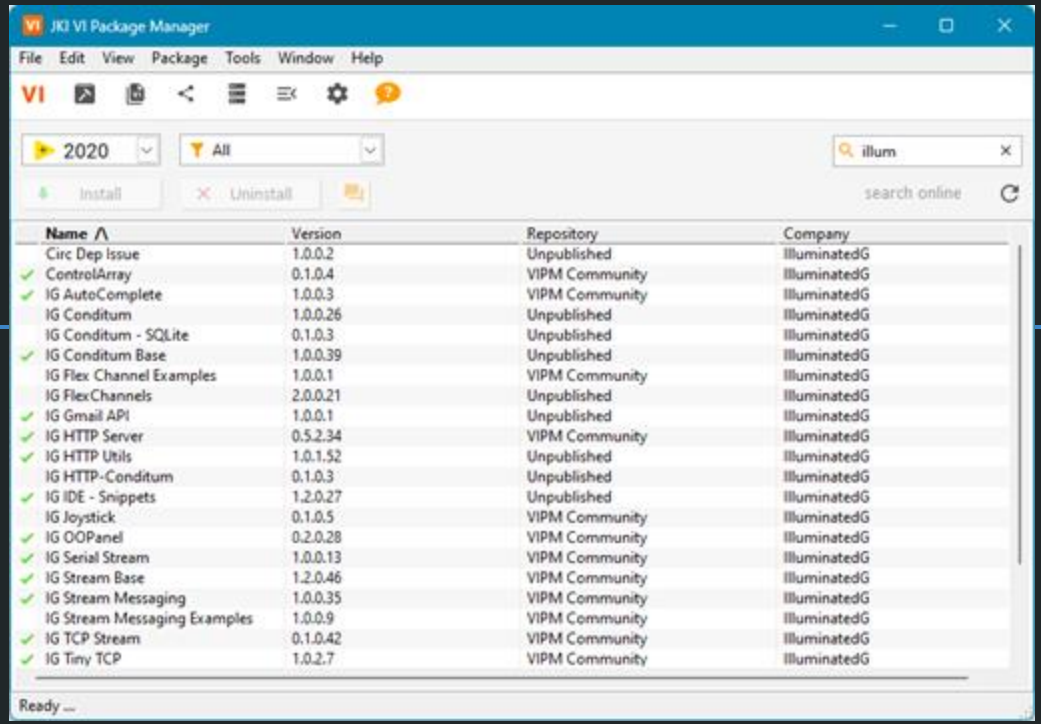
Putting a bow on your reusable code by packaging with VIPM

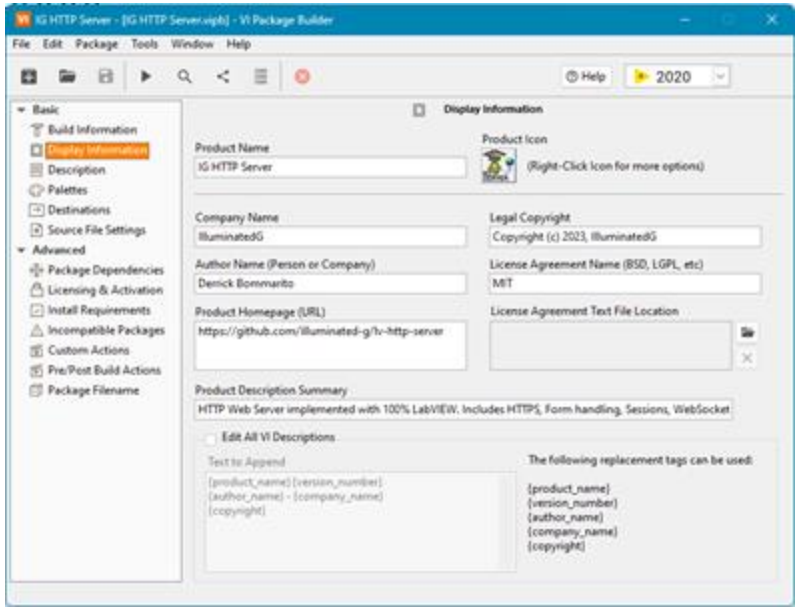
NI Connect 2023

Derrick Bommarito

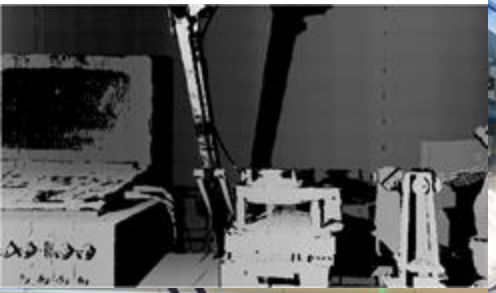


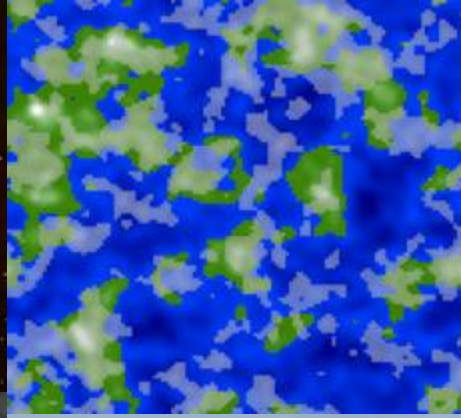
What shall we talk about?

- Me
- Shameless Plugs
- Why package reusable code?
- vipm.io
- Comparison with NI Package Manager
- The Simple™ Case
- To split or not to split packages
- Package Polish
- Extra tips



Me!



