Programming in

LabVIEW™

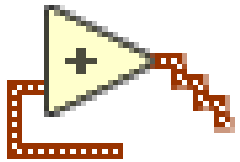
faster

Nick Murray

G DEV CON ANZ

About Me

- BE (Hons) Electrical & Electronic
- Post Grad Cert in Connected Environments IoT.
- Started Programming in LabVIEW in version 6i.
- Experienced in Product Test Engineering / TestStand
- Worked for CPE Systems for 3 years
- CLD



- There is LabVIEW running on the Navy Boats

Acknowledgments / Credit

- Darren Nattinger (The Quick Drop Guy) https://www.dnatt.org/home

  I Find Your Lack of LabVIEW Programming Speed Disturbing

  An End to Brainless LabVIEW Programming

  Become the World's 3rd Best LabVIEW Presenter

  Darren's LabVIEW nuggets are also a good learning resource on the forums.
  https://forums.ni.com/t5/Community-Documents/Darren-s-Nuggets/ta-p/3524320

- Alejandro

#OurGiantsAreFemale - Beatrice Tinsley

1941 -1981

British-born New Zealand astronomer and cosmologist, and the first female professor of astronomy at Yale University,

whose research made fundamental contributions to the astronomical understanding of how galaxies evolve, grow and die.

Photo Credit

Agenda

Various tips on how to program in LabVIEW faster…

- Philosophical ones first
- Then more practical examples
- The future of faster programming.

This presentation will NOT be:

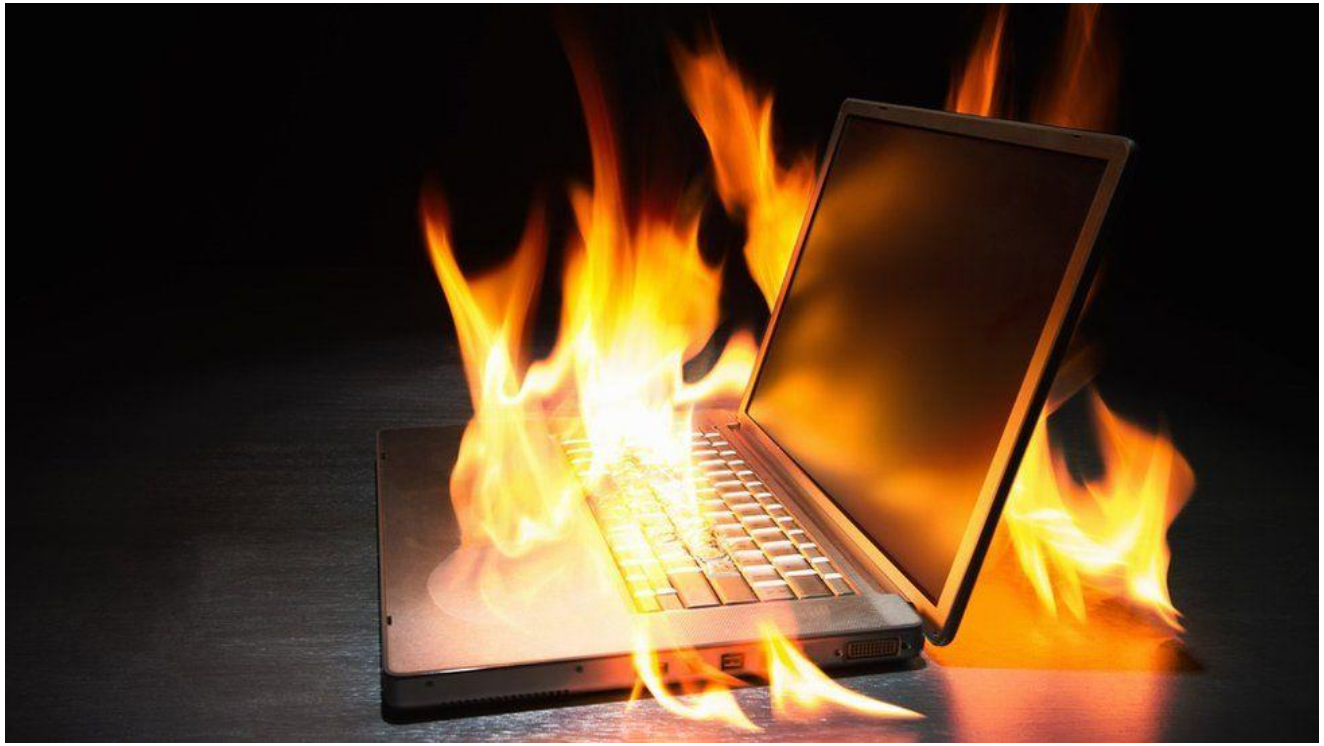
How to improve the runtime performance of your code
I'm showing you how to write code faster (not write faster code)

- There is another session on faster performant code

Don’t Lose Code You Have Written

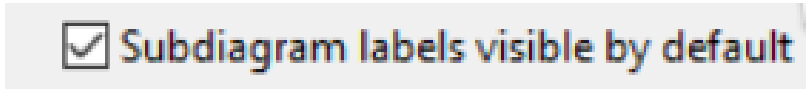
- Back up
- Source Code Control
- Be careful to avoid cross linking when copying code
  - Check VI paths are as expected in Project.

Yes – just last week my test system blew up my Laptop!

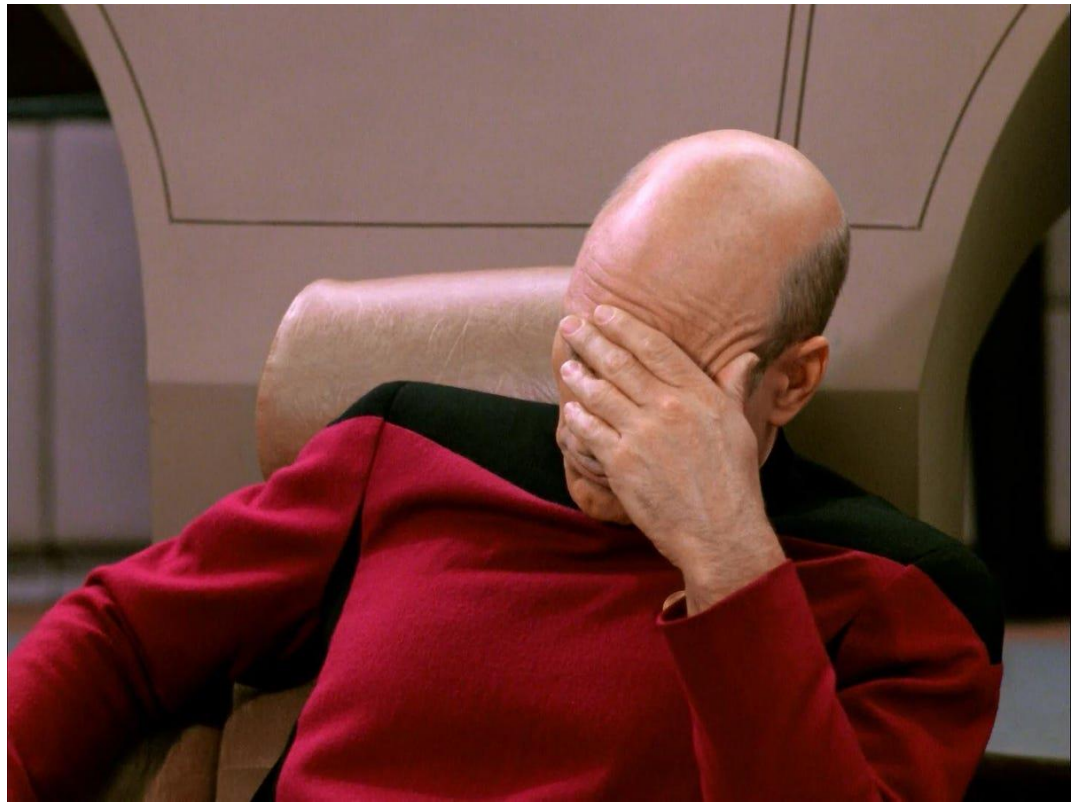


Love the Code You Wrote

- What was I thinking when I wrote that code a few years ago?
- That code would so much better using that new feature such as sets…
- Comments – Comment your structures!



