

#OurGiantsAreFemale - sammancoaching.org

Emily Bache



Clare Sudbery



Stop Straightening Wires

Focus on what matters

Questions/Comments

Speak up to ...

- Ask for clarification

Wait until the end to ...

- Add your experience/wisdom
- Argue with everything I'm saying
- Nitpick

I appreciate these things and I am probably going to cover it in the next slide.

If it at the end I missed something, we can talk about it then.

Are Zebras Black with White Stripes or White with Black Stripes?



2 Questions

Rock Climbing

- Does it work?
- Is it safe enough?

Code

- Does it work?
- Is it readable enough?

Premise

A lot of our discussion around style is arguing about zebras.

Our lives would be much easier with an autoformatter.

Opportunity Cost

It's not that we shouldn't strive for straight wires.

We shouldn't be manually moving them pixel by pixel.

Opportunistic Laziness

Which is “better”?

OnTop



OnLeft 

 OnRight



OnBottom

More Choices

Left Aligned



Centered



Right Aligned



Left Aligned



Centered



Right Aligned



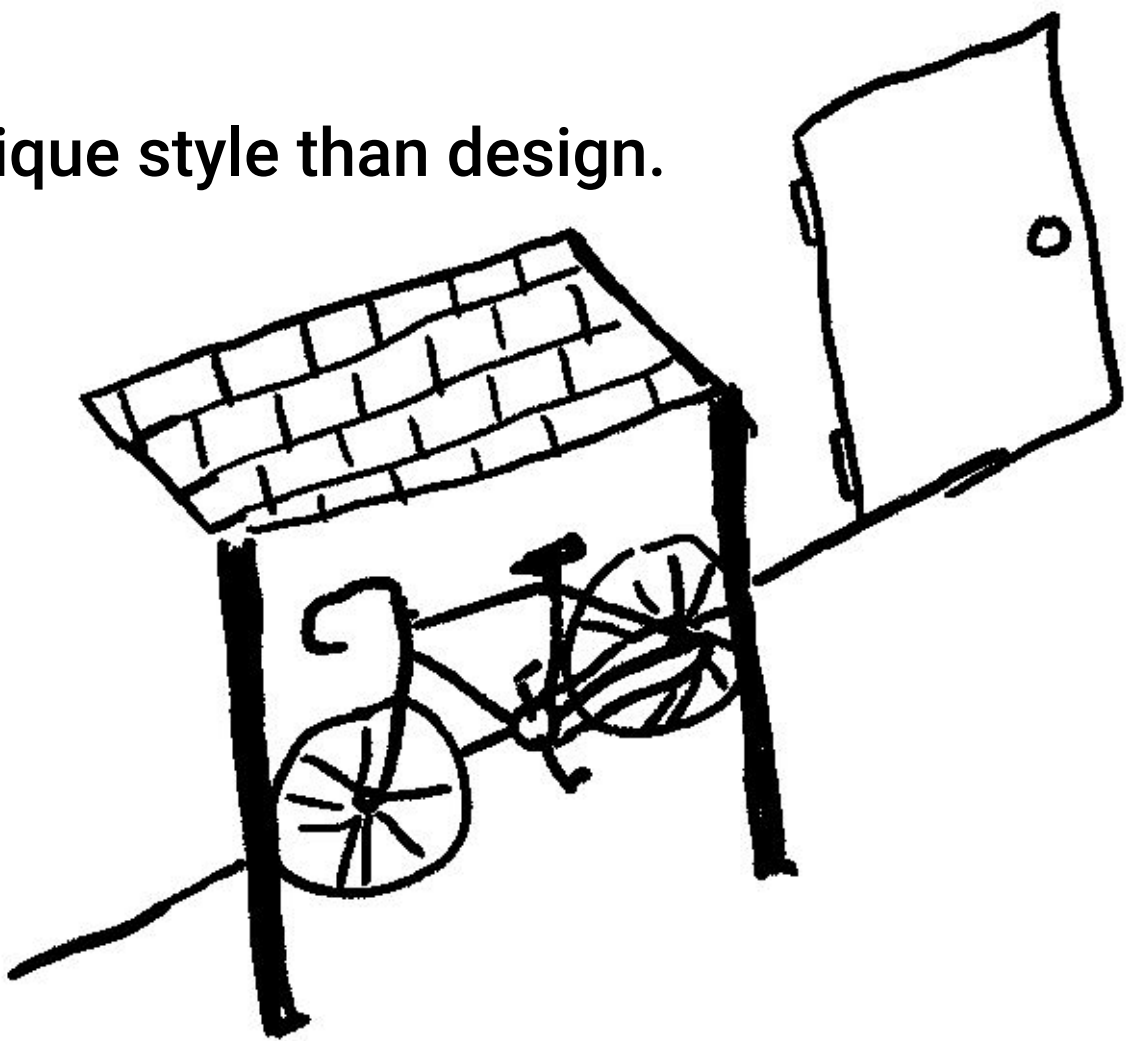
More pedantic

- Fonts and font size
- Spacing/Caps-> title, snake, camel, kebab case
- Abbreviations
- Units ie “time” vs “time_in_ms” vs “time(ms)”
- Default values
- Do arrays get []? What about 2D arrays - [][]?
- Do booleans get an is and/or a ? ie. “is valid” vs “valid?” vs “is valid?”
- Max number of characters

We haven't even left control/indicator labels yet



It's much easier to critique style than design.



Code Review Findings

Style Findings

A list of VIs missing VI descriptions

“my-subvi.vi” should be “My SubVI.vi”

This one subVI has the wrong conpane

This one subVI has a default icon

You have too many wire bends in this VI

Architecture Findings

Circular Dependency between Class A and Class B

Our God Class has too many responsibilities

Our messages are not atomic leading to race conditions

Our VIPC file is missing some key dependencies

You are duplicating functionality that exists in JSONtext

Show of Hands

Which side is:

- Easier to spot?
- Easier to fix?
- Adds more value when fixed?