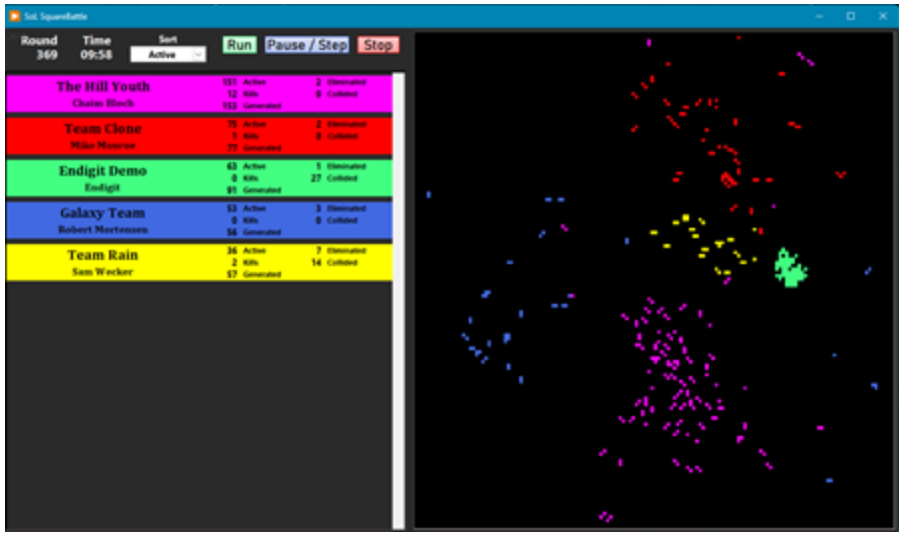


Summer of LabVIEW

Programming competitions and challenges to grow skill and excitement for LabVIEW development.

Being run publicly in partnership with G Central.

SquareBattle project template available on VIPM.



https://discord.gg/KxtUER4NGS
SoL Discord Server

15 recertification points for submitting a working entry!

BLUE ORIGIN

FOR THE BENEFIT OF **EARTH**

We want YOU!

Many positions across all skill levels. (Not just LabVIEW)

DSCE, the team I'm on, has lots of work to do!

Help make a LOT of cool stuff possible.

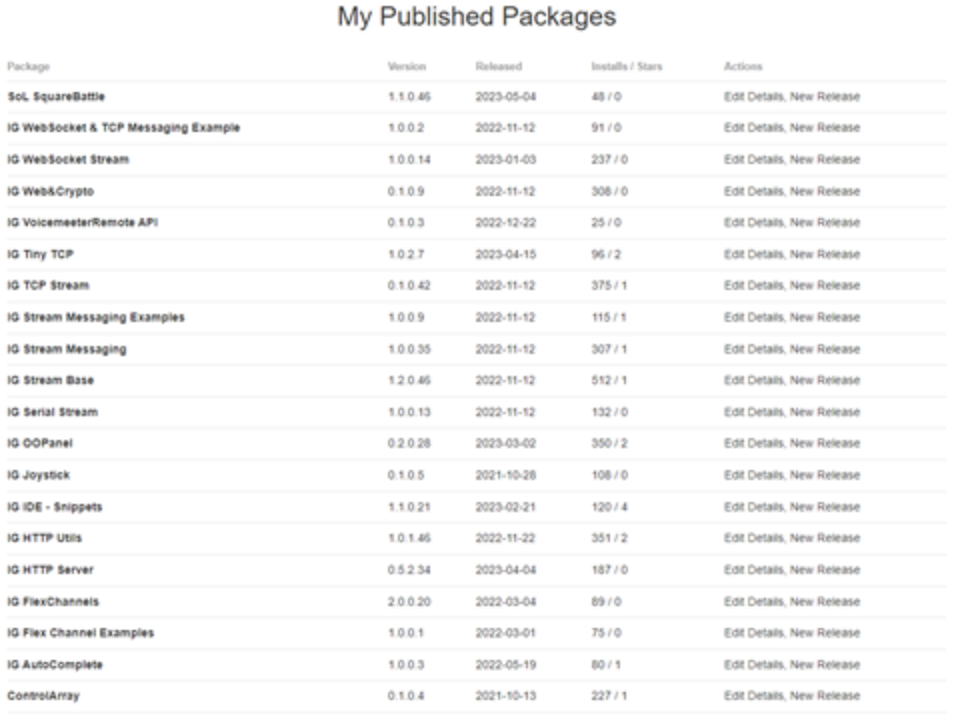
https://blueorigin.wd5.myworkdayjobs.com/BlueOrigin?q=labview

Disclaimer!

I am not affiliated with JKI or VIPM, I just have a bunch of packages posted, some that get used by others.

I'm just a LabVIEW dev here to encourage more open-source LabVIEW development and try to rise the tide for all ships.

Packaging of development dependencies is also just very convenient for a host of reasons.



Why create packages?

Pros

- Easy to distribute
- Easy to version
- Easy to consume
- Vipm.io packages are open source
  - Inner-source (private repos) available with VIPM Pro
- VI Package Configurations

Cons

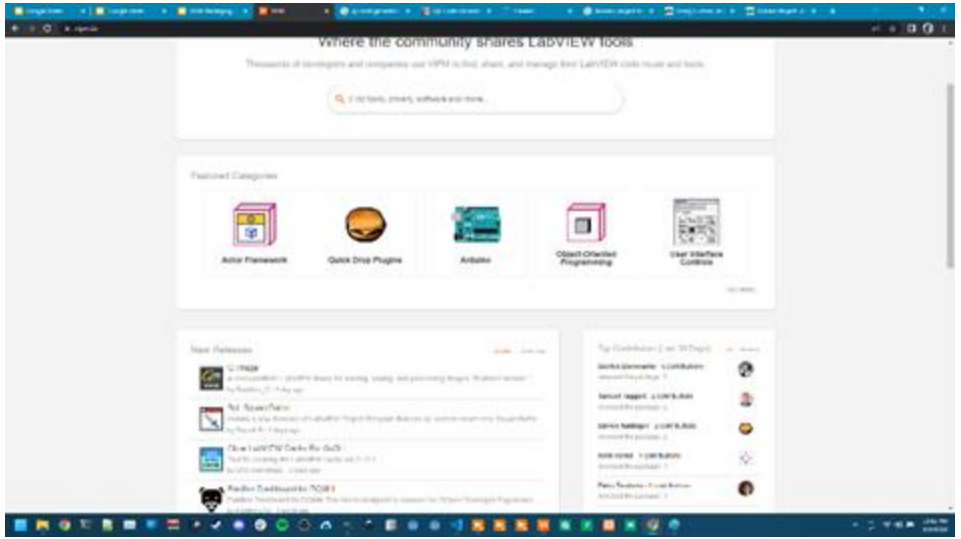
- Extra steps
- Not easy to switch back (Not that you'd want to)