- If it worked and has been debugged already – just reuse it! Of course, it could be better – good enough is quicker.

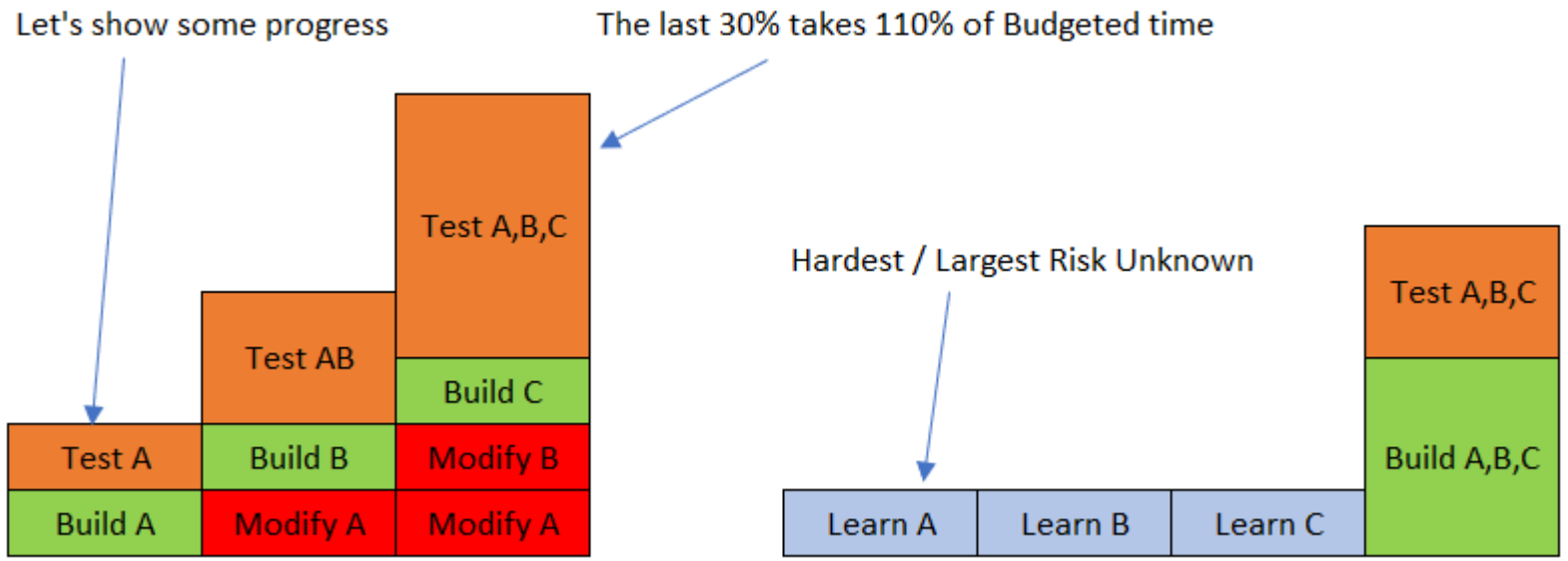


Project Planning

Spend some time upfront to plan the best way forward.

Rapid Learning Cycles: (**Katherine Radeka**)

- Close Knowledge Gaps Quickly
- Aims to reduce rework

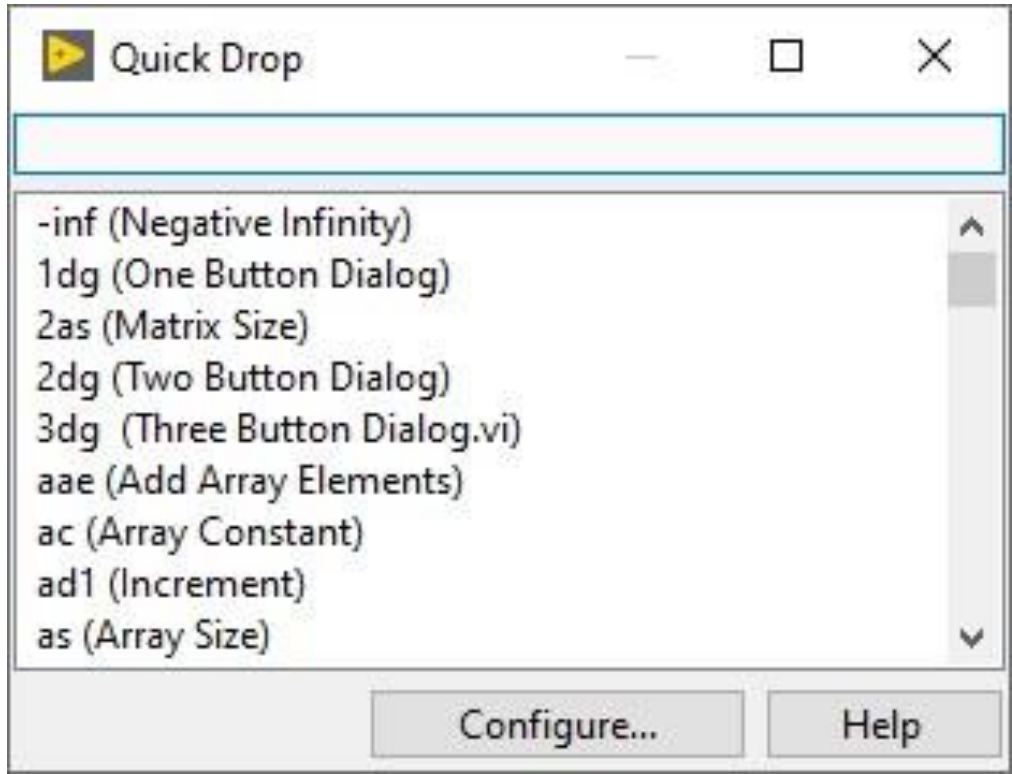


Use Quick Drop

- Its much faster if you learn the short codes
- Faster to get to deeply nested VIs than browsing though the pallets.
- Returns useful VIs that you didn't know existed.

