LevyLab
Instrument Framework

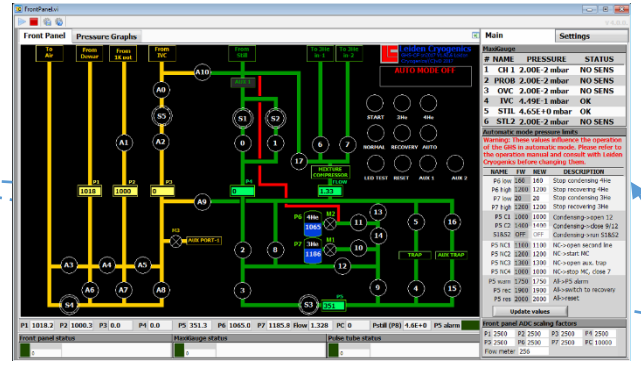
Patrick Irvin

June 29, 2020

Experiment Overview

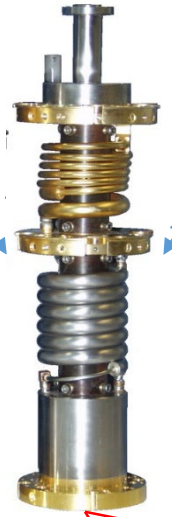
Citadel Historical Database

Front Panel (FP) Valve Controller



LabVIEW Instrument (FP)

Dilution Refrigerator (T=20mK)



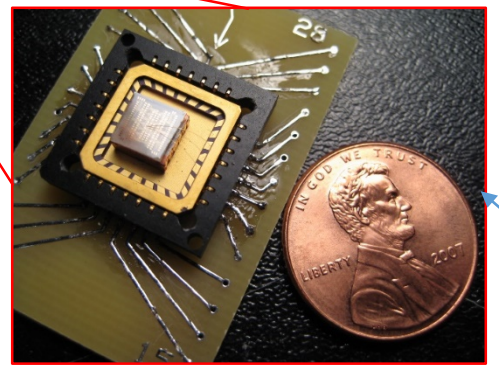
Temperature Controller (TC)

LabVIEW Instrument (TC)

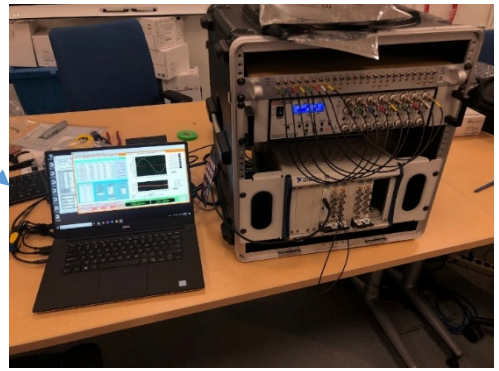
LabVIEW Instrument (PS)

Superconducting Magnet Power Supply (PS)

Superconducting Magnet



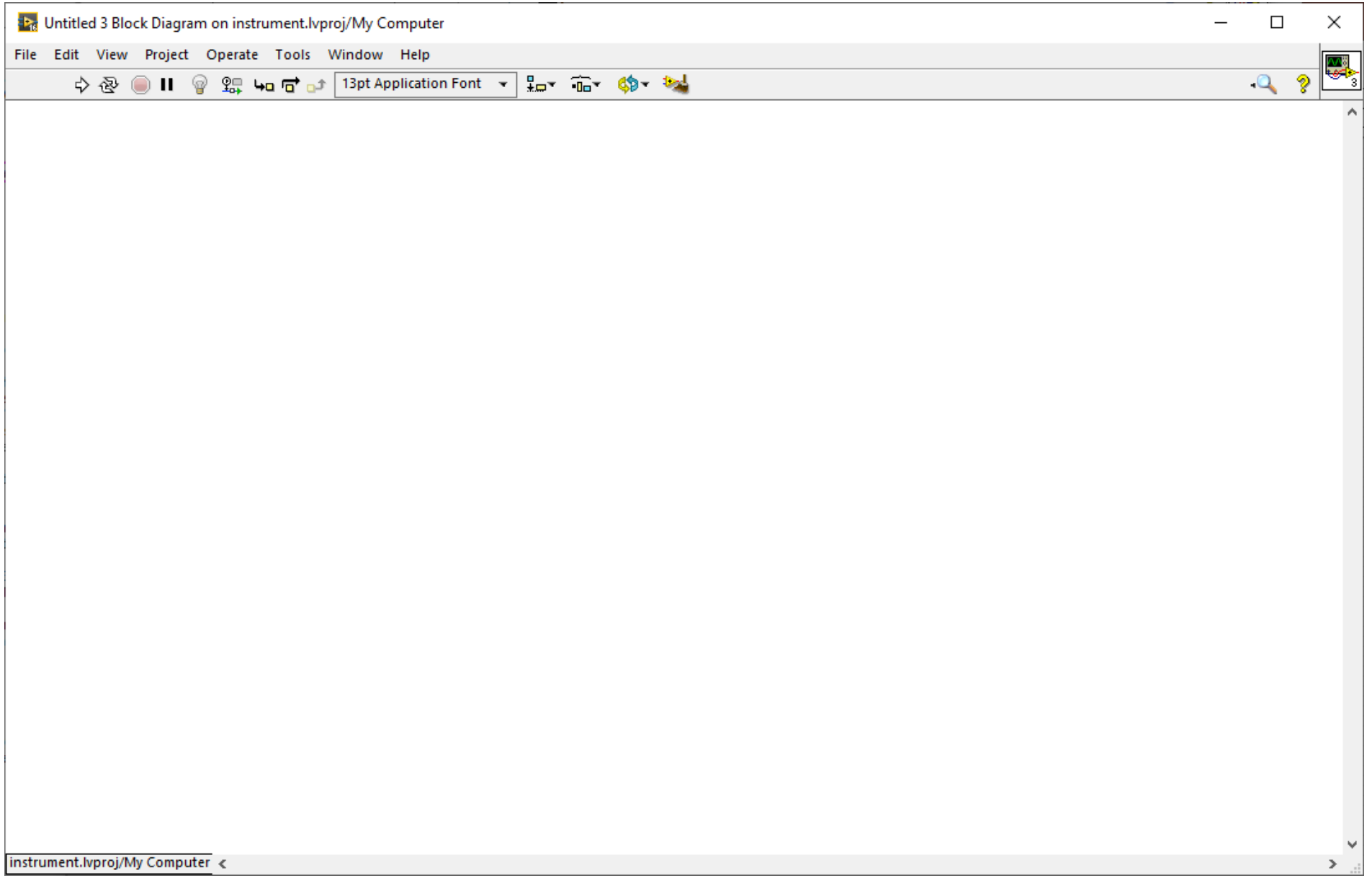
sample



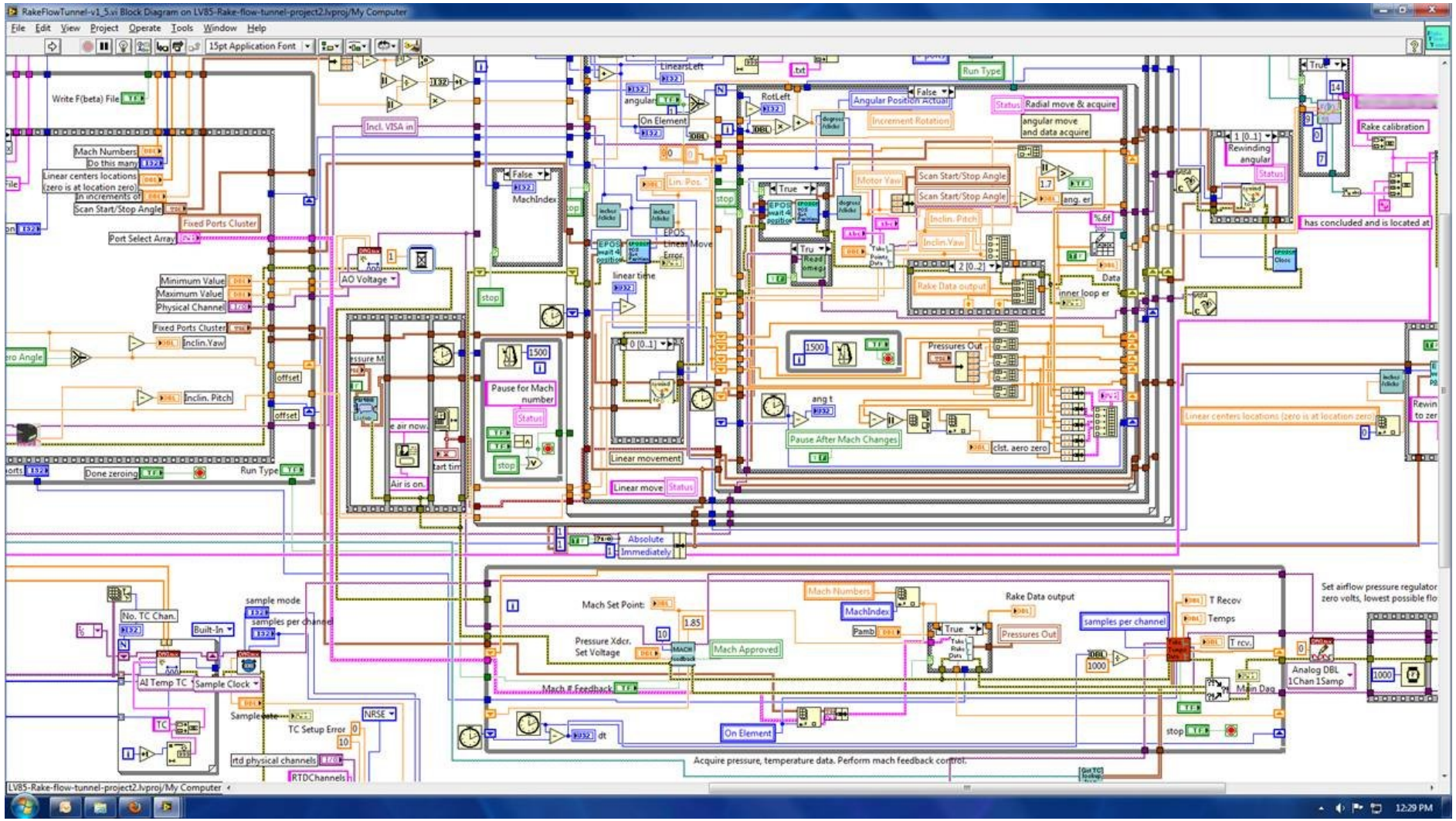
Data Acquisition (DAQ)

Software Frameworks and Design Patterns

LabVIEW

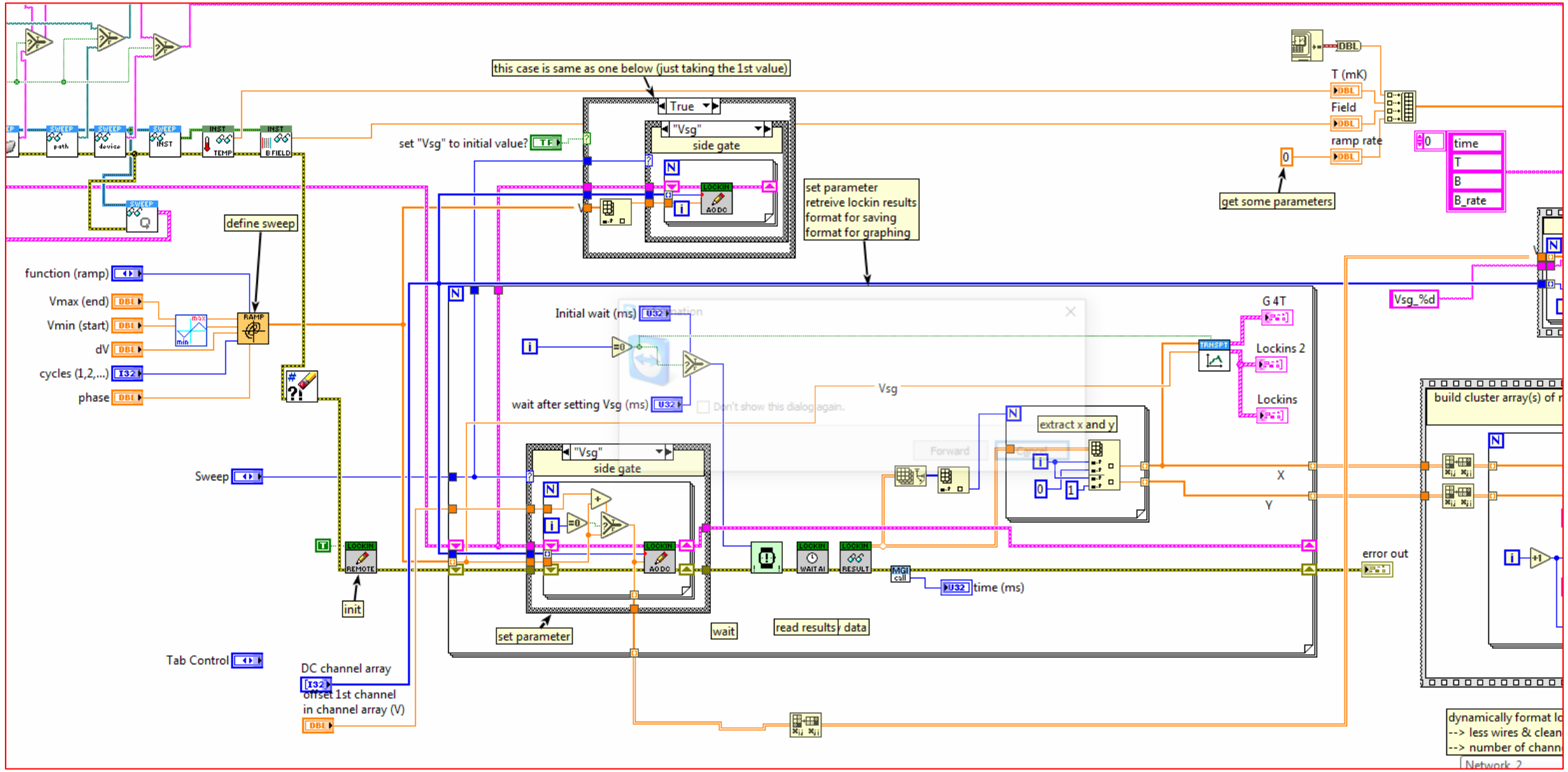

Untitled 3 Block Diagram on instrument.lvproj/My Computer
File
Edit
View
Project
Operate
Tools
Window
Help
13pt Application Font
instrument.lvproj/My Computer

LabVIEW…



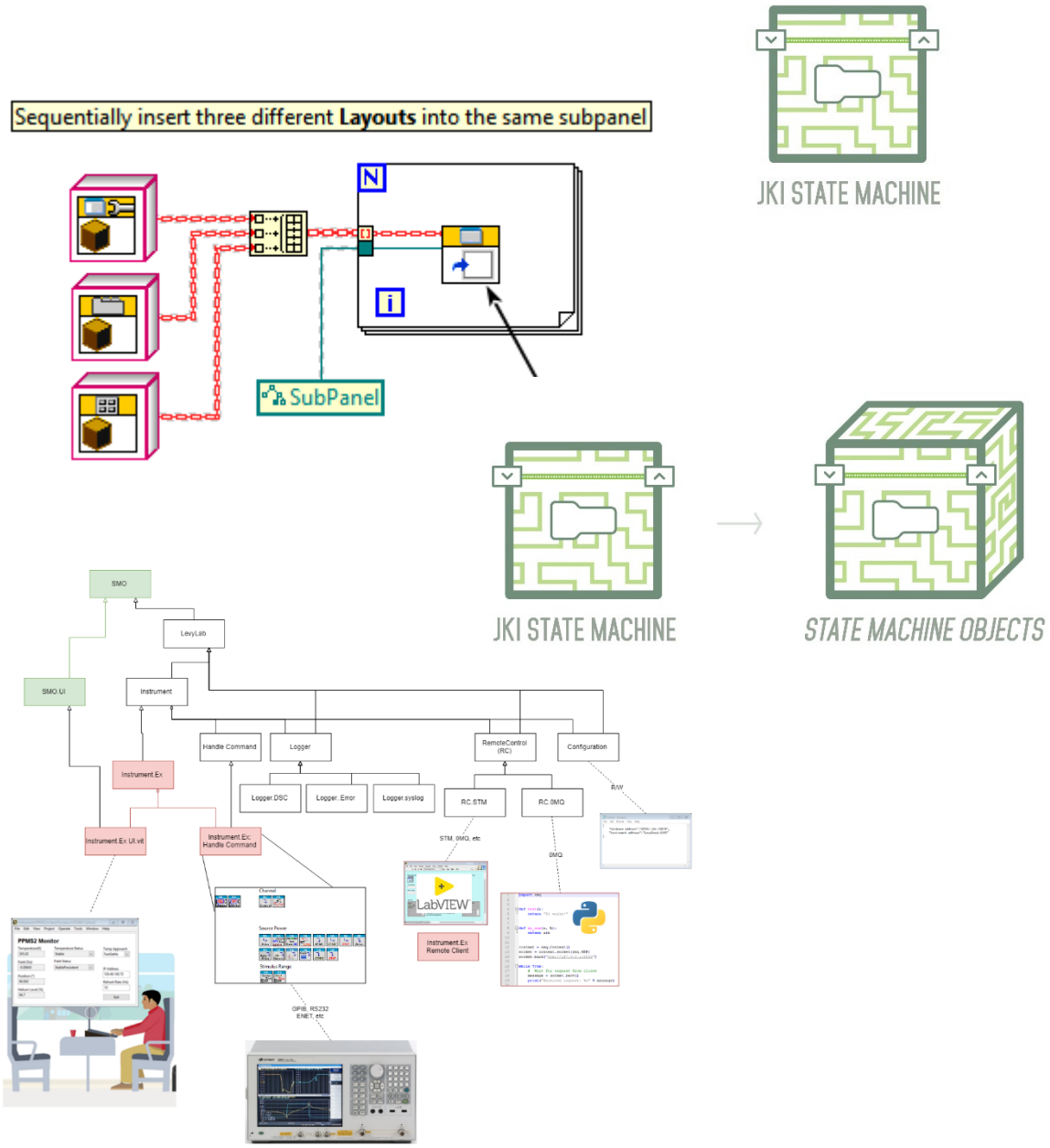
http://www.ni.com/newsletter/51735/en/

LabVIEW…


this case is same as one below (just taking the 1st value)
set "Vsg" to initial value?
True
"Vsg"
side gate
set parameter
retreive lockin results
format for saving
format for graphing
get some parameters
T (mK)
Field
ramp rate
time
T
B
B_rate
define sweep
function (ramp)
Vmax (end)
Vmin (start)
dV
cycles (1,2,...)
phase
Initial wait (ms)
wait after setting Vsg (ms)
Vsg
extract x and y
G 4T
Lockins 2
Lockins
X
Y
Vsg_%d
Sweep
init
set parameter
wait
read results / data
time (ms)
error out
build cluster array(s) of r
Tab Control
DC channel array
offset 1st channel
in channel array (V)
dynamically format lo
--> less wires & clean
--> number of chann
Network 2

Software Frameworks Can Help Us!

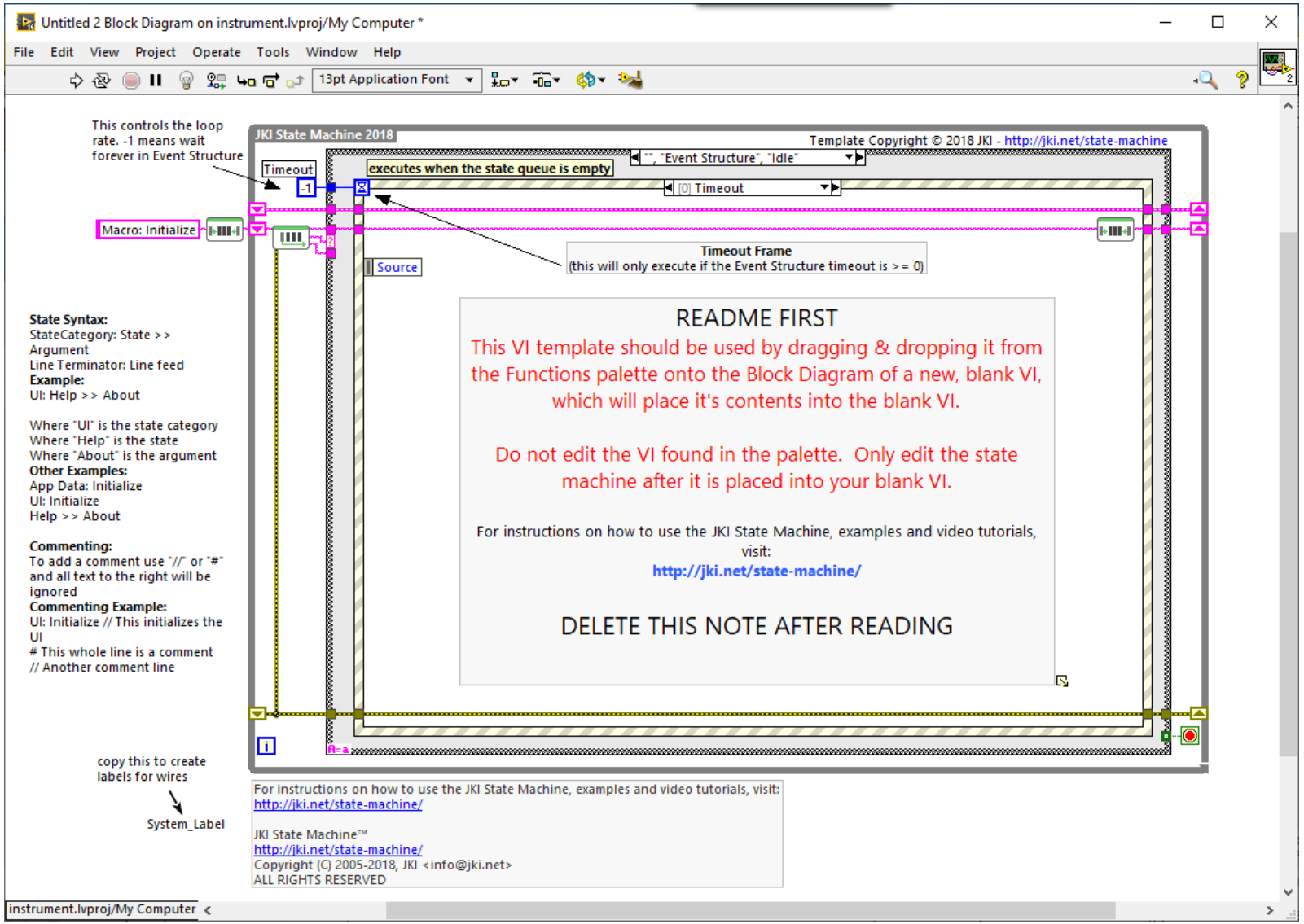
- **JKI State Machine**: A powerful state machine design template
- **Object Oriented Design**: An approach to designing a system of interacting objects
- **State Machine Objects**: Turn JKI State Machines into asynchronous objects!
- **LevyLab Instrument.lvclass**: A framework using JKI State Machines, SMO, that solves specific needs of LevyLab Research