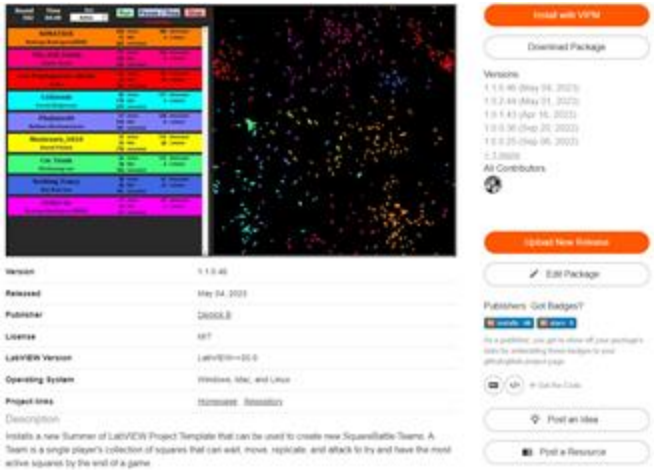
https://vipm.io

More than just package search:

- Links to source code repos
- Ability to post ideas to authors*
- Authors & others can post pages about packages
- Version history and notes

* My opinion is that this is best accomplished with issues on the source code repo.





What about NI Package Manager?

Pros

- Packages can depend on products and toolkits besides just LabVIEW
- Useful for built applications
- Better remote management (SystemLink)
- Complex dependencies
- LabVIEW Package Build spec
- Installers

Cons

- Unable to target multiple versions of products

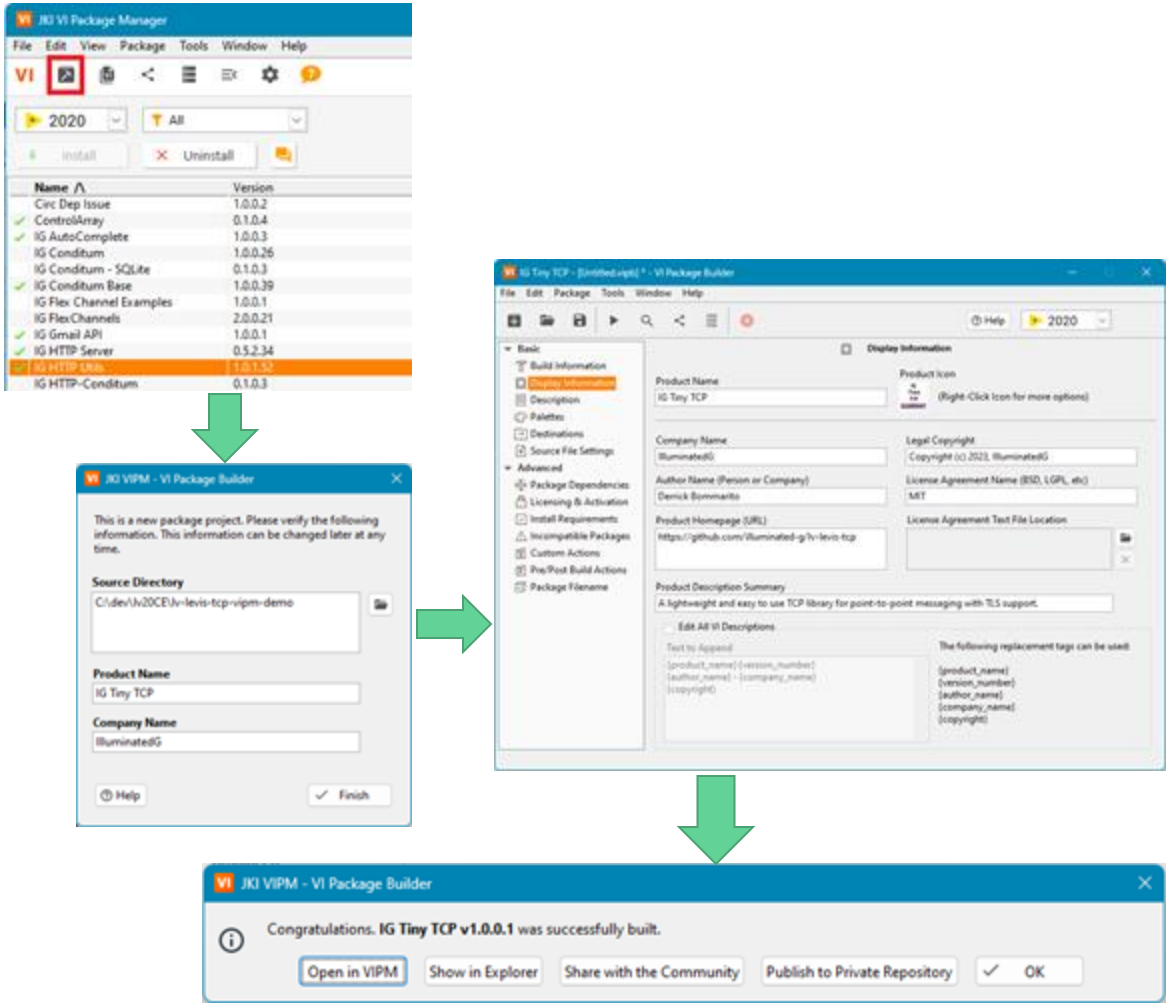
The Simple™ Case

Creating a New Package

1. Select source folder
2. Fill out author information
3. Fill out package description
4. Build!

You now have a .vip file you can install with VIPM and share, congratulations!

We'll talk about everything else you should do for a package shortly.



The Not-So-Simple Cases

To split or not to split?

Should you split layers of a set of libraries into multiple packages or not?

