Nugets Workshop

One massive feature of VL is its ability to consume almost any .NET library. Most of them are available as nugets. As patchers, this gives us access to thousands of libraries to solve our problems. Sometimes though, those libraries might not exactly fit the approach we're used to from visual programming : taken out of the box, nodes can use exotic types, or simply be way too complicated to use in a high-level dataflow approach.

In this workshop, we'll learn what is the nuget ecosystem, and with hands-on examples, download nugets and make them fit our patching habits.

1. Introduction

We'll learn what is the nuget ecosystem, and how a library in structured.

There are three ways to get new nodes in VL :

- Write your own C# code
- Use a library from the GAC (Global Assembly Cache)
- Use a nuget

GAC

.NET already comes with many libraries : this is known as the GAC (Global Assembly Cache).

Hands on :

- Let's press `CTRL` + `SHIFT` + `E` and look for the `mscorlib` and `System` libs
- Let's create a few nodes : `Is64BitOperatingSystem`, `Is64BitsProcess`, `MachineName`, and so forth
- We can have documentation about those on the MSDN website

The NuGet ecosystem

- Think of nuget as a database for .NET libraries (similar to JS' `npm` or python's `pip`)
- Those nugets come in `.dll` format, and can be used with any language of the .NET framework. Like C# or Visual Basic, VL is a language for the .NET framework.

Hands on :

- Open the Nuget Gallery
- Browse, show a nuget page
- Observe the "structure" of the nuget page : versions, project site, source repository, etc.

Small demo : String.Extensions

- Let's install StringExtensionsLibrary, a convenience nuget that contains many functions that allow us to do string manipulations
- We read the Github descriptions, and see many usefull functions that could be usefull in our patches
- We install the nuget, reference it in our patch, and new category shows up in the node browser
- Let's look at the node browser and the Github page side by side : that's a match!

- Let's use a cool node and see it works instantly in our dataflow approach

2. Tailoring a nuget for VL usage

Coming from the textual programming world, nugets might not always fit our patching habits out of the box. In this section, we'll se how we can overcome those problems and wrap .NET libs to make them patch-friendly.

Incompatible types

.NET libraries might return types that are not convenient for us to use in VL. Sometimes, they're not even compatible, even though they have the same name. Let's see how we can overcome that.

Note that The Gray Book has info about this.

Hands on : GeometryTools

- Open the example patch : we want to color a line when the circle intersects it.
- After some searching, we've found this nuget, called `GeometryTools`.
- The `GeometryTools` nuget has a `IsPointOnLineSegment` operation, but its input are not compatible with our Vector2 IOBoxes
- By looking at the lib's source code, we learn that the library uses `System.Numerics.Vectors` to express Vector2 values. We need to convert those from and to the Vector2 that VL knows : let's create wrapper nodes that takes care of the type conversion

Hands on : ColorThief

- Open the example patch : we want to retrieve a color palette from an image.
- After looking around on the net, we find the ColorThief library that seems to do what we want.
- Let's look at the code example provided on the repo
- We install it, find the `GetPalette` node, but its inputs are not that VL friendly...
    - The node turns pink : it's a warning. Here, the node expects a `ColorThief` instance to operate on, but we did not provide one. Hence the node complaining!
- We create a wrapper node that first takes care of creating this instance for us, and makes it more suited for our patching life
- Let's expose the color count and give it a default value
- Note that nodes must always return a Spread

Events

There are many ways in C# to express the concept of events. In VL, the prefered paradigm is Observables. If a library uses the .NET Core Event Pattern, VL automatically converts it to an Observable. There are other cases though where we'll need to adapt the nodes so we can cosume them.

Hands on : DuckDuckGo Instant Answers

- We want to use DuckDuckGo's Instant Answer feature in our patch. After digging, we find the DuckSharp nuget that gracefully wraps this API in C#
- We install the nuget and reference it in our doc
- The lib's README on Github shows a basic usage, let's see how we can patch it

Unmanaged dependencies

Some .NET libraries might in turn use libraries themselves. Sometimes, those libraries are not written in C# but rather in an unmanaged language such as C++. Those are called "native dependencies". For quite some time, there was no clear recommandation for nuget packages as where to put those native dependencies : anyone could come up with their own folder structure. As a result, gamma won't be able to pick those up automatically : we have to explicitely tell it where to look for those dlls.

For more informations on this topic, head other to the Gray Book.

3. Create a wrapper lib

We now know how to use existing nugets and tailors them for further usage in a VL patch. But how could we build re-usable blocks that are not tied to the project we're working on? Here we'll see how we can create our own VL library.

Hands on : AetherPhysics

- We'll start by downloading the `Aether.Physics2D` nuget
- Then, we'll create a new document named `VL.2d.Physics` : just by referencing this VL doc in our project, we'll have access to all the features of the physics library
- Now we'll need to forward Classes, it means that our `VL.2d.Physics` document will then expose those classes to the other documents that are referencing it
- We'll need to adapt a few things : create conversion operation for Vectors, change `MutableList<T>` to `Spread<T>`, etc
- Now, let's create a document that has a reference to our `VL.2d.Physics` document : see how the forwarded class appear in the node browser
- Let's write some documentation for the utils we've created

4. In a nutshell

If you must remember something from this workshop, it must be those points:

- Whenever possible, go to the Github page of the nuget you're trying to use. You might find code examples and/or test files that can show you how the library is meant to be used. If you're planning to wrap a nuget to create a fully fledged library, it's better if the code is properly maintained (does the repo owner reply to issues? has the code been recently updated?)
- If a node returns a type you don't know, simply type it in the node browser and see what nodes the library has to offer. Most of the time you can deduce the innards of lib by looking at its nodes
- If a node expects an input that and you don't know what to feed it, go backwards : start from the input of the node and explore the node browser until you can connect to the data you already have. See for that example how we managed to retrieve the `Radius` out of a `Body` in our `2D.Physics` wrapper : the `Radius` node takes a `Shape` as input. But what is a `Shape`? If we type it in the node browser, we get a `Shape` node, cool, but what do we do with it? It takes a `Fixture` as input. Ok, then we do the same by typing `Fixture` in the node browser, and so forth until we find our `Body` again!