media mongrels
DIGITAL IS IN OUR DNA

Malleable VIs

Sam Sharp

MediaMongrels Ltd

sam@mediamongrels.com

NATIONAL INSTRUMENTS Alliance Partner

ARCHITECT

DEVELOPER

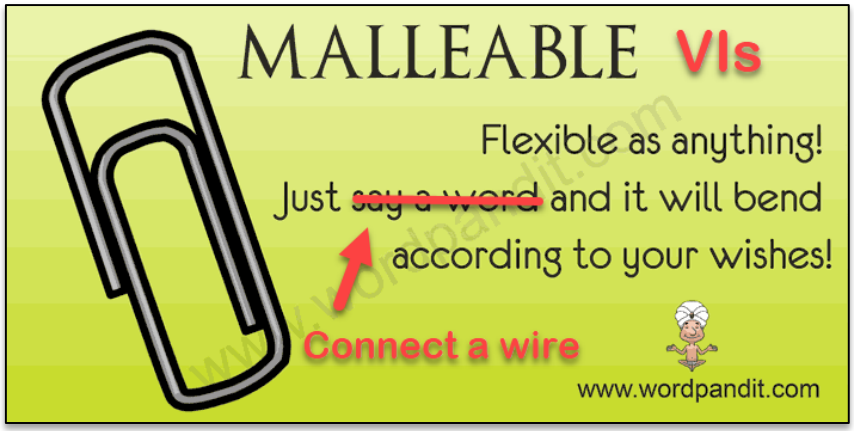
DEVELOPER

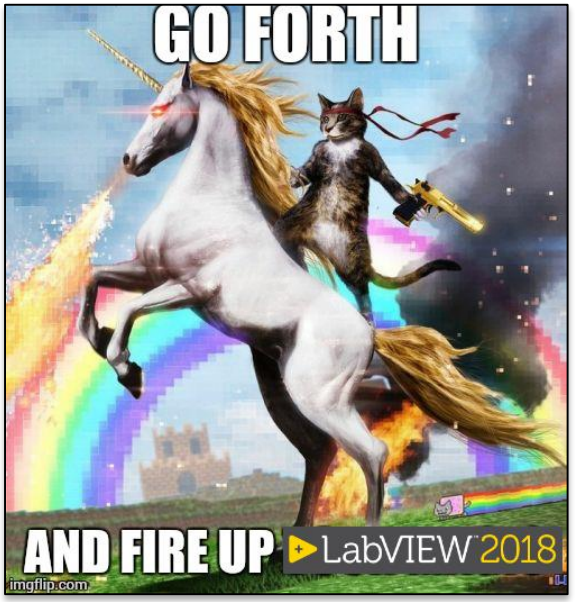
Introduction

This presentation…

- …is an introduction to Malleable VIs
- …is a summary of some blog posts available on my website
- …is similar to Stephen Loftus-Mercer's CLA Summit / NI Week presentation
- **but I am going to show some practical examples from my own code**
- …is intended to provide **inspiration** on how you can improve code re-use in your own projects
- …should give you confidence to start using and writing your own Malleable VIs

https://www.mediamongrels.com/blog



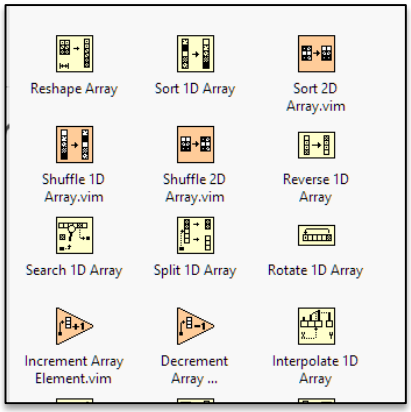


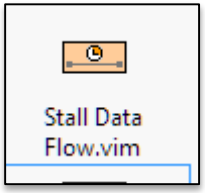
What is a Malleable VI?

