

Especially during development it may happen that you crash an extension and it needs a restart. You can restart all extensions at once using the shortcut `Shift` `F9`.

Best Practice

Here is a list of best-practice articles on various topics:

3d Rendering

- Text Rendering

Video

- Capturing video input
- Playing back video
- Video synchronization
- Recording output as video

Development

- Deploying to a Raspberry Pi
- Version control using Git

Video Playback

vvvv offers two distinct ways for video playback:

- Video files
- Image sequences

Video files

- Referencing the VL.Video nuget (shipping with vvvv) adds the **VideoPlayer** [Video] node
- This node plays a wide range of video containers and codecs out of the box
- If you're missing any codecs for playback, see if they are available in this Mediafoundation Codec Pack.
- Consult the help browser for examples on using the node

Advantages

- Quick and easy to use
- Forgives unstable rendering framerates, by dropping frames if needed

Disadvantages

- No seemless looping (may work with certain codecs but not with others)
- No seemless switching between sources (may work with certain codecs but not with others)
- May show micro-jitter, most notably in fullscreen playback, because timing is not coupled to the v-sync
- No network sync

Note

For the playback of HAP video files the thirdparty VL.HapPlayer nuget is required.

Image sequences

- Depending on the rendering engine you are using, the following nodes are shipping with vvvv:
  - VL.Stride (3d engine): **ImagePlayer (Stride)** [Video] or **ImagePlayer (FrameBased Stride)** [Video]
  - VL.Skia (2d engine): **ImagePlayer (Skia)** [Video] or **ImagePlayer (FrameBased Skia)** [Video]
- The ImagePlayers for VL.Stride prefer images in the DDS format (see below for conversion tools)
- Both support the playback of JPG, PNG and BMP files
- Consult the help browser for examples on using the nodes

Advantages

- Nodes exist in two variants: timebased and framebased, see below

- Can do seemless looping
- Allows to implement seemless switching between sources
- The playback of image sequences on multiple PCs in a local network can be synchronized

Disadvantage

- Less comfortable handling of media assets (ie. thousands of image files)
- Audio tracks need to be played separately using VL.Audio

Timebased vs. Framebased

Time based

- The default, simpler to use option
- Use for scenarios where the video is not playing fullscreen but rather is part of a scene
- Forgives unstable rendering framerates, by dropping frames if needed
- May show micro-jitter, most notably in fullscreen playback, because timing is not coupled to the v-sync

Frame based

- Use for scenarios where the video is playing fullscreen
- For frame-perfect, v-sync coupled playback
- Requires a perfectly stable rendering framerate

DDS conversion tools

- Texconv: Commandline tool
- TexconvGUI: A GUI for the above
- Intel's Texture Works: A Plugin for Photoshop
- NVIDIA Texture Tools Exporter
- AMD Compressonator

Video Capture

For capturing video from webcams and capture cards, vvvv offers different solutions:

VL.Video

This NuGet is shipping with vvvv. Referencing it, gives you the **VideoIn** node which supports all USB cameras that have a UVC 1.1 driver.

VL.Devices.Decklink

For capturing from Blackmagic Decklink devices.

VL.Devices.uEye

For capturing from IDS Imaging uEye Cameras.

Video Recording

Here are a couple of options for screen recording.

XBox Game Bar

Typically installed on Windows 10, otherwise available via the Microsoft Store.

Press `Windows` `G` to open it. Allows you to record fullscreen video plus audio.

NVidia Cards

If not yet installed with your drivers, download and install Geforce Experience.

Press `ALT` `Z` to open it. Allows you to record fullscreen video plus audio.

AMD Cards

If not yet installed with your drivers, download and install Radeon Software.

Press `ALT` `R` to open it. Allows you to record fullscreen video plus audio.

OBS Studio

Free and open source software for video recording and live streaming: https://obsproject.com/

VL.ScreenRecorder

If you're looking for a recording option as part of your application, have a look at the VL.ScreenRecorder NuGet.