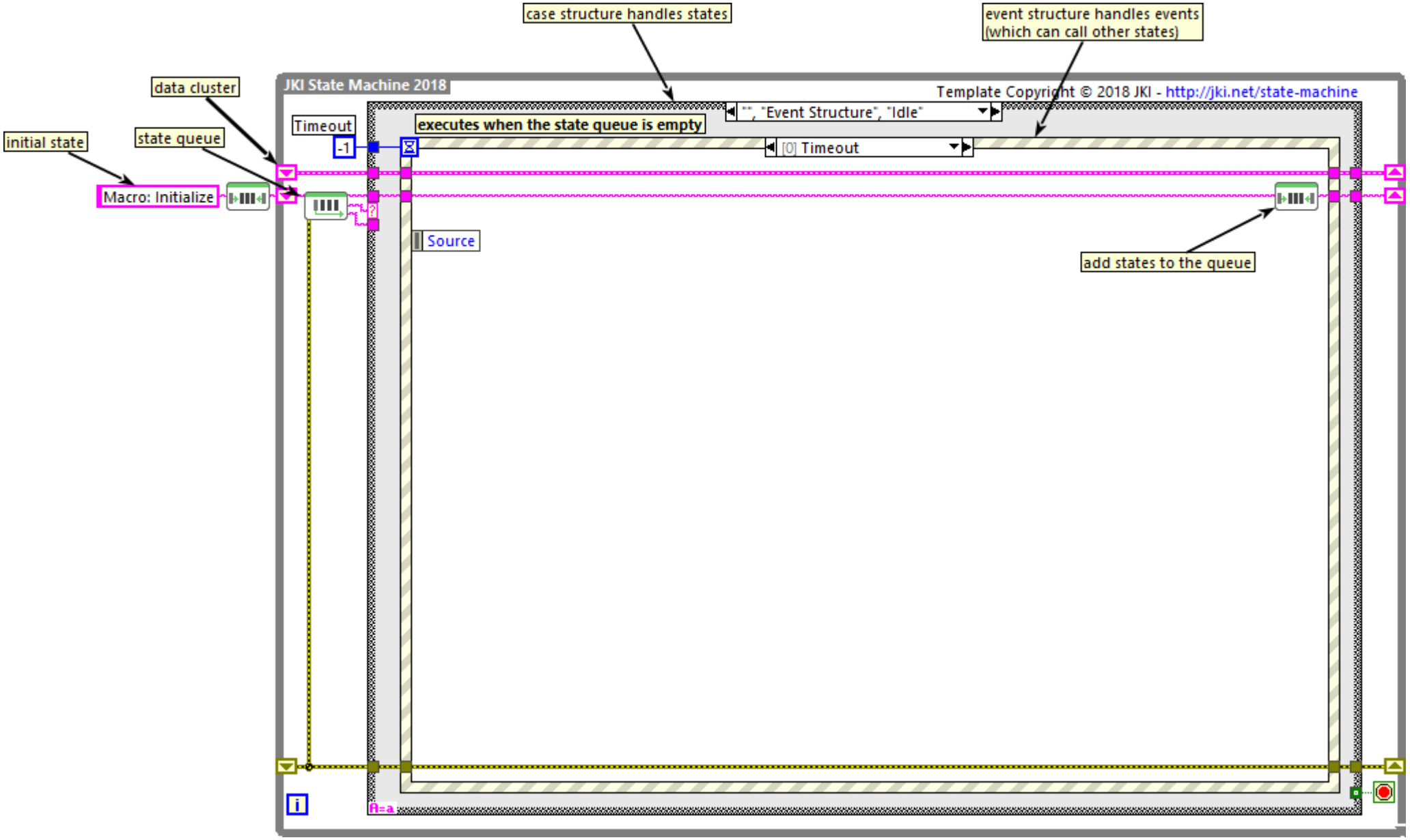
Software Frameworks and Design Patterns

JKI State Machine

JKI State Machine

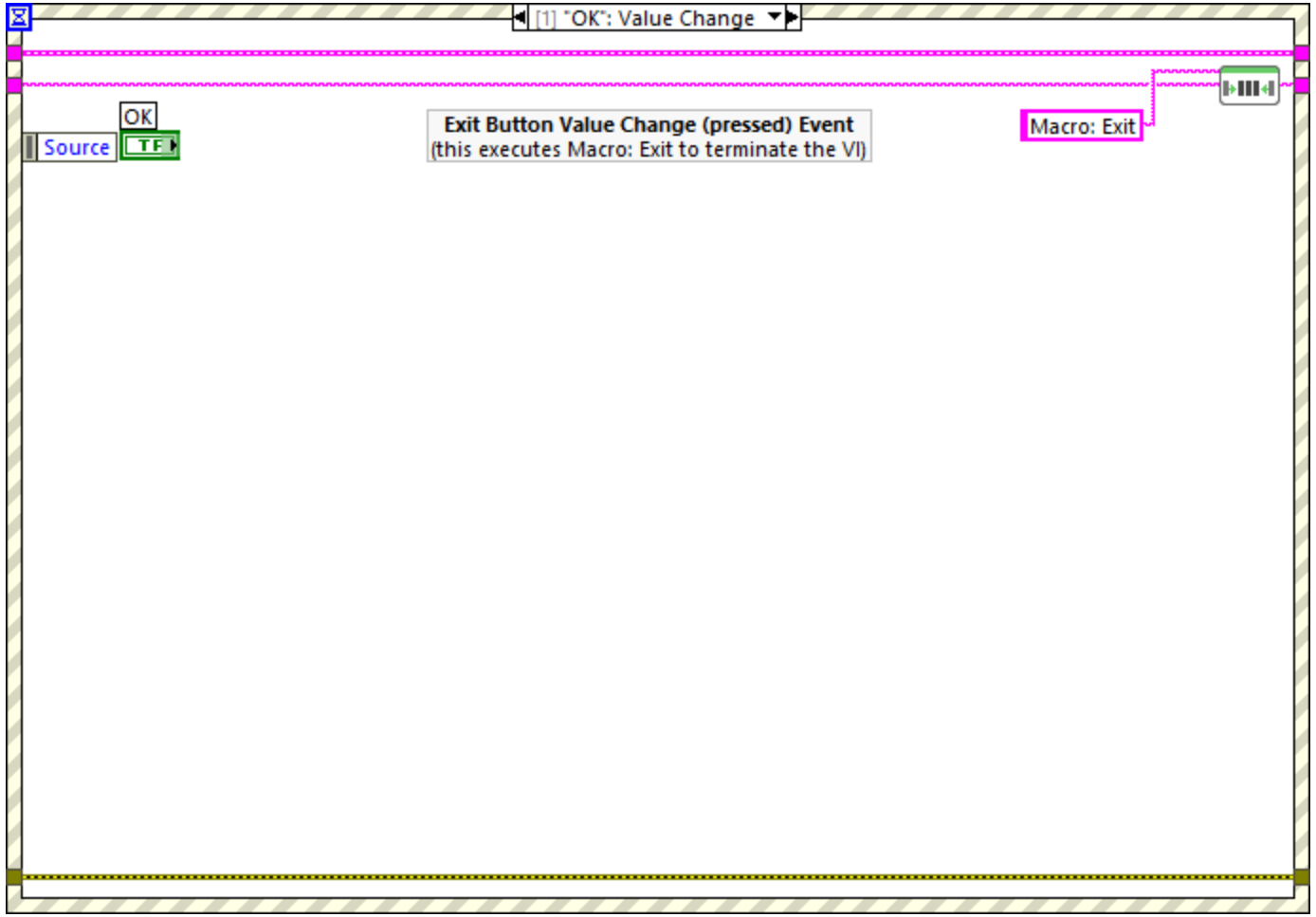


JKI State Machine



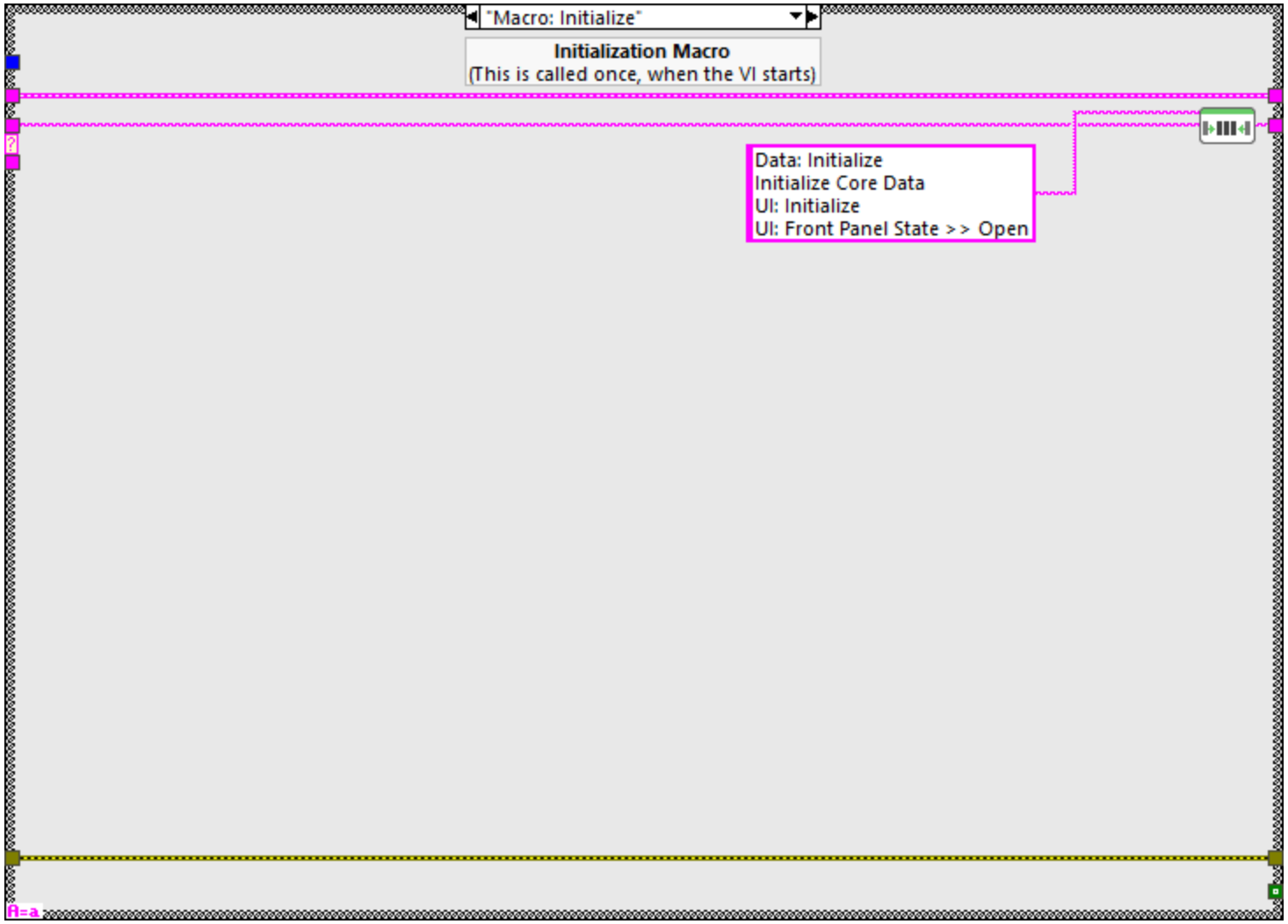
JKI State Machine

Event example:

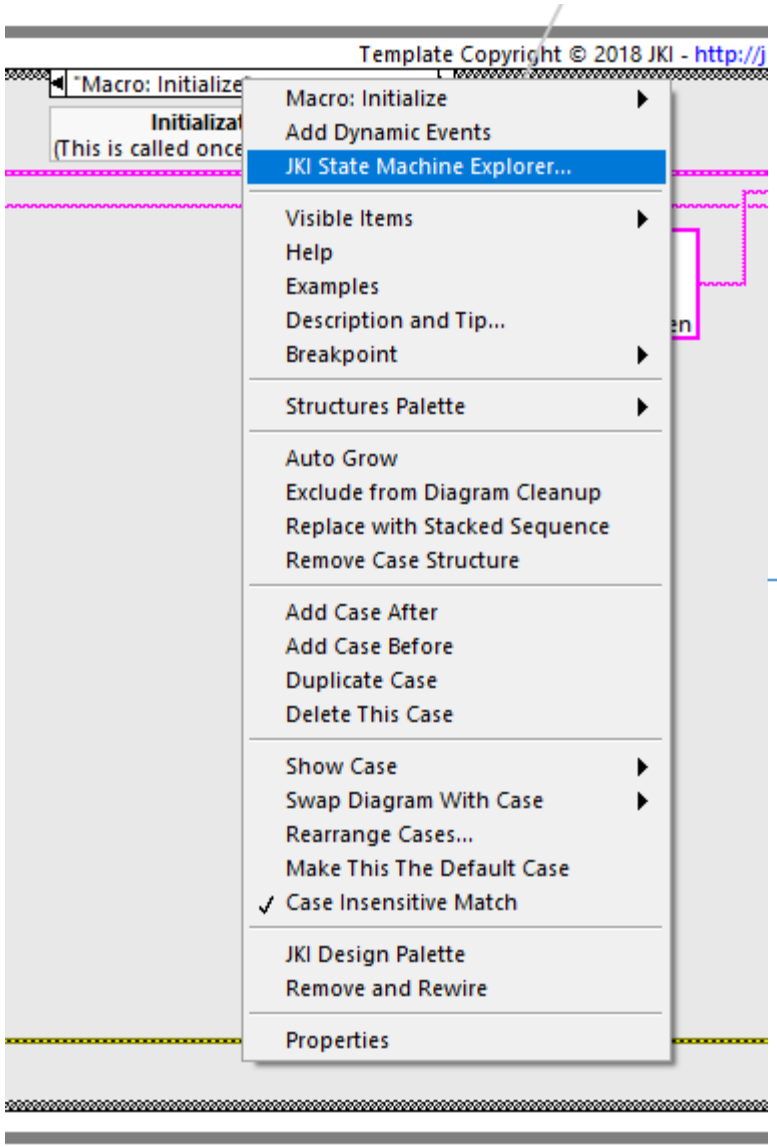


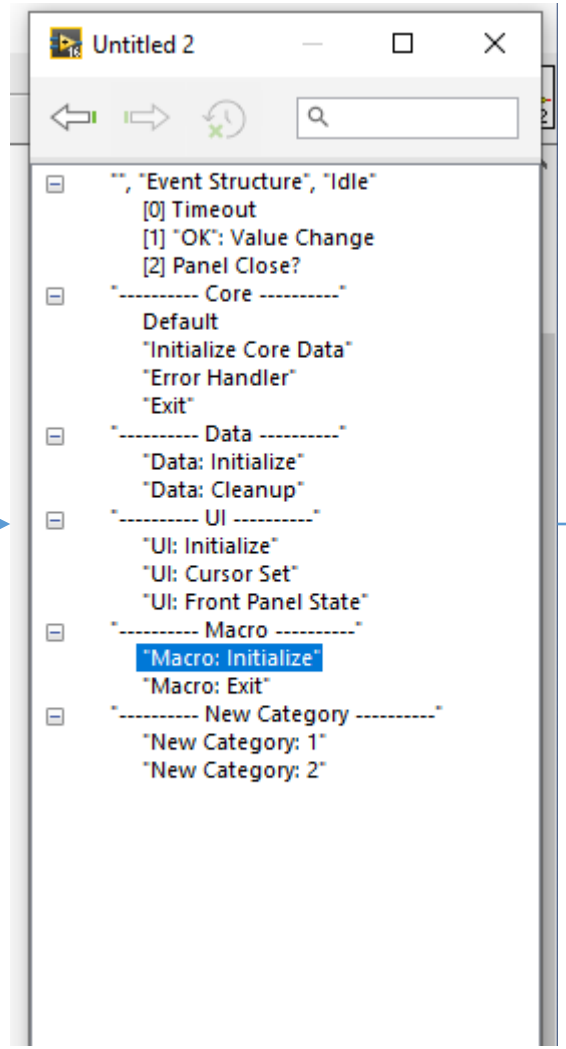
JKI State Machine

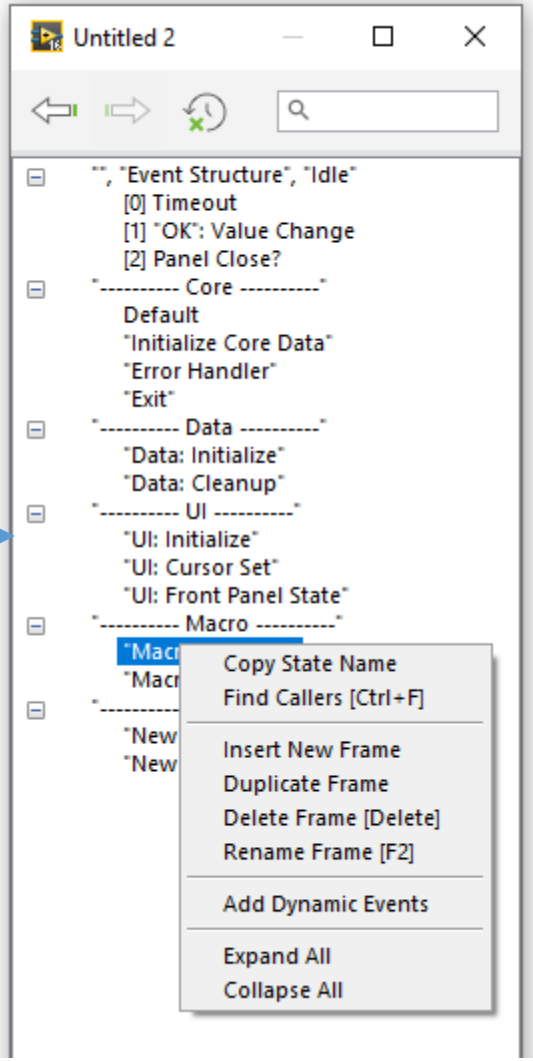
State example:



JKI State Machine Explorer







JKI State Machine Best Practices

JKI State Machine Best Practices
http://blog.jki.net/products/state-machine/jki-state-machine-best-practices

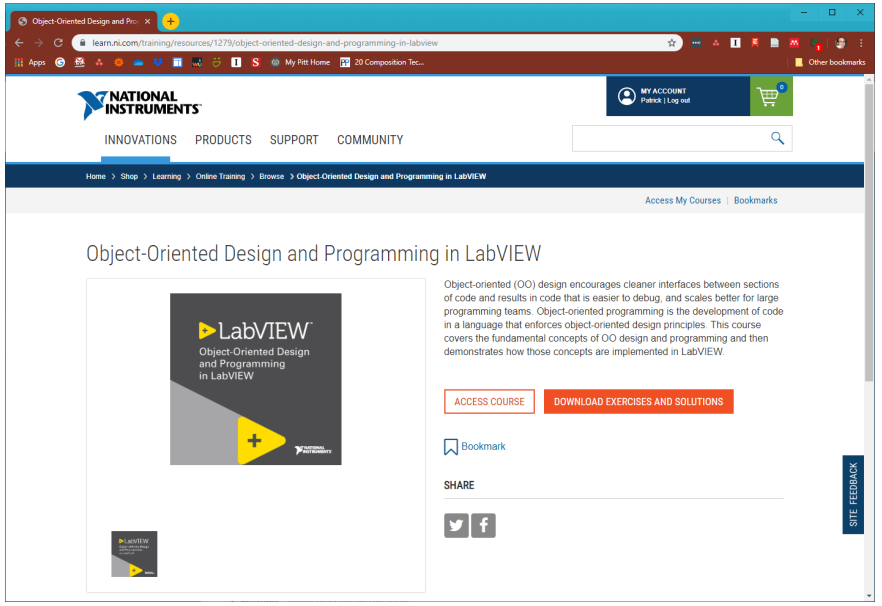
1. Don’t hide your state strings in subVIs
2. Don’t add code and logic inside the Event Structure
3. Keep the Original Size (i.e. don’t grow the structures)
4. Use macros instead of “chaining” together sequential states
5. Left-justify State Strings instead of Right-justify

Software Frameworks and Design Patterns

Object Oriented Programming with LabVIEW

Object Oriented Programming LabVIEW

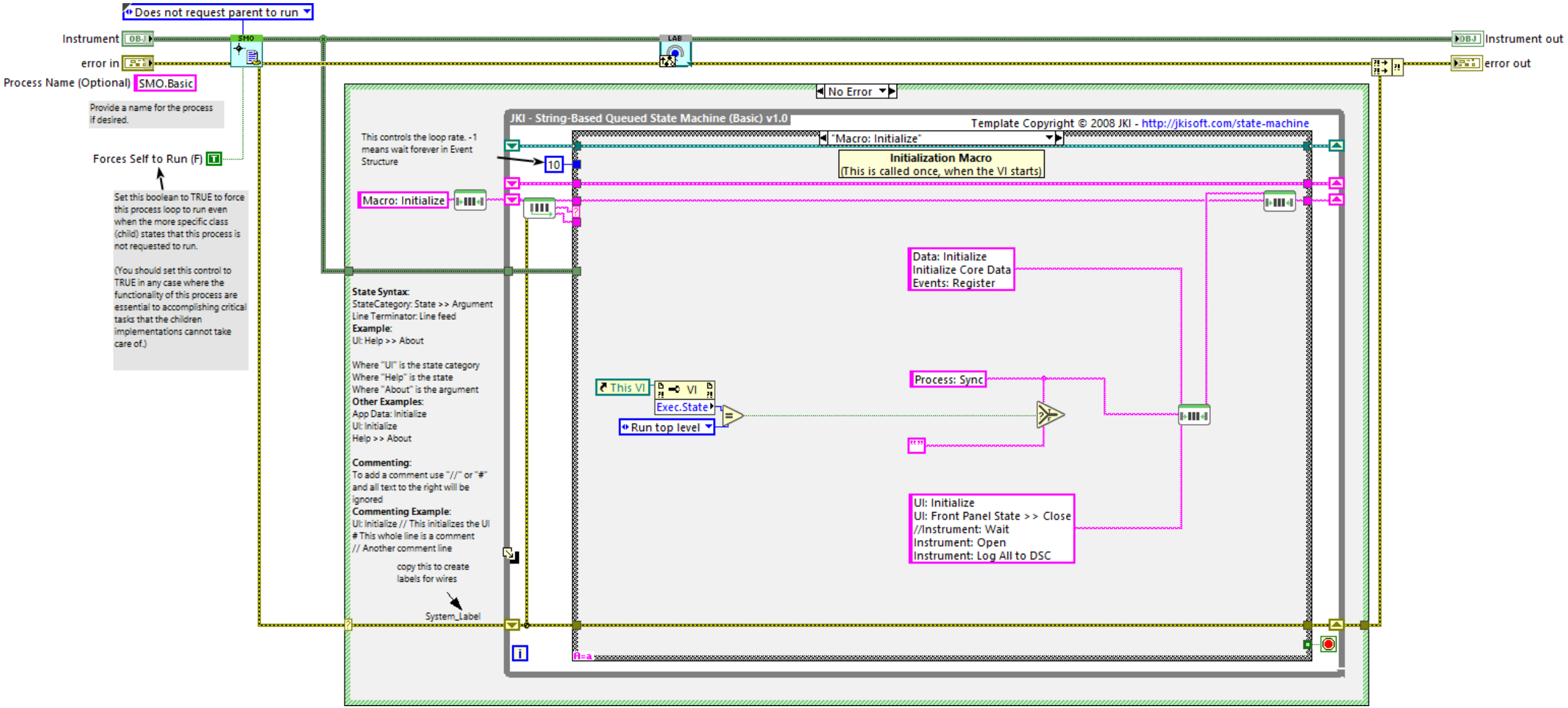
- NI OO Course
- https://www.bloomy.com/support/blog/object-oriented-labview-inheritance-part-1-3-part-series
- https://www.bloomy.com/support/blog/object-oriented-labview-encapsulation-part-2-3-part-series
- https://www.bloomy.com/support/blog/object-oriented-labview-polymorphism-part-3-3-part-series