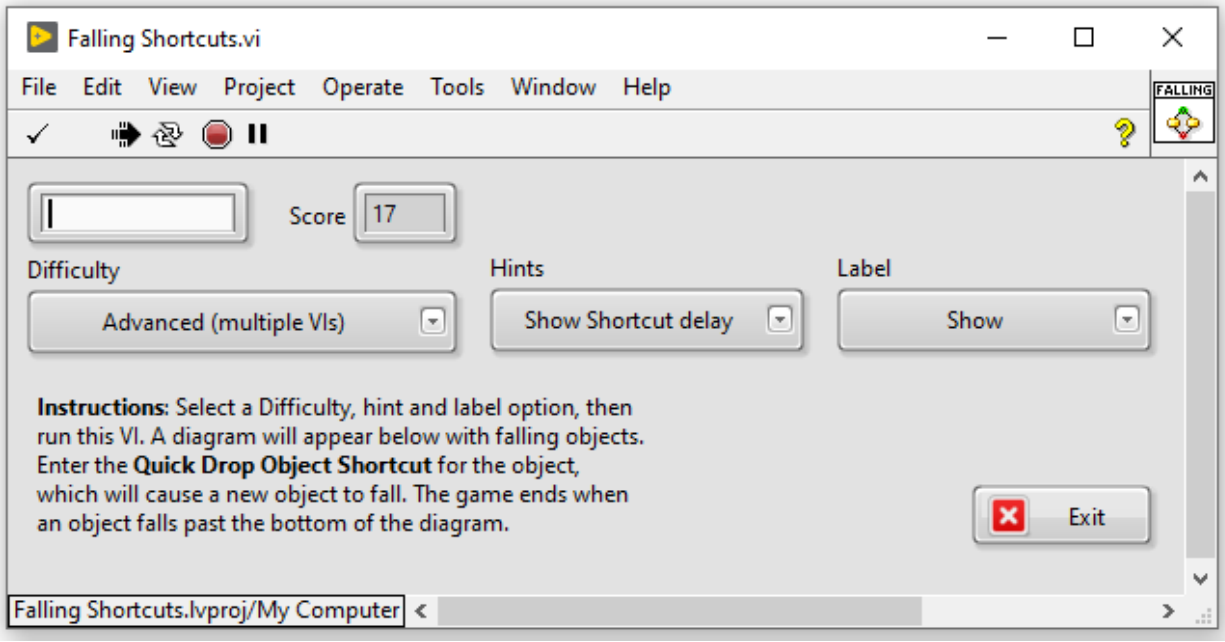
Average time to drop an object from the palette:
2.00 seconds
Average time to drop an object with **Quick Drop**:
0.50 seconds
For an average drop transaction, Quick Drop is
4 times faster than the palettes.

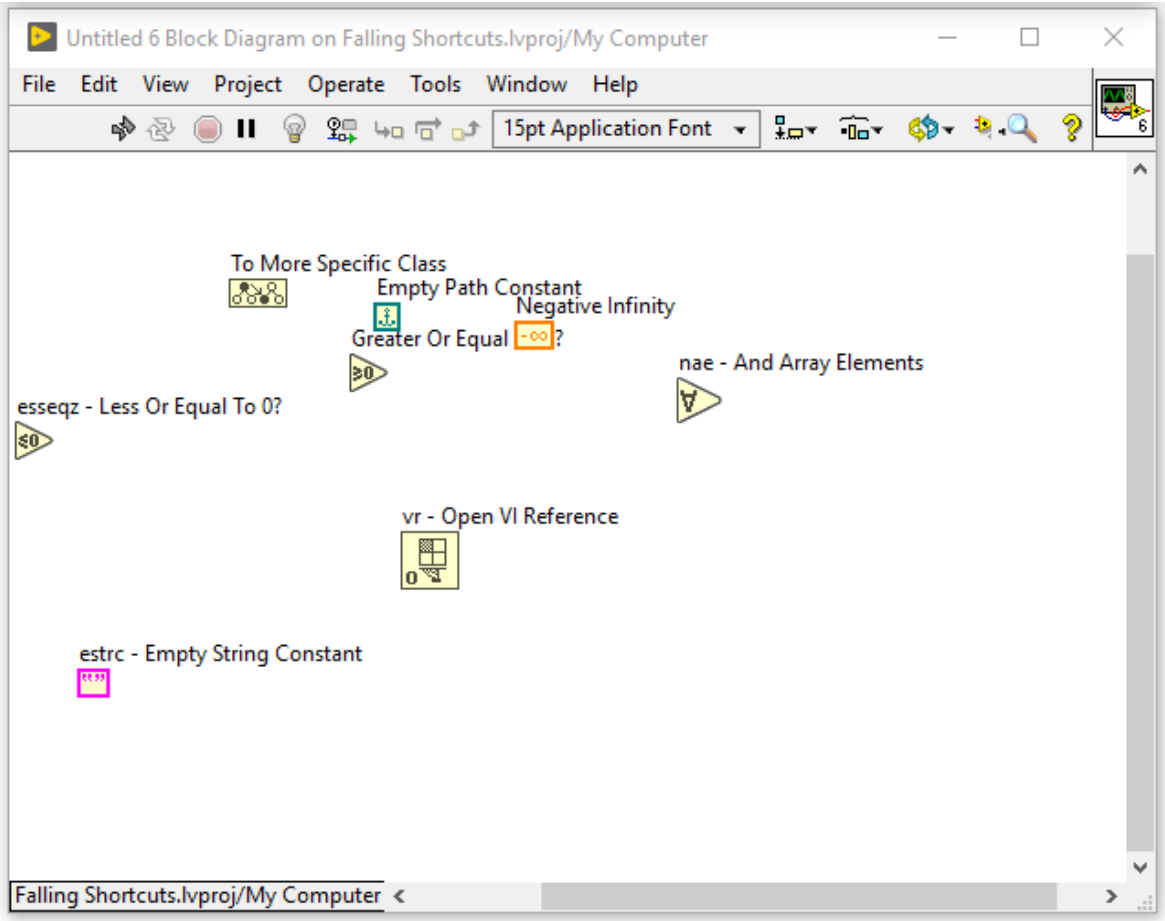
But how do I learn the shortcuts?



Falling Shortcuts Game

https://forums.ni.com/t5/Quick-Drop-Enthusiasts/Falling-Shortcuts/td-p/3537204





Quick Drop Shortcuts

- QD object shortcuts – Installed by default in LabVIEW 2016 and later
- Press Ctrl-Space, then...
  - Ctrl-B – Set property/invoke node class
  - Ctrl-Shift-B – Set property/invoke node property/method
    - Including dotted properties in LabVIEW 2019 and later
  - Ctrl-W – wire multiple objects together
    - Ctrl-Shift-W will also clean up the wired code
  - Ctrl-I – insert an object on one or more wires
    - Ctrl-Shift-I – smart insert a single object on multiple wires
  - Ctrl-R – remove selected object(s) and rewire pass-through types
  - Ctrl-P – replace one or more selected objects
  - Ctrl-D – create controls/indicators (type ‘req’ for creation of required inputs only)
  - Ctrl-F – Frame block diagram window on primary monitor (LV 2020 and later)

Learn the Short Cuts

- Ctrl – U for Diagram & Front Panel Cleanup
- Control Click to swap inputs / connector pane terminals
- Printout Shortcut Posters

LabVIEW
Keyboard Shortcuts

Objects and Movement

Shift-click	Selects multiple objects; adds an object to the current selection.
↑↓→← (arrow keys)	Moves the selected objects one pixel at a time.
Shift-↑↓→←	Moves the selected objects several pixels at a time.
Shift-click (drag)	Moves the selected objects in one axis.
Ctrl-click (drag)	Duplicates the selected objects.
Ctrl-Shift-click (drag)	Duplicates the selected objects and moves them in one axis.
Shift-resize	Resizes the selected object while maintaining aspect ratio.
Ctrl-resize	Resizes the selected object while maintaining center point.
Ctrl-Shift-resize	Resizes the selected object while maintaining center point and aspect ratio.
Ctrl-drag a rectangle in open space	Adds more working space to the front panel or block diagram.
Ctrl-A	Selects all front panel or block diagram items.
Ctrl-Shift-A	Performs last alignment operation on objects.
Ctrl-D	Performs last distribution operation on objects.
Double-click open space	Adds a free label to the front panel or block diagram if automatic tool selection is enabled.
Ctrl-mouse wheel	Scrolls through subdiagrams of a Case, Event, or Stacked Sequence structure.
Spacebar (drag)	Disables preset alignment positions when moving object labels or captions.
Ctrl-U	Reroutes all wires and rearranges block diagram objects automatically.