A list of VIs missing VI descriptions

“my-subvi.vi” should be “My SubVI.vi”

This one subVI has the wrong conpane

This one subVI has a default icon

You have too many wire bends in this VI

Circular Dependency between Class A and Class B

Our God Class has too many responsibilities

Our messages are not atomic leading to race conditions

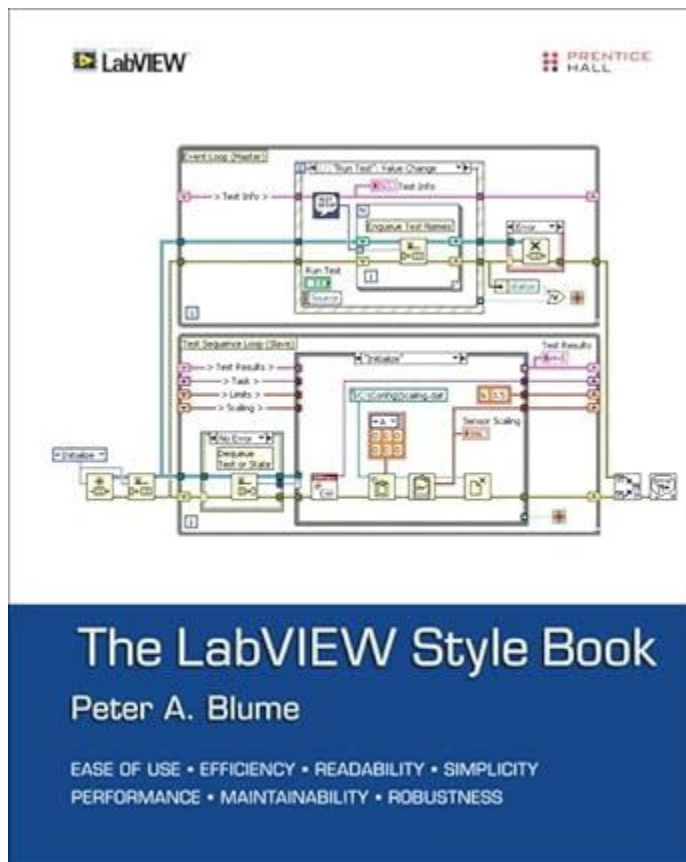
Our VIPC file is missing some key dependencies

You are duplicating functionality that exists in JSONtext

Style is not a goal but a means to an end.

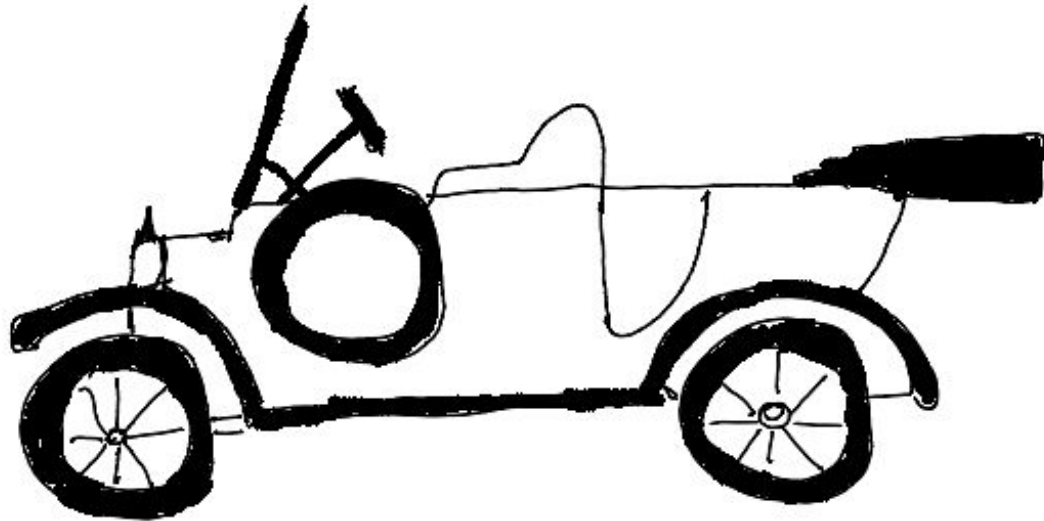
Does the actual choice (label location, font, camel vs kebab, etc) matter as much as consistency?

How do we achieve consistency now?



Limiting choices in pursuit of a goal.

*“You can have any color you want as long as it is
Black”*





<https://www.theatlantic.com/family/archive/2023/06/grocery-shopping-option-overload/674502/>