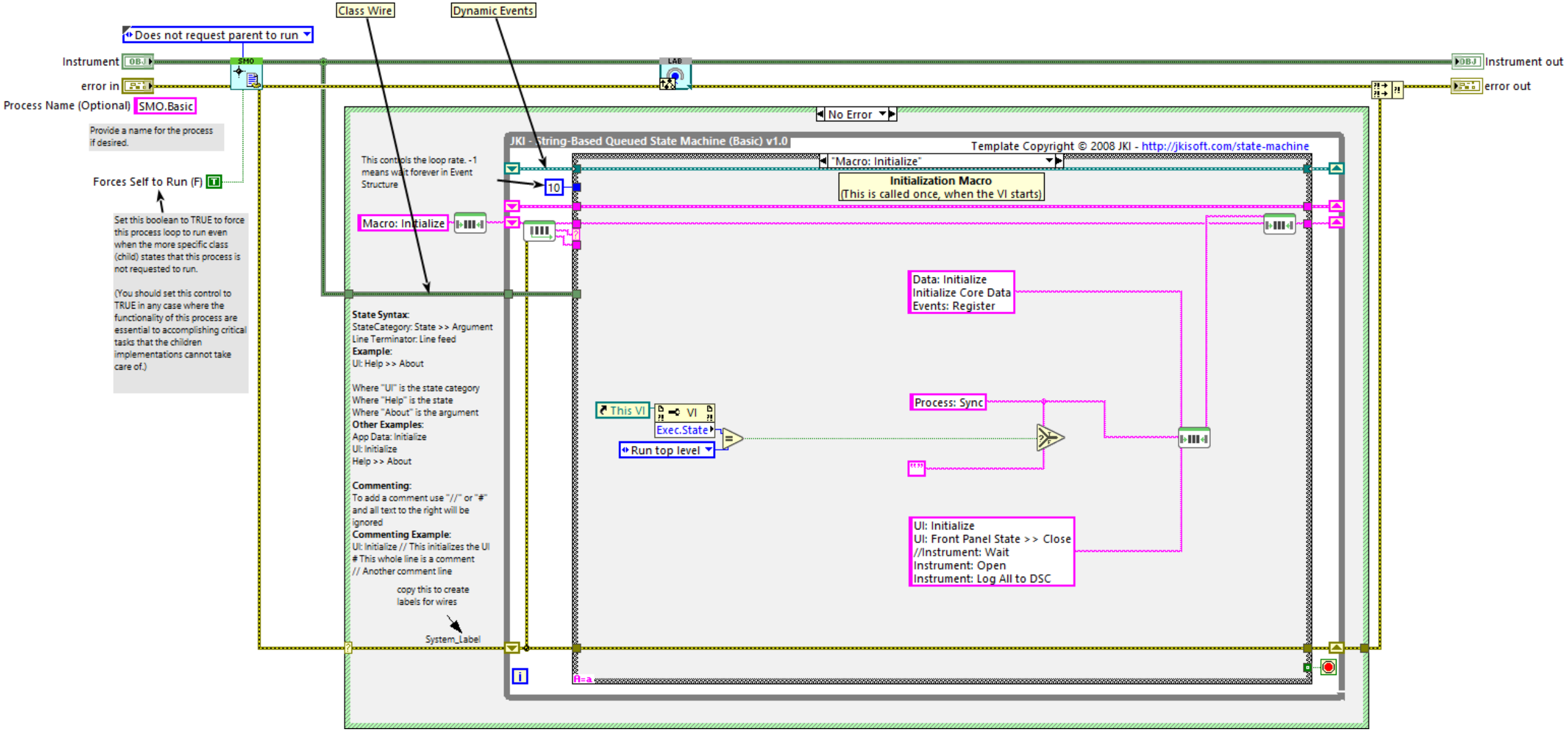
Software Frameworks and Design Patterns

State Machine Objects

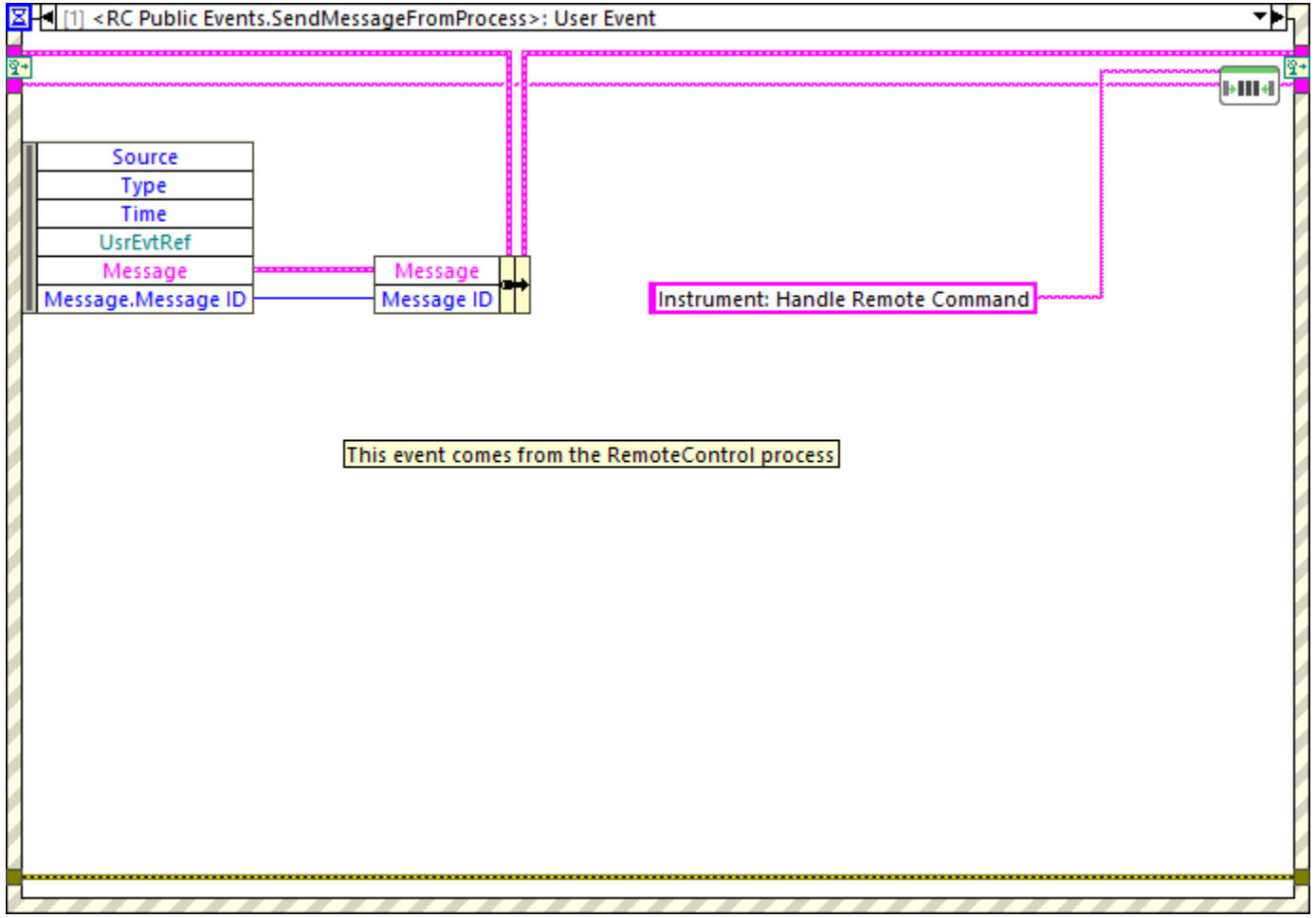
State Machine Objects



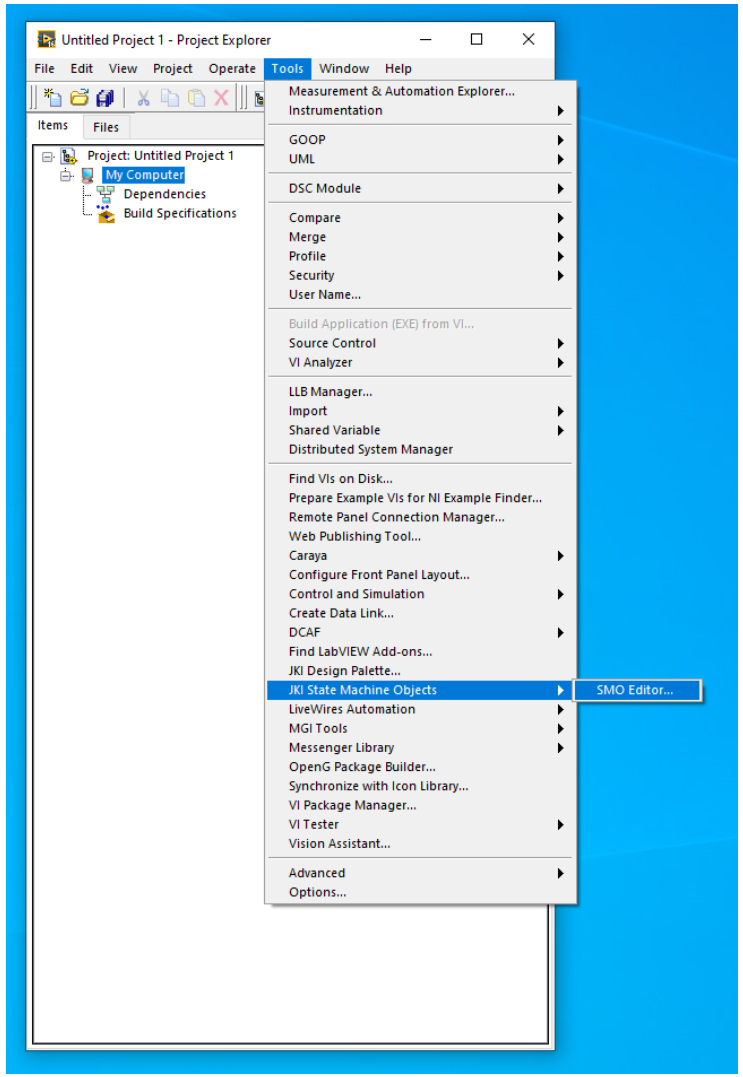
State Machine Objects



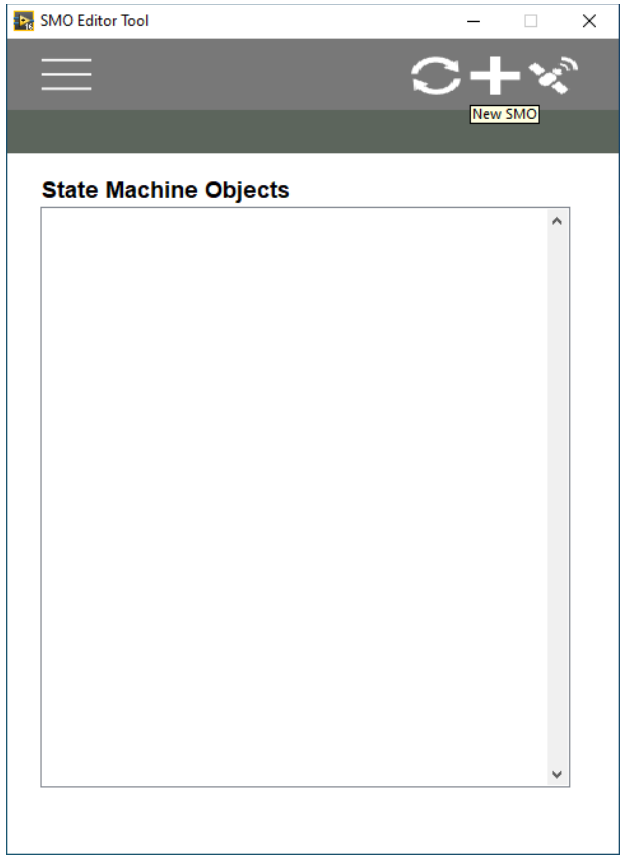
State Machine Objects: Dynamic Events



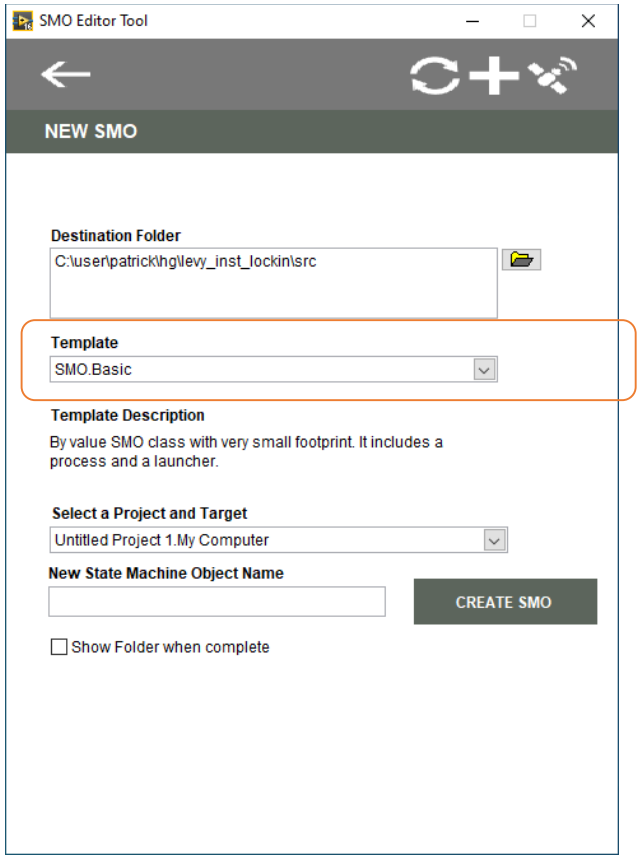
Awesome! How do I make an SMO?



1. Open the SMO Editor

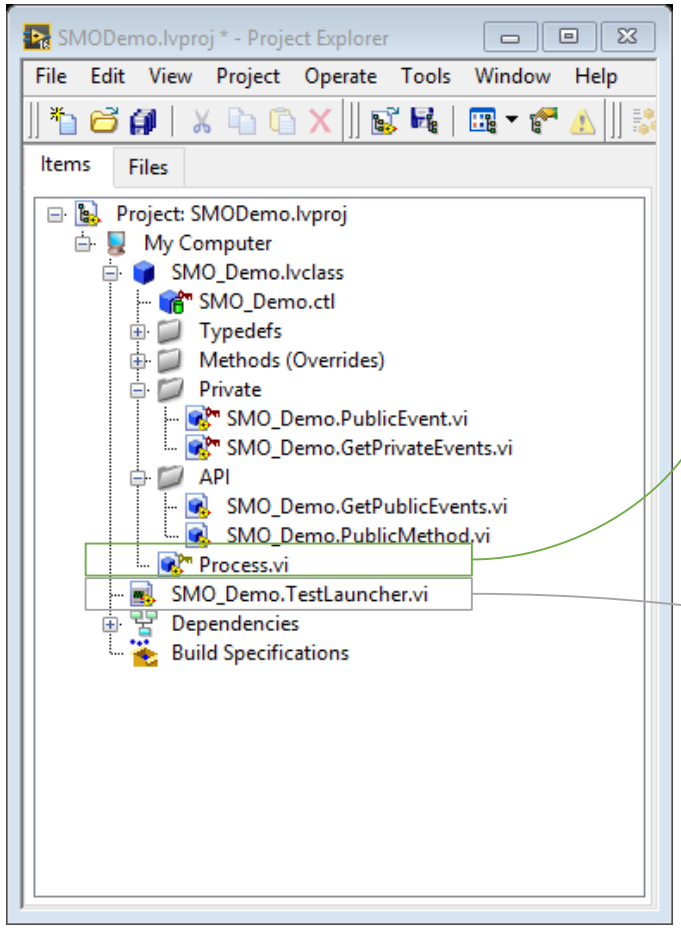


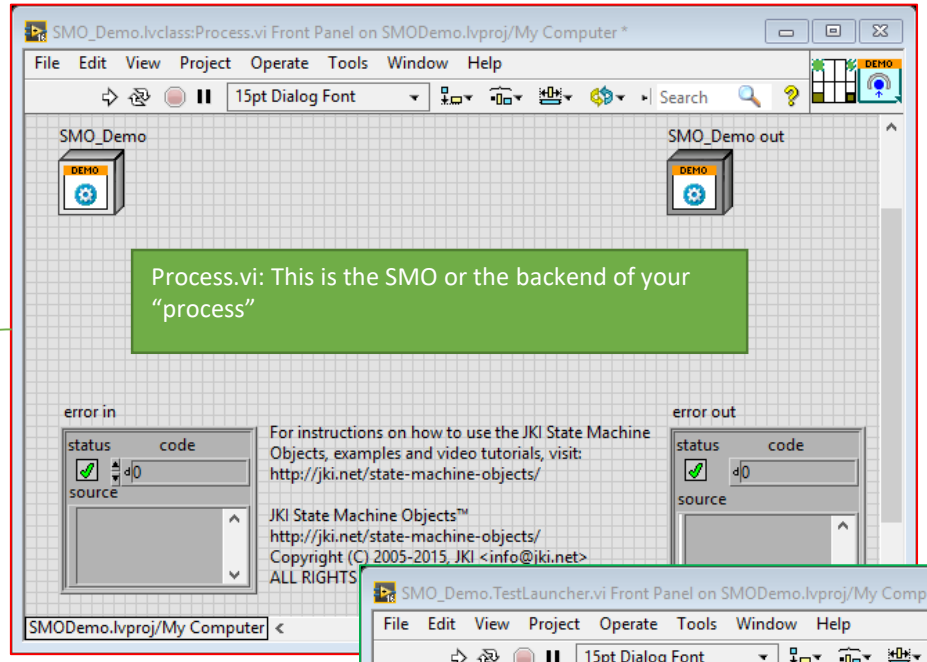
2. Click “New SMO”

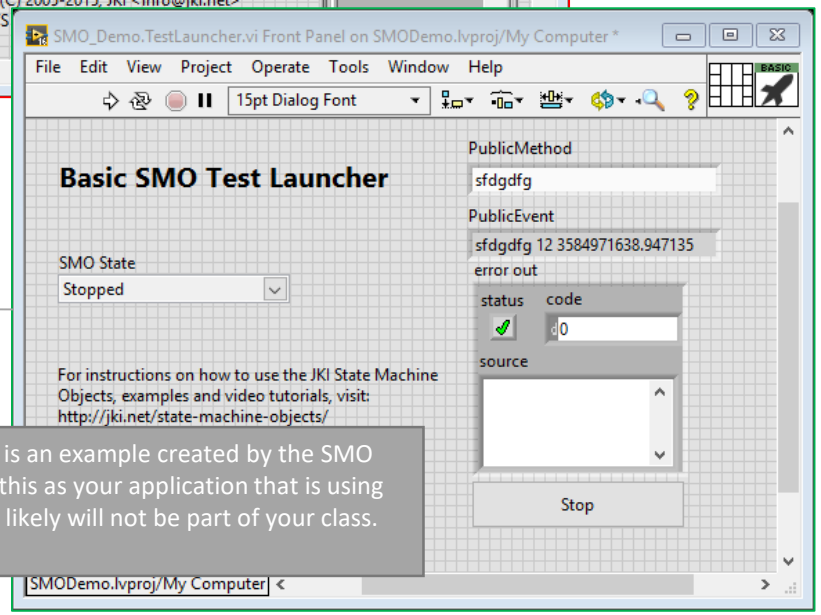


3. Choose SMO Type

SMO Intro 1

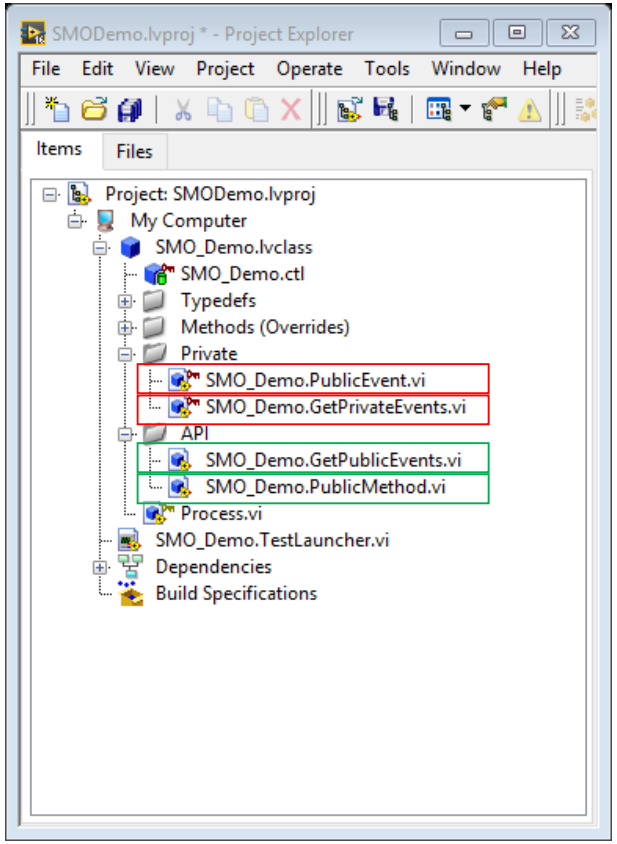






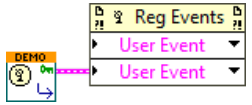
TestLauncher.vi is an example created by the SMO Editor. Think of this as your application that is using the SMO, which likely will not be part of your class.