Debugging

Ctrl-↓	Steps into node.
Ctrl-→	Steps over node.
Ctrl-↓	Steps out of node.

Basic Editing

Ctrl-Z	Undoes last action.
Ctrl-Shift-Z	Redoes last action.
Ctrl-X	Cuts selected objects.
Ctrl-C	Copies selected objects.
Ctrl-V	Pastes last cut or copied objects.

Navigating the LabVIEW Environment

Ctrl-E	Displays the block diagram or front panel window.
Ctrl-#	Enables or disables grid alignment. (Mac OS) Press the Command-* keys.
Ctrl-/	Maximizes and restores the window.
Ctrl-T	Tiles front panel and block diagram windows.
Ctrl-F	Finds objects or text.
Ctrl-G	Searches VIs for next instance of object or text.
Ctrl-Shift-G	Searches VIs for previous instance of object or text.
Ctrl-Shift-F	Displays the **Search Results** window.
Ctrl-Tab	Cycles through LabVIEW windows.
Ctrl-Shift-Tab	Cycles through LabVIEW windows in reverse order.
Ctrl-Shift-N	Displays the **Navigation** window.
Ctrl-I	Displays the **VI Properties** dialog box.
Ctrl-L	Displays the **Error list** window.
Ctrl-Shift-E	Displays current VI in **Project Explorer** window.
Ctrl-Y	Displays the **History** window.
Ctrl-Shift-W	Displays the **All Windows** dialog box.
Ctrl-Space	Displays the **Quick Drop** dialog box. (Mac OS) Press the Command-Shift-Space keys.

Navigating the VI Hierarchy Window

Ctrl-D	Redraws the window.
Ctrl-A	Shows all VIs in the window.
Ctrl-click VI	Displays the subVIs and other nodes that make up the VI you select in the window.
Enter †	Finds next node that matches the search string.
Shift-Enter †	Finds previous node that matches the search string.

† After initiating a search by typing in the VI Hierarchy window.

File Operations

Ctrl-N	Opens a new, blank VI.
Ctrl-O	Opens an existing VI.
Ctrl-W	Closes the VI.
Ctrl-S	Saves the VI.
Ctrl-Shift-S	Saves all open files.
Ctrl-P	Prints the window.
Ctrl-Q	Quits LabVIEW.

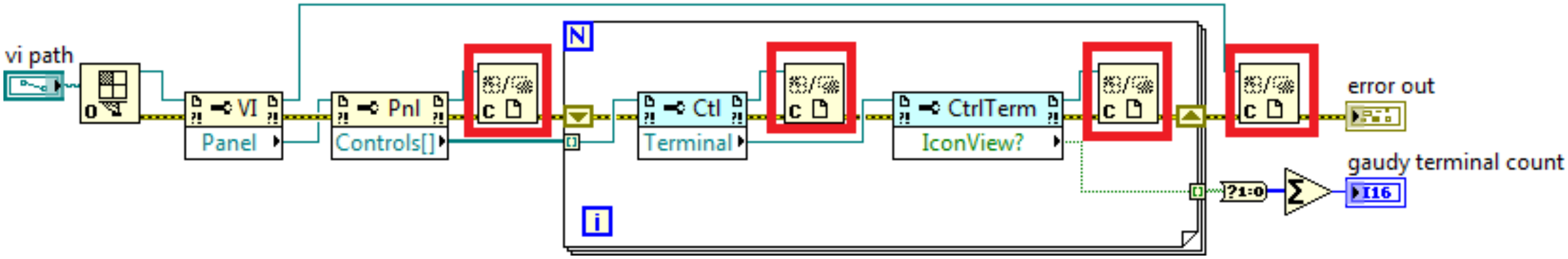
Help

Ctrl-H	Displays the **Context Help** window. (Mac OS) Press the Command-Shift-H keys.
Ctrl-Shift-L	Locks the **Context Help** window.
Ctrl-? or **F1**	Displays the **LabVIEW Help**.

Refer to the **LabVIEW Help** for keyboard shortcut variations on other system locales and keyboard layouts.

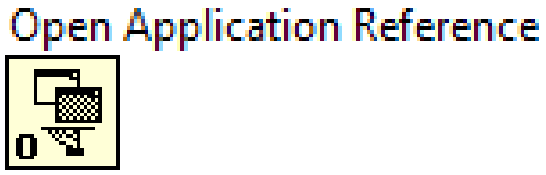
Close All References?

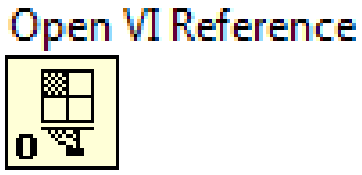
- If you get a reference, close it when you're done.
- It's a commonly accepted best practice.
- Easiest way to avoid memory leaks.

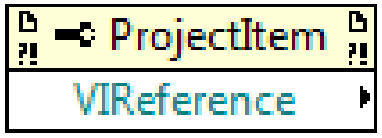


Always Close These References

- ActiveX/.NET/hardware
  - For ActiveX/.NET, you usually need to close the references in order, from child to parent, once you're done with them.
- App/VI that are explicitly opened or obtained from properties/methods.



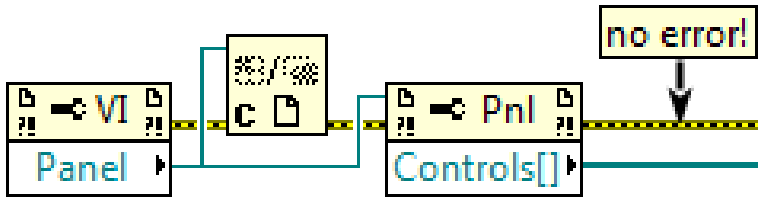




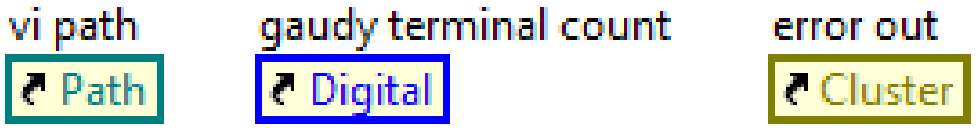
- ProjectItem
- Project

Don’t Worry About These Refs

- *Anything* that inherits from **GObject** (it’s actually a no-op!)

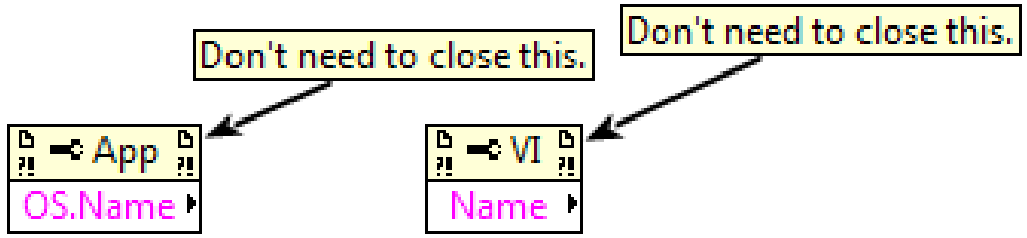


  - Including static control references.