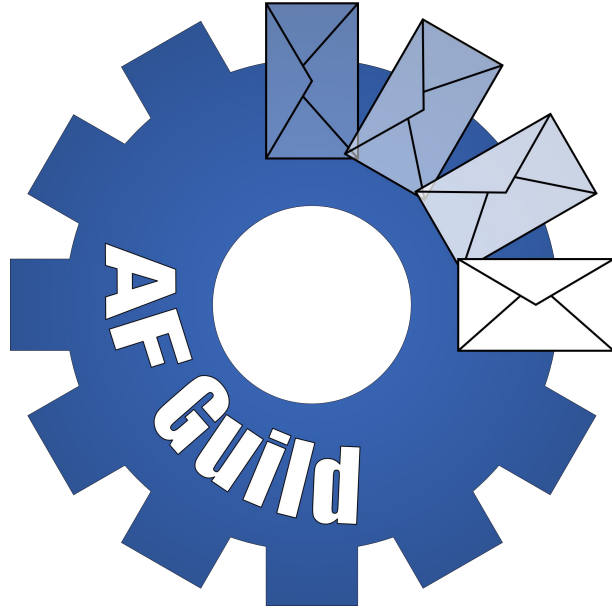


Southwest

N8854Q

Southwest.com

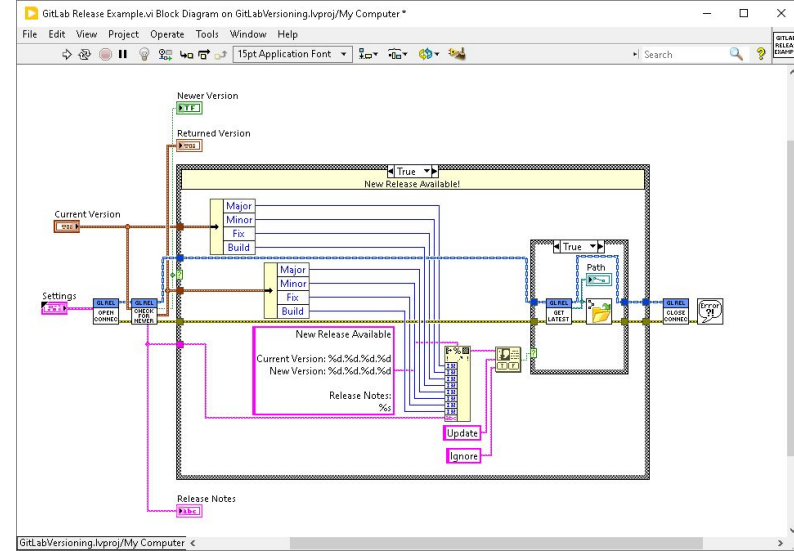
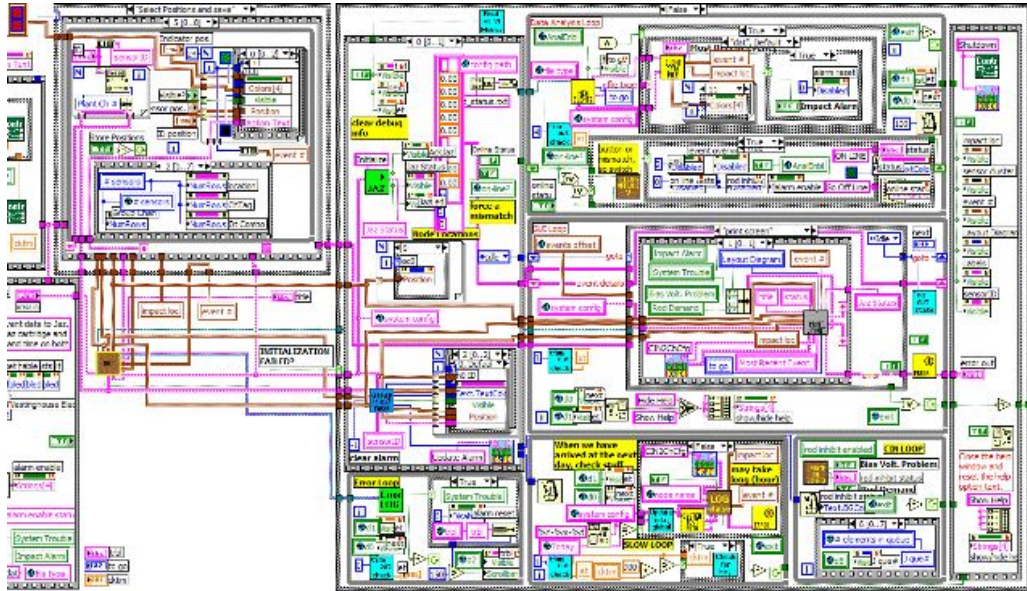
Frameworks limit our choices



What are our goals? (aside from consistency)



How do we end up with style guides?



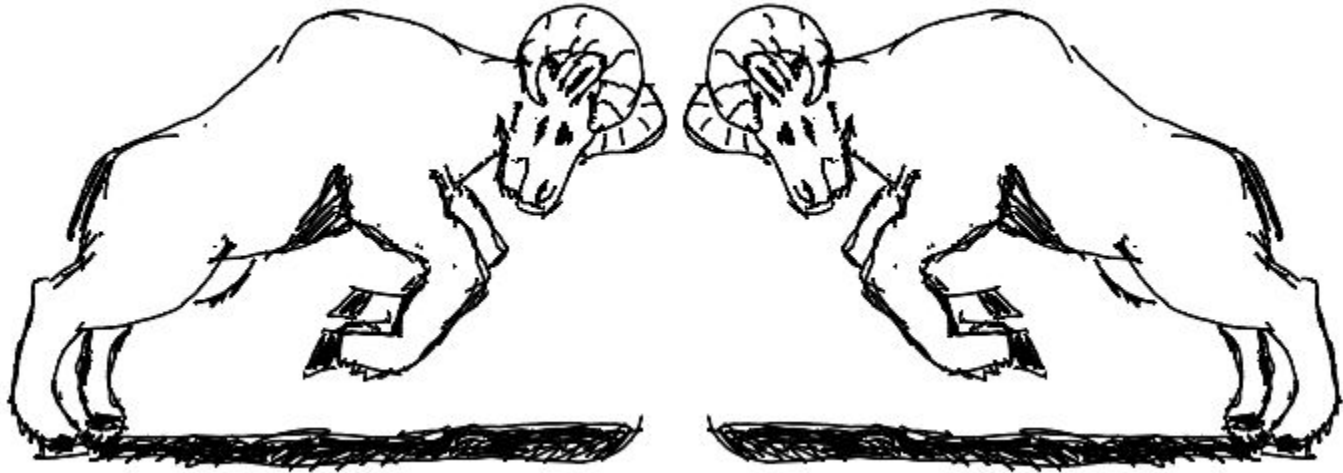
Everyone has a style.



**If you are a lone wolf developer
you don't necessarily need a style guide**



Two heads are better than one, until you disagree.



This is getting out of hand



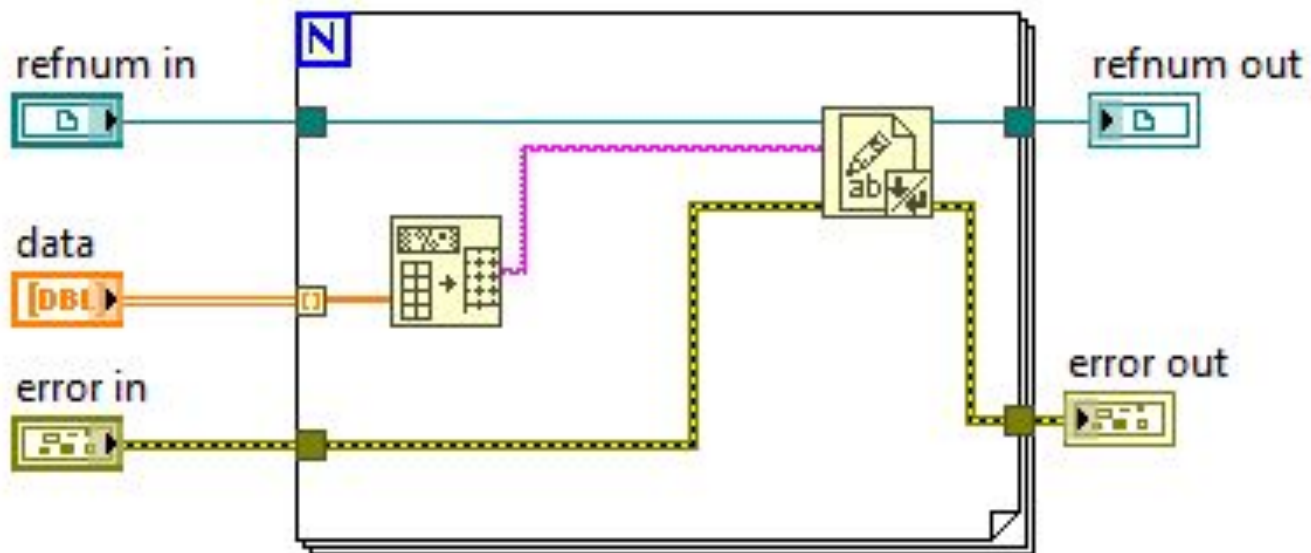
SAS ~/Desktop/example (main)

>> git graph

- * 3315401 (HEAD -> main) Went through and replaced all spaces in file names with underscores - you are welcome!
- * 4a9cf35 You both are wrong - terminal labels should be on the side.
- * b99a5a0 Right-Aligned terminal labels - easier to read that way
- * 0be056f Left-Aligned terminal labels
- * ac186a0 initial commit

SAS ~/Desktop/example (main)

>> |



Step back - what are style guides trying to achieve?