Splitting Benefits

- Smaller downloads
- Faster CI/CD
- Less palette clutter
- Flexibility in fixes

Splitting Cons

- Increased development effort
  - Workflow equivalent of using PPLs
- CI/CD for LabVIEW still primarily driven by private companies that treat it as their IP
- Potential versioning hell

Not-splitting Pros

- Easier Development
- They're typically just dev libraries so extra download doesn't impact builds

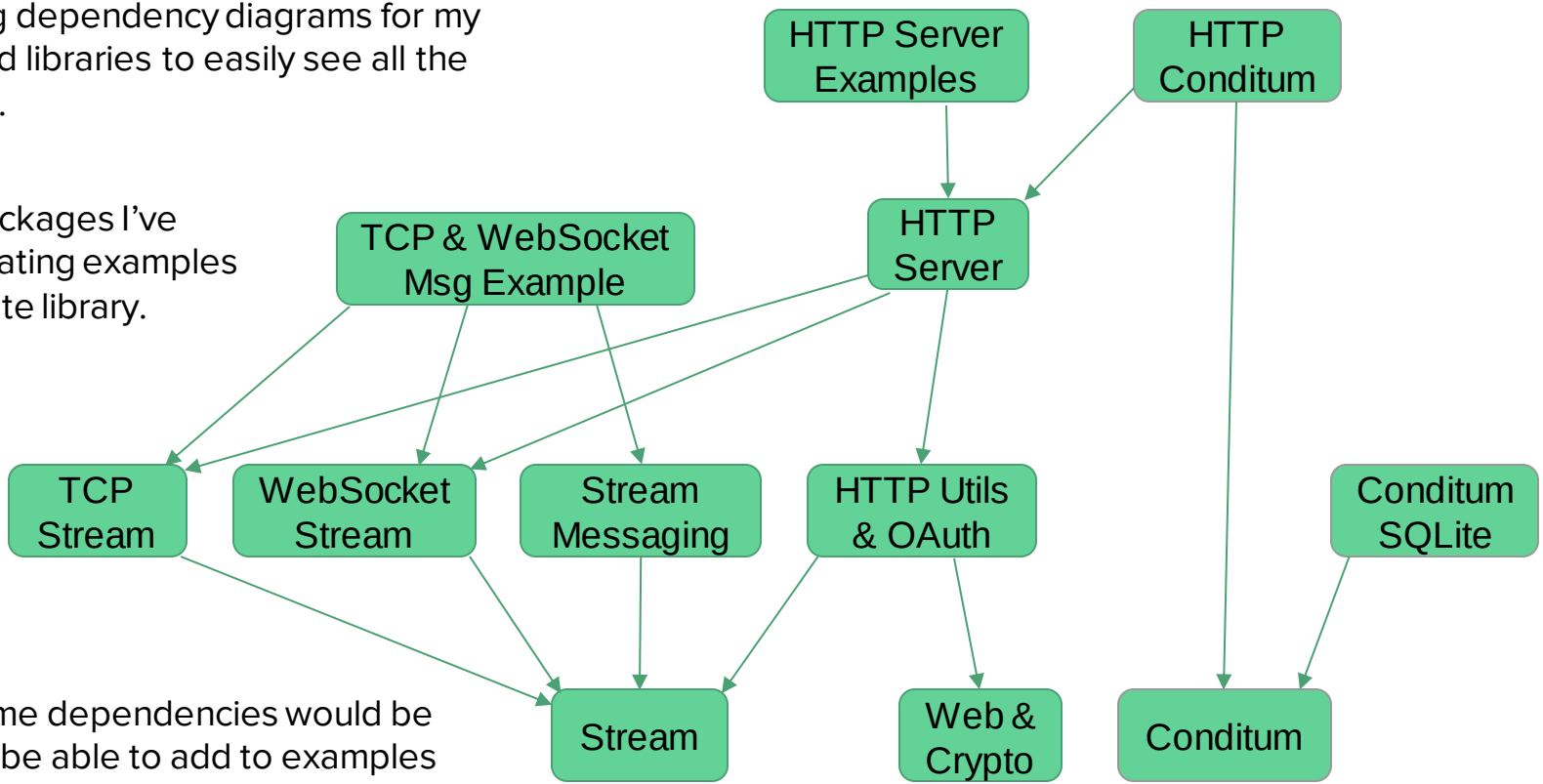
Not-splitting Cons

- Less likely to be designed to be easily extendable

My Web Packages

I like creating dependency diagrams for my packages and libraries to easily see all the relationships.

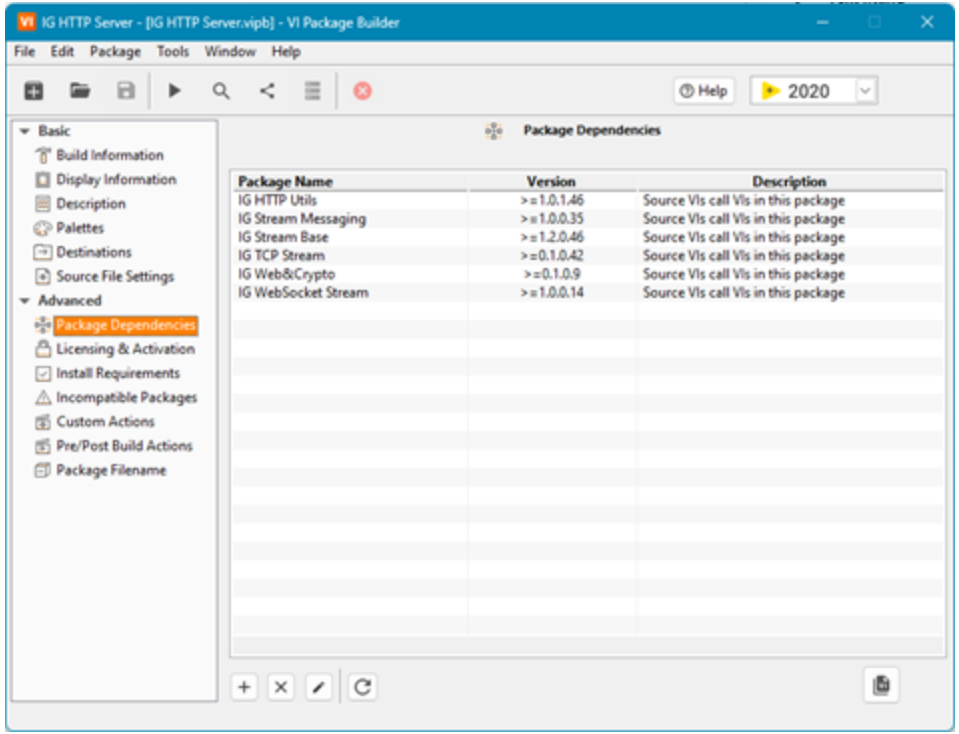
For larger packages I've begun separating examples into a separate library.