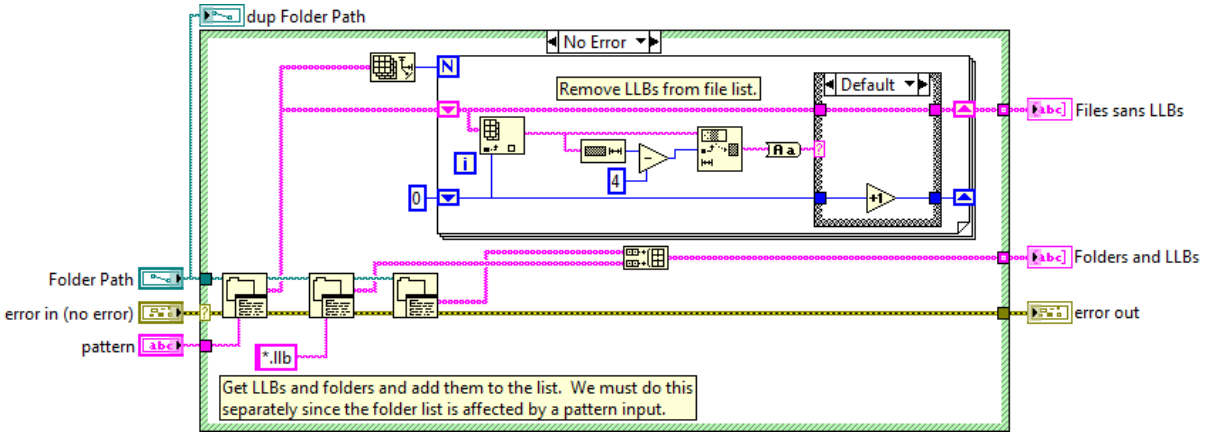


- “This VI” and “This Application” static references.

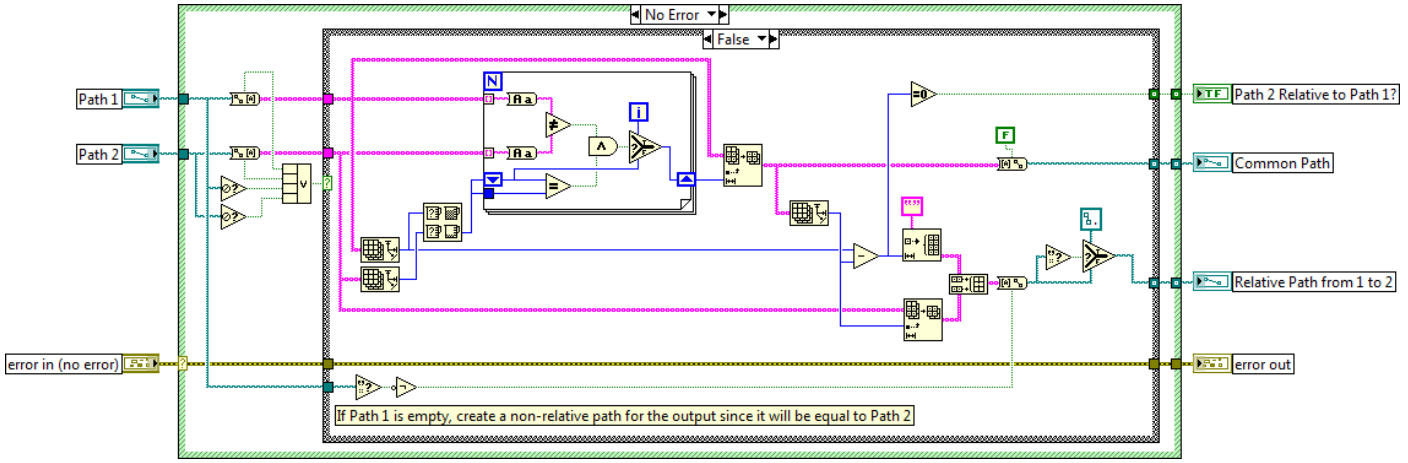
- Outputs of implicit App and
- VI property/invoke nodes.



All VIs Need Error Case Structures



- Ensures subVI code never executes if there is an incoming error.
- Currently the norm in most subVI template VIs, like class accessor VIs.

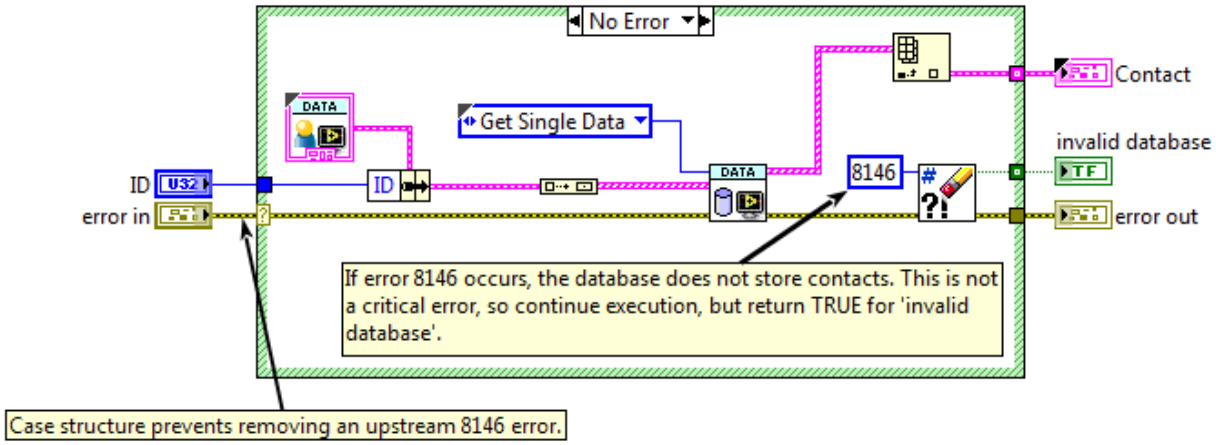


Hardly any VIs need Error Case Structures.

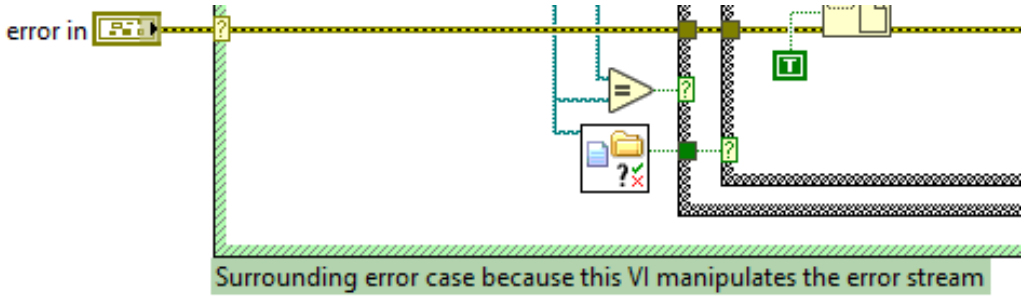
- Any time you can remove a decision point from a diagram, it opens up all sorts of potential compiler optimizations.
- Less diagram noise
- Less time spent constructing diagrams
- It's in the class template because it's easier to remove code than add code. Just because it's in the template doesn't mean you should keep it.
  - Pro-tip: If you do want to remove it, Use Quick Drop - just click the case structure and press **Ctrl-Space, Ctrl-Shift-R** (faster than *right-click > Remove Case Structure)*

Hardly any VIs need error case structures.