- Consistency
- Avoid ugly, unreadable code
- Create code that is easier to change
- Create code that is less buggy
- Avoid Squabbles/noise in Git
- Avoid some easy to spot bugs
- **Make devs interchangeable**



Style Guide Costs

It takes time to create

Can be mitigated by starting with an existing one

It requires lots of decisions

Decision Fatigue is real.

The rules need to be clear and not vague

Requires more decisions. Leads to bloat.

It requires management

You need a change process. New features are added.

You need to get consensus and buy-in

How do you get everyone to agree?

It can be difficult to remember all the rules

Style guides can be quite large.

You need to deal with optional rules

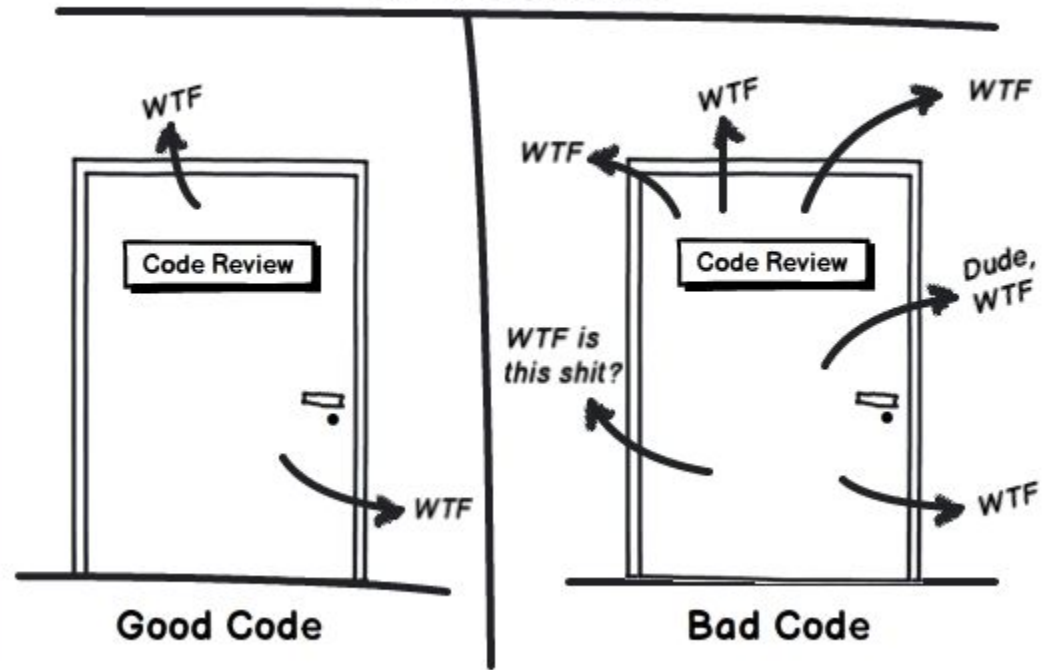
Should you even have them? How to enforce them?

It requires enforcement

People can be lazy or forgetful. It doesn't enforce itself.