I do this anytime dependencies would be affected or to be able to add to examples without bumping main library versioning.

Splitting up packages

- Follow the Simple Case steps for the lowest level package(s).
- [Re]move packaged source from disk.
- Open next package.
- Ensure everything relinks to vi.lib properly
  - *Should* happen automatically, LabVIEW searches vi.lib for missing dependencies by default.
- Package, specify package dependencies.
- Install.
- Repeat.



Polishing things up

Large Classes and APIs

I've heard that a lot of people rely on the automatic class palette.

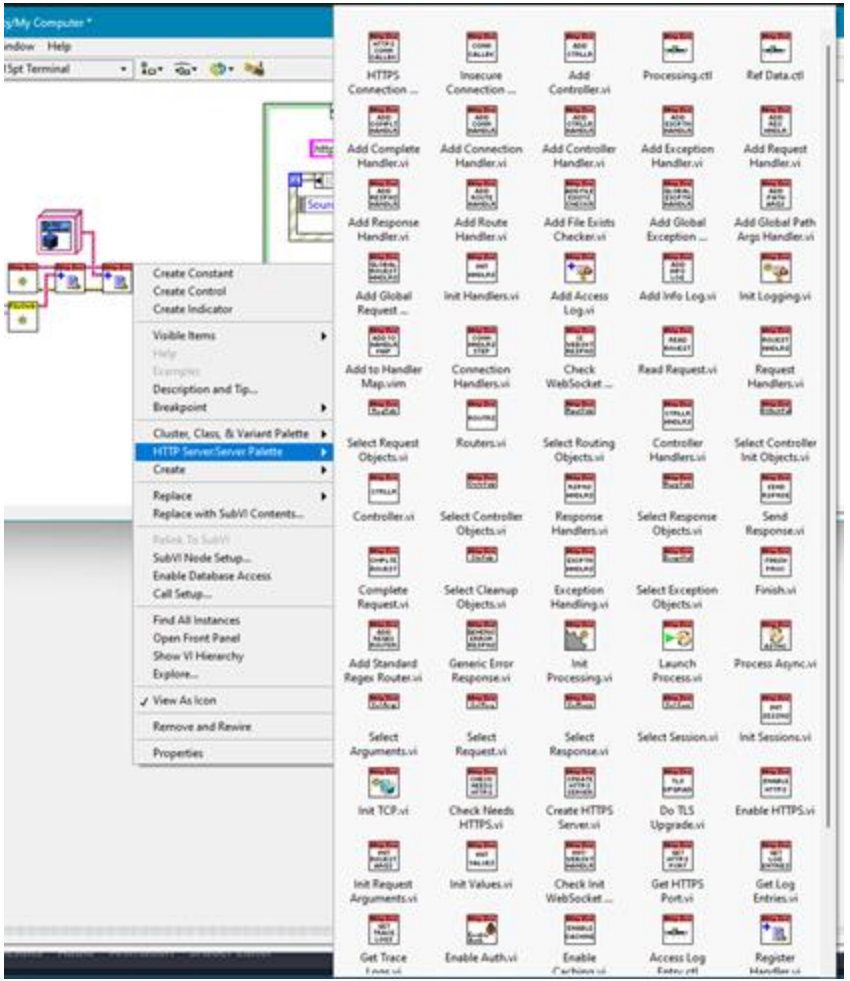
I personally can't stand it.

No scope control (most of this isn't accessible outside the class).

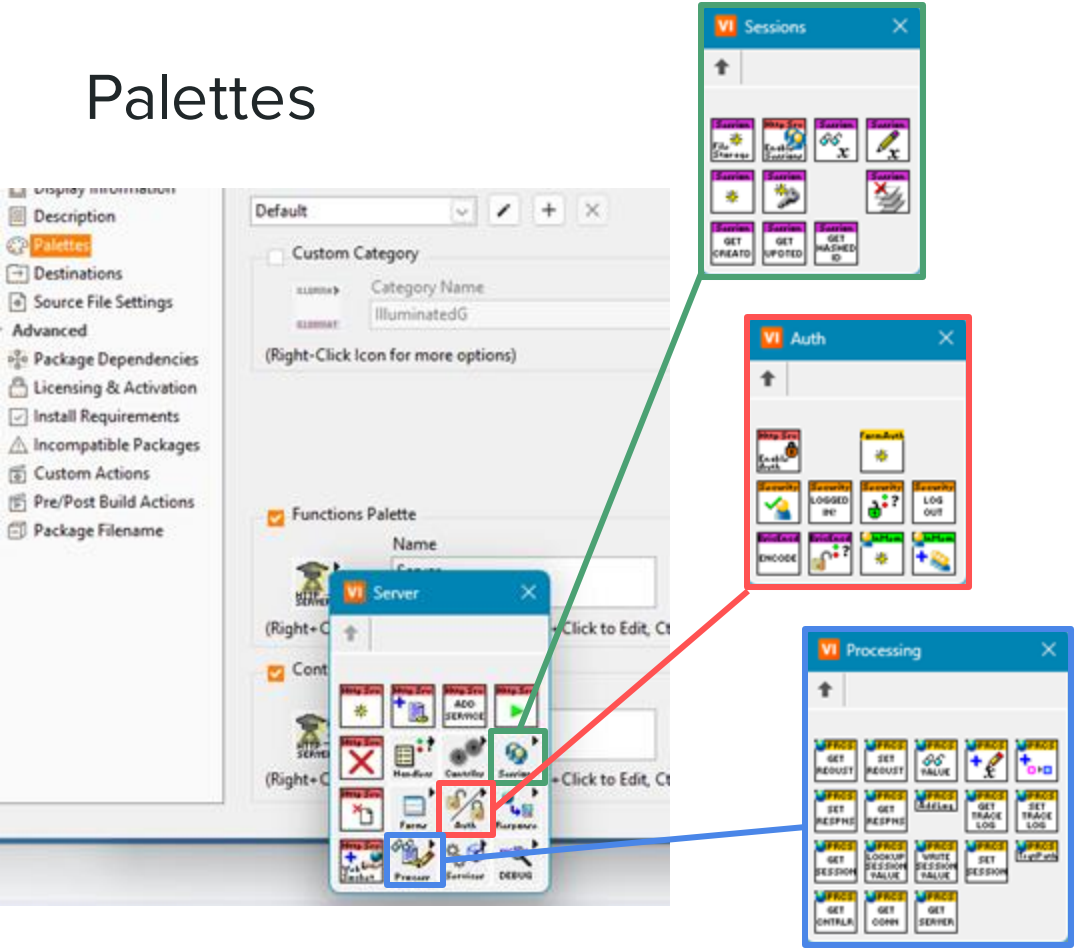
No organization.

