The vvvv API

Patching a Tooltip

- Tooltips are often about data visualizations
    - On Pins and Pads

- No editing, just viewing
- So let’s not talk about tooltips,

  but about the value viewers that are used inside

- Value viewers are not only used in tooltips
- You now can place them on the patch as well

Patching a Value Viewer

- Value Viewers for existing Types
  - Value Types (colors, numbers, vectors, ...)
  - String
  - Skia Types (Layer, Path, Effects)
  - Image, Texture

- Which ones do we miss?
  - Let’s brainstorm together

Patching a Value Viewer

We can add Value Viewers for

- our own Types
- but also for existing Types!

Which prominent type would you like to have a viewer for?

- You can give it a try yourself and get some attention
  by the community and the devvvvs, guaranteed ;)

Patching a Value Viewer

Viewers are made out of Widgets

- Value Viewers translate an Value of your type to a Widget
- There is a whole bunch of Widgets that help you to layout and design your viewer
- Fluttr was an inspiration
- Some prominent Widgets
  - TextWidget, VectorWidget, SKImageWidgets, ... (covering the basics)
  - Column, Row, EmptyWidget, Flexible, Panel, ... (for layouting)
  - Some more are hidden in other libraries

Open the “Patch a Tooltip” help patch!

Patching a Value Viewer

Value Viewers get called from the user code (tracing)

- Can be any thread
- Let’s build an immutable description that captures the state of the object in that particular moment. These guys are called Widgets.
- Look at the “Sample a Path” help patch!
- See that a mutable object gets traced several times per update. We want to visualize the mutation between the calls…

Tooltips and the IOBoxes draw themselves on the UI Thread. These are the mutating views that sync to the IWidget description and are of type IWidget (View). As long as you don’t build your own widget you don’t need to know.

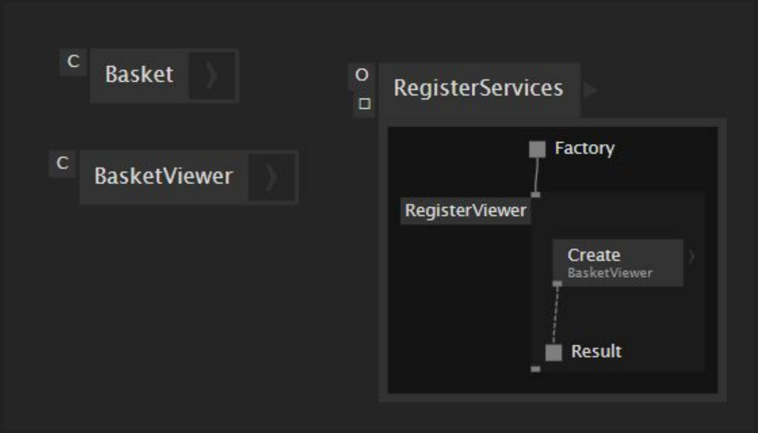
Patching a Value Viewer

- The service is called IValueViewer
- Actually there is IValueViewer<T> and IValueViewer (Ungeneric)
- We need to implement the generic version
    - This mainly helps us to get away without Cast nodes and makes our implementation easier as we have some static type safety - which generics are all about
- But the system typically is only interested in the nongeneric one
    - It typically just has an Object at hand that it tries to visualize
    - So at compile time we don't know the type and can't make use of generics
    - Each object has a runtime type which is used to look up the right value viewer at runtime

Registering a Value Viewer

Registration goes via

- A global operation RegisterServices. This will be picked up by the system
- A RegisterViewer region inside. It registers both the generic and nongeneric viewer services
- Do NOT use RegisterService yourself.



Registering other Services

Registration goes via

- A global operation RegisterServices.
  This will be picked up by the hotswap-aware factory

- A RegisterService region inside

- Whenever you change the patch the factory gets updated.
  Sometimes however you need to get rid of old instances that got created by an older version of the factory

Obtaining your Service

You now can look up the service via

- TryCreateService [IVLFactory]
- You pass an Object that you need a service for
- The factory looks at the runtime type to figure out the right service
- The factory can be obtained via the Node Context
- The Node Context gets fed into the Create operations of any Class, Record or Process
  - To pick it up you need a pin on Create named “Node Context”
  - If everything went well you end up with a pin of type NodeContext

Reacting on Patch Selection & Hovering

- GetElementsInActiveCanvas [Session] pushes
    - A Spread of Selected LiveElements
    - Whenever the selection changes or the active canves changes
- HoveredElement [Session] pushes
    - An Object
    - Which can be a LiveElement
    - You need to try to Cast with CastAs
- A LiveElement can be
    - A LiveDataHub
    - A LiveNodeApplication
- The DataStream of a
    - LiveDataHub is an Observable of Values that flow through the Pin or Pad
    - LiveNodeApplication is an Observable of the State Reference of the Process Node

More to play with

- IVLObject
- IVLTypeInfo
- IVLFactory
- Session (e.g. SetPinValue)
- Warn nodes
- TreeNodeChildrenManager & TreeNodeParentManager

Please note that the API is in a preview phase and can still change

Some more words are here: https://vvvv.org/blog/vvvv-the-tool