Story Time

Code Quality Measurement: WTFs/Minute

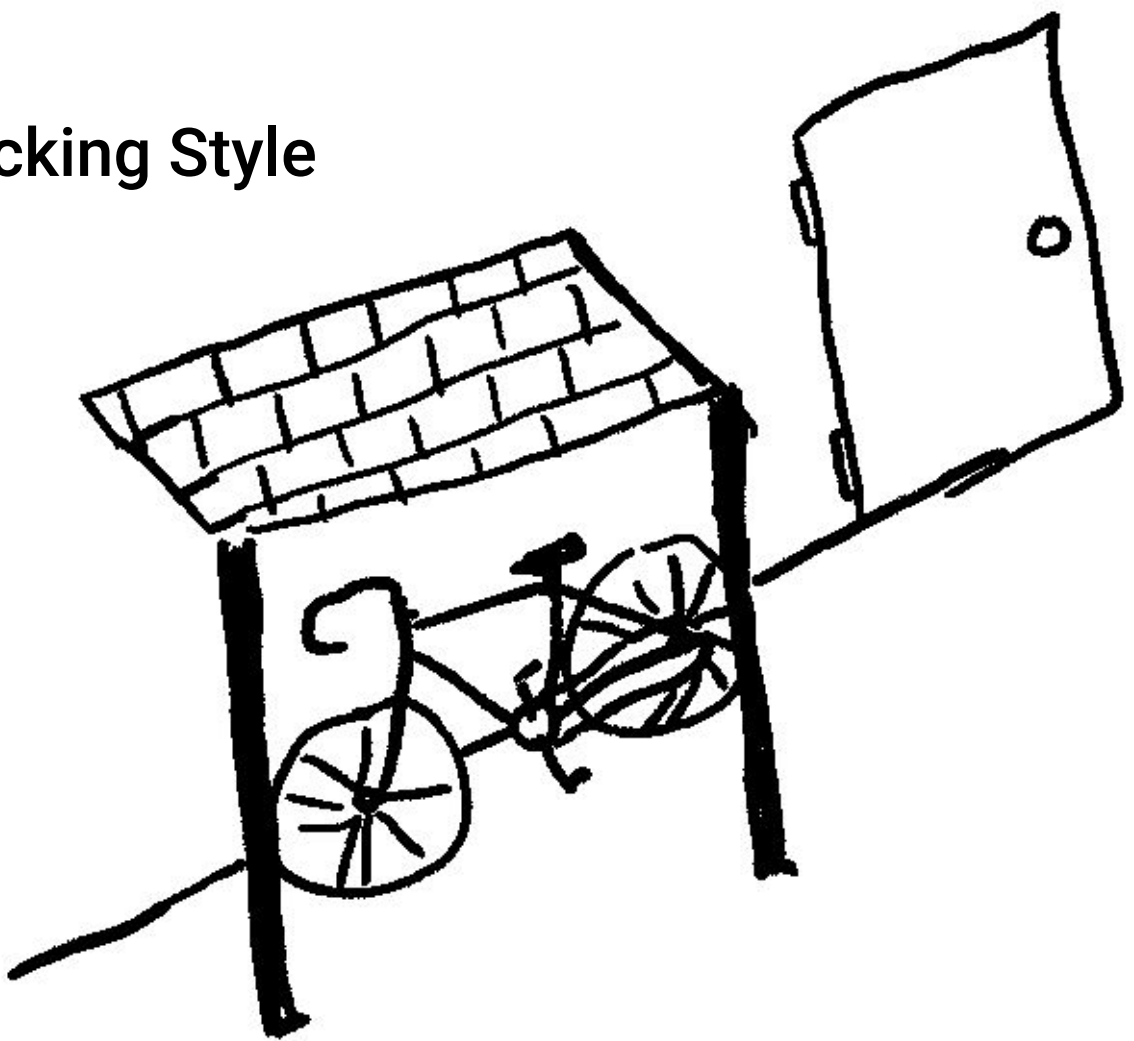


<http://commadot.com>

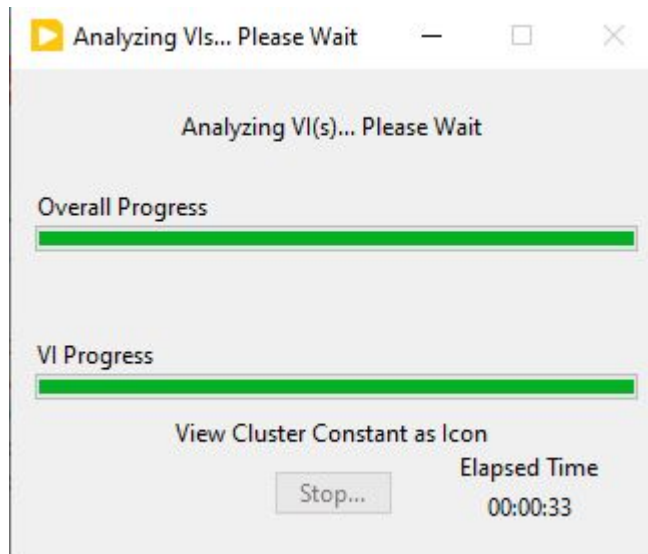
<https://commadot.com/wtf-per-minute/>

Code Reviews for Checking Style

- Why did you choose that:
 - Framework
 - Datatype
 - Design Pattern
 - Dependency
- How does this solve the customers problem?
- Did you consider this edge case?
- What other alternatives did you consider?



Enter VI Analyzer



[Solutions](#) ▾[Products](#) ▾[Perspectives](#) ▾[Support](#) ▾[Community](#)[About](#)[Contact Us](#)[ST](#)[HOME](#) / [COMMUNITY](#) / [USER GROUPS](#) / [SPECIAL INTEREST GROUPS](#) / [VI ANALYZER ENTHUSIASTS](#) / [VI ANALYZER ENTHUSIASTS DOCUMENTS](#) / [DOCUMENT](#)

VI ANALYZER ENTHUSIASTS DOCUMENTS



Taggart

DOCUMENTS ▾

Search the community

SEARCH

Document options ▾

List of Community VI Analyzer Tests

by Jordan_M on 03-14-2013 05:25 PM - edited on 11-03-2023 02:33 PM by: Darren

The following is a list of community-contributed VI Analyzer Tests:

[Anti-Aliased Plots](#)[Auto Error Handling_Detect or Correct](#)[Block Diagram Background Color](#)[Broken VI Extended](#) (exclude FPGA VIs)[Case Sensitive String_case structures](#)[Check Captions and Labels Match](#)[Check for Whitespace in FP names](#)[Check VISA read/write Synchronous mode](#)[Check whether Tunnels use default values if unwired](#)[Checks whether references are closed](#)[Comparing Variants](#)[Constant Documentation](#)[Copyright](#)[Custom Control / TypeDef Style](#)[Deprecated Functionality](#)[Detect Mechanical Action](#)

The Costs of VIAN

It is slow

Even a medium projects can take a long time.

It can be overwhelming

Too much noise. Too many Options. Irrelevant Tests.

Scripting skills are required

Custom tests require a scripting ninja.

Manually adding pacifiers is wasteful

Why can't this be done automatically?

Fixing issues is an iterative process

Fixing one issues often causes another to fail.

It is unclear what to do with failures

Do you fail CI/CD if any test fails? Some threshold?

It doesn't play well with legacy code

It's working, do you want to manually change it all?

Module Name Const
Module Name Const
art Module VI
op Module VI
op Module VI
ster VI
ster VI
Settings

that returned the Reply Payload. This information can help when debugging.

Click the **Fix Issue** button below to correct this issue by modifying the code for these Request and Wait for Reply event VI(s) to return the Module ID of the instance that returned the reply.

performed.