A lot of footprint taken by optional features.

Save anyone using your package some sanity!



Palettes



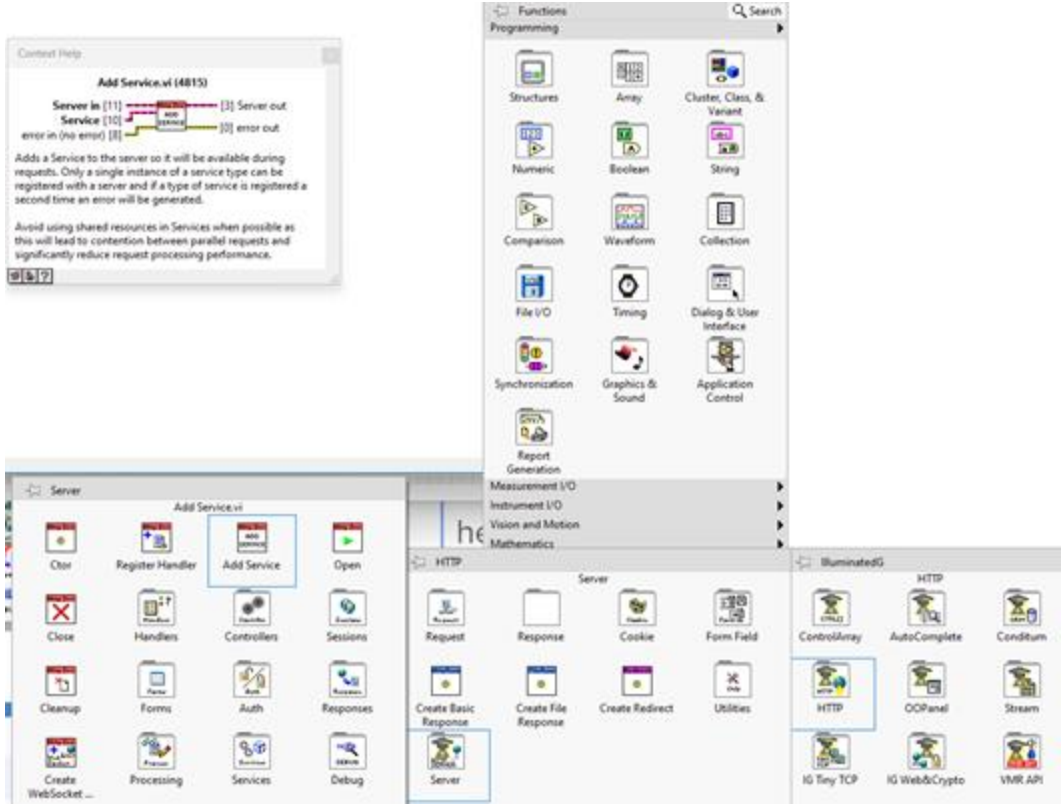
Organize your code and features based on related functionality.

Palettes are generally organized with initialization in top-left, cleanup/close towards top-right.

Good VI names and context help are crucial for quick browsing of palettes.

Aiding Palette Browsing

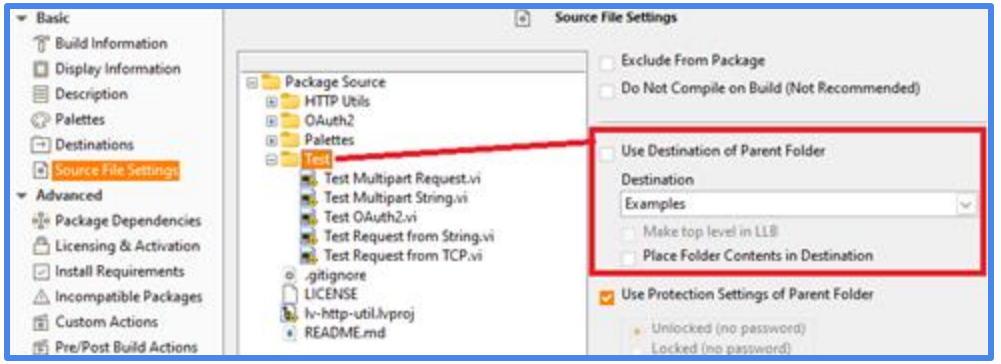
Good VI names and VI/Ctl documentation are crucial for browsing the palettes.