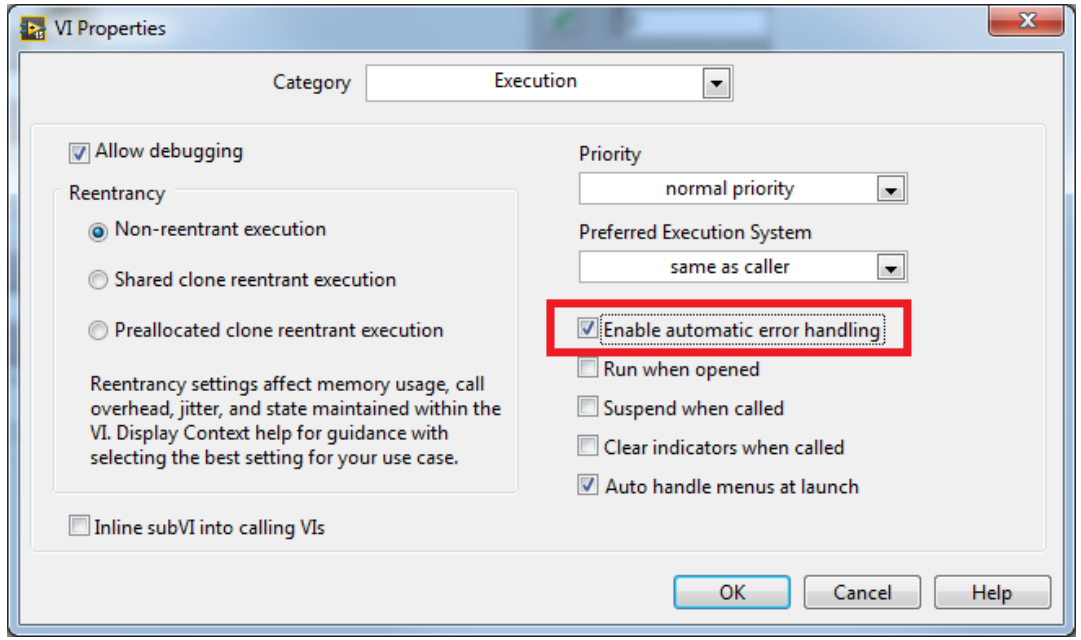
- If your diagram is performing custom error generation or propagation, then use an error case structure around its contents so that code doesn't run (and potentially mess with an incoming error).



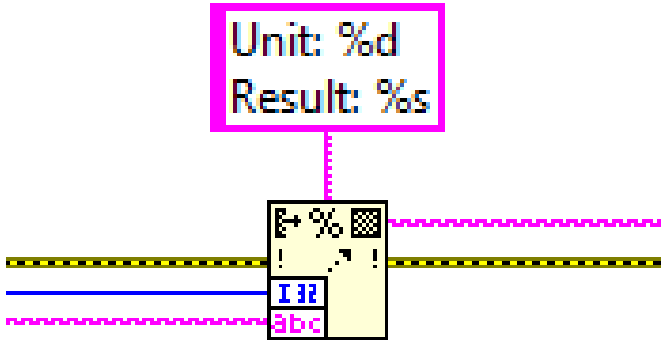
or

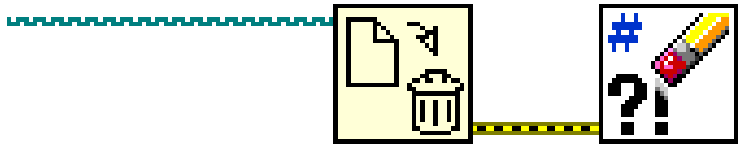


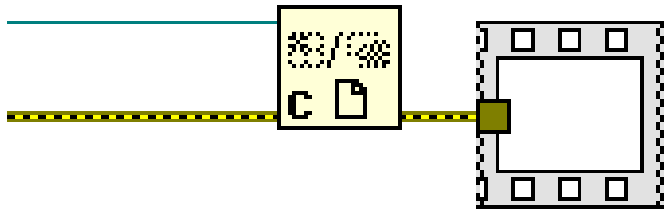
Wire Every Error Output Terminal



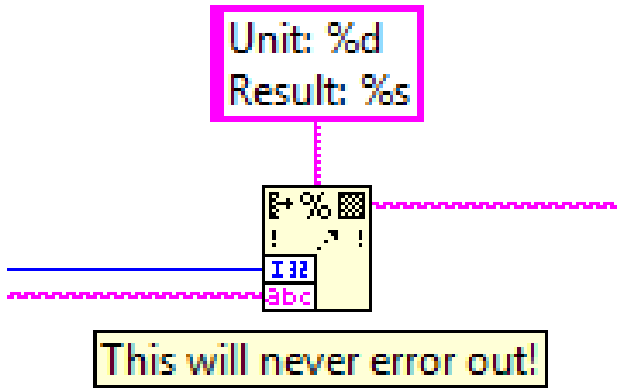
- Make sure there aren't any errors you're missing.
- Prevent auto error handling dialogs popping up for errors you don't care about.

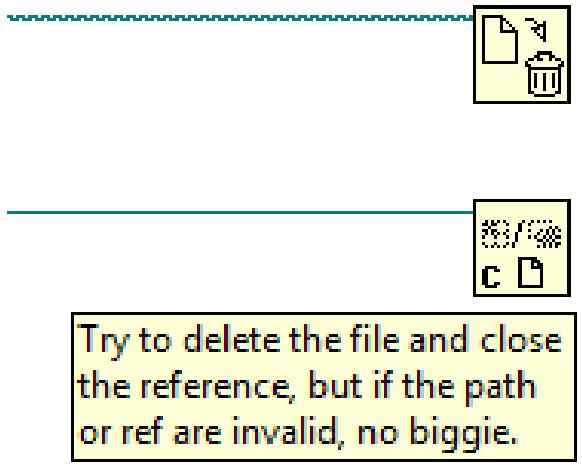






Don’t Wire Error Terminals That Don’t Matter



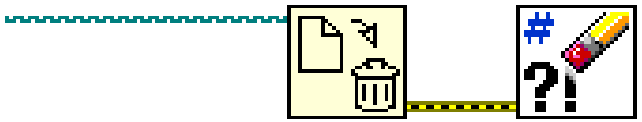


- If you know a function can’t reasonably error out, don’t wire its error terminal. More opportunities for compiler optimization and parallelism.

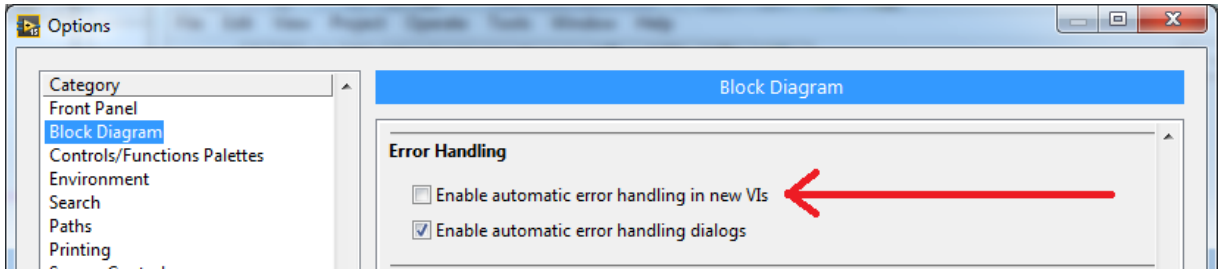
- If you know a function might error out, but you don’t care, don’t wire its error terminal.

Don’t be a Slave to Auto Error Handling!

- Automatic error handling (more like *Brainless* error handling) is the worst feature ever. Don’t write silly code because of it!



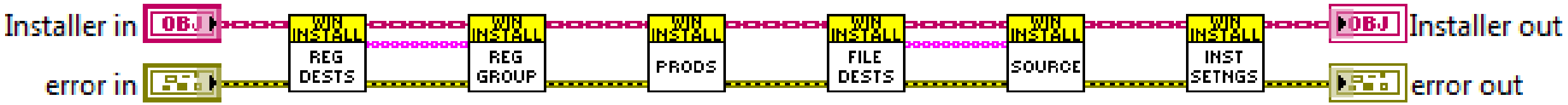
- You can turn off Auto Error Handling in the *Tools > Options* setting that enables it on new VIs.