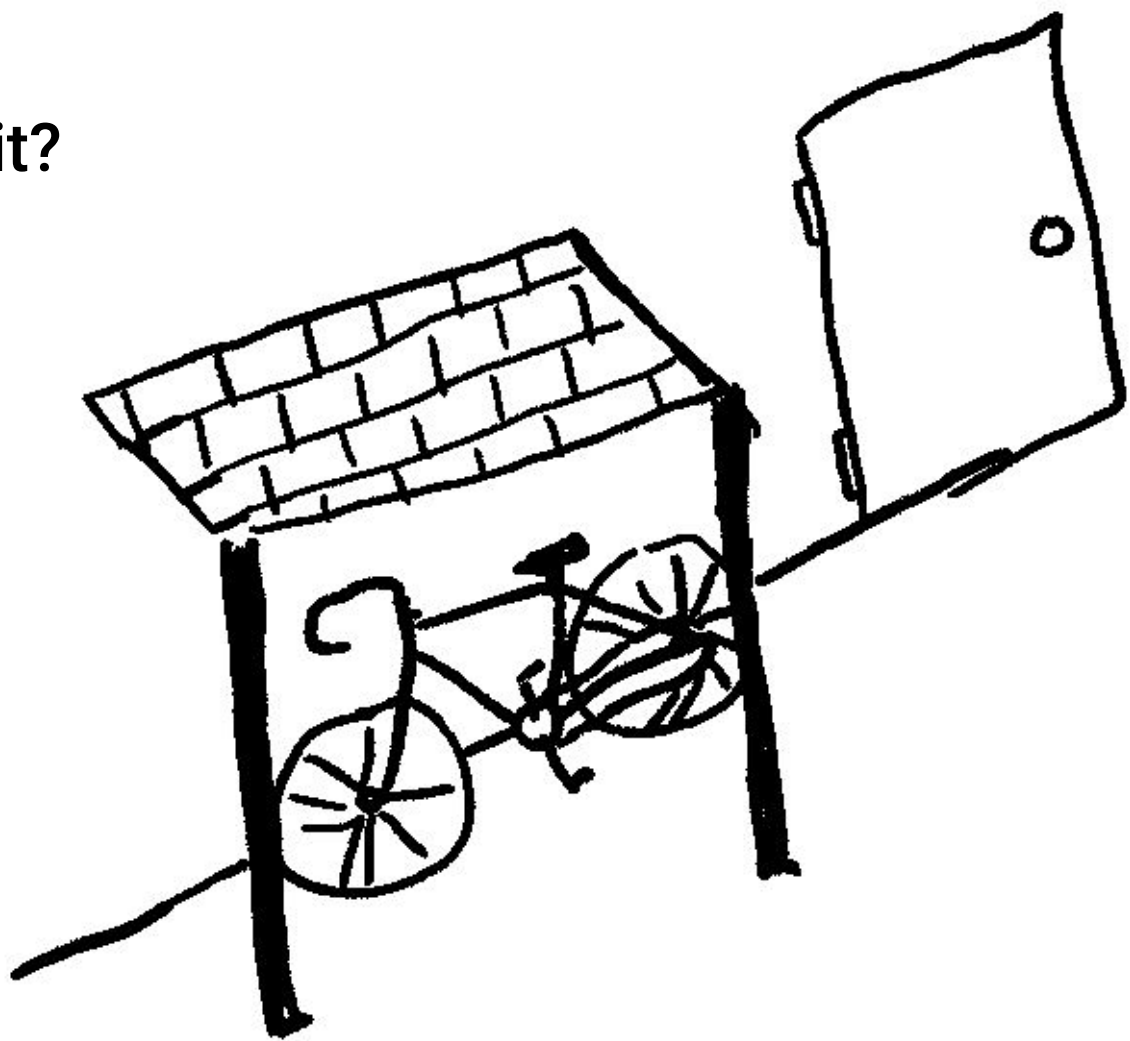
All Modules

Fix Issue

Close

Help

Is all this really worth it?



Are style guides/VIAN the best way to achieve our goals?

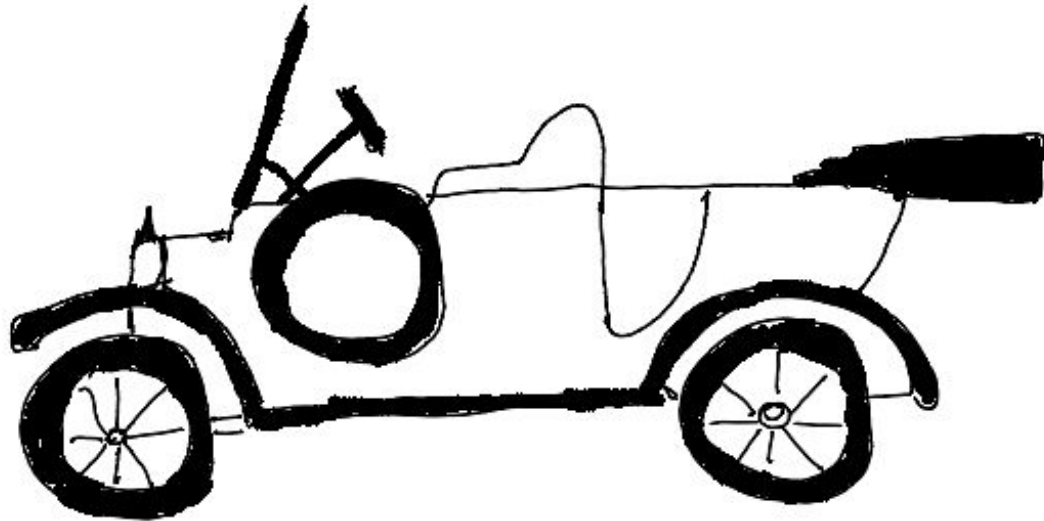
- Consistency
- Avoid ugly, unreadable code
- **Create code that is easier to change**
- Create code that is less buggy
- Avoid Squabbles/noise in Git
- Avoid some easy to spot bugs
- Make devs interchangeable



Can we get the same benefits without the
cost?

Limiting choices in pursuit of a goal.

*“You can have any color you want as long as it is
Black”*



A better way

Why an autoformatter?

Its consistent

Very little or no configuration. Always runs so everything always looks the same.

It is proactive versus reactive

VIAN runs after you've written the code. This runs as you are writing the code. There's no going back after the fact to fix things.

It requires no extra thought or decision making

It runs automatically and just fixes things for you. No more input required.

It plays well with legacy code

Less work and less risk than manual edits

It frees up time to focus on important stuff

Avoids time spent on:

- bikeshedding
- creating style guides
- configuring VIAN
- creating custom VIAN tests
- going back and fixing things after the fact



The Uncompromising Code Formatter



"Any color you like."

Black is the uncompromising Python code formatter. By using it, you agree to cede control over minutiae of hand-formatting. In return, *Black* gives you speed, determinism, and freedom from `pycodestyle` nagging about formatting. You will save time and mental energy for more important matters.

Blackened code looks the same regardless of the project you're reading. Formatting becomes transparent after a while and you can focus on the content instead.

Black makes code review faster by producing the smallest diffs possible.

```
Git Bash
~/Desktop/lv-format-on-save/pytest (main)
>> black .
reformatted C:\Users\SAS\Desktop\lv-format-on-save\pytest\test_CLI.py

All done! 🌟📁🌟
1 file reformatted.

~/Desktop/lv-format-on-save/pytest (main) mod: 1
>> |
```

Tools > BlackConnect

- Trigger when saving changed files
- Trigger on code reformat

Connection Settings

Hostname: localhost Port: 45484

Use SSL (e.g. blackd behind nginx)

Check connection

Local Instance (shared between projects)

Start local blackd instance when plugin loads

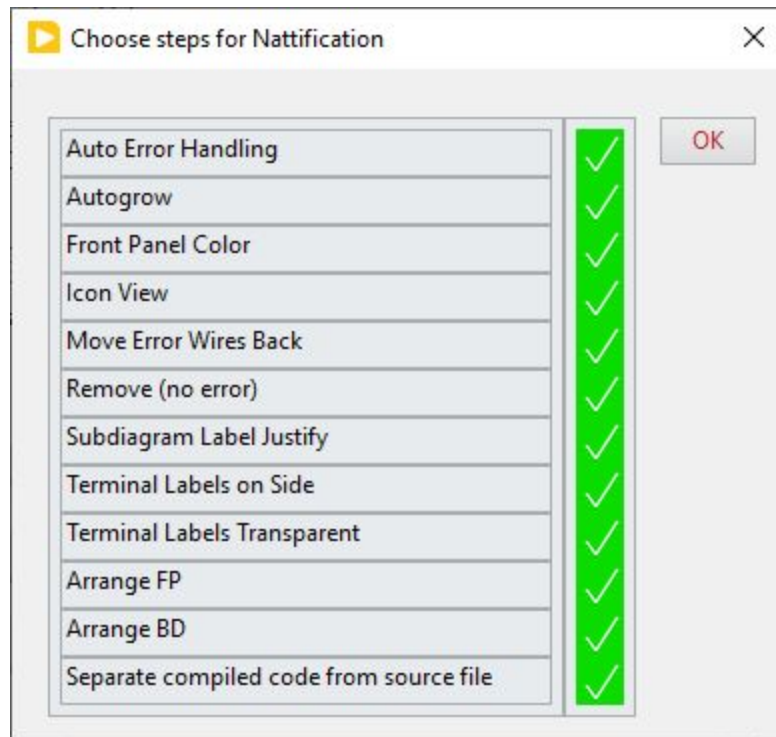
Bind on: localhost Port: 45484

Path: \$USER_HOME\$/.local/bin/blackd Detect

No instances are running at this moment.