SMO Intro 2a



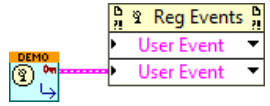
Startup:

Test Launcher



Register for public events

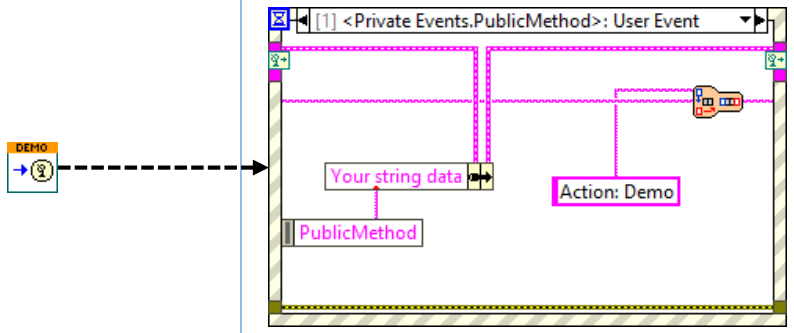
Process.vi



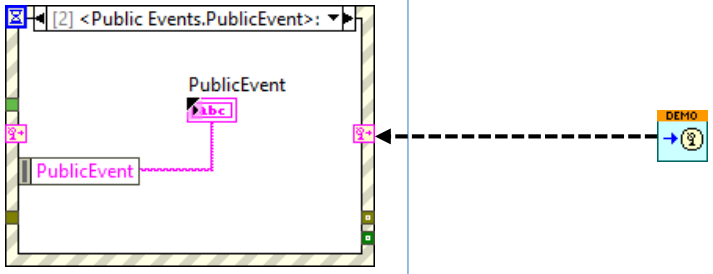
Register for private events

During Execution:

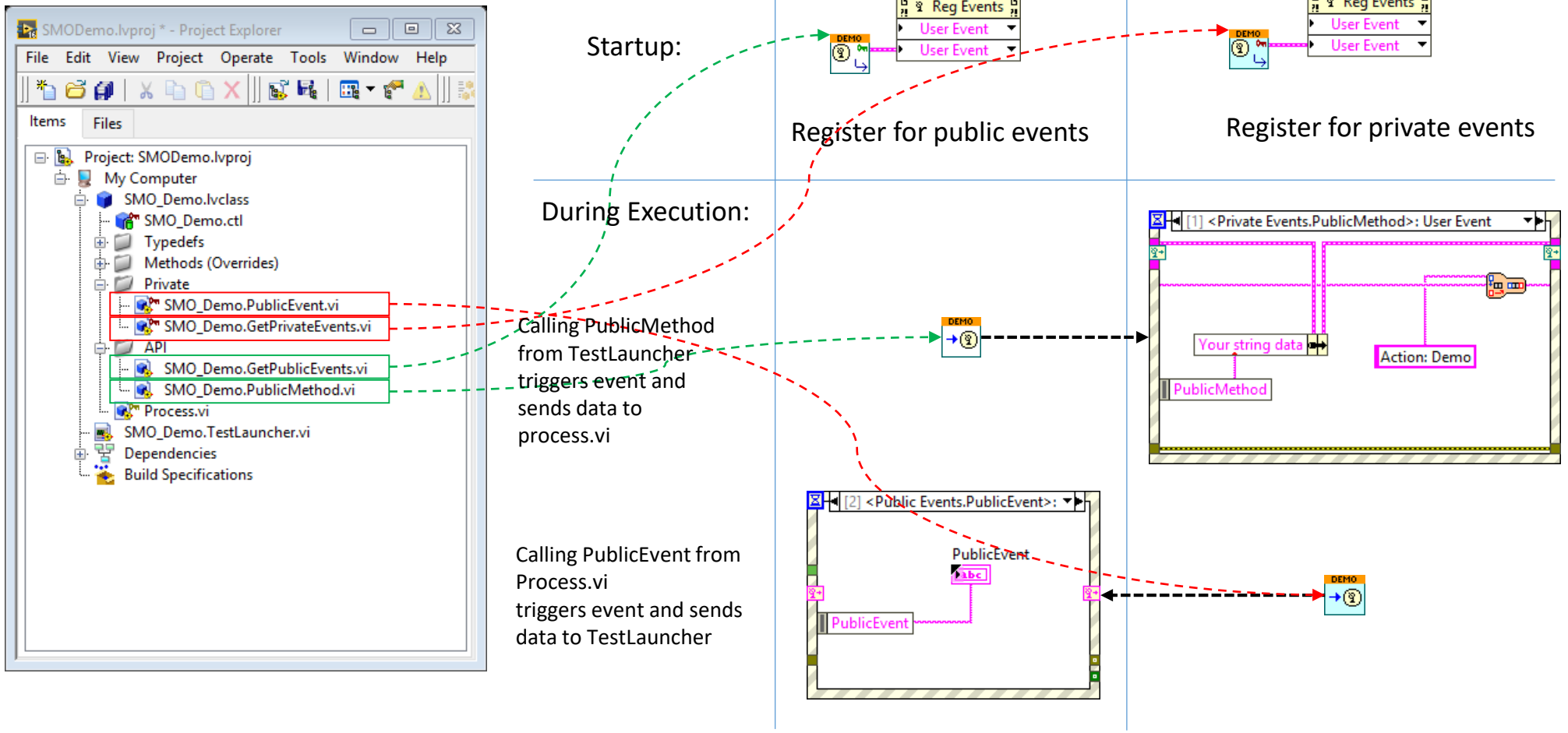
Calling PublicMethod from TestLauncher triggers event and sends data to process.vi



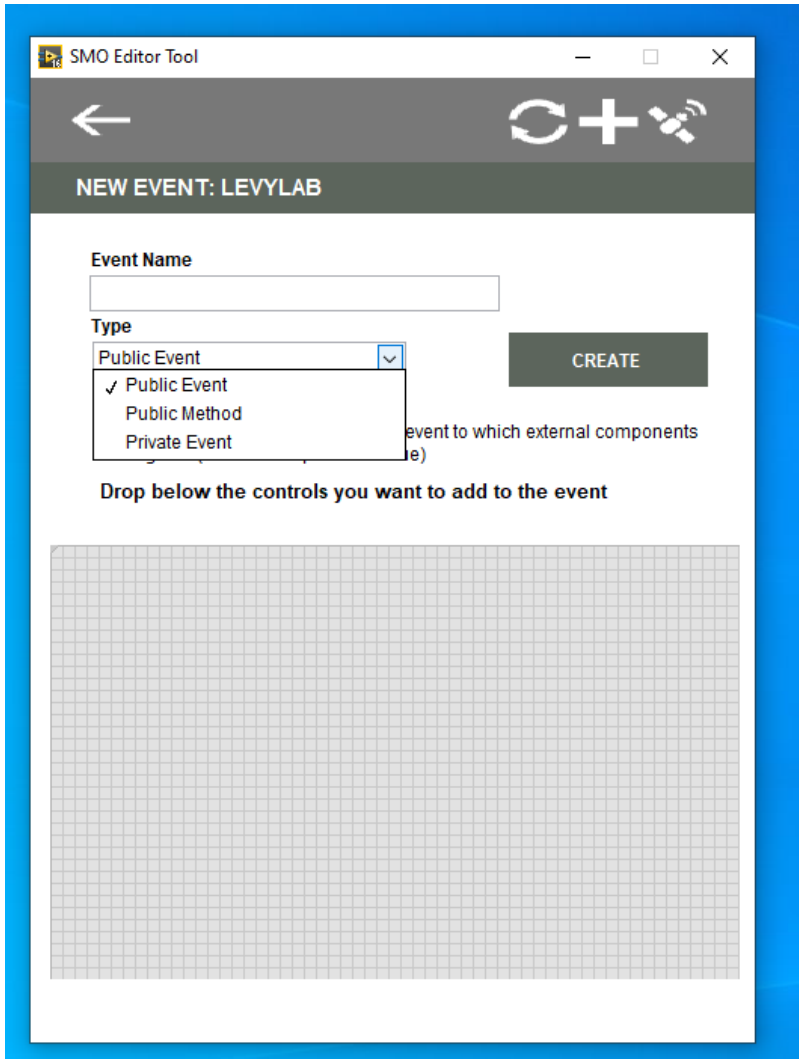
Calling PublicEvent from Process.vi triggers event and sends data to TestLauncher



SMO Intro 2b

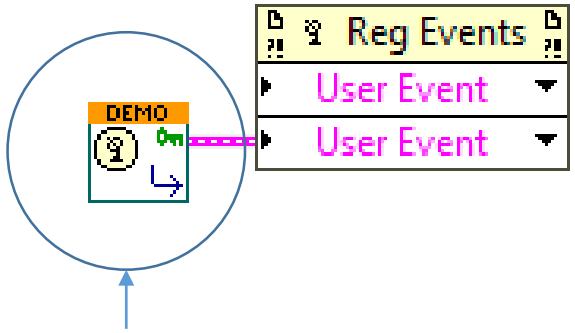

SMODemo.lvproj * - Project Explorer
File Edit View Project Operate Tools Window Help
Items Files
Project: SMODemo.lvproj
My Computer
SMO_Demo.lvclass
SMO_Demo.ctl
Typedefs
Methods (Overrides)
Private
SMO_Demo.PublicEvent.vi
SMO_Demo.GetPrivateEvents.vi
API
SMO_Demo.GetPublicEvents.vi
SMO_Demo.PublicMethod.vi
Process.vi
SMO_Demo.TestLauncher.vi
Dependencies
Build Specifications
Test Launcher
Process.vi
Startup:
Reg Events
User Event
User Event
Register for public events
Register for private events
During Execution:
Calling PublicMethod from TestLauncher triggers event and sends data to process.vi
[1] <Private Events.PublicMethod>: User Event
Your string data
Action: Demo
PublicMethod
Calling PublicEvent from Process.vi triggers event and sends data to TestLauncher
[2] <Public Events.PublicEvent>:
PublicEvent
PublicEvent
DEMO

Use SMO Editor to create Public Events and Public Methods

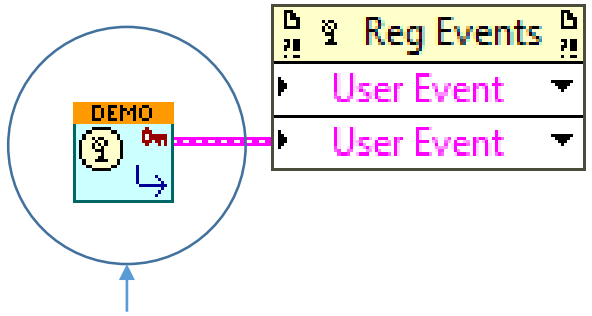


SMO Intro 3

Change your icons to make them useful!

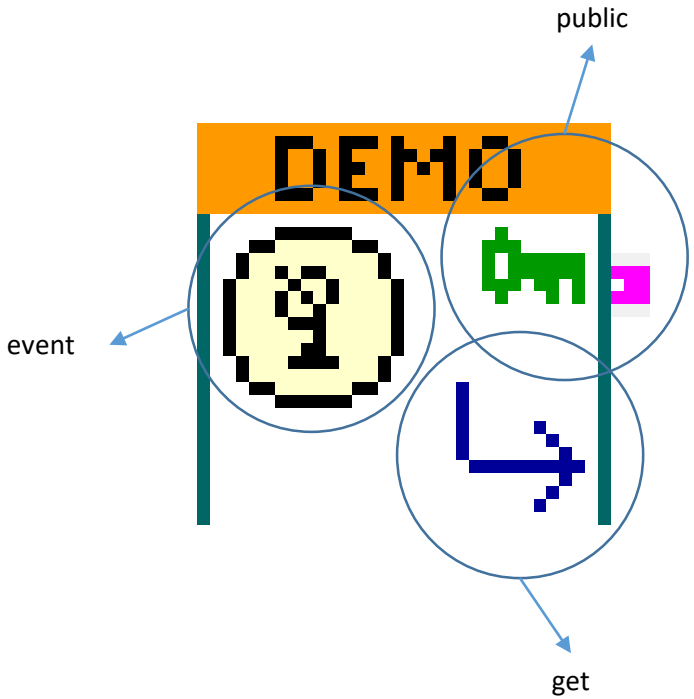


White background = public method



Blue background = private method

Use glyphs provided by LabVIEW to make your life easy!



SMO Resources

- JKI State Machine Objects on GitHub
  - Getting Started with JKI's State Machine Objects Framework
- Patrick's Short Introduction to JKI SMO
- LevyLab SMO Tutorials

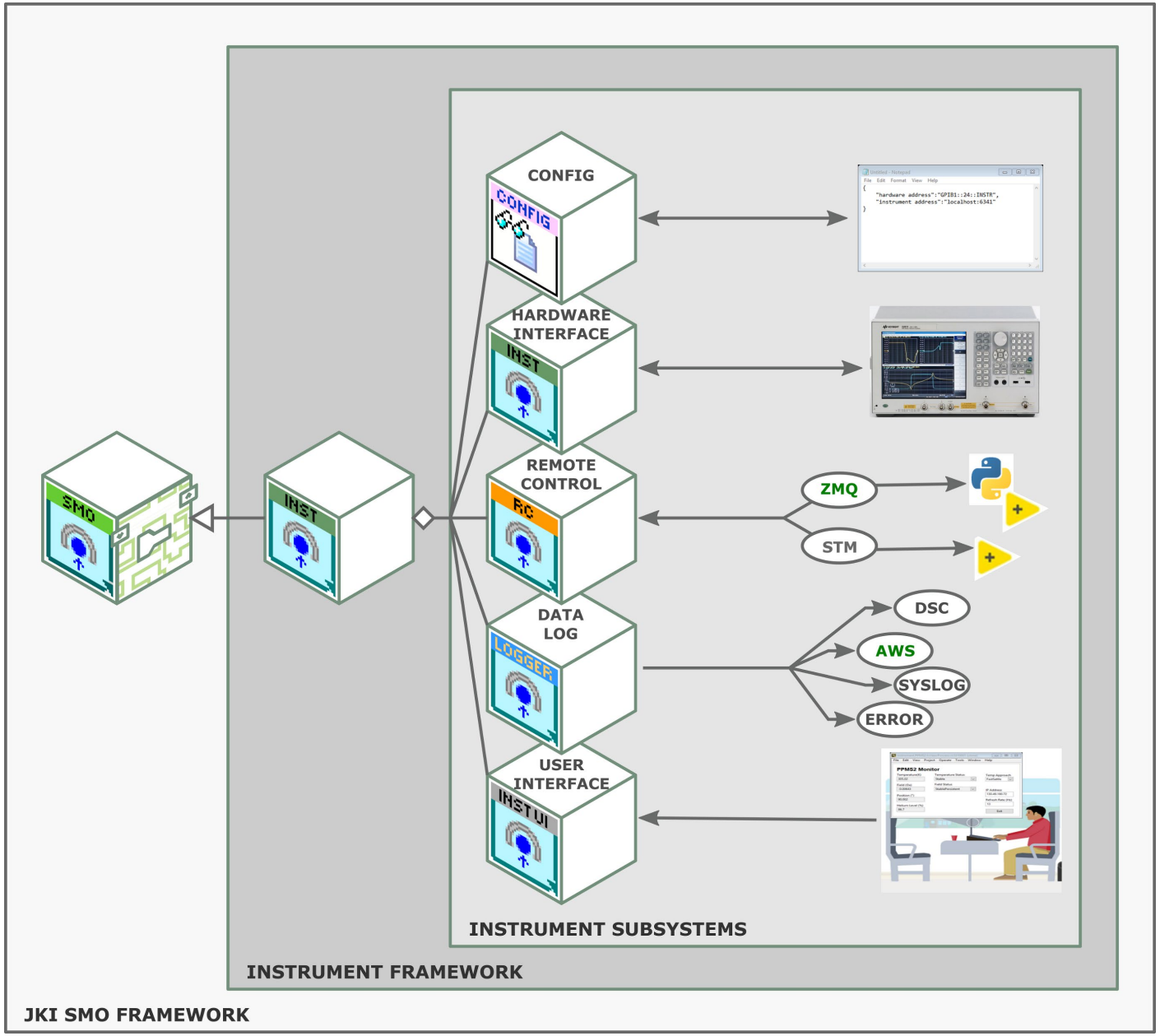
Software Frameworks and Design Patterns

Instrument.lvclass

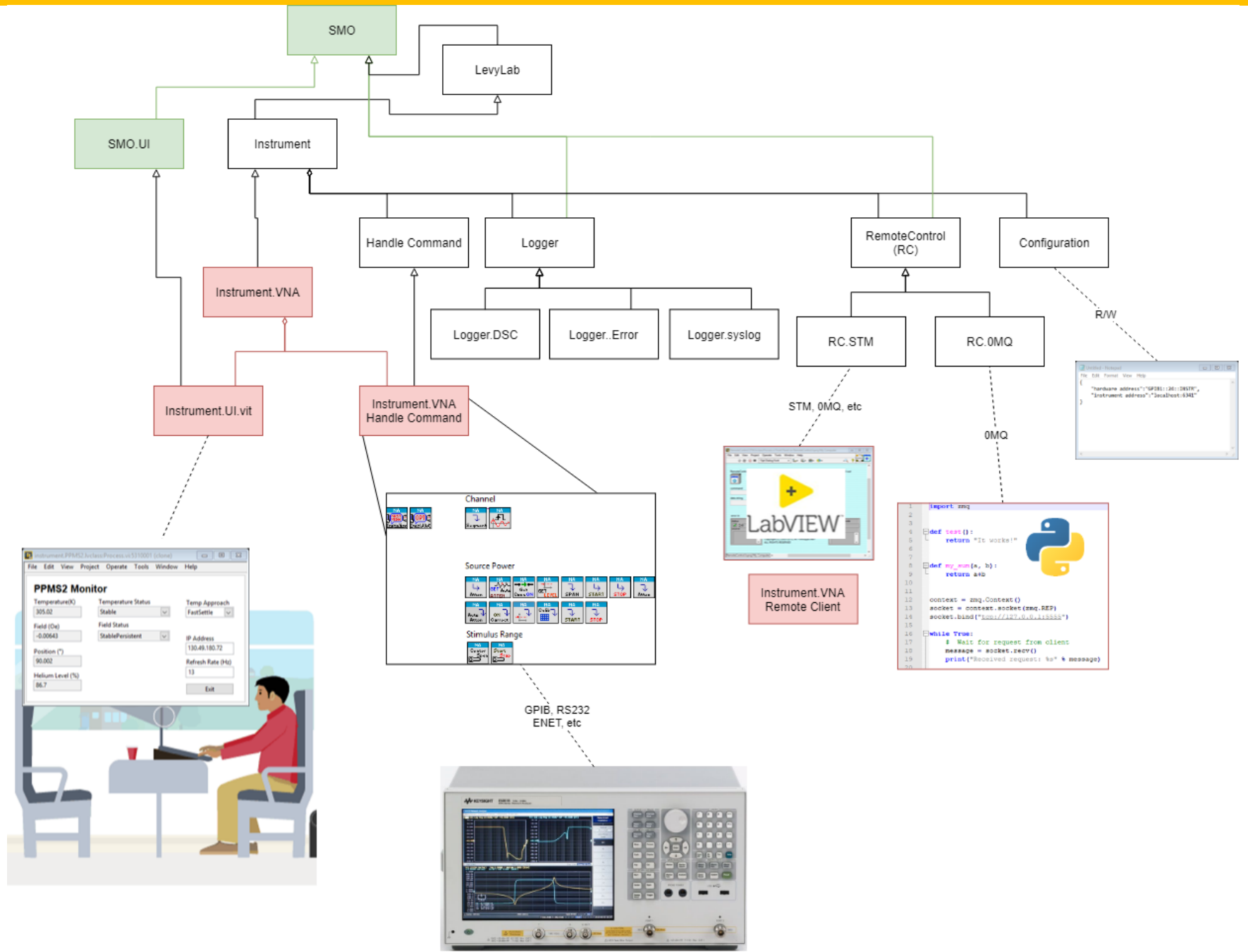
Instrument Framework

- https://github.com/levylabpitt/Instrument-Framework
- An Instrument developed using this framework will have access to the following capabilities:
  - Methods for reading and writing configuration settings.
  - A hardware abstraction layer (HAL) for interfacing with hardware.
  - Data logging: poll and log instrument configuration and settings to DSC database
    - In development: logging to Amazon AWS
  - Defines an API to allow external programs to control compiled instances of the application and remotely across a network
    - Currently implemented using National Instruments' Simple Messaging Library (STM), however a cross-platform protocol is being developed using 0MQ.
  - User interface framework.
    - In development: an embedded subpanel UI framework (MAUI)
- The LevyLab Instrument Framework makes extensive use of JKI State Machines and JKI State Machine Objects.

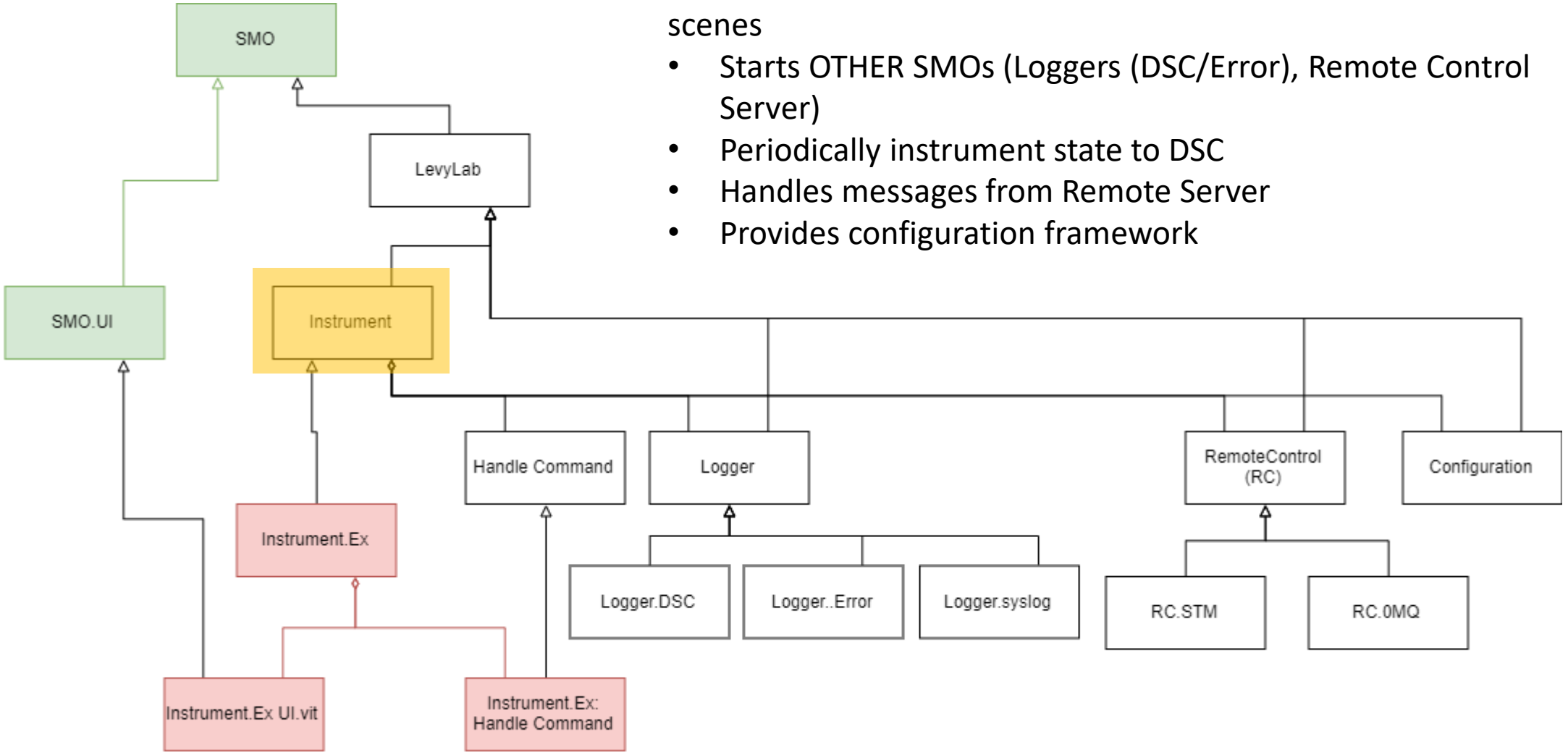
Instrument Framework Diagram



Instrument.lvclass



Instrument.lvclass



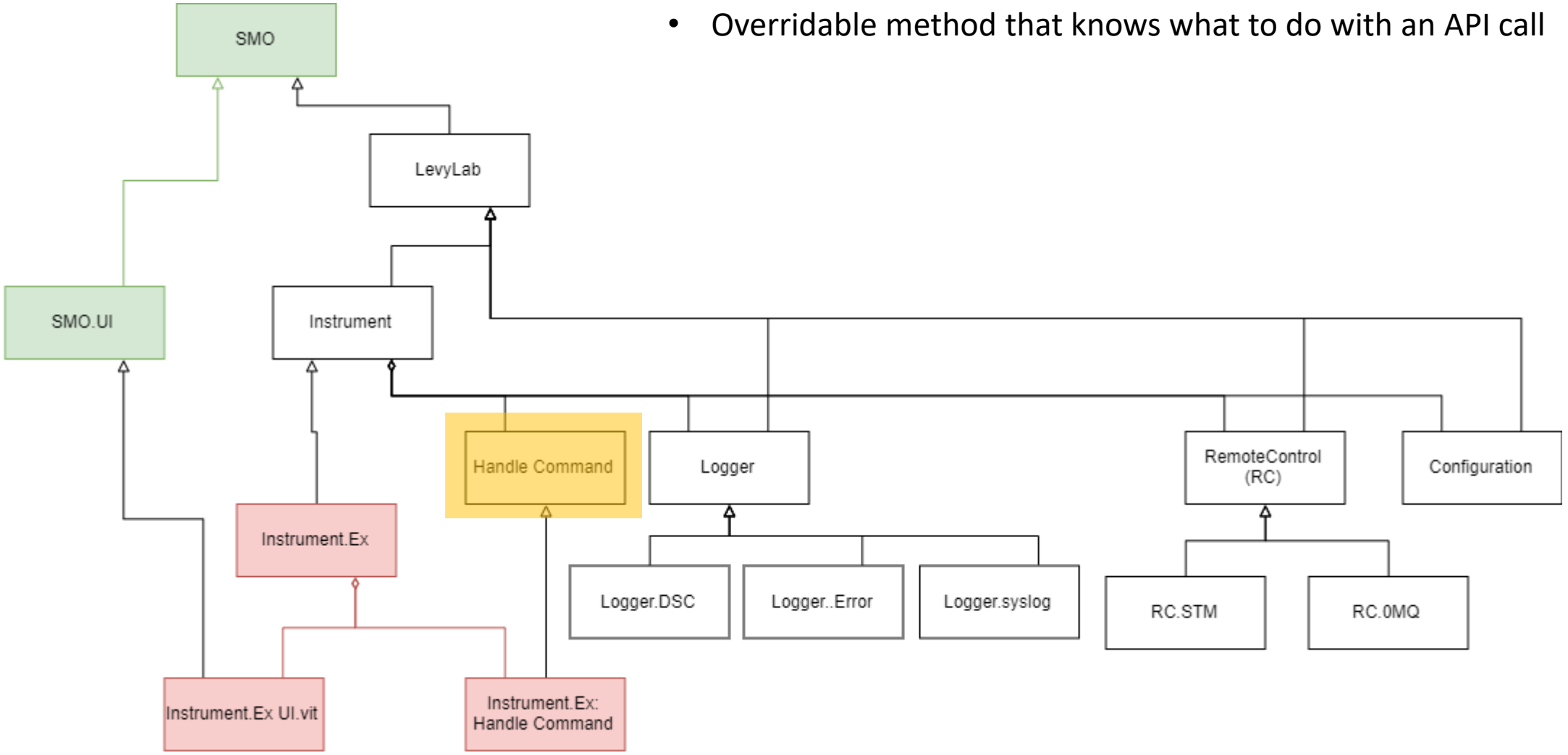
Instrument Process (SMO): The main process that runs behind the scenes