- Test - Auto Error Handling Detect or Correct - NI Community: a VI Analyzer test that will turn off this setting on all VIs in an existing codebase.

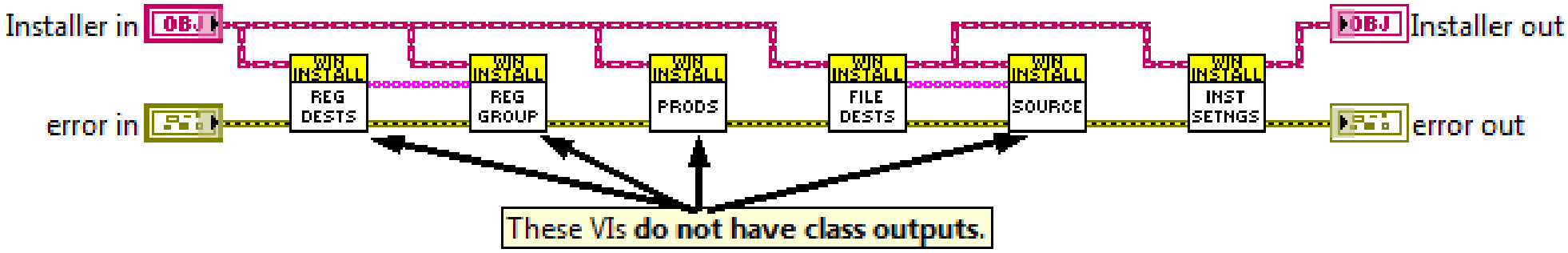
Method VIs Need Pass-through Class Wires

- "Train tracks" are visually pleasing.
- It's convenient to always have the class wire available as an output from any class member VI.
- *Everybody's code looks like this!*



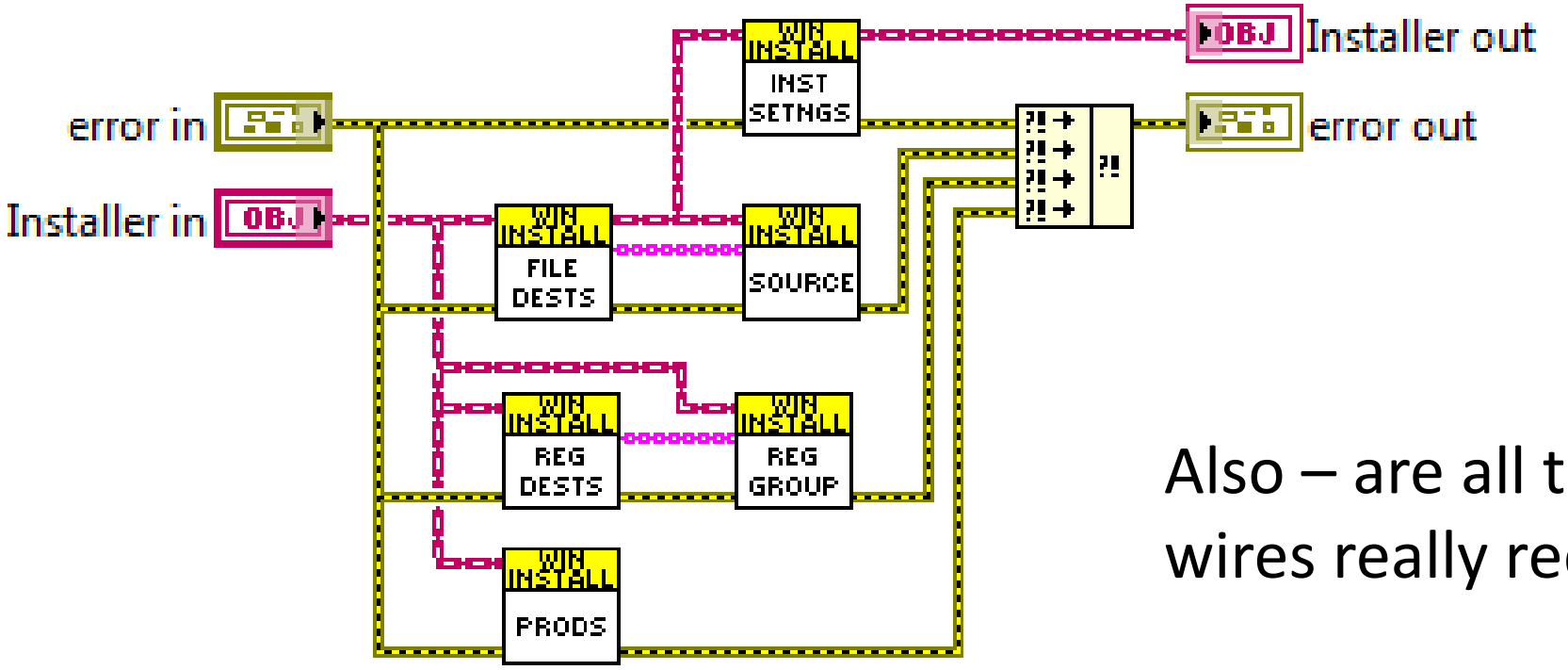
Method VIs Don’t Require Class Outputs!

- If a method VI does not manipulate the class data, **don’t make it a pass-through wire!**
- Dynamic dispatch VIs are perfectly happy only having a class input!
- Method VIs that don’t change pass-through class data slow debugging down! Only 2 VIs here modify class data



Don’t be Tied to the Train Tracks.

- If the subVIs in the previous API didn’t need to be serialized, then you could modify the code further to take advantage of parallel execution:



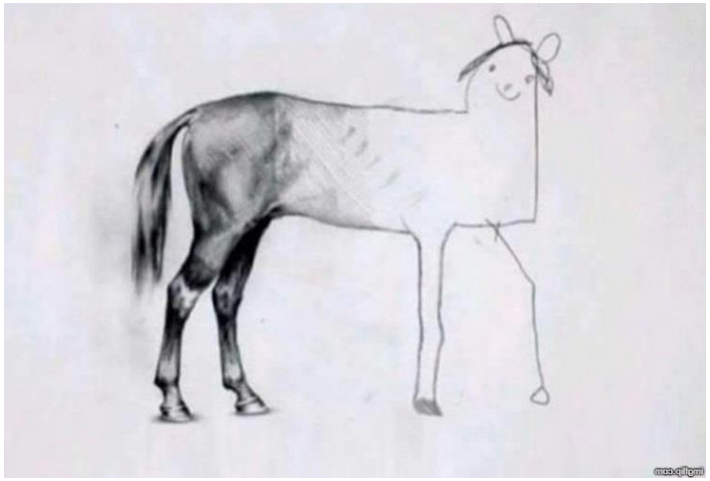
Also – are all the error wires really required?

Avoid Global Variables at all Costs!

- Globals are evil!
- They can be easily abused!
- They lead to race conditions!
- Never use globals!
- Use functional globals instead.