Start Stop

Nattify



Letting Go

OnTop

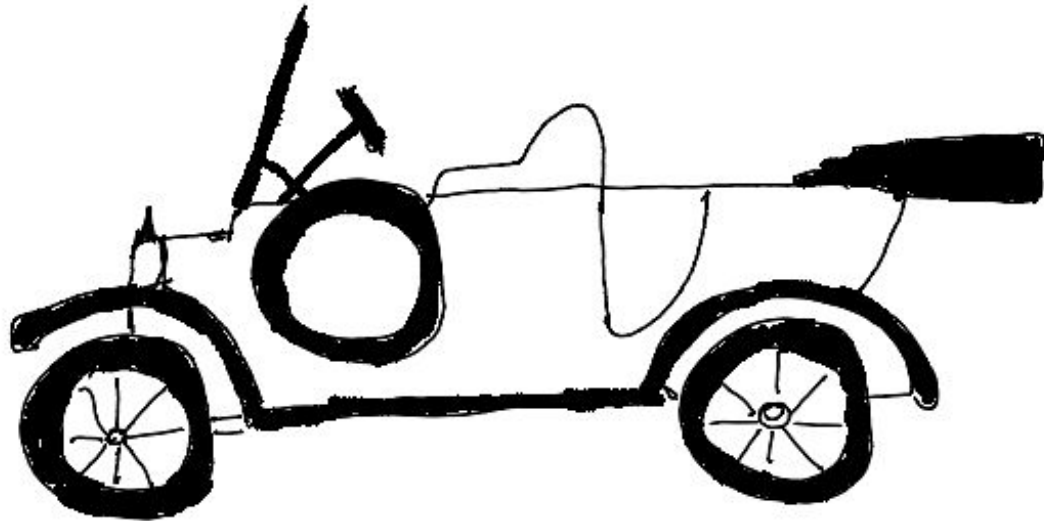


OnLeft



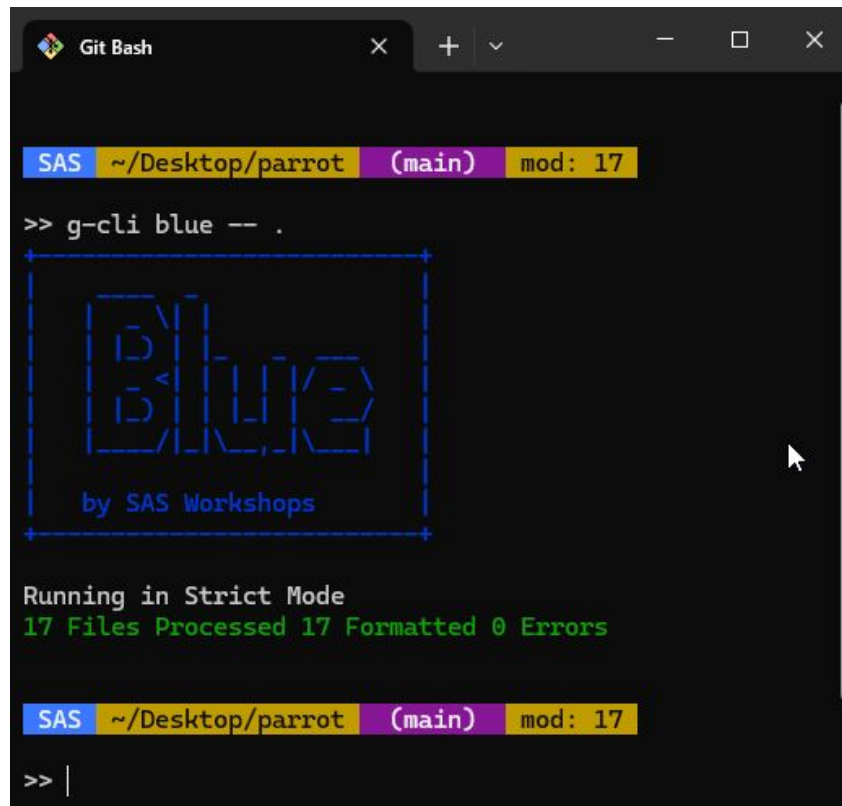
Limiting choices in pursuit of a goal.

*“You can have any color you want as long as it is
Black”*



**It's not about losing creativity, it's about
focusing it on what really matters**

Blue



```
Git Bash
SAS ~/Desktop/parrot (main) mod: 17
>> g-cli blue -- .
┌───────────────────────────────────────────────────────────────────────────────────┐
│                                     Blue                                       │
│                                     by SAS Workshops                          │
└───────────────────────────────────────────────────────────────────────────────────┘

Running in Strict Mode
17 Files Processed 17 Formatted 0 Errors

SAS ~/Desktop/parrot (main) mod: 17
>> |
```

BlueMon



Getting Started with Blue

Blue - Like Black, but for LabVIEW

Latest Release 1.0.4 pipeline passed  installs 71  stars 3

This is a LabVIEW autoformatter inspired by Python's [Black](#) and [this](#) from Felipe Pinheiro Silva. I also borrowed some code from the various [Nattify VIs](#)

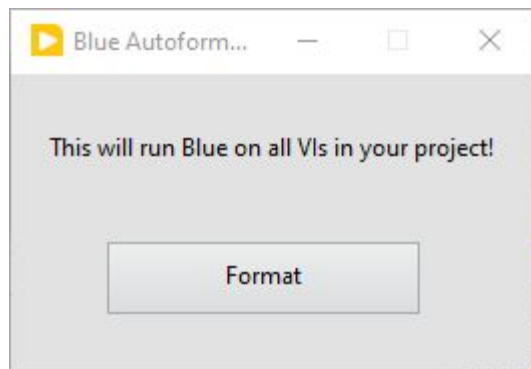
This repository is for the g-cli tool which can be run from the command line. There is a [bluemon project](#) which is a file monitor that will allow you to monitor a directory and everytime a VI is saved in that directory or one of its subdirectories the monitor will run blue on it. That is still in progress.