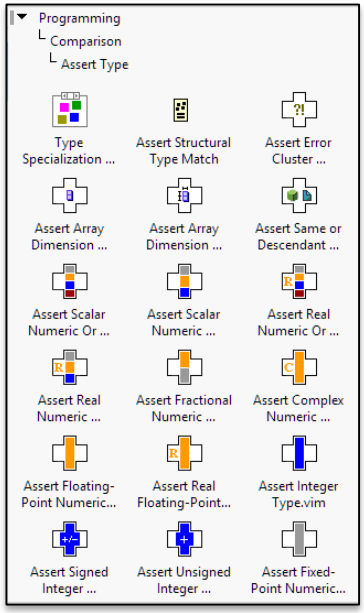
- A special type of VI where the input/output terminals can adapt to the wired data type
- Introduced in LabVIEW 2017
  - Made better in 2017 SP1 and 2018
- Created by:
  - New… -> Malleable VI
  - Save as '.vim' on existing VI and enable in-lining*
    *there are some caveats around this I'll discuss later
- Similar to 'generics' (Java/C#) or 'templates' (C++) in other languages
- Before Malleable VIs, this functionality was implemented with polymorphic VIs or variants (also xnodes!)

Basic Example - Demo

- LabVIEW 2017+ shipping examples (tan background)
  - Array Palette
  - Timing
  - Comparison
- LabVIEW 2018 brings new structure
  - Type Specialisation
  - Assert Type primitives
    (more on these later!)
- Detailed examples included in LabVIEW

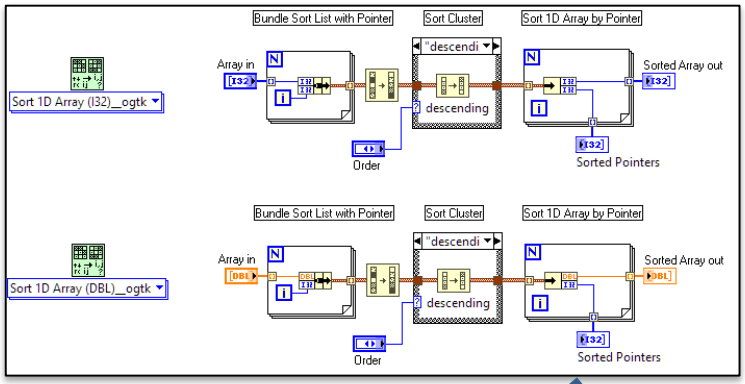






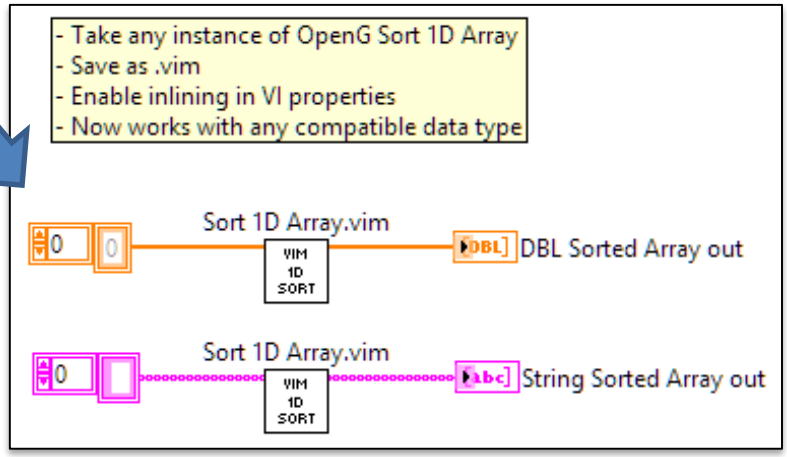
Why use Malleables?

- #1 Reason: Improve code re-use



30+ VIs to maintain

What about FXP or array of clusters?