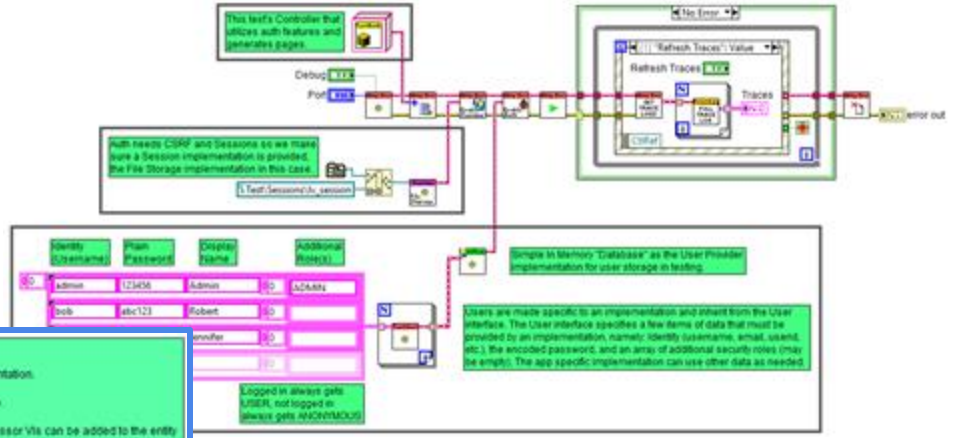


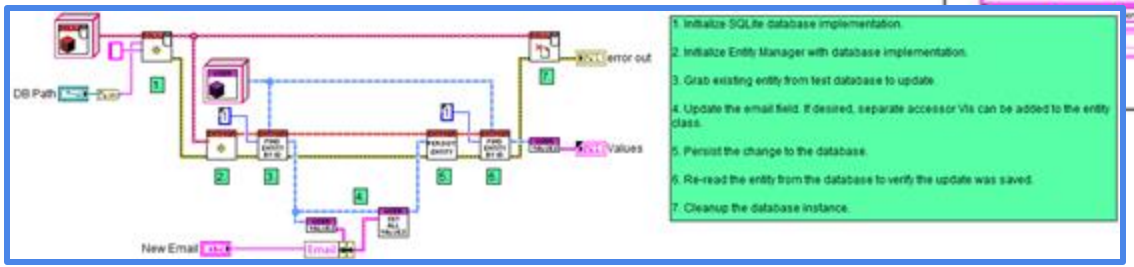
Examples

I call them “tests” during development and my style of tests also serve as usage examples. Of course you can have dedicated examples as well.

Throw down some comments in your tests and set them to be installed to the examples folder.







Project Templates

When they're applicable, they're a great way for getting new users ramped up on your tool / framework.

Read the KB for getting started, reference existing templates in LabVIEW.

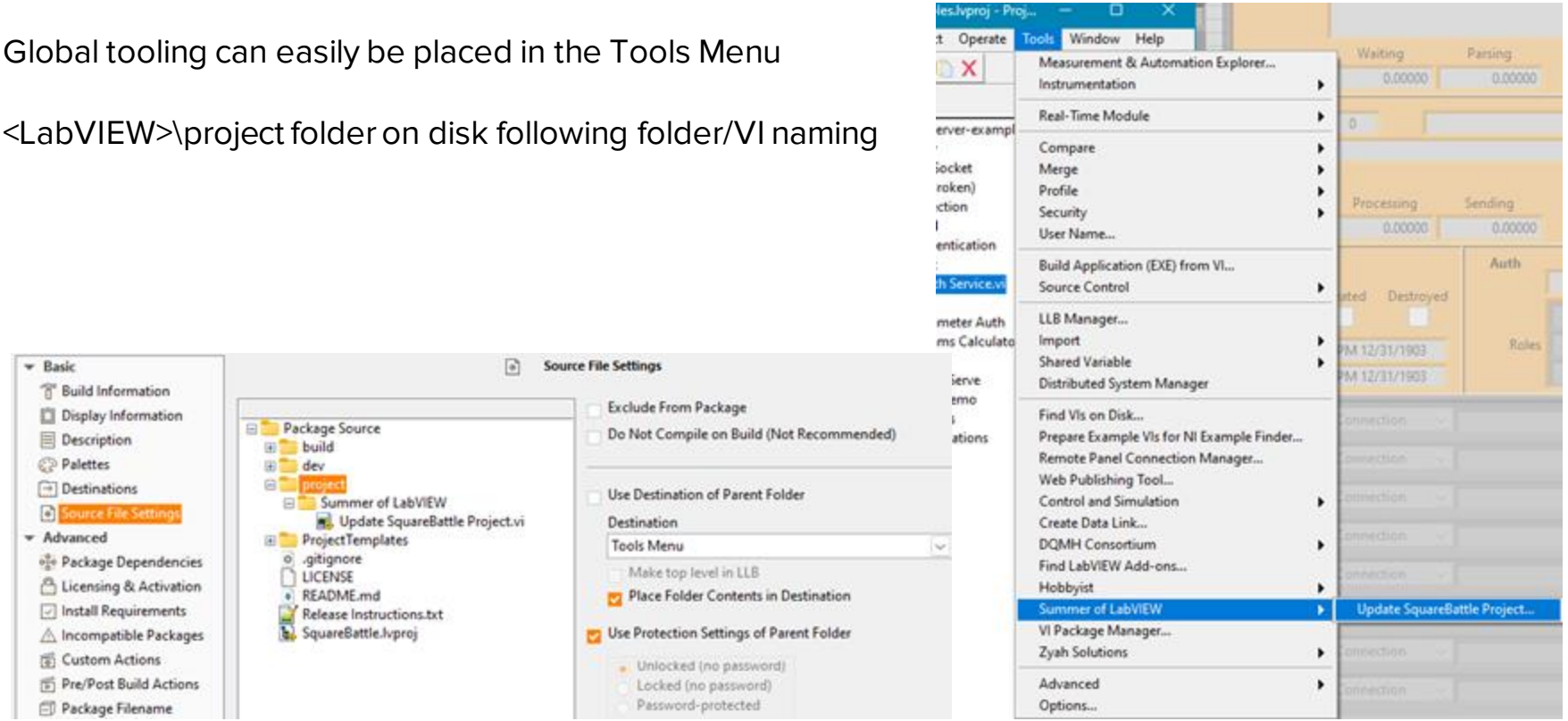
Specify a folder to put in the Project Templates Destination in VIPB.