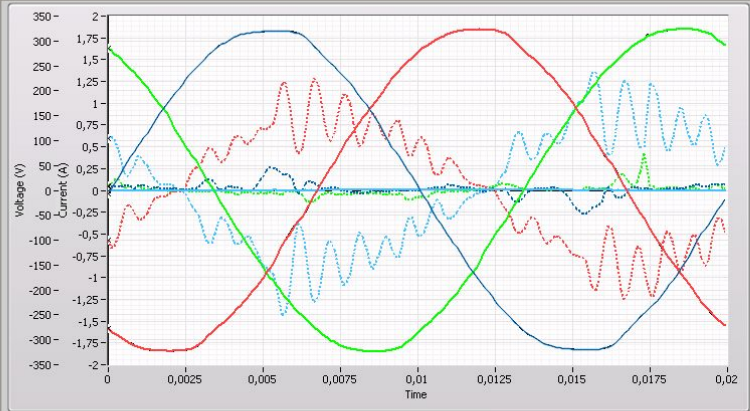
Table of Contents

- [Blue - Like Black, but for LabVIEW](#)
 - [Table of Contents](#)
 - [Roadmap](#)
 - [Requirements](#)
 - [Installation](#)
 - [Quick Start](#)
 - [Running only on changed files](#)
 - [Usage](#)
 - [Help](#)
 - [Formatting a directory](#)
 - [Verbosity](#)
 - [Quiet](#)
 - [Verbose](#)
 - [Very Verbose](#)
 - [Very Very Verbose or Debug](#)
 - [Ignoring Specific Steps](#)
 - [Getting a List of Steps and Macros](#)

Things You Might Be Worried About

I'm afraid of the CLI





- U L1
- U L2
- U L3
- U N
- I L1
- I L2
- I L3
- I N

Aggregations E160160

Aggregations Custom

Spectra THD

Power Quality

aggregation settings 1

Interval: 1

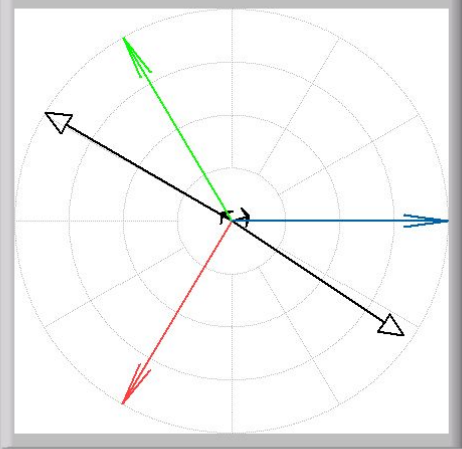
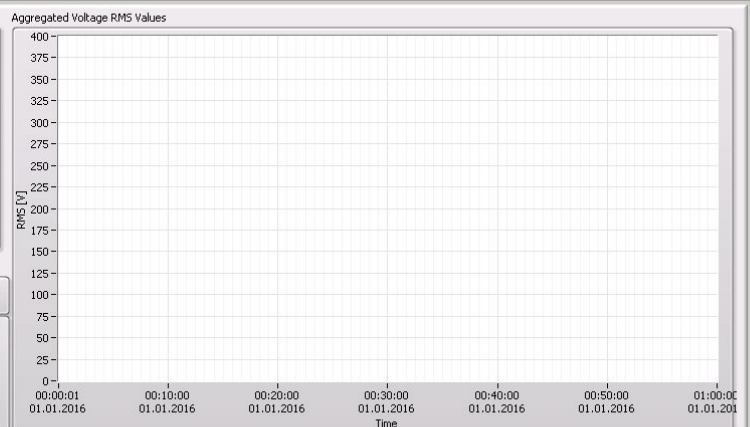
Interval unit: Seconds

synchronization of intervals: Not synchronized

algorithm: RMS

Cursor 0

Phase	X	Y
Phase A		
Phase B		
Phase C		



Cursors:

Cursor	X	Y
Cursor 0	U L1	0 -7,71693
	U L2	0 -277,602
	U L3	0 286,093
	U N	0 -0,01702

AC Voltage

229,3

230,59

230,06

AC Current

0,08

0,64

0,06

T0

7,5 10 12,5 15 17,5 20 22,5 25

L1 L2 L3 Topology

230 230 230 Line

HalfZ T1On T2On T3On T4On

aggregation settings 2

Interval: 1

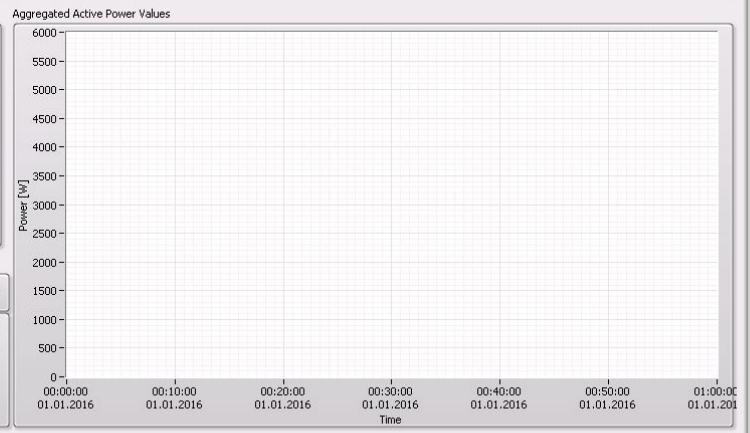
Interval unit: Seconds

synchronization of intervals: Not synchronized

algorithm: Average

Cursor 0

Phase	X	Y
Phase A (L1)		
Phase B (L2)		
Phase C (L3)		



Will it Change How My Code Runs?

Maybe?

- Auto Error Handling
- Separate Compiled Code

How Do I fix that? (ie ignore or skip certain steps)

Several ways:

- #blueignore bookmarks
- .blueignore files
- `--ignore`

Get Involved!

Ways to Help

- Try out Blue
- Provide Feedback
- File Issues on GitLab
- Volunteer to help write or design code
 - Looking to add more features to tools menu GUI
 - Looking for better way to run automatically
 - Looking to add more steps beyond just Nattify Steps
- Lobby NI for an OnSave Hook

<https://forums.ni.com/t5/LabVIEW-Idea-Exchange/Allow-a-general-purpose-quot-on-Save-quot-hook/idi-p/4380317>