VL.LoopTool

A collection of nodes designed to simplify the creation of repeating loop animations. Includes animation nodes that respond to a global sequencer, making it easy to create smooth tween animations. Provides animated camera and scene presets. Captures video, image sequences and images.
NuGet: VL.LoopTool

High resolution Texture/Image Writer

If you want to write high-resolution image sequences in non-realtime, use the the following nodes:

- for VL.Stride: TextureWriter in combination with SceneTexture
- for VL.Skia: ImageWriter in combination with Renderer (Offscreen)

Use those in also combination with the MainLoop node set to "Is Incremental" and specifying the "Incremental FPS" you need. This makes sure that the timing of all nodes that are depending on a clock (like LFOs, Filters,...) is correctly advanced, no matter how long it takes, to write each of the images to disk.

Note

This technique does not work for scenarios where your visual content relies on realtime parameters like audio analysis or realtime sensor data.

Gif Recorders

For recording animated gifs, try the LiceCap screen capturing tool.

Video Synchronization

Using either the **ImagePlayer (Stride)** or **ImagePlayer (Skia)** nodes (or their framebased variants) found in the Video category, it is possible to synchronize the playback of videos that are playing on different PCs, when connected via a local network.

How it works

Both server and clients run their own playback mechanism (timebased or framebased) and the server sends control info (play, seek, loop from/to) to the clients. In addition, the server sends its current stream position which the clients can adapt to in case they diverge too far from it.

When the server sends a "play" message, all clients will be already in perfect sync depending on 3 conditions:

- Did the "play" message arrive at the same time on all clients
- Are server and clients frame synced via hardware, ie, using Quadro or FirePro cards
- Do server and clients have a perfectly stable framerate

If either of the conditions is not true, there may be an offset from the start, or an offset may happen over time. To compensate for this offset, the clients will have to adapt to the servers stream position.

Time based

In this case, the clients time is continuously adapted to the servers time if necessary so they cannot diverge much. This may though interfere with v-sync timing and increase the chance of micro-jitter which timebased playback has anyway already.

See `HowTo Synchronize players between multiple PCs` in the Helpbrowser.

Frame based

In this case, to figure out an offset on the client side and decide when to jump, you can come up with different strategies. The FramePlayer node implements one of them.

See `HowTo Synchronize framebased players between multiple PCs` in the Helpbrowser.

Text rendering

2d Graphics

For VL.Skia you have the following options:

- Skia itself comes with an extensive set of nodes for high-quality simple text rendering
- Try VL.RichtextKit for rendering richttext
- Using VL.CEF.Skia to render html content allows for complex formatted text to be rendered

3d Graphics

For VL.Stride you have the following options:

- For quick, simple Text rendering use Text [Stride.Models] (experimental)
- Use any of the above (2d Graphics) options via a SkiaRenderer or SkiaTexture node in Stride
- Using VL.CEF.Stride to render html content allows for complex formatted text to be rendered
- Use VL.Stride.Text3d for rendering extruded 3d text
- Try VL.BMFont
- Try FontStashSharp (Text rendering library addon for Stride)
- For the best available option go with the Slug library. Requires a separate license from them. If you need help with the implementation, get in touch.

Deploying to a Raspberry Pi

As of version 5.0 you can now export console apps to Linux, which makes the Raspberry Pi an excellent target.

In the Application Exporter then specify:

- Output type: Console Application
- Target: Linux

Deployment modes

As explained in Deploy .NET apps on ARM single-board computers there are two modes of deploymnet:

Framework dependent

This is the default used by the Exporter. It requires you to install .NET on the Raspberry Pi (Follow steps 1. and 2.) in order to be able to run the export on it.

After a successful export, copy the generated files over to the PI and there on a commandline run the program by typing:

```
dotnet myprogram.dll
```

Self-contained

Using this option will not require you to install .NET!

Run the export normally once, then press the "Show Details" button. At the very top you'll see a line looking like this:

```
dotnet publish -c Release --self-contained false /clp:ErrorsOnly /nologo
PathToYourProject.csproj"
```

Copy this, open a Command prompt and run the command modified like this:

```
dotnet publish -c Release -r linux-arm --self-contained true /clp:ErrorsOnly /nologo
PathToYourProject.csproj"
```

After a successful export, copy the generated files over to the PI and there on a commandline set execute permissions on the executable:

```
chmod +x myprogram
```

and then run it:

`./myprogram`

Automatic deployment of files

Copying the files over to the PI after every build can be automated. In the Exporter UI press `Advanced build configuration` and add the following lines inside the `<Project>` tag:

```
<PropertyGroup>
  <SourceFolder>PATH-TO-YOUR-PROJECTS-EXPORT-FOLDER</SourceFolder>
  <DestFolder>PATH-TO-YOUR-DESTINATION-FOLDER-ON-THE-PI</DestFolder>
</PropertyGroup>

<ItemGroup>
  <FilesToCopy Include="$(SourceFolder)\**" />
</ItemGroup>

<Target Name="Deploy" AfterTargets="Publish">
  <!-- copy all files from the source folder to the dest folder that are newer or don't exist in
  <Message Importance="High" Text="Copying files to Raspberry PI..." />
  <Copy SourceFiles="@(FilesToCopy)" DestinationFiles="@(FilesToCopy->'$(DestFolder)\%(RecursiveI
    <Output TaskParameter="CopiedFiles" ItemName="Copied" />
  </Copy>

  <ItemGroup>
    <OutdatedFiles Include="$(DestFolder)\**" Exclude="@(Copied)" />
  </ItemGroup>
  <Message Importance="High" Text="Deleting files..." />
  <Delete Files="@(OutdatedFiles)" />
</Target>
```

Specify `SourceFolder` and `DestFolder`. Then this will copy modified files over to the PI after every build.

Autostart

To have your application autostart when starting the Pi, you need to install it as a service (other options like rc.local or a .desktop file don't seem to work).

For installing a service, refer to chapter "4.4 Using A Systemd Service" of boot.pdf.

Pitfalls

- If your program is accessing files, make sure to set the `WorkingDirectory` to where your application resides on disk
- `ExecStart` needs to have absolut paths to both "dotnet" and your application. eg.: `/home/pi/.dotnet/dotnet /home/pi/MyApp/myapp.dll`

Map Pi as network drive

To map your Pi users home directory to the Z drive on your windows machine, in a command prompt run:

```
net use Z: \\[hostname]\[username]
```

Useful NuGets

- System.Device.Gpio: for GPIO, I2C, SPI, PWM, Serial port
- Iot.Device.Bindings: for higher-level specific device support
- VL.IO.RCP for remote controlling parameters of the application from a web browser
- Any of the libraries in the IO category
- SFML.Net: for audio playback and recording (NOTE: version 2.5.0 of the NuGet has audio recording broken. Compile yourself from sources to get this working)

Useful links

- Setup Raspberry Pi SSH Keys for Authentication
- .NET IoT Libraries documentation
- Debug .NET apps on Raspberry Pi

Version Control with Git

As soon as you're working on something that is more than a quick sketch, you should consider putting your .vl documents under version control using Git. There are other version control systems, but Git is by far the most widely used. It takes some getting used to, but once you get the hang of it, there is no turning back.

So instead of saving your progress under files with incremental names, like foo_1.vl, then foo_2.vl, then foo_3.vl,... where you pile up endless versions of your work but you can't remember which was which, version control allows you to:

- save the state of your work (even spanning multiple .vl documents) in one "commit"
- add a human readable message to each commit
- view your history of commit messages
- revisit any step of your work that you commited earlier

And all this, without being confused by multiple versions of the same file in one folder. As a bonus you can push your work to a cloud service for these additional benefits:

- backup
- access your work from another PC
- share it with co-workers

If you need some further convincing you may want to watch some introductory videos.

Prerequisites

Software

Essentially what you need is:

- Git itself
- A Git UI client

You can use Git via the commandline, ie. without a GUI client (some people prefer so) in which case it is enough to just download Git. In the case you decide to go with a GUI client, most of them probably also install Git for you, ie. you won't need to download Git extra.

Which GUI client to choose? In the end you may want to try different ones and see which you prefer. The vvvv development team has been happy with GitExtensions for many years. The best diff/merge tool we found is P4Merge which you can choose to use as diff/merge tool with GitExtensions (and maybe also other GUI clients).

Cloud Service

If you want to backup your git repositories in the cloud, which also allows you to easily share them with others, sign up with one of these git cloud providers.

Most of vvvv's library repositories are on GitHub, so if you want to contribute to those at some point, you'll need a GitHub account.