1 VIM to maintain

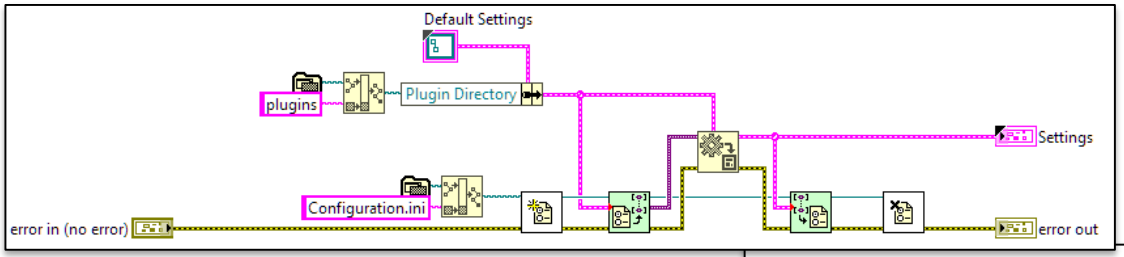
No broken run arrow = valid input

Malleable VIs – Use Cases

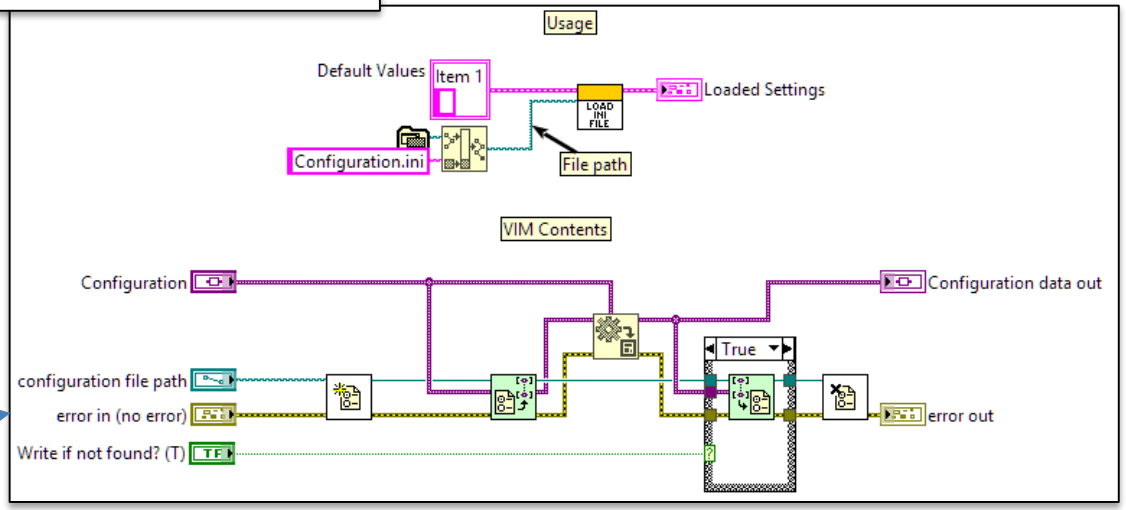
- Any situation where you File->Save As… and replace the input/output types
- Numeric/Mathematical Functions
- Calibration Functions
- Array Manipulation
- Wrapping communications primitives (e.g. Queue/Notifier)
- Debugging
- Logging
- Application Frameworks?!
- …and many more!

Practical Example 1: Configuration Library

- Simple library for loading/saving a configuration cluster to INI file
  - Based on OpenG Variant Configuration VIs
  - Uses ‘Default Values’ for missing INI keys



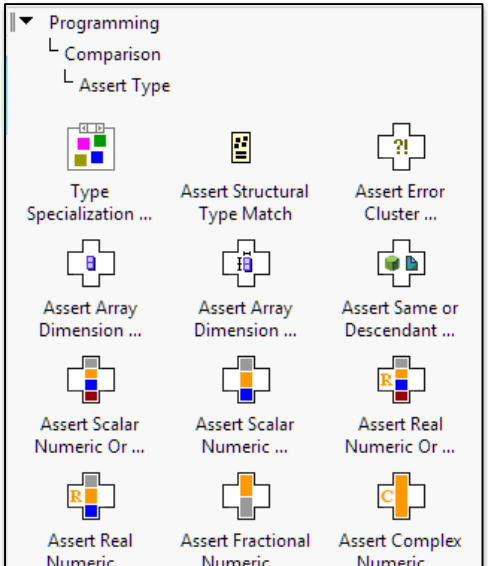
Project-Specific Load VI



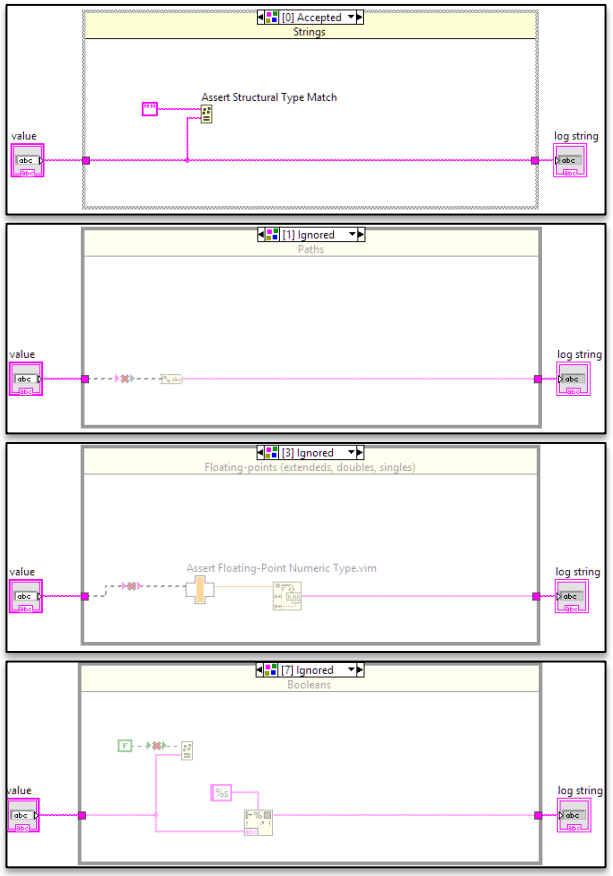
Reusable VIM

Type Specialisation Structure

- New Structure for Malleable VIs in 2018
- Allows special handling of certain data types
- Accepts first frame that will compile
- Use Assert VIs to force data type match



- Example:
  - Scalar to String.vim



Strings

Paths

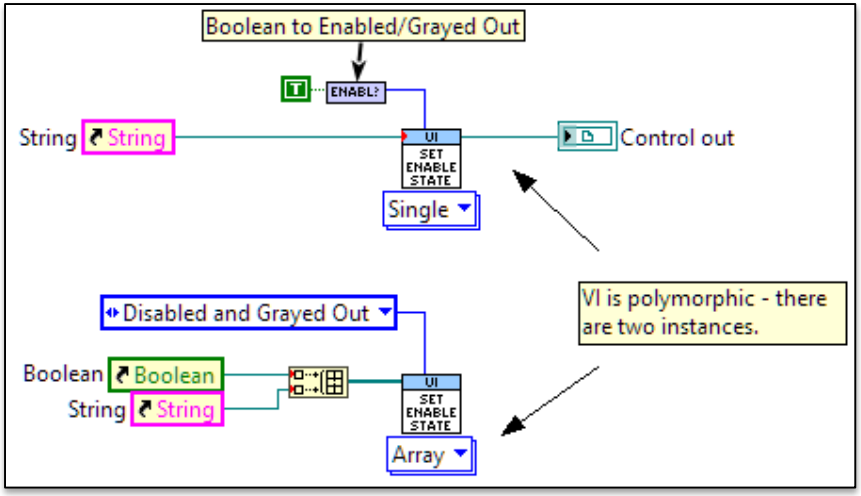
Floats

Booleans

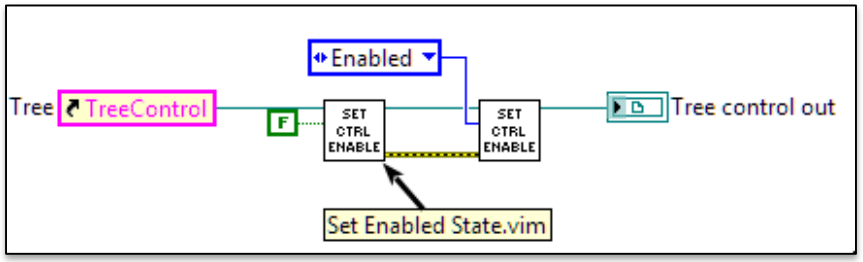
Practical Example 2: Set Enabled State.vim

- No property nodes in a Malleable VI (due to inlining)
  - But you **can** wrap into a standard SubVI!

Old UI Library

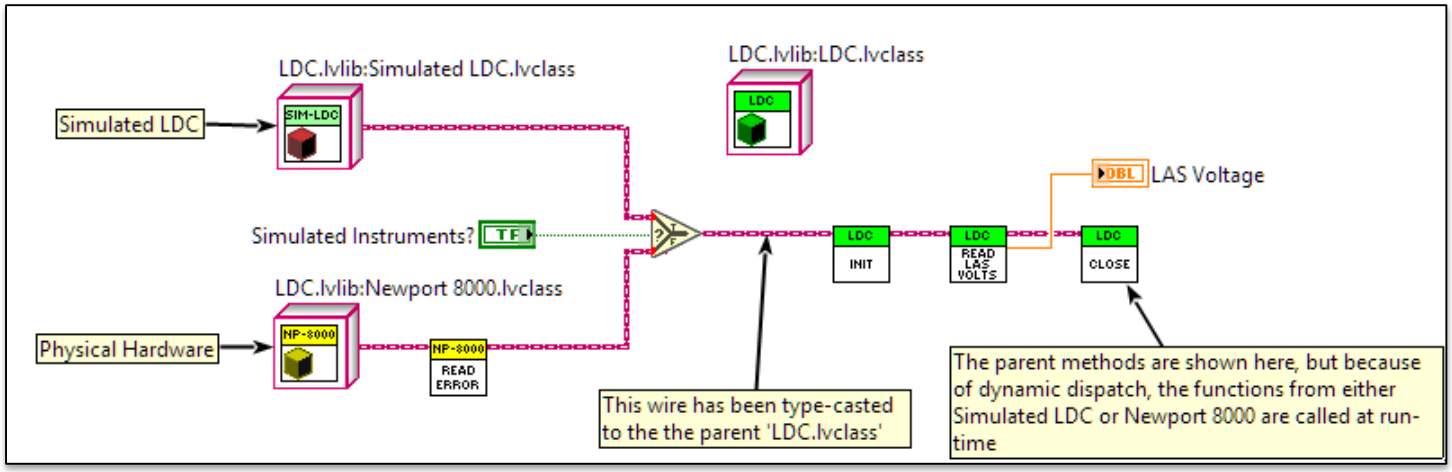


New (Demo)



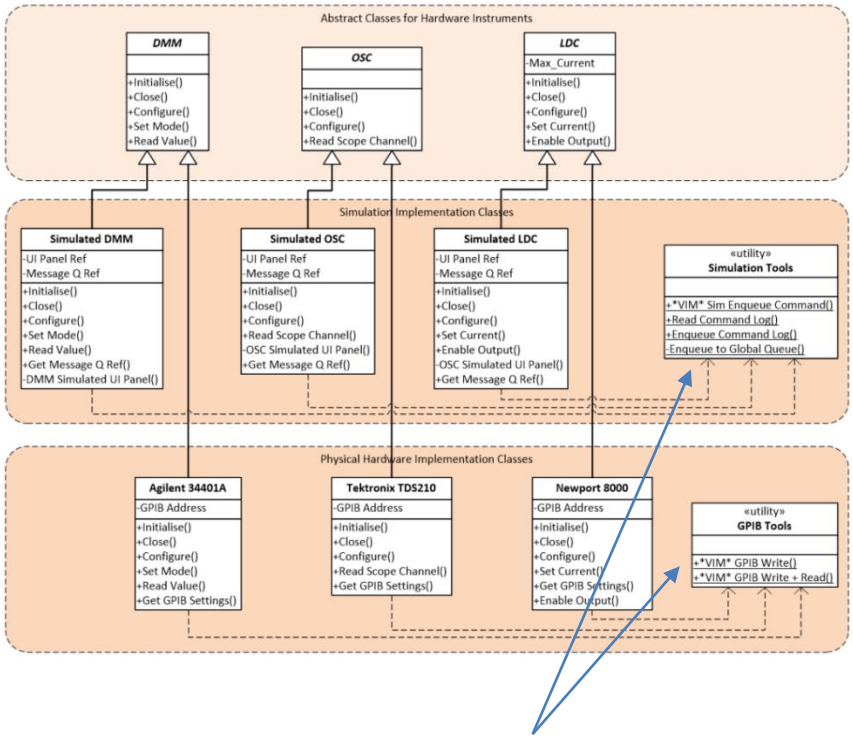
Malleables & LVOOP: Hardware Abstraction (HAL)