- Starts OTHER SMOs (Loggers (DSC/Error), Remote Control Server)
- Periodically instrument state to DSC
- Handles messages from Remote Server
- Provides configuration framework

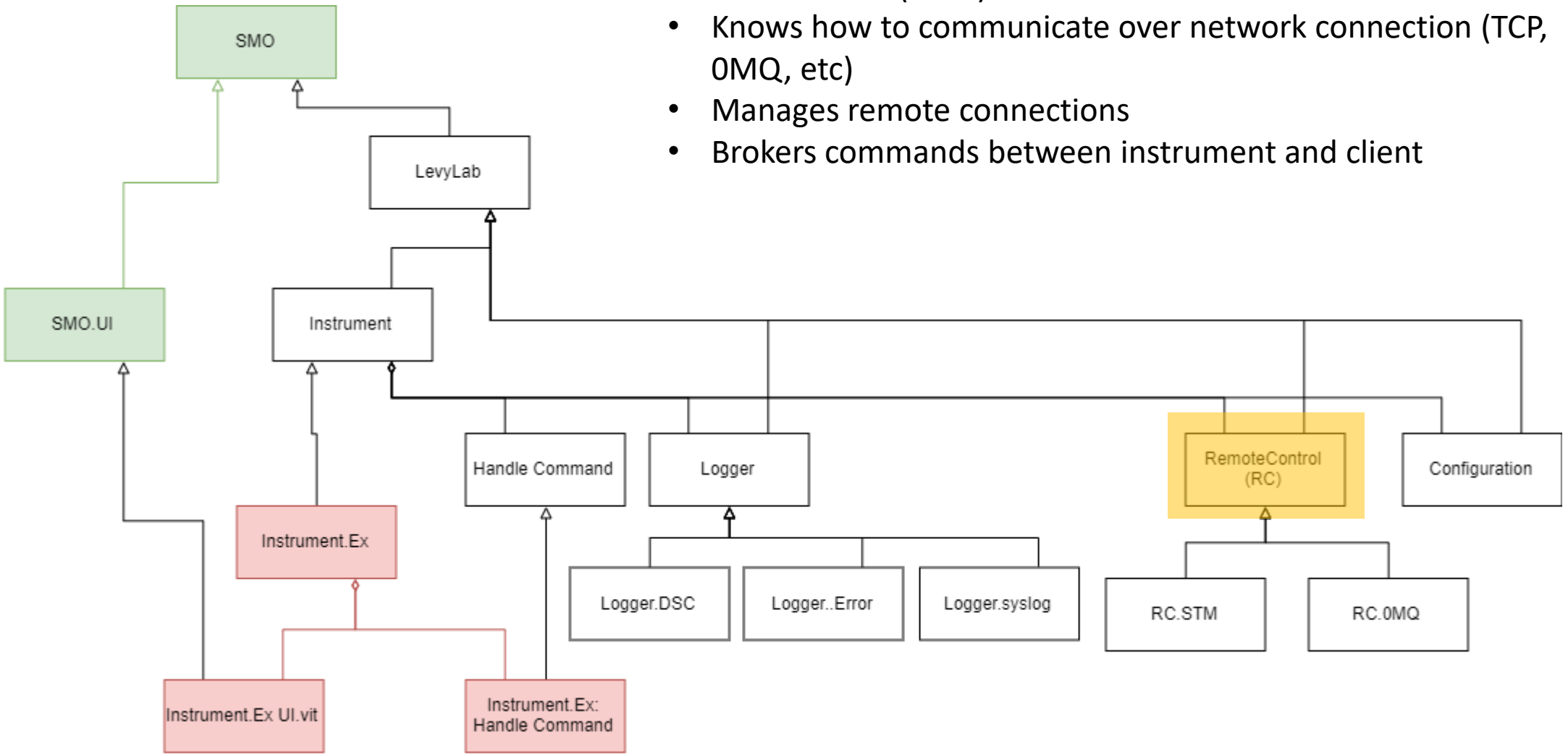
Instrument.lvclass

Handle Command.vi

- Overridable method that knows what to do with an API call



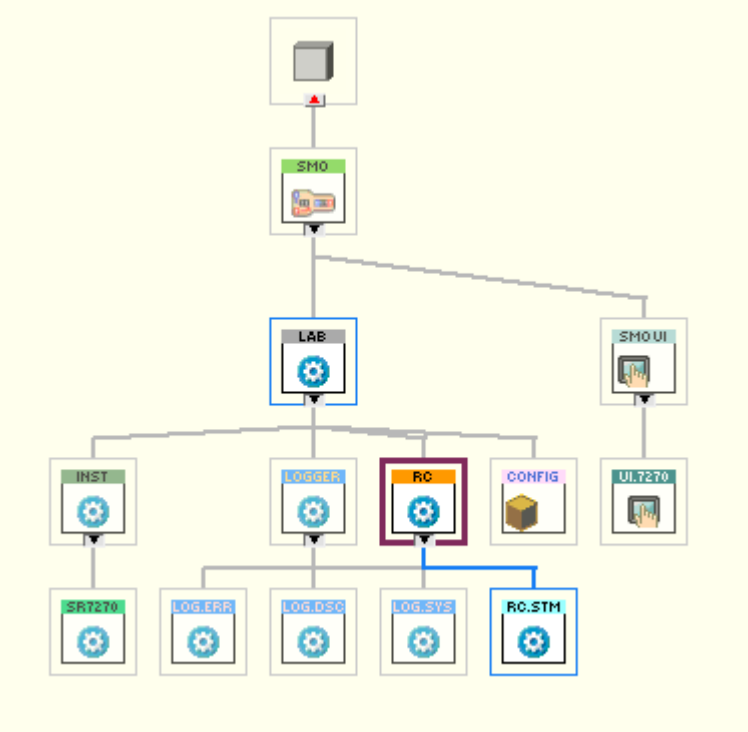
Instrument.lvclass



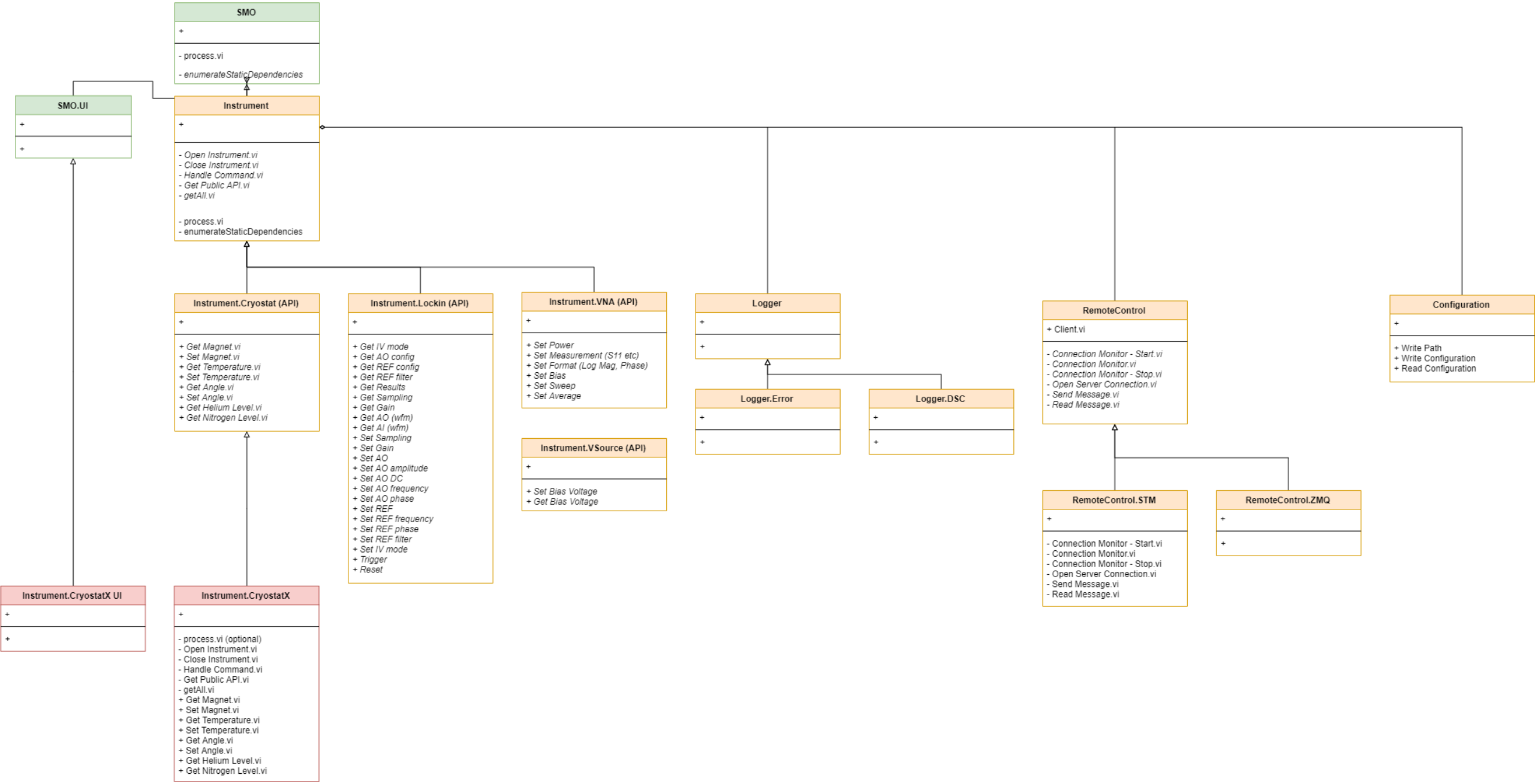
Remote Control (SMO)

- Knows how to communicate over network connection (TCP, 0MQ, etc)
- Manages remote connections
- Brokers commands between instrument and client

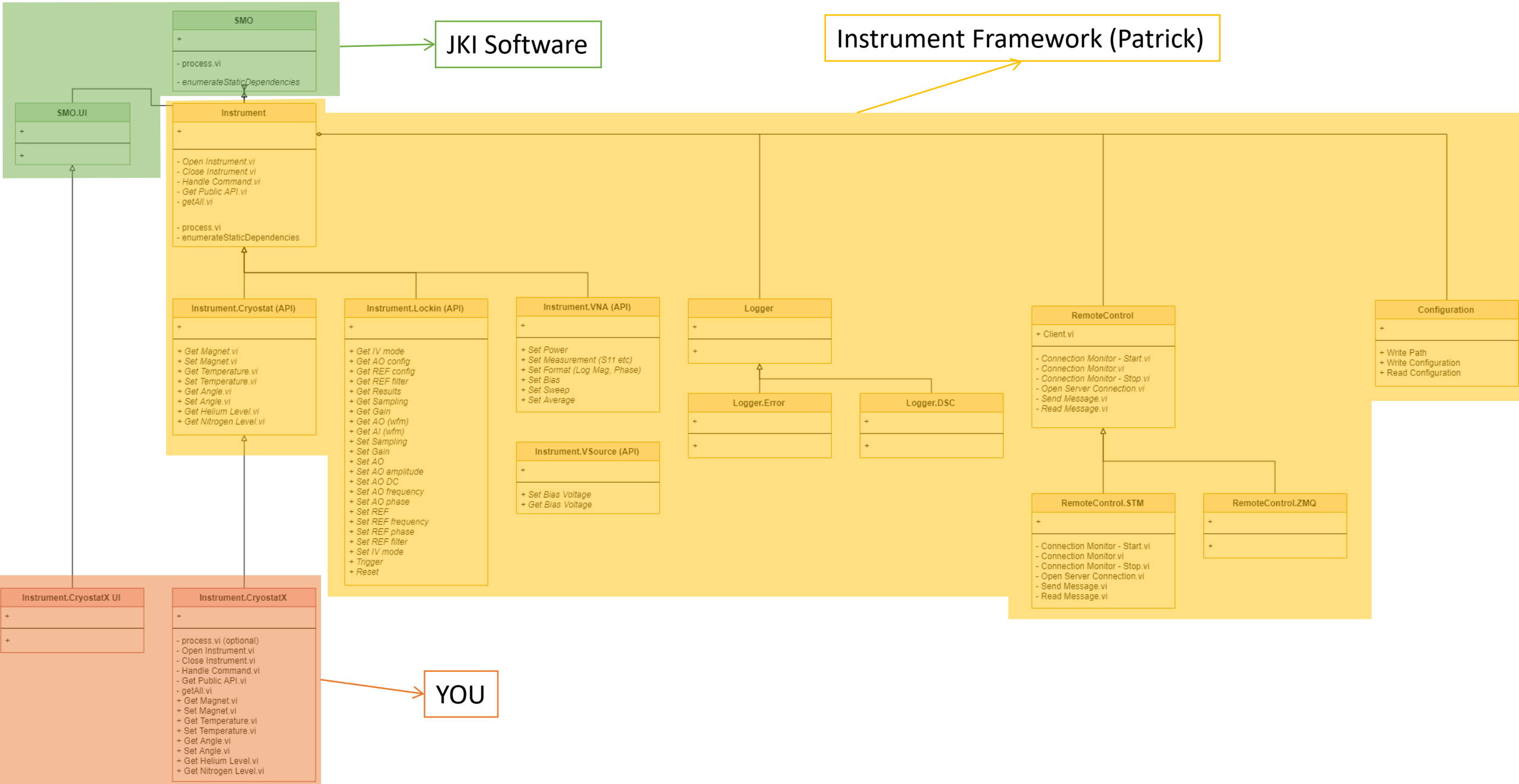
Instrument.lvclass



Instrument.lvclass: UML Diagram



Instrument.lvclass: UML Diagram (Responsibilities)



Create your own Instrument

Instrument.lvclass Prerequisites

- Packages:
  - JKI State Machine Objects (SMO)
  - JKI SMO Editor
  - LevyLab Instrument Template (Base with UI)

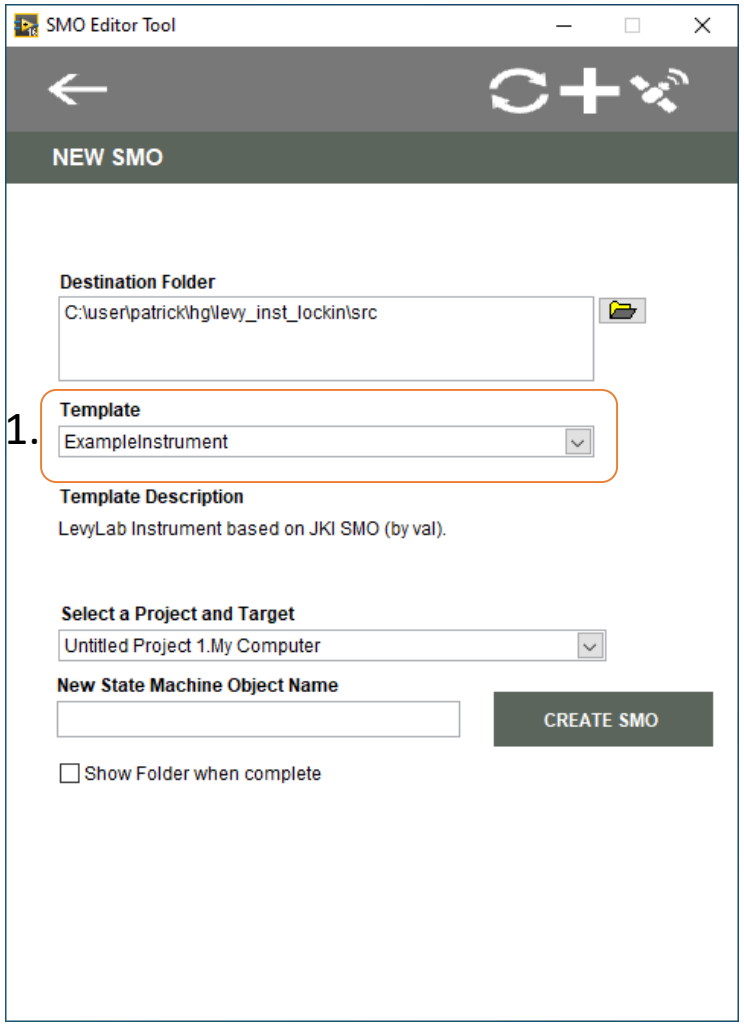
Create Instrument.*NewInstrument*.lvclass

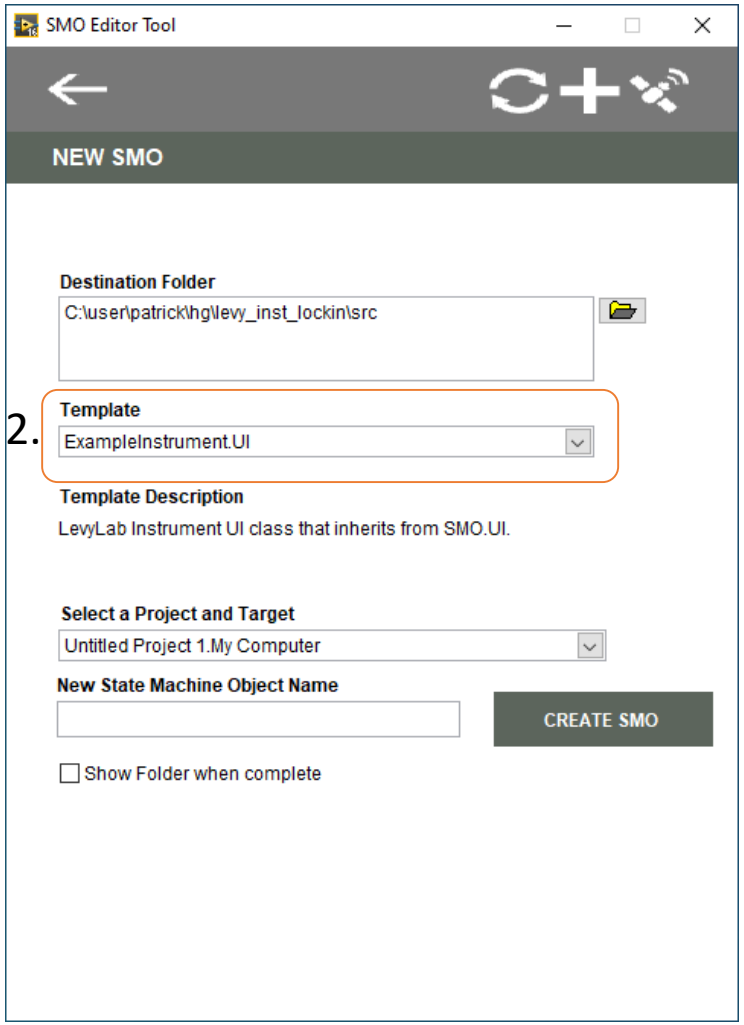
1. Setup repository in Bitbucket (\levylab_inst_*NewInstrument*)
2. Setup codebase using the SMO Editor
3. Modify overridable methods
4. Inside AppLauncher.vi:
    1. Your Instrument must have an SMO Name (HWName_S/N)
    2. ON HOUSESOFTHEHOLY: Create a Process in the DSC that matches the SMO Name above (example: SR7270_09337296)
    3. Change class object to your NewInstrument
    4. ~~#TODO: "CreateSMOName.vi" should be part of instrument class.~~ ~~**child must override**~~
    5. ~~#TODO: ensure children must override CERTAIN protected methods~~
5. getAll
    1. getAll data must correspond to variables inside the DSC
    2. ON HOUSESOFTHEHOLY: create variables with logical names that match the getAll datatype cluster
6. Make a UI

1. Setup repository in Bitbucket (\levylab_inst_*NewInstrument*)

1. Option 1: If this will be a new instrument, create a new repository on Bitbucket and clone to a working directory on your computer (ie, c:\user\levylab_inst_*NewInstrument*
2. Option 2: If you are refactoring an existing instrument, you can make a new branch and work from there.