Terminology

- Each project is stored in a **repository**
- To mirror a local repository to a cloud service, you give it a **remote**, ie. a URL to the remote repository
- Whenever you feel like having your work in a good state, you save it in a **commit** to the repository
- A repository stores your history of **commits**
- The last commit in a repository is referred to as **HEAD**
- To upload your commits to a remote repository, you run the **push** command
- **Cloning** is the act of making an initial local copy of a remote repository
- To download commits from a remote repository, you run the **pull** command
- To revisit a particular state of your work you **checkout** the respective commit

Getting Started

Creating a new repository

TODO

Forking and/or Cloning an existing repository

TODO

Commiting

Here are some general thoughts regarding commits:

- Try to commit changes that you can describe well in a commit message
- Avoid commiting changes that involve multiple "tasks"
- One commit per task/fix/change is recommended
- Never commit changes to a file you did not intentionally change (files can get changed while working on something else by accident, or you may forget about something you tried somewhere but did not intend to commit)
- Always check the changes you are about to commit to make sure they match what you are intending to change

Working on your own

As long as you're working on a project on your own, everything is mostly straight forward. A typical workflow will look like this:

- Create a new repository for a new project
- Commit the initial state of work
- Make changes, make a new commit
- Make changes, make a new commit
- Make changes, make a new commit
- At the end of the day push your commits to remote, to have it backed up
- In case you have to move to another PC, just clone the repository on there
- Make changes, make a new commit
- Make changes, make a new commit
- Push your commits to remote
- Get back to the first PC, pull the remote commits you pushed from PC 2, and continue working

Now, what git allows you to do, is to switch to any state of your work without having worry about loosing your latest state. You do this by running a **checkout** of any particular commit of your history. When you're done looking at the older state, you can go back to your latest state easily by checking it out again.

There is much more to git, but the above should give you an idea of the most simple workflow. Practice this on your own projects, before moving on to work with a team, where things can get a bit more juicy.

Working with a team

When working with a team, depending on everyone's git expertise, it may help if you're agree on one Git client to use. So in case there are problems you can help each other out more easily.

Now the obvious problem that may arise in a team, is that people may be working on the same .vl document independent of each other:

Say you have your local changes to a file in a commit which you try to push, but before you can, git tells you, you need to pull in remote changes. As long as the file you changed locally is not touched in any of the remote commits you're pulling in, all will be fine. Otherwise git will now try to merge the changes of the remote commit into your local file. And here is where things can go wrong!

You'll see three scenarios:

- Best case: the merge goes through fine
- Worst case: git claims the merge went fine, when it actually didn't
- Fine case: git tells you it cannot do an automatic merge and asks for your help

The thing is that git is made for textual programming languages. vvvv is storing its .vl documents in XML format which luckily is text. But unfortunately git doesn't understand the structure of XML, ie. treats it as

normal text and may in some situations corrupt your .vl files in case of a merge.

So the key really is to avoid merge conflicts as good as you can. Remember, as long as work is happening simultaneously but in different files, there will be no conflicts, therefore:

Split a project into multiple .vl documents

While with vvvv you can create an entire project in just one .vl document, it is good practice to split projects into multiple documents following these guidelines:

- Have one master .vl document that does not have any definitions, but only applications
- Have multiple topical .vl documents that hold all the definitions but have an empty application
  - Examples could be: InputHandling.vl, Scene1Logic.vl, Scene2Logic.vl, LightControls.vl, Audio.vl, Rendering.vl,...
- Define owners for the individual files, ie. only an owner is allowed to commit a change to a file
- Reference the definition documents as dependencies in the master document and build the master application there

Obviously the master .vl document is a classic point of conflict because everyone needs to work on it to see their changes working. But try this: If the part you're working on is sufficiently separate from the other parts, use your own local copy of the master document to work in your parts. Here you can mess around as you please. This is just for you to test, you'd not commit this file. Just when your part is integrated as planned you make sure you have the latest master document pulled, tell everyone you'll now be pushing your part and simply copy/paste over your part in one go. Like this you even don't risk a potential merge. Obviously this doesn't work for all scenarios.

Communicate

There will be times where changes need to touch many documents at once. In those cases make sure that everyone is aware this is going to happen. Make everyone commit and push their latest state. Then one team member makes the changes touching multiple documents. If in doubt, do these changes together, while screensharing, so everyone knows what's going on.

Branching

Branching is not something you should start with.

Merging

Merge Tool for VL documents