Globals are Just One Tool in a Big Tool Box

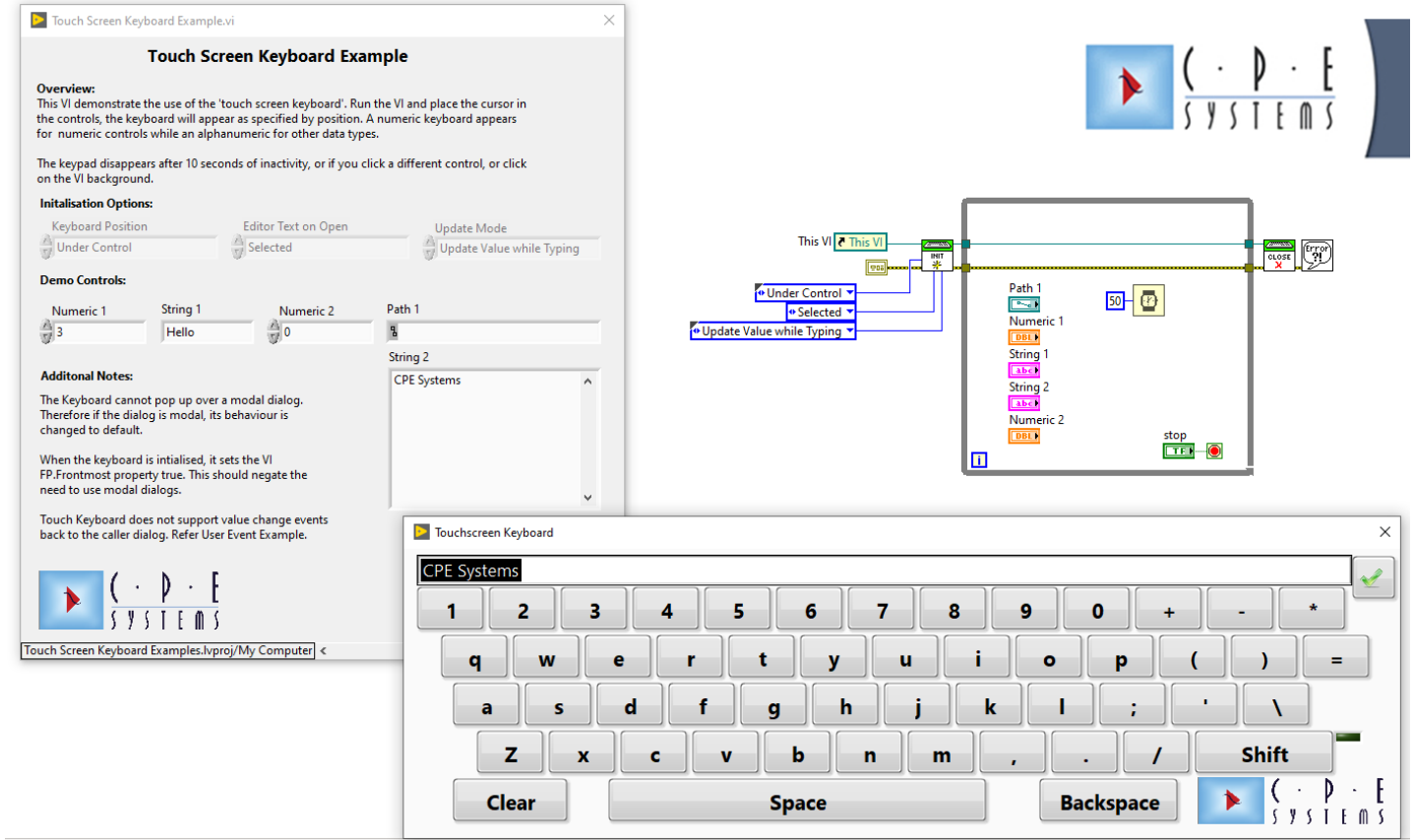
- Horses for Courses
- Global variables are very easy to use.
- You have the potential to use them in an unsafe, accidental manner.
  - The same could be said for driving a car.
  - We still drive, we just use our brains when we do so.



- The safest way to use globals is to write to them in 0 or 1 places.
- Don’t brainlessly replace them with read/write functional globals.
  - Same exact functionality, but take longer to implement.
- If you’re really paranoid, use the ‘privately scope the global to a library and wrap the *write* operation with a private VI’ trick.

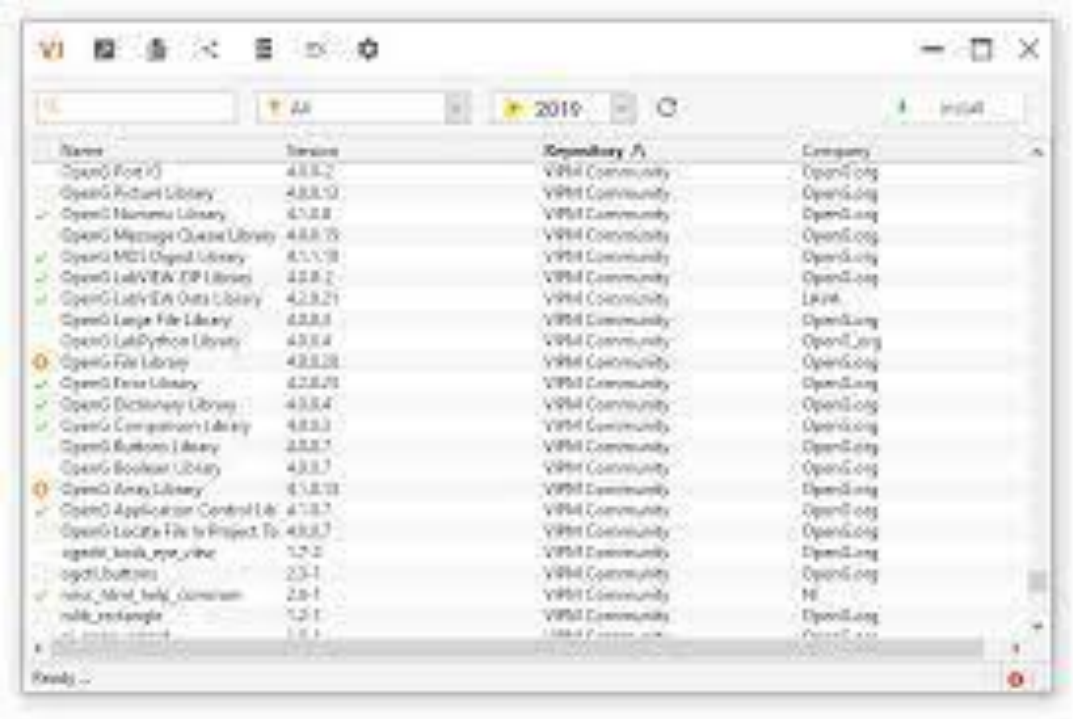
Should The Code Already Exist?

- Look first
- CPE Systems Touch Screen Keyboard – it can't be that hard right?



Don’t Reinvent the Wheel

- ni.com/hiddengems
- Unsupported (but useful) VIs that ship with LabVIEW
- Tools Network download to add hidden gems to Quick Drop/palettes
- Search VIPM



The Future… AI

AI such as ChatGPT / CoPilot is here…. It does know about LabVIEW

```
LabVIEW (short for Laboratory Virtual Instrument Engineering
Workbench) is a programming language and development environment
specifically designed for creating custom virtual instruments for
data acquisition, instrument control, and industrial automation.
```

Because LabVIEW is a graphical programming tool, I guess its really hard for you to help me...

```
As a language model, I can certainly help answer questions about
LabVIEW programming language and concepts, although as you
mentioned, it may be more challenging for me to provide specific
guidance on designing or debugging a LabVIEW program because of its
graphical nature.
```

LabVIEW R&D is working on Nigel

NI Connect 2023 Day 2 - Closing Keynote

https://www.youtube.com/live/Bk-dmXEp5xk?si=BLxLRJn7jNutTWoj

Nigel Review

- Code looked AI generated – I assume it is trained on the connections between VIs rather than layout (Which Block Diagram cleanup could have helped with)
- Hope Block Diagram cleanup with AI can get code closer to what humans prefer

Keep Learning

- Check out new features
- Check out Forum
- Attend LUGs & GDevCon!

Thank You