KB for getting started making custom Project Templates

Tools Menu

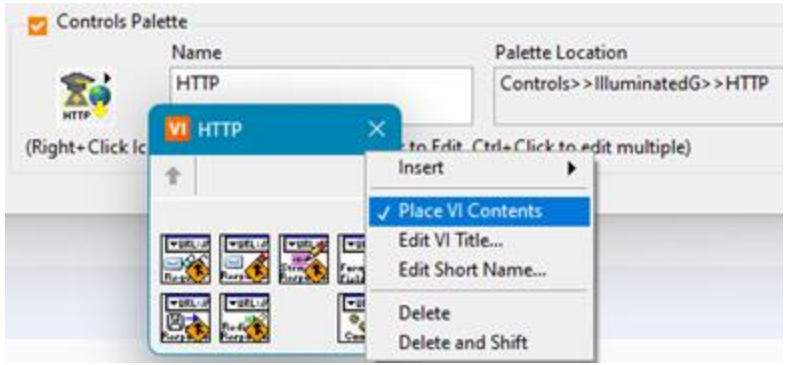
Global tooling can easily be placed in the Tools Menu

<LabVIEW>\project folder on disk following folder/VI naming

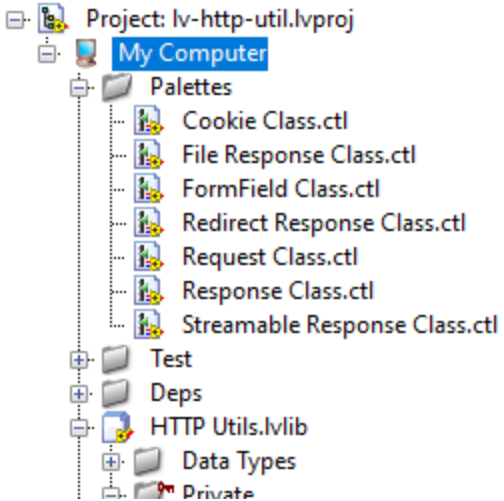


Extra Tips

“Place-contents” Palette Items



You can right-click a palette menu entry in VIPM and enable the option to “Place VI Contents”. This is useful for code-snippets and placing class controls on VIs.



I’ll often create a Palettes folder in the project specifically for these.

Custom install actions & replacing shipped NI code

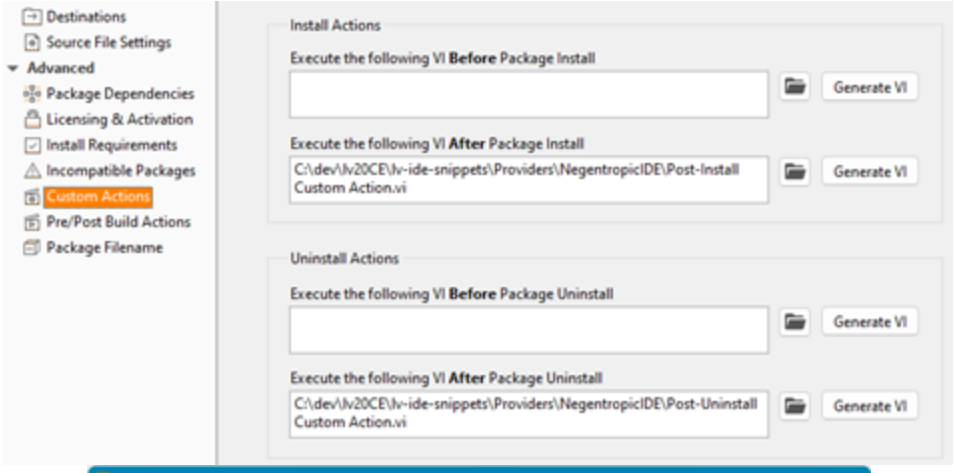
You should never have VIPM directly overwrite VIs that ship with LabVIEW. VIPM will not replace the old file when uninstalling the package.

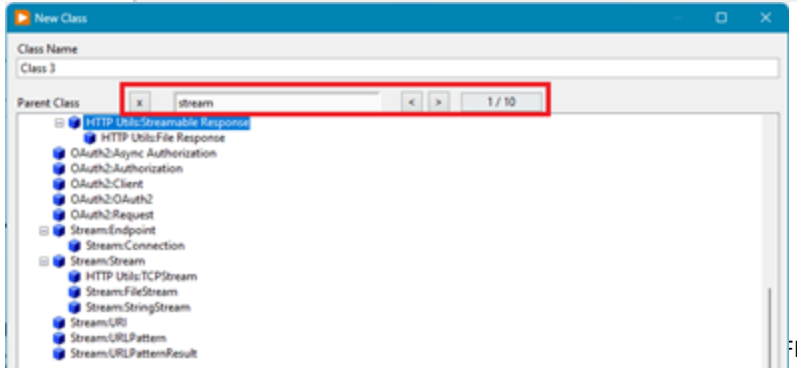
Install to a “standard” VIP destination and use post-install scripting to backup the shipped VI, move the new VI into place.

Use post-uninstall scripting to restore the backed up original VI after package is uninstalled.

E.g. A package that replaces class member VI template and custom class override VI scripting to get everyone using the same standard templates without the full-VI error case.

Or new class parent search.

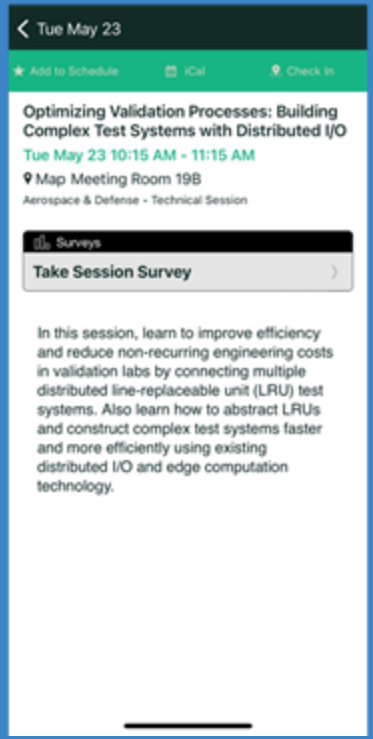




Questions?

Give us your feedback!
Quick 2 Question Survey

In the mobile app, click into the session you would like to provide feedback for



Click "Take the Session Survey"