2. Setup codebase using the SMO Editor

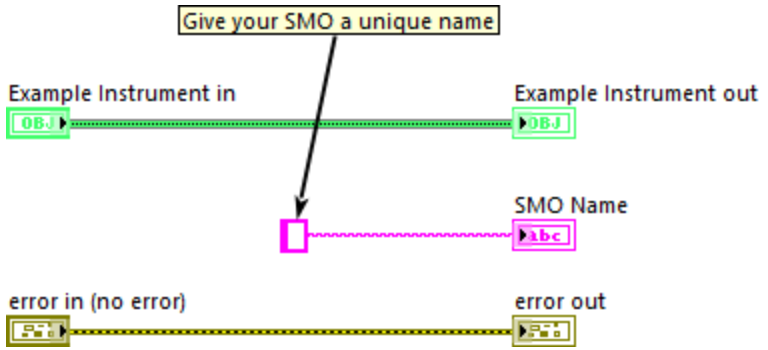




3. Modify overridable methods

1. Modify these VIs:
    1. Create Instrument SMO Name.vi
    2. Open Instrument.vi
    3. Close Instrument.vi
    4. Handle Command.vi
    5. getAll.vi
2. Modify these Typedefs:
    1. *NewInstrument*.Commands--Enum.ctl
    2. *NewInstrument*.Configuration--Cluster.ctl
    3. *NewInstrument*.getAll--Cluster.ctl
3. Optional
    1. Configure Instrument.vi (and subVIs)
4. Other (do not need to modify)
    1. Get Public API

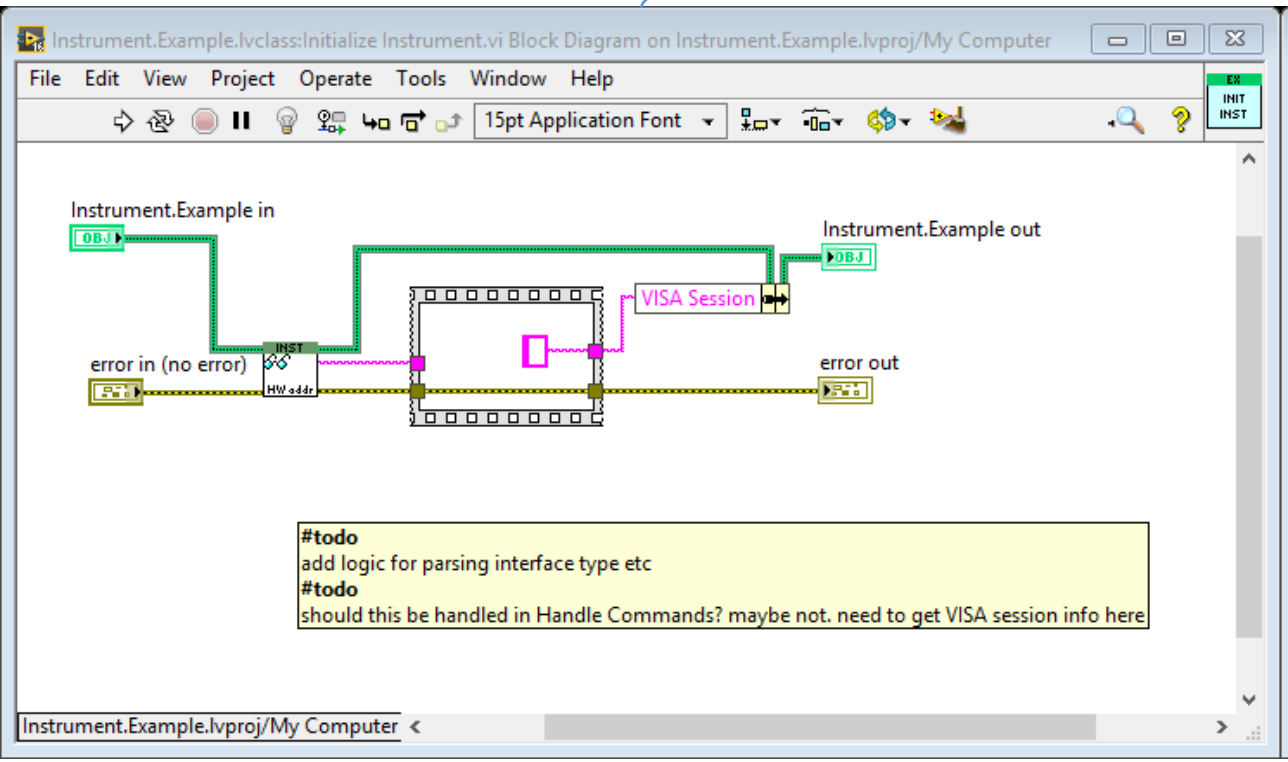
1.1 Create Instrument SMO Name.vi

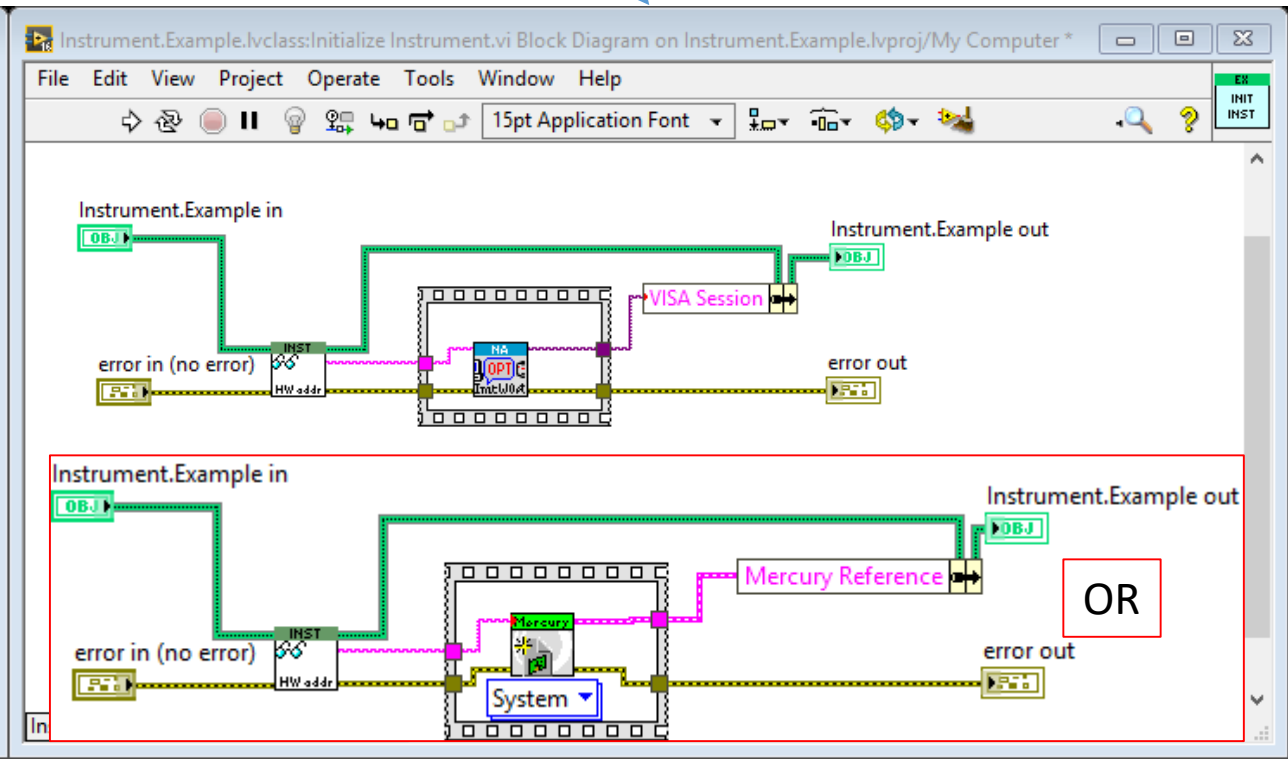


*Your SMO name needs to match an existing **process** in the DSC*

1.2 Open Instrument.vi

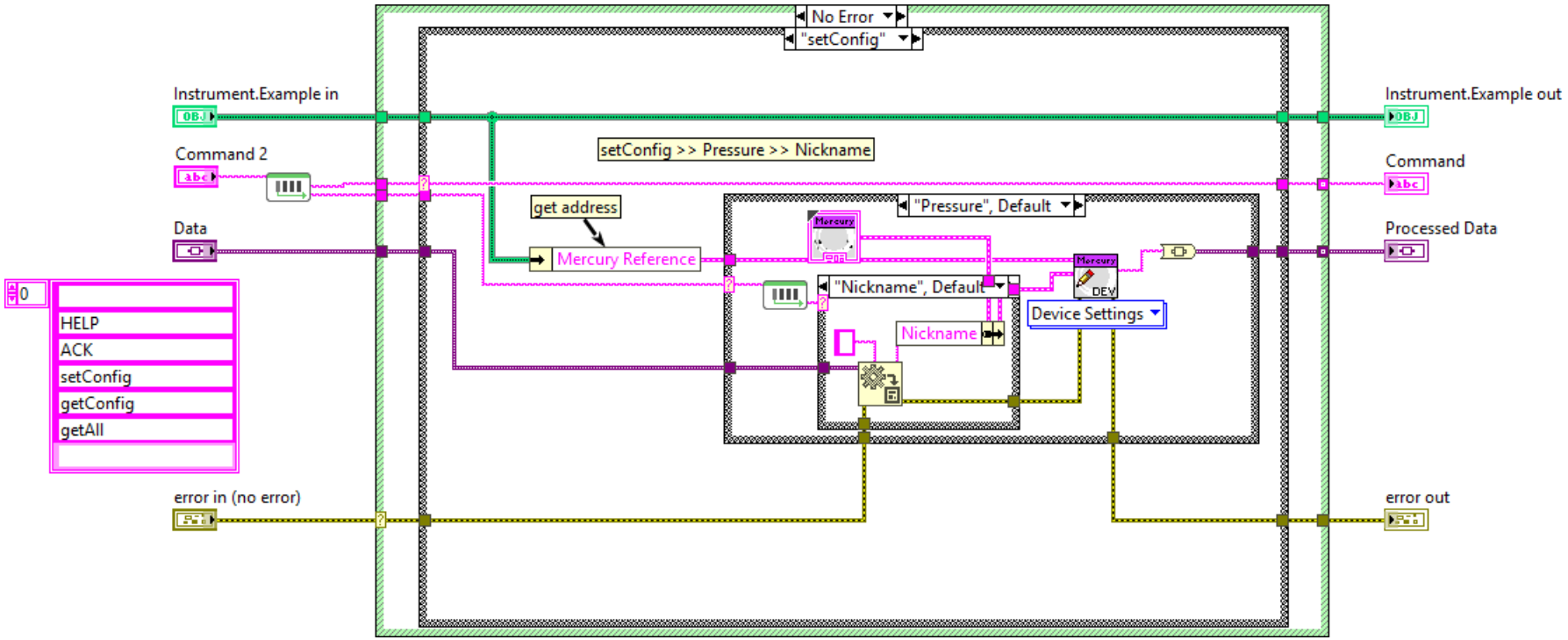
- Called by Instrument.lvclass:Process.vi's state Remote: Init (called by Macro: Initialize
- Add code to open connection to hardware
- Example:





1.4 Handle Command.vi

- Modify code to respond to command strings
- Could talk to hardware directly or send Public Method to another SMO
- Example:



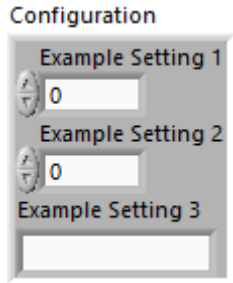
Typedefs

1. *NewInstrument*.Commands--Enum.ctl

an enum containing a list of supported API commands

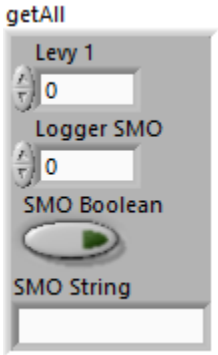
2. *NewInstrument*.Configuration--Cluster.ctl

any special configuration settings your instrument needs. Hardware address?



3. *NewInstrument*.getAll--Cluster.ctl

this is a cluster that will be written to the DSC. The name of each element needs to match an existing trace in the DSC

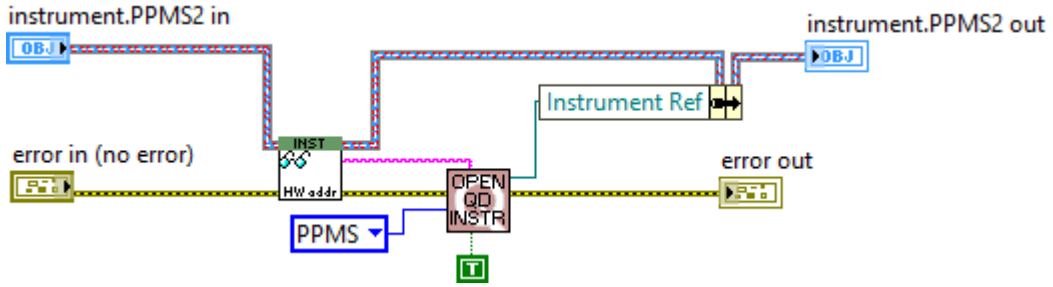


PPMS2

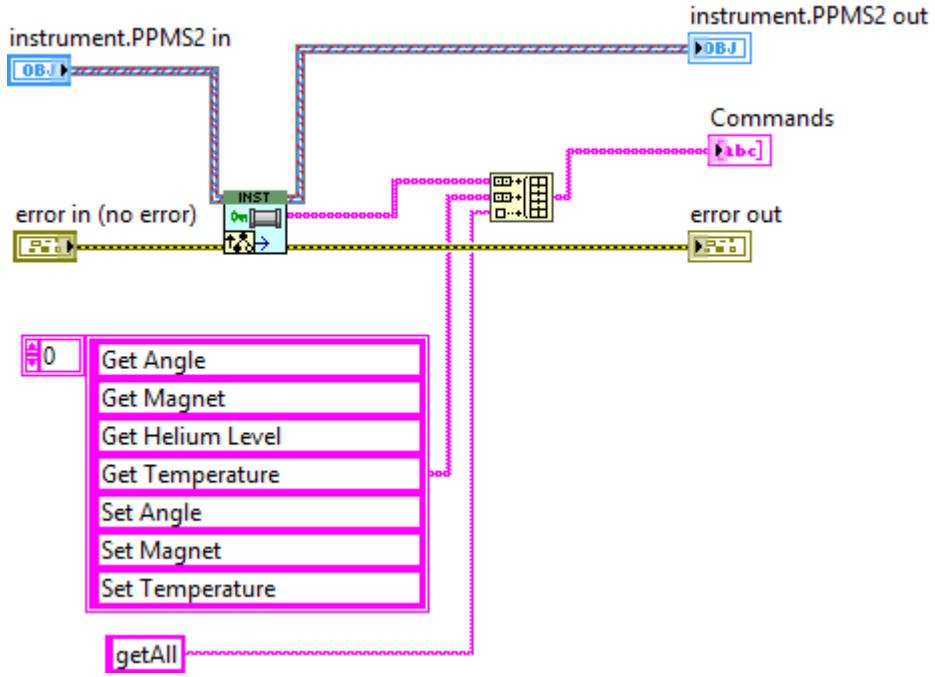
- Instrument.PPMS2.lvclass already exists…What is the plan?
  - Copied Instrument.Example and *.Example.UI merge code….
- Make existing *.PPMS2.lvclass inherit from Instrument.lvclass >v1.1.0.9 check
- Make inherit from Instrument.Cryostat.lvclass check
- Make new overrides (set Field, etc) check
  - Get Angle, Get Field, Get Helium Level, Get Temperature, Set Angle, Set Field, Set Temperature
- Override “~~Initialize~~ Open Instrument”
  - Copy code from “PPMS2: Initialize” state (?) check
  - Add Instrument Ref to Instrument.PPMS2 class private data check
- Override “Get Public API” check
- Override “Handle Command” check
  - getAll: code from “PPMS2: Get Status” state check
  - (make override method “getAll”) check
- Update APIs to use remote client check
- Override “Create Instrument SMO Name.vi”
  - Return name “PPMS 2” check

Overrides

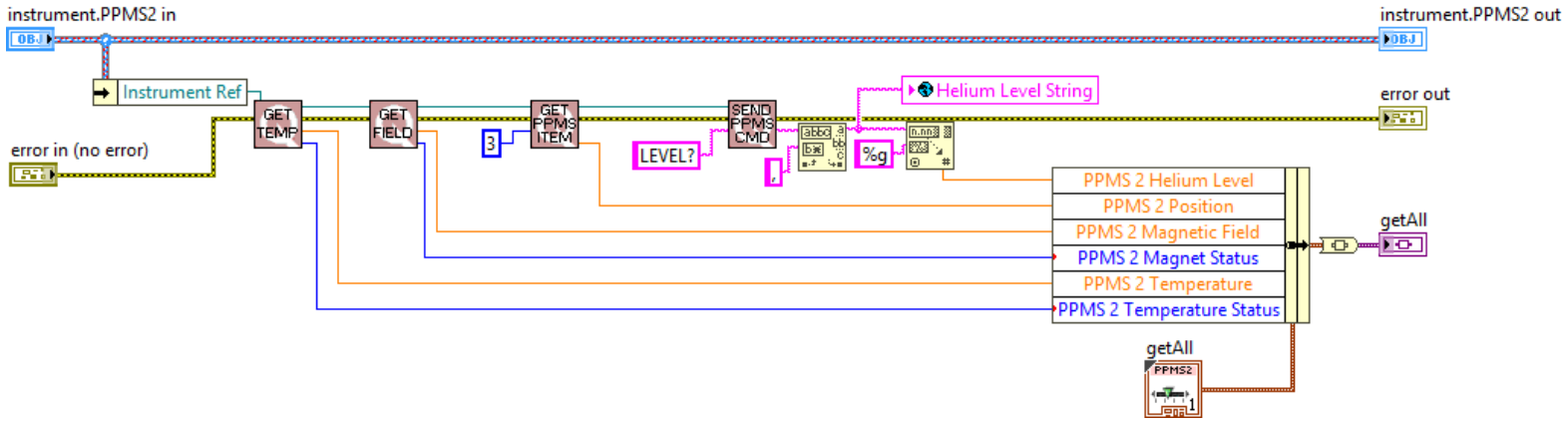
Open Instrument.vi



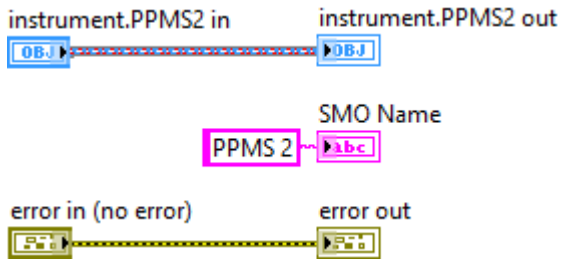
Get Public API.vi



getAll.vi

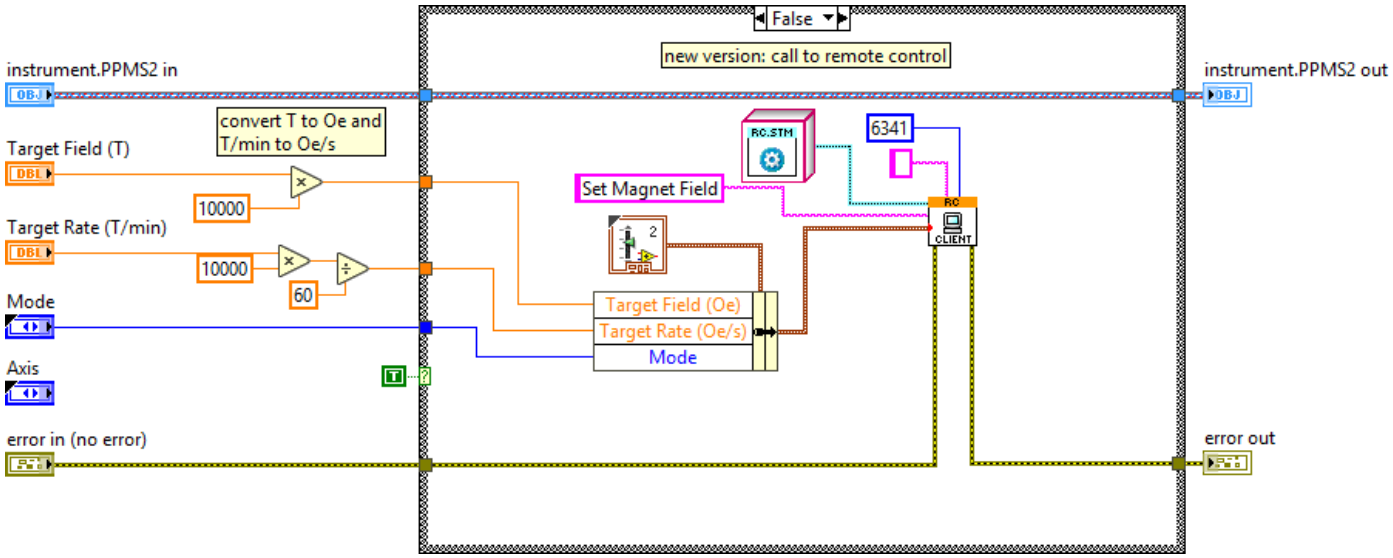


Create Instrument SMO Name.vi

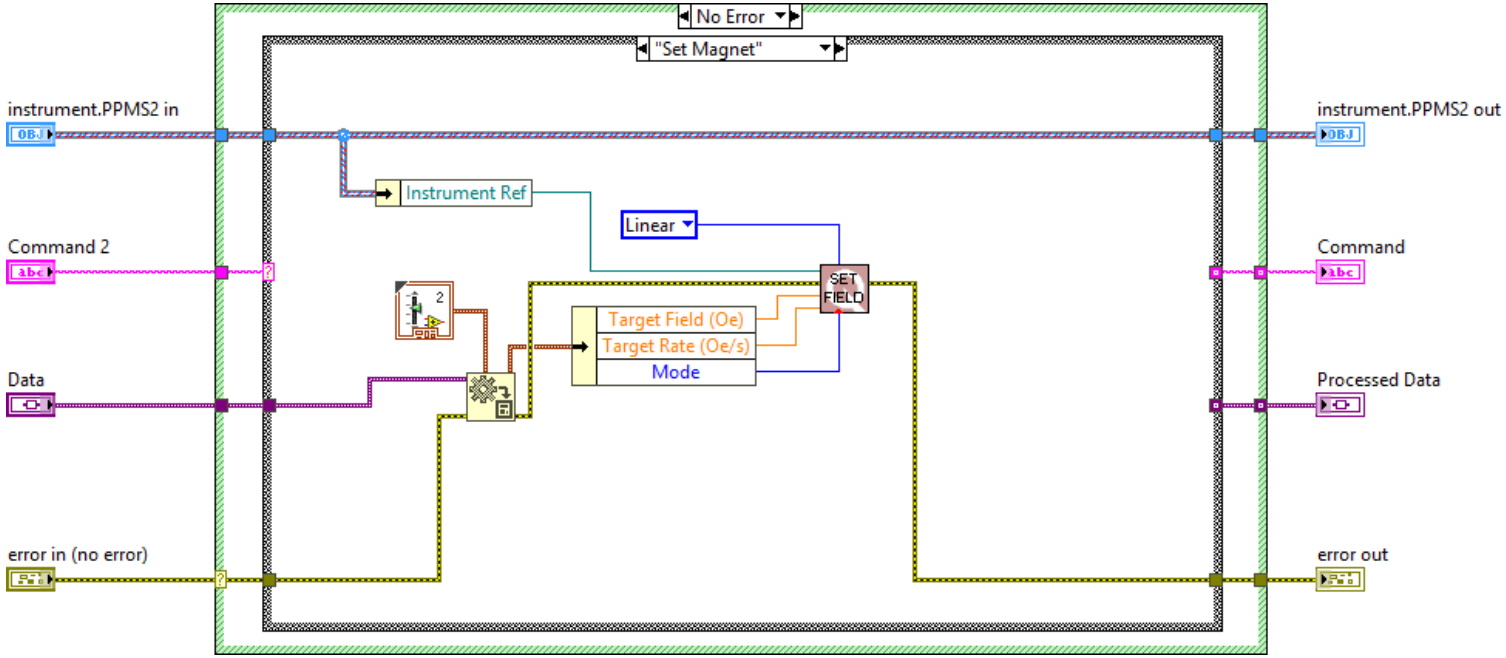


API & Override “Handle Command.vi” (Examples)

Set Magnet Field.vi (API)