- If you don't currently use LabVIEW classes…
  - Do not be afraid!
  - Following examples use classes for hardware abstraction
  - Good starting point for LVOOP
  - Allows substituting of VIs at run-time using Dynamic Dispatch (e.g. Simulated vs Physical Hardware)



Practical Example 3: TestStand Project

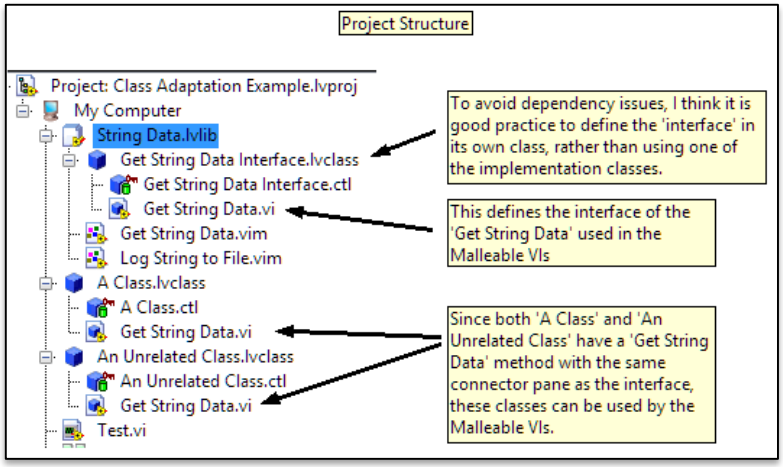
- Parametric Test Station
  - ~8 different types of measurement hardware with requirement to support 2/3 different devices
  - Working remotely, no access to hardware
  - Most hardware is GPIB, but there are some exceptions (e.g. RS-232, USB)
  - Using Hardware Abstraction for Simulated/Physical hardware
- How to implement common code between simulated devices (e.g. simulation panel) and GPIB instruments
  - CS ‘Mixin’/‘Interface’ using Malleable VIs
  - LV does not support multiple inheritance (another possible solution)

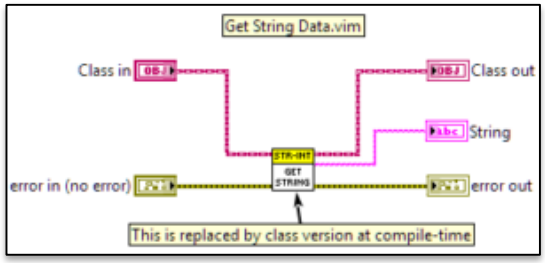


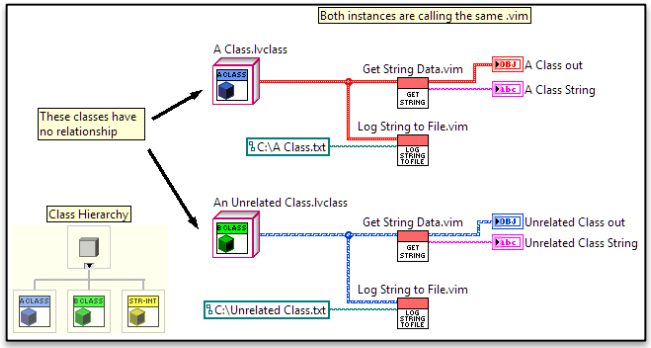
How to implement this functionality?

Malleable VI Class Adaptation

- Malleable VIs can also adapt to class wires to call methods
- Classes can be unrelated (i.e. no inheritance)
  - VI Name & Connector Pane must match
- LabVIEW 2017+ Example – HVAC System
- My Demo
  - Get String Data.vim and Log String to File.vim can be used with any class with a ‘Get String Data’ method (think Serialisation!)



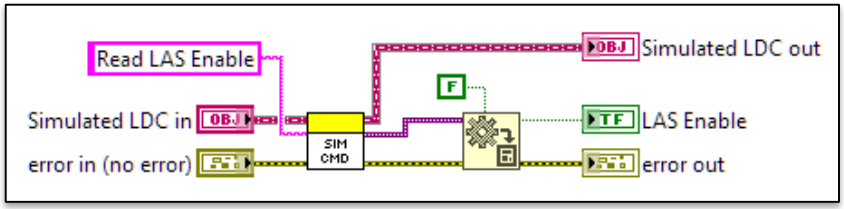




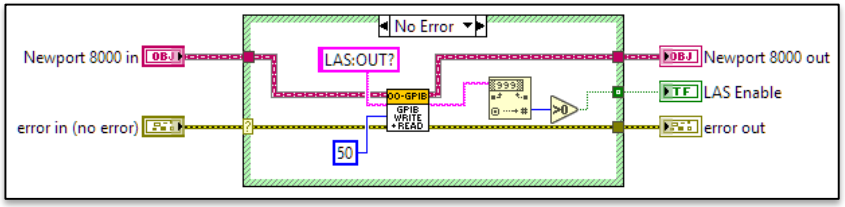
Practical Example 3: TestStand Project

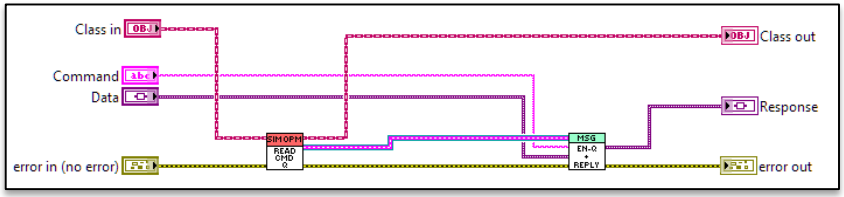
- Simulation Utility – Enqueue Command.vim
  - Sends a command to the instance of the simulation panel + waits for reply
  - Calls ‘Get Simulation Queue’ method of simulation classes
  - Used by every simulation class
- GPIB Write / GPIB Write & Read.vim
  - Message & Message+Reply GPIB communications
  - Calls ‘Get GPIB Settings’ method of hardware classes
  - Used by every GPIB instrument class

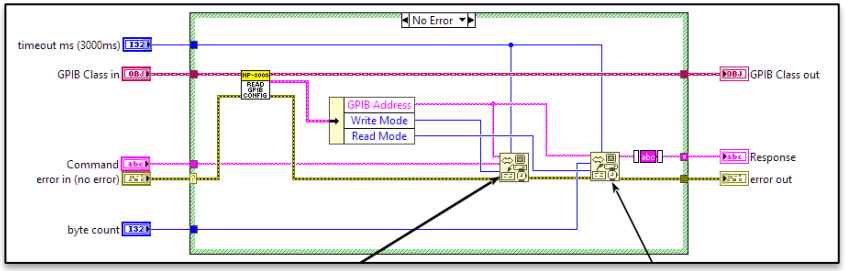
Read Laser Enable.vi - Simulated



Read Laser Enable.vi – Newport 8000 (GPIB)

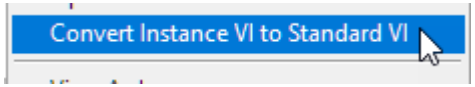






Considerations

- Requires in-lining therefore:
  - No property/invoke nodes ☹ (see workaround)
  - No automatic error handling
  - No debugging
  - No recursion…recursion…recursion…
- Malleable VIs are somewhat confusing to debug
  - Deliberately creating broken code
  - Use ‘Convert to standard instance of VI



- No run-time performance impact – Malleables are ‘flattened’ during compilation (should also work on FPGA/RT)
- ‘New Feature’ – beware potential undiscovered bugs
- If using class adaptation, suggest creating dummy interface class to avoid dependencies on your implementation classes
- Malleable VIs cannot be called directly in TestStand 2016/2017…
  - …but code modules can contain malleable VIs
  - …but beware possible deployment errors (build early + often)
- Not yet available in LabVIEW NXG

Summary

- Introduced Malleable VIs
  - Malleable VIs improve code re-use and provide a method to implement OO ‘interfaces’
- Features
  - VI that adapts to type
  - Type Specialisation Structure (TSS)
  - Class Adaptation
- Practical Examples
  - Configuration Library
  - UI Utility Library
  - TestStand Project
- Highlighted some additional considerations for their use

Thanks for listening!

Questions?

Ideas for Malleable VIs?

www.GDevCon.com