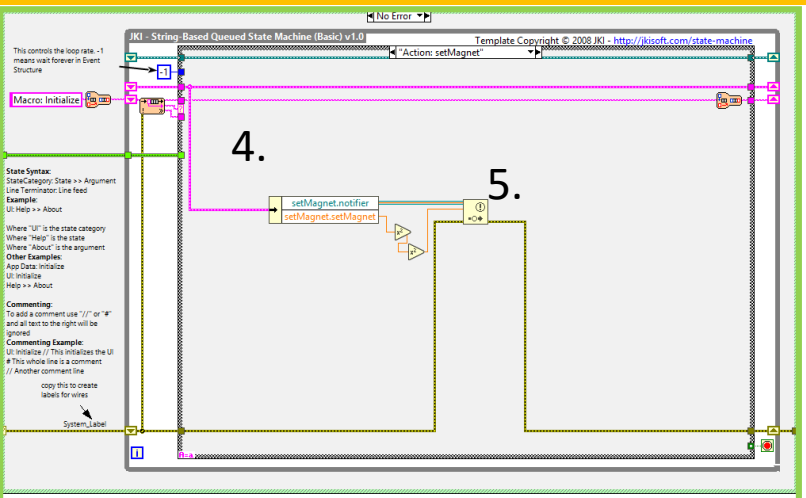
Handle Command.vi: Set Magnet



Non-Standard Stuff…

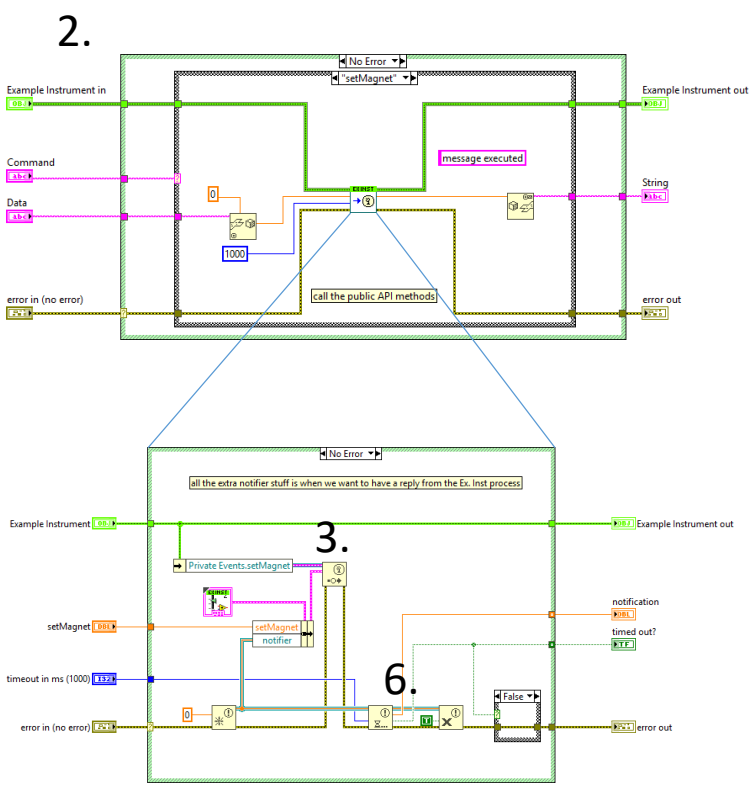
- Instrument.PPMS2: PPMS Monitor and Control.vi
  - Keep this one, but make code match Instrument.Example:AppLauncher.vi
- Instrument.PPMS2: Process.vi
  - Ultimately I’m not sure sure if this one should be doing anything. Move UI elements to →
- Instrument.PPMS2 UI.lvclass
  - Create this class. This will contain the UI
  - This class already exists as Instrument.Example UI.lvclass. Just rename that stuff

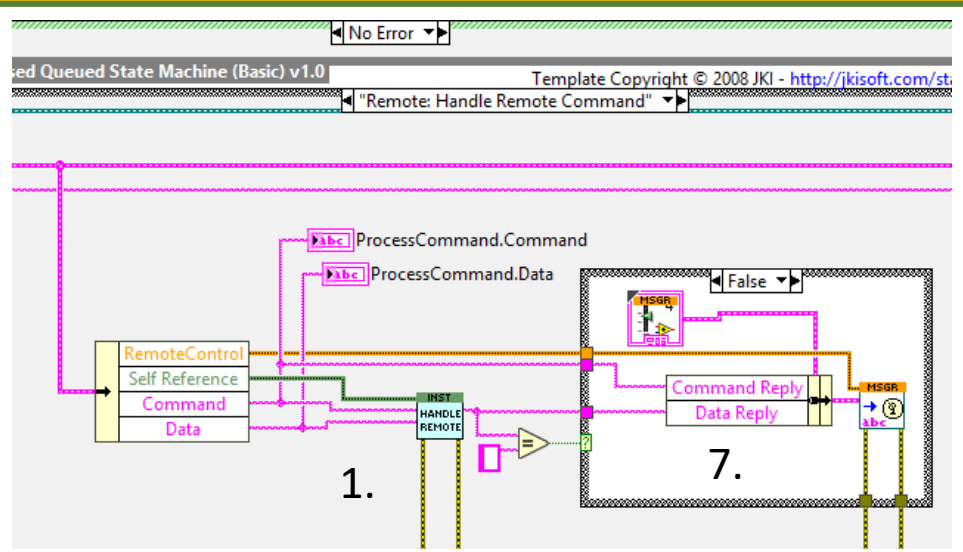
AnyInstrument.lvclass ←→ Instrument.lvclass



AnyInstrument.lvclass:process.vi

- Override "**Handle Remote Command**" and "**Get Public API**" to list of supported commands
- Commands are processed in "**Handle Remote Command**"
- If **Handle Remote Command** needs to communicate with **AnyInstrument**'s process it uses methods provided by **AnyInstrument**





1. Handle Remote Command* is called the by **Instrument's process.**
Is an overridable method by child class.
7. Any responses are forwarded to **RemoteControl** by calling **RemoteControl.CommandResponse**

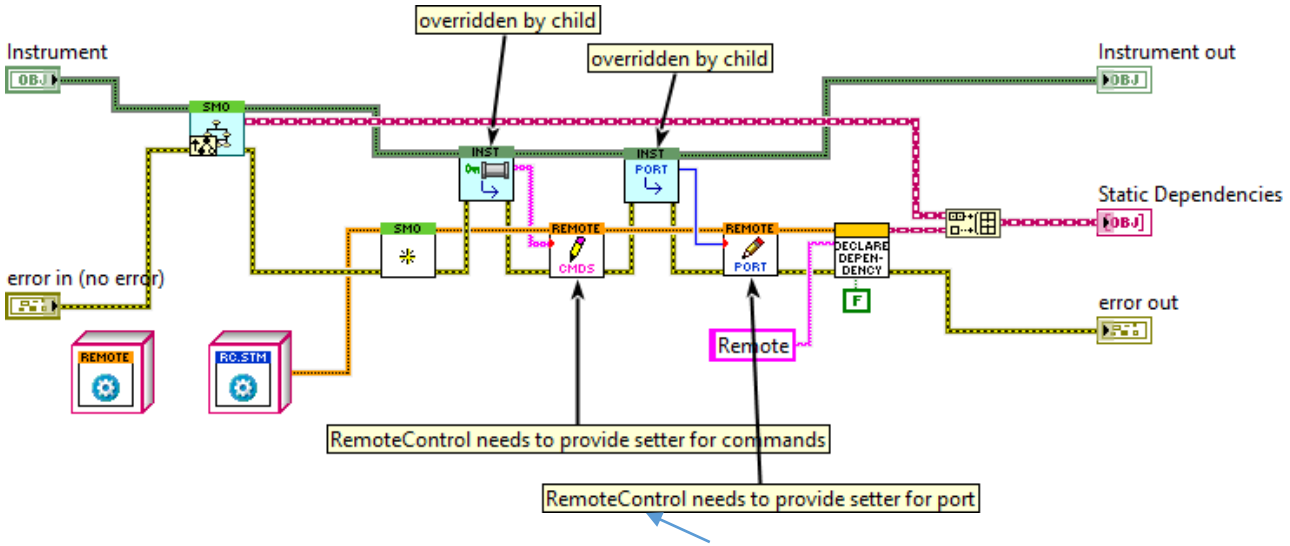
1. **Instrument** process receives event from **RemoteControl** and calls **Handle Remote Command**
2. **Handle Remote Command** decides what to do with command. In this example it called **AnyInstrument:setMagnet**
3. **AnyInstrument:setMagnet** fires event
4. **setMagnet** event is received by **AnyInstrument**.
5. **AnyInstrument** does its work and *responds using notifier*** provided by **setMagnet***
6. **setMagnet** receives notifier from **AnyInstrument**
7. **Instrument** calls **RemoteControl:SendMessageToProcess** to send processed data

*Handle Remote Command is protected.
Can also have "Script Remote Command" that is a public method

**This programming structure provides an easy way for instrument to respond to the API when it is finished

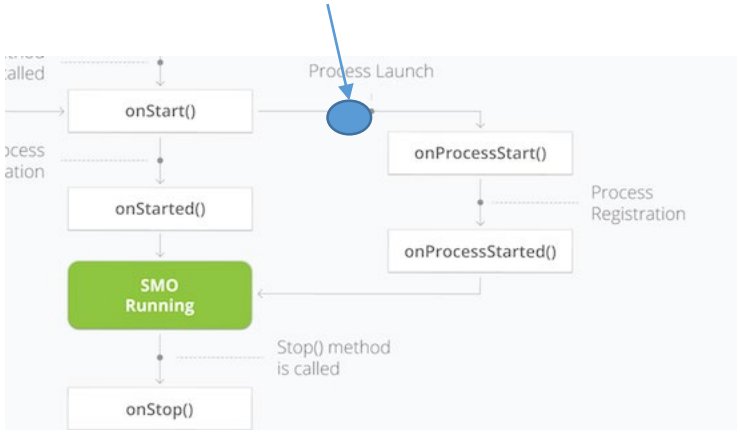
Instrument.lvclass: EnumerateStaticDependencies.vi

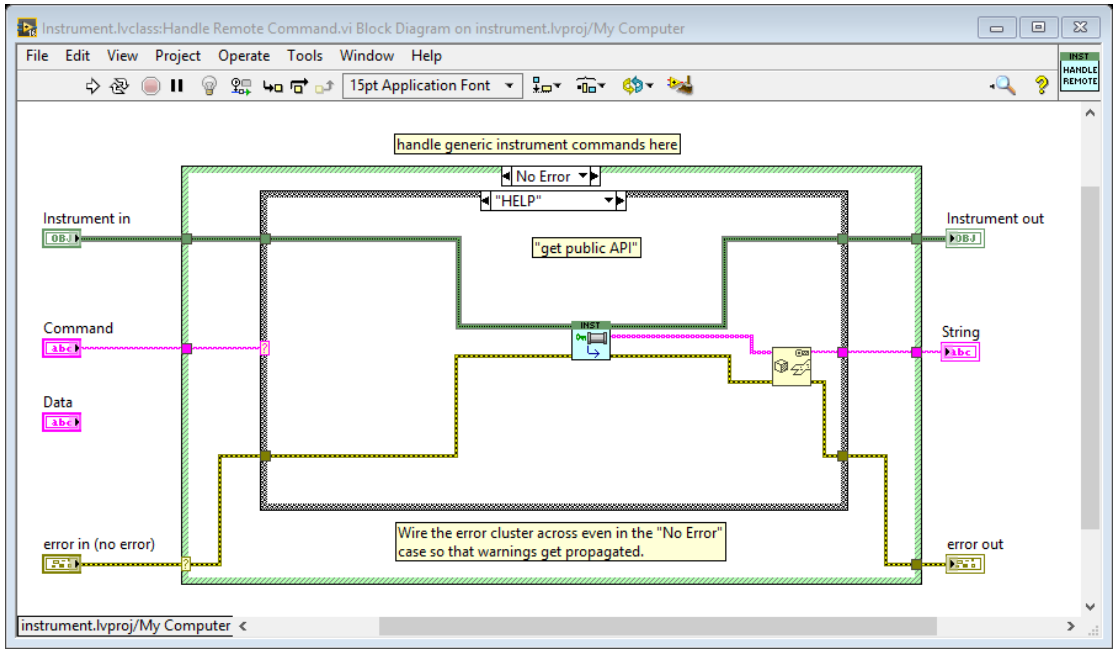
EnumerateStaticDependencies

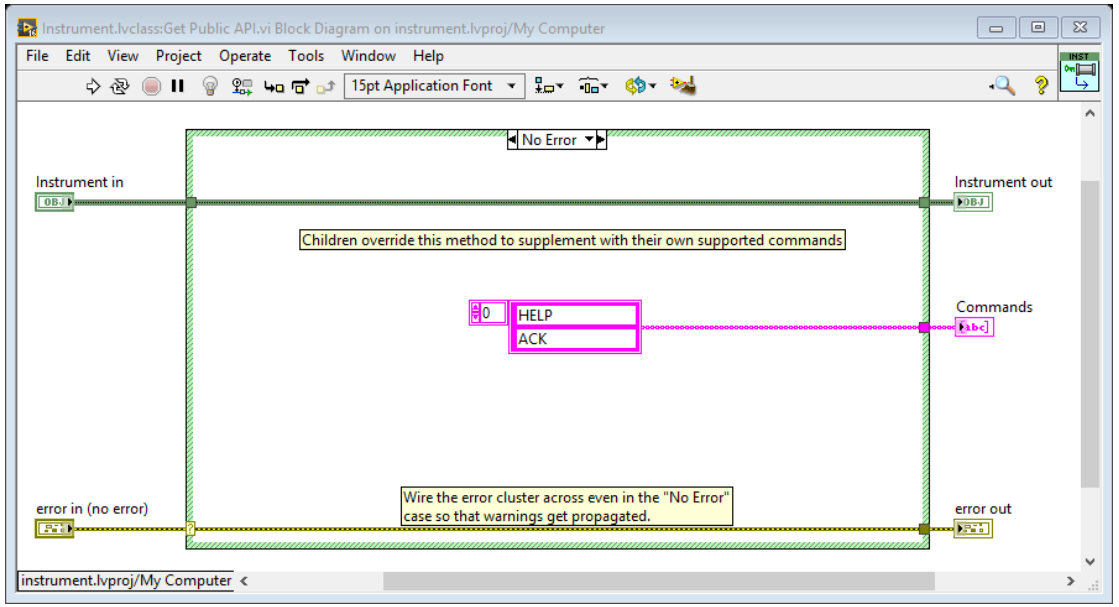


Make RemoteControl.lvclass a dependency of Instrument
Call write commands and port here to initialize RemoteControl

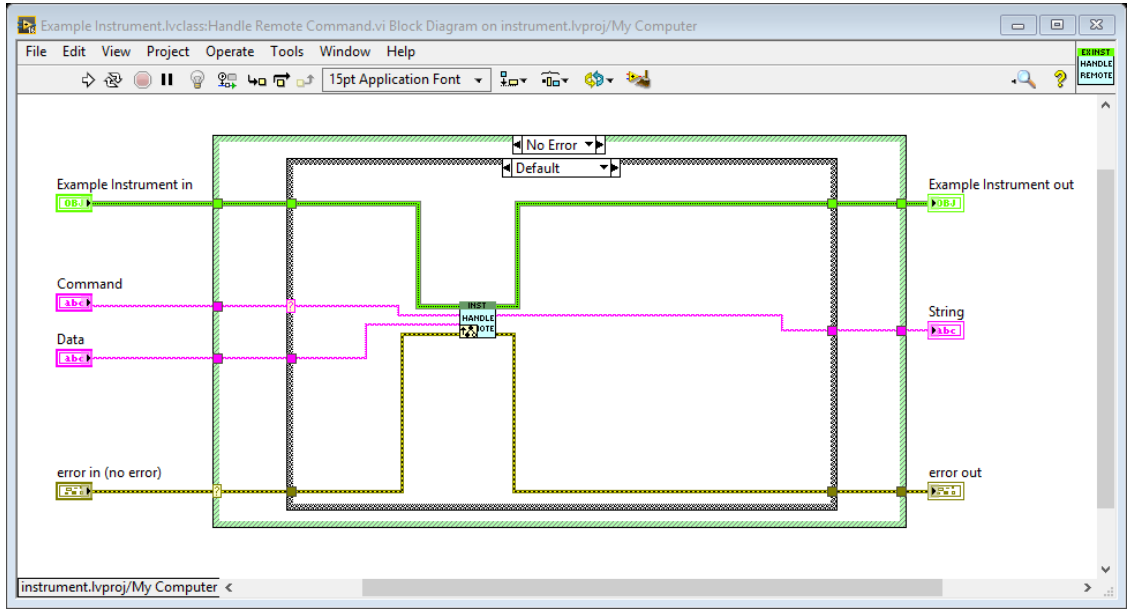
EnumerateStaticDependencies called here

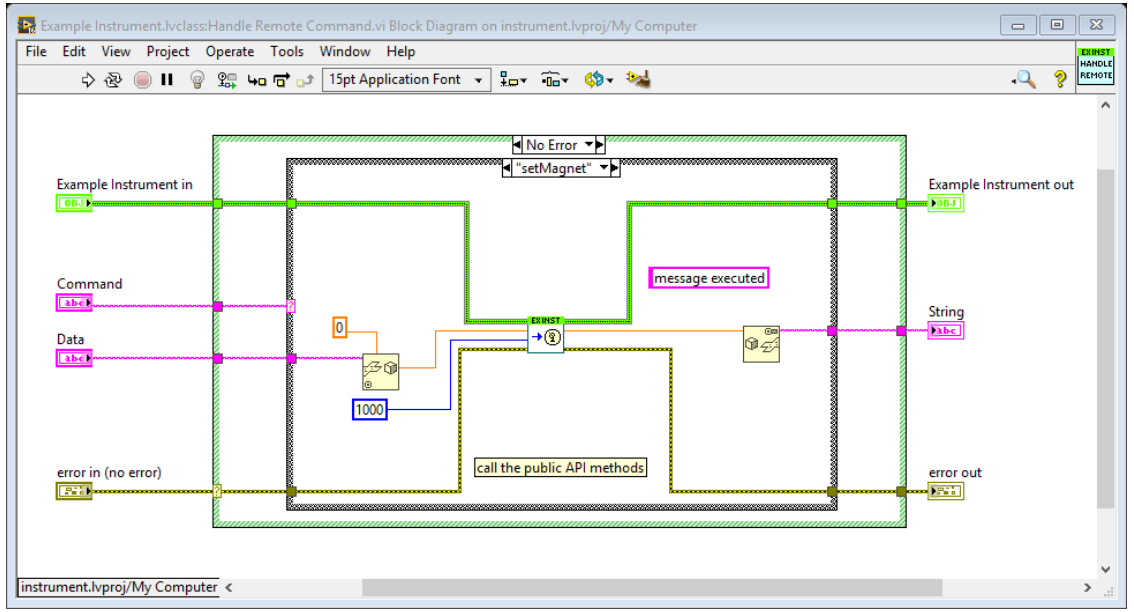


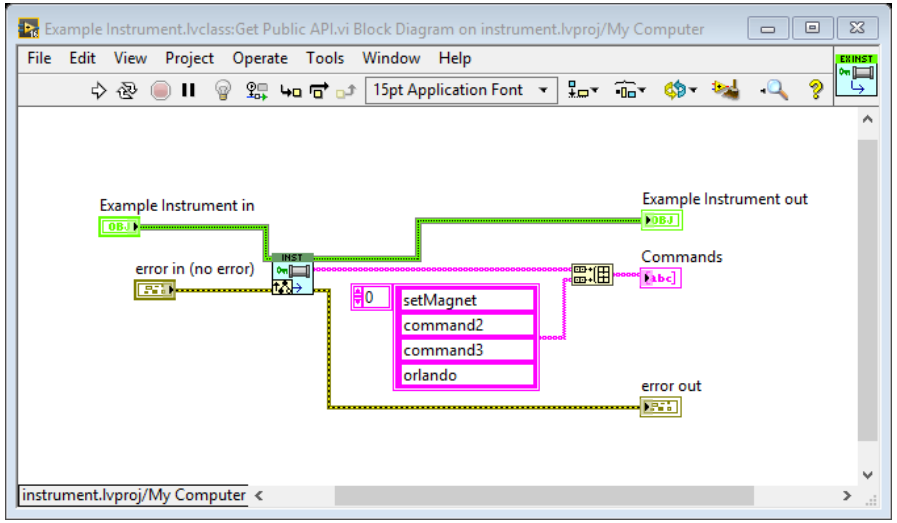


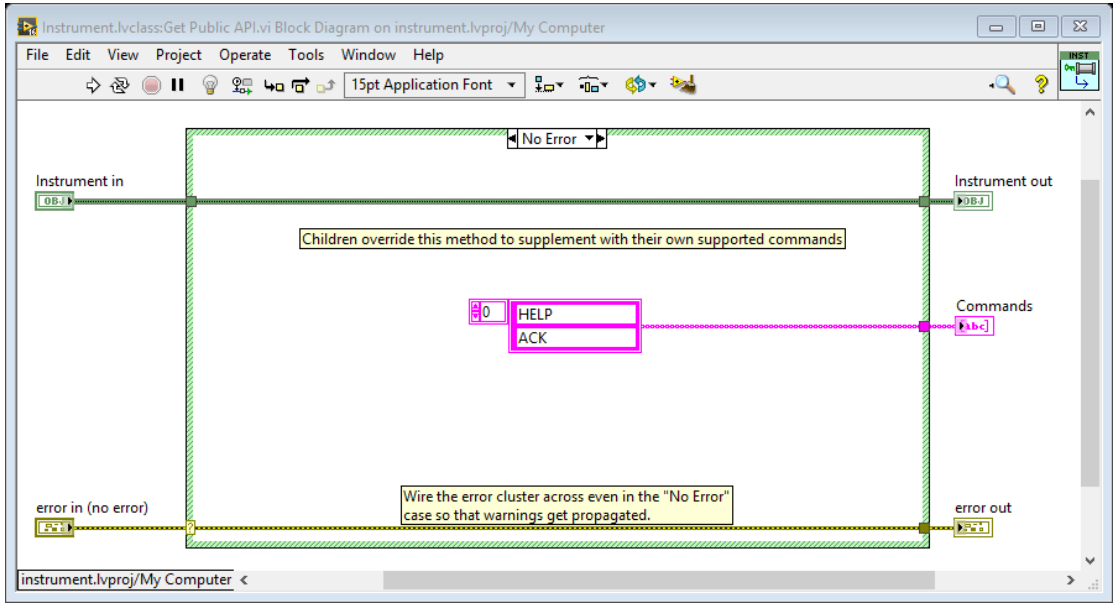


Instrument.lvclass: Handle Remote Command

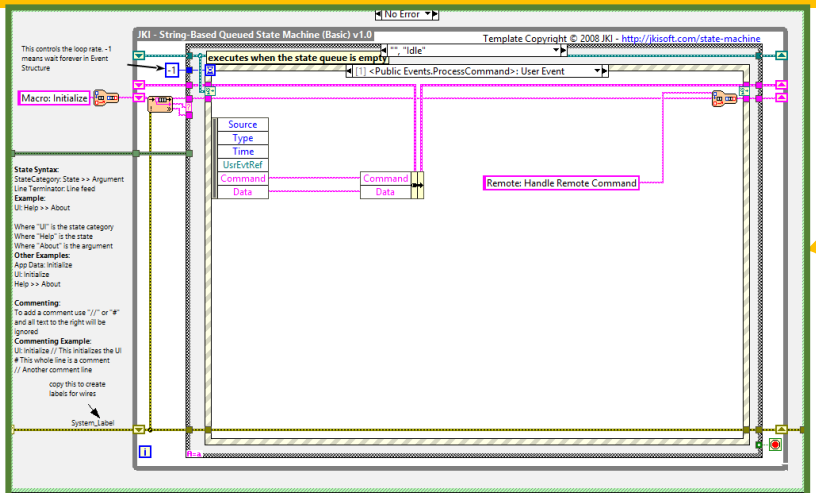








Instrument.lvclass ←→ RemoteControl.lvclass



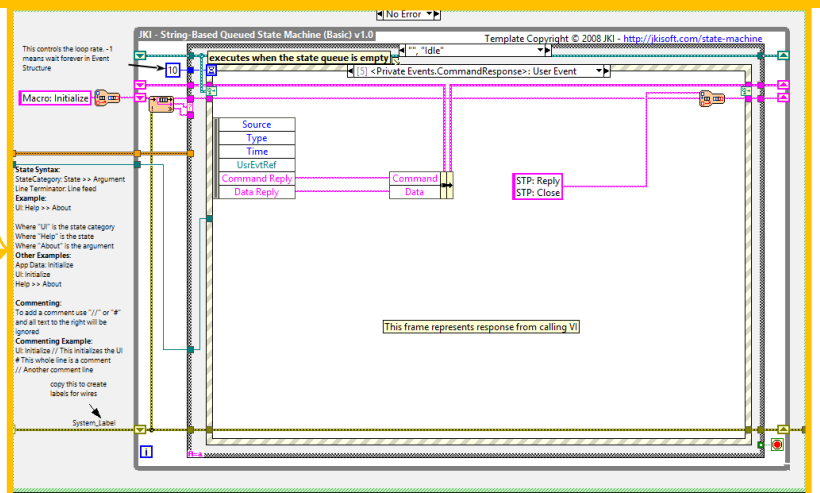
Instrument.lvclass:process.vi

- Uses **RemoteControl** as dependency (declared by enumerateStaticDependencies.vi (overridden from SMO))
- Processes commands received from **RemoteControl.SendmessageFromProcess**
- List of supported commands maintained by overridable method “Get Public API”
  - children of Instrument override with their own list
- Commands are processed by overridable method “Handle Remote Command”
- Responds by calling **RemoteControl.SendMessageToProcess**

RemoteControl.SendMessageFromProcess

RemoteControl.SendMessageToProcess

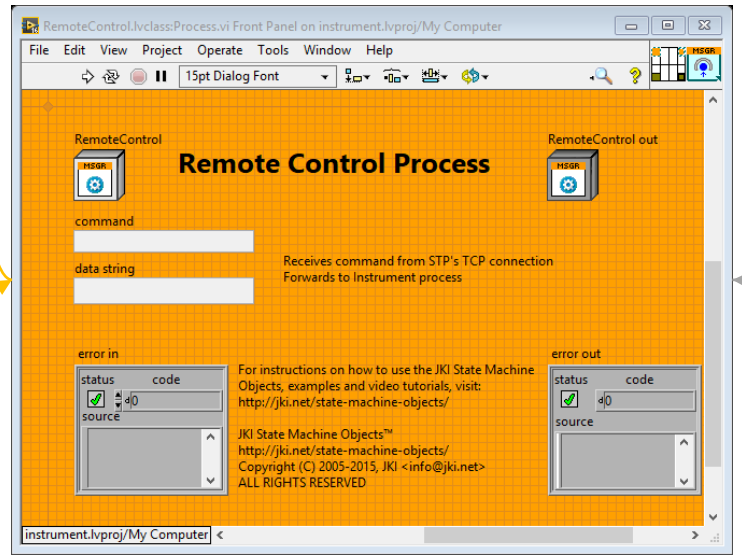
Methods are owned by RemoteControl.



RemoteControl.lvclass:process.vi

- Acts as command server between external processes (clients) and owner process
- Should be protocol agnostic (should work for raw TCP, STM, 0MQ by choice of child class

RemoteControl.lvclass ←→ RemoteControl (.STM) or (.0MQ)



RemoteControl.SendMessageFromProcess

Instrument

RemoteControl.SendMessageToProcess

Read/Write

Write/Read

STM Client

---or---

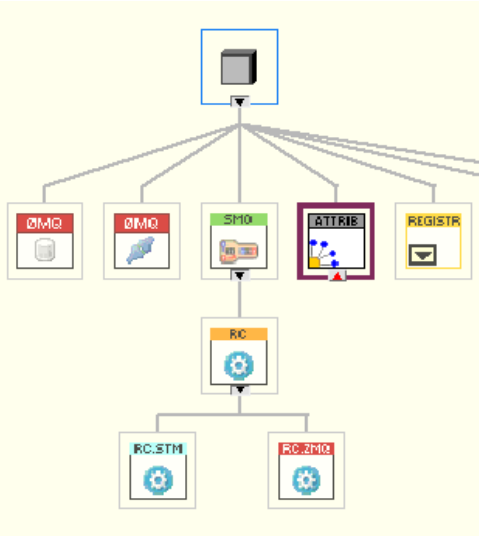
0MQ Client

RemoteControl.SendMessageFromProcess

RemoteControl.SendMessageToProcess

RemoteClient

RemoteControl Diagram (11/29/2018)



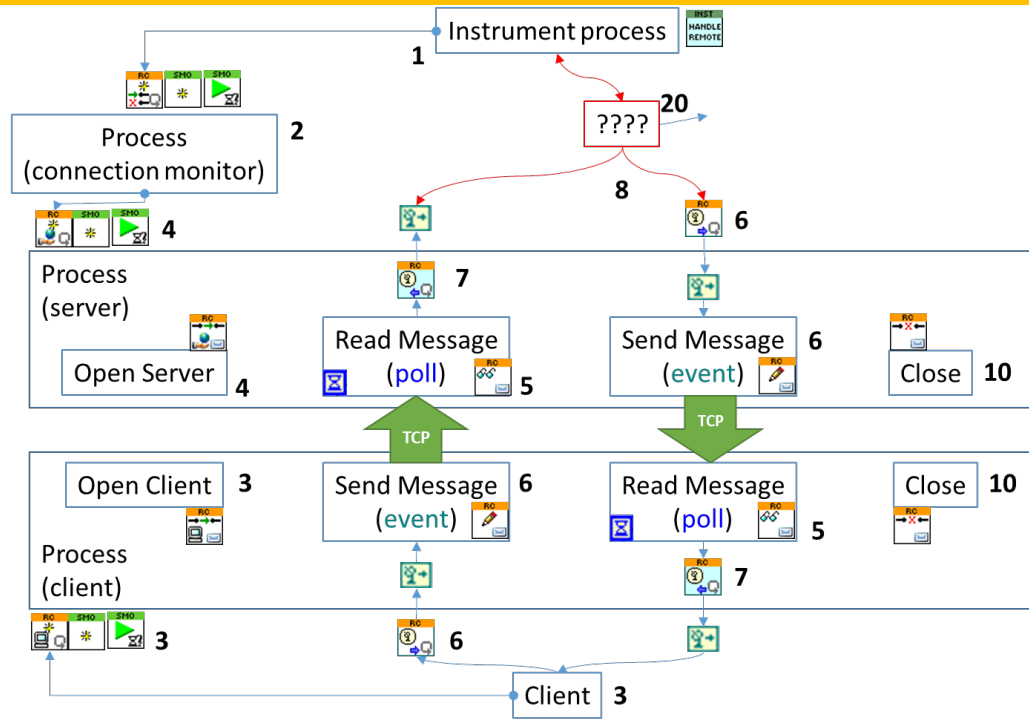
#Javier:
Future:
Client could request a certain connection type from connection monitor (ZMQ, TCP, network stream, whatever)

Instrument process
1
INST HANDLE REMOTE
Process (connection monitor)
2
????
20
Instrument process knows about connection monitor
Connection monitor knows about server process
How can Instrument know about server??
8
SendMessageToProcess
6
4
Process (server)
SendMessageFromProcess
7
Open Server
4
Read Message (poll)
5
Send Message (event)
6
Close
10
TCP
TCP
Open Client
3
Send Message (event)
6
Read Message (poll)
5
Close
10
Process (client)
7
SendMessageFromProcess
3
SendMessageToProcess
6
Client
3

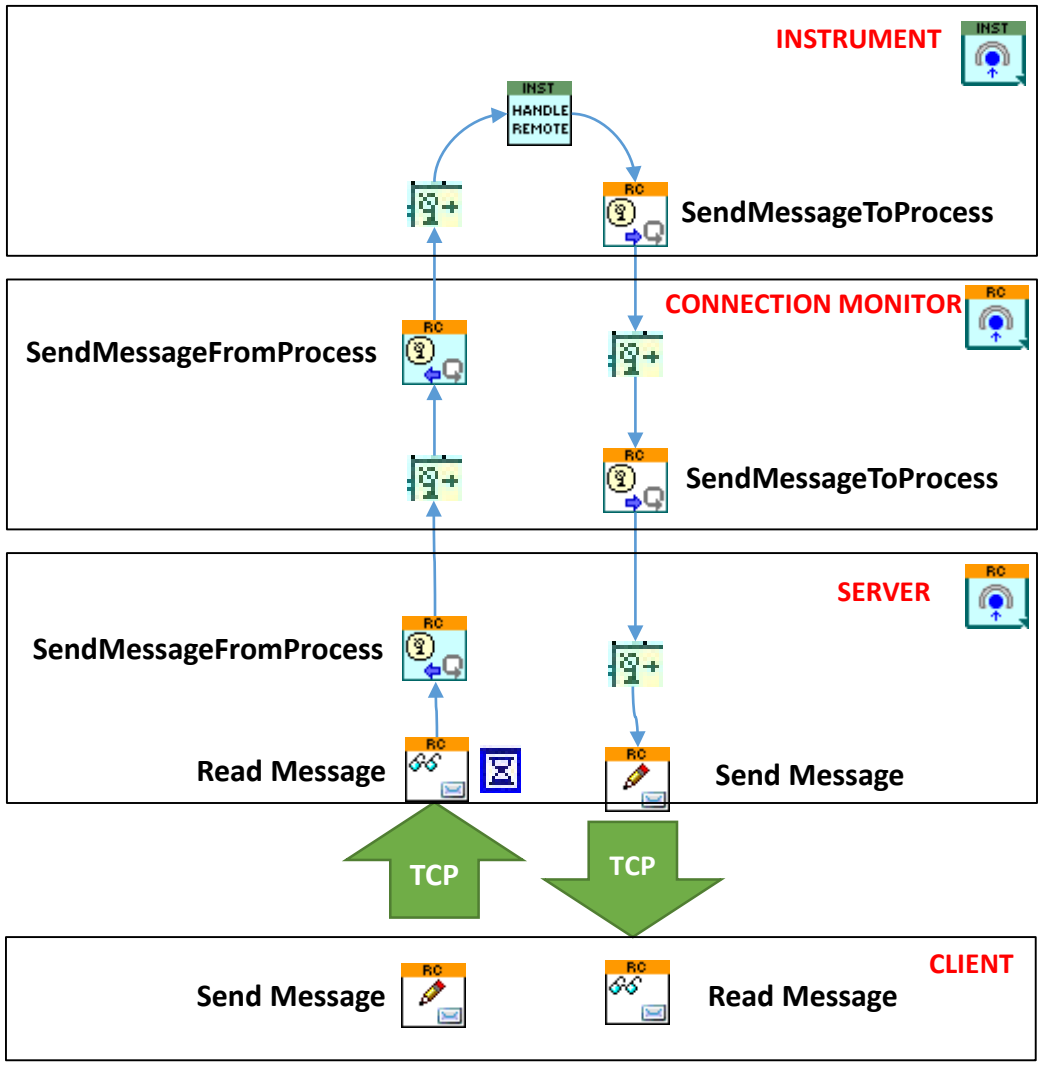
RemoteControl Description (11//29/2018)

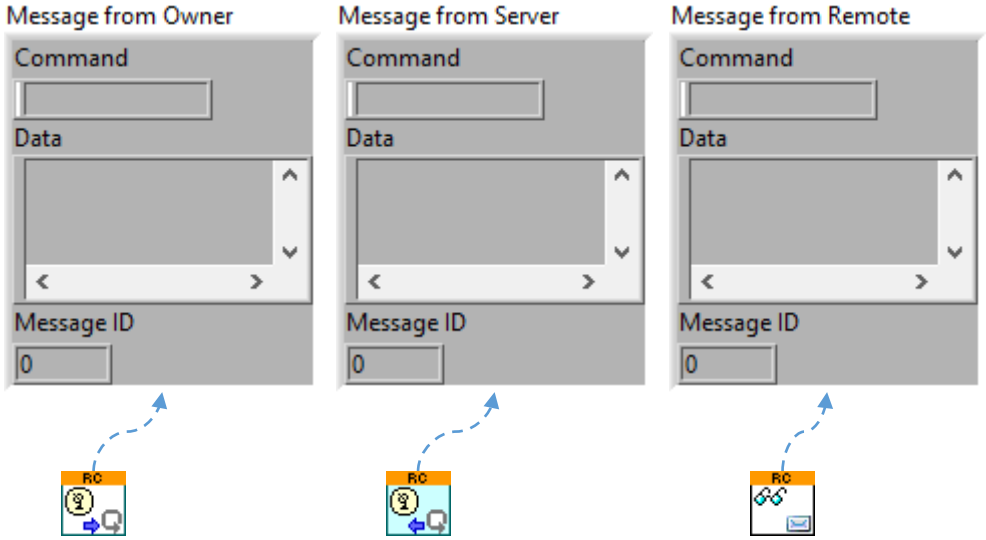
1. “Instrument” starts a RemoteControl (RC) Connection Monitor SMO
2. Connection Monitor listens for clients to connect
3. “Client” starts a RC Client SMO
4. Connection Monitors receives the connection from Client and starts a RC Server SMO
5. Both RC Server and Client poll Read Message according to the timeout
6. Client sends a message by generating a private user event. Client process catches event and calls Client’s Send Message to send message over TCP to Server’s Read Message (5).
7. Server generates public event that is caught by Instrument Process.
8. Instrument processes message and generates private user event to talk back to Client through same call chain described in 6&7.

20. (*how does Instrument know about Server or vice versa?*)

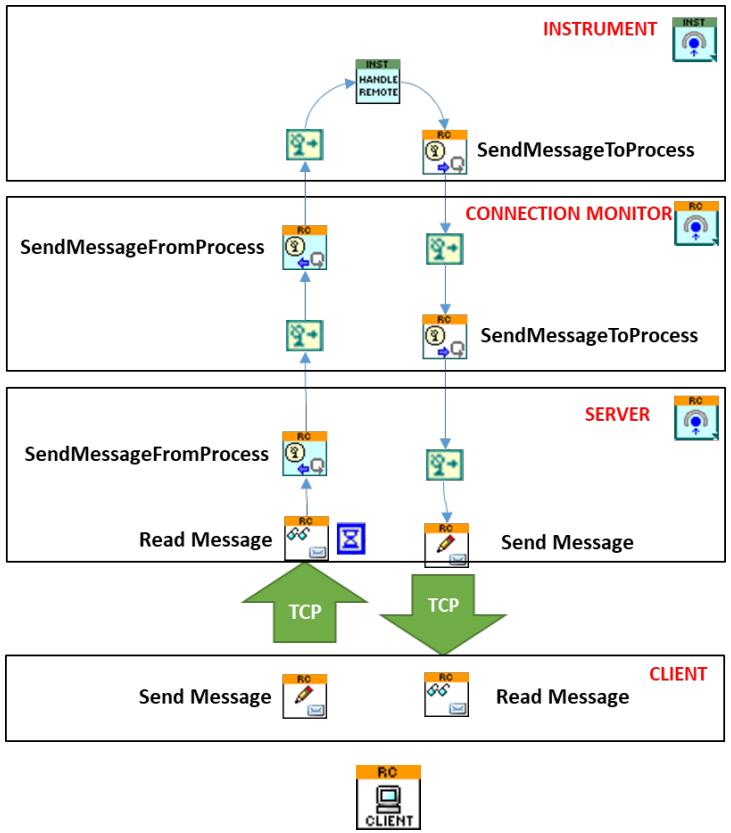


RemoteControl Diagram (6/26/2019)

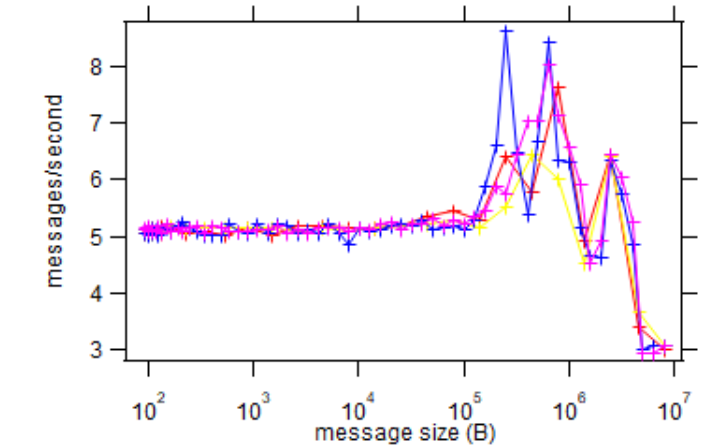


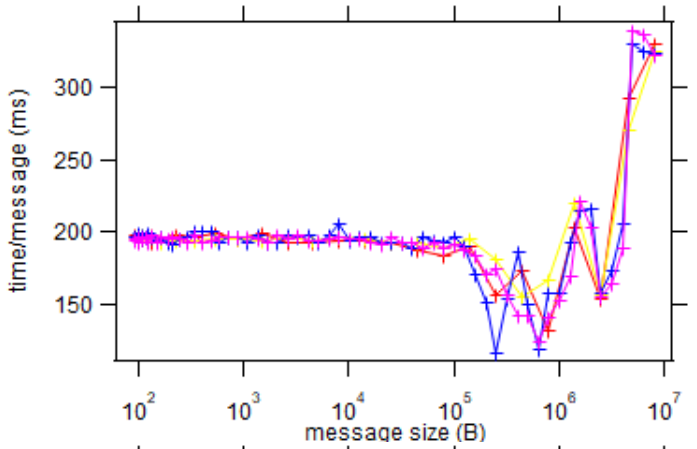


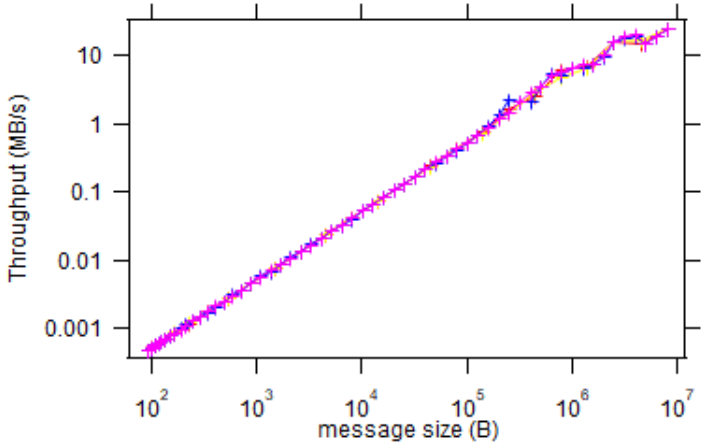
RemoteControl Tests



1. Start Connection Monitor
2. Client send 100 messages of 100 B size and reports average number of messages/second, time/message, and throughput (MB/s)
3. Increase message size and repeat up to ~10MB

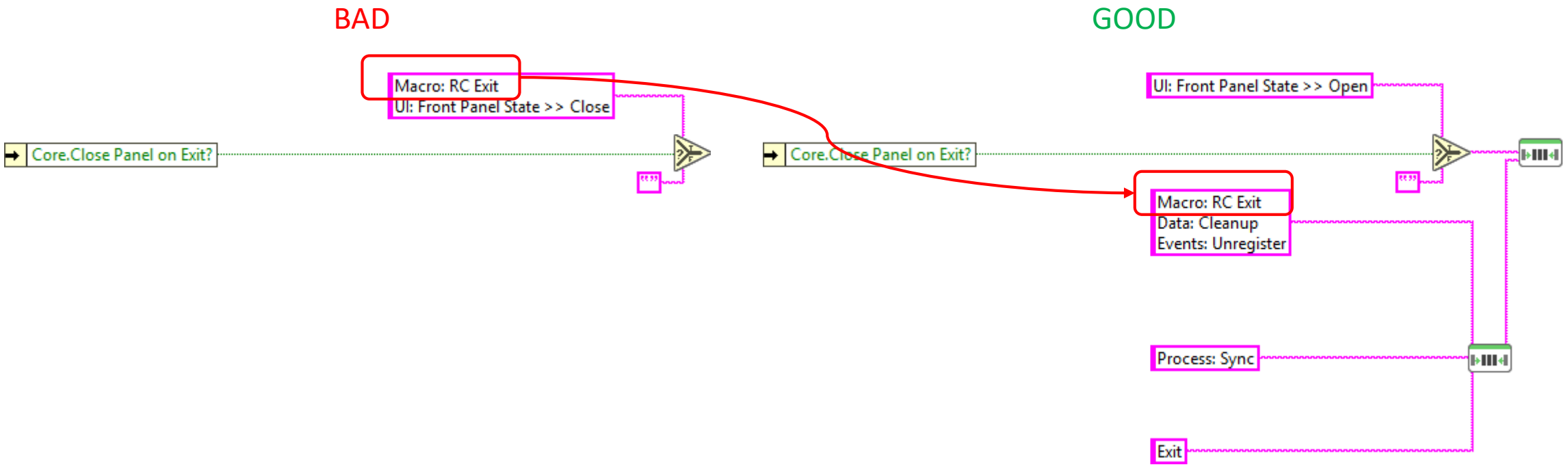






Discussion with Javier 6/27/2019

- RemoteControl Issue:
  - Server process would only stop half of the time
  - If Client closes the connection, Server receives error 66, Server process calls Macro: RC Exit, Macro: RC Exit calls SelfTerminate.
  - SelfTerminate was not being called every other time, because Macro: RC Exit was being called before a selector controlled by “Core.Close Panel on Exit?”



Discussion with Javier 6/27/2019

- Build issue
  - \Templates\ExampleInstrument\ and \Templates\ExampleInstrumentUI\ have some cross-linking. They need to be build in one package
  - ?? – can ExampleInstrumentUI.lvclass be named ExampleInstrument.UI.lvclass? Maybe this would fix the issue with needed to change the class wires after the class is created from the template.

RemoteControl (RC) Diagram

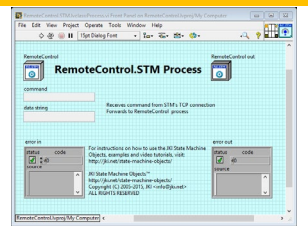
SMO.LevyLab.lvclass
(owns SMO.RC)

SMO Create/Start

SMO Config

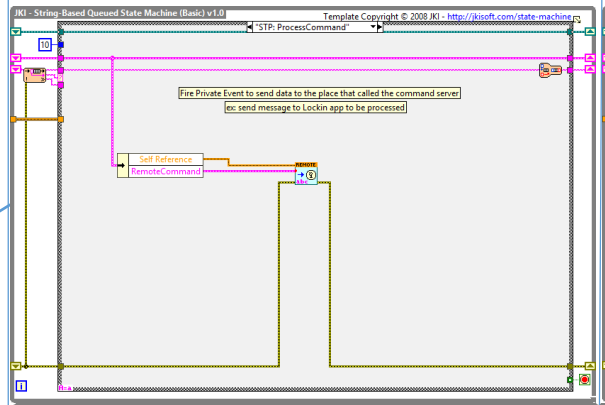
SMO Stop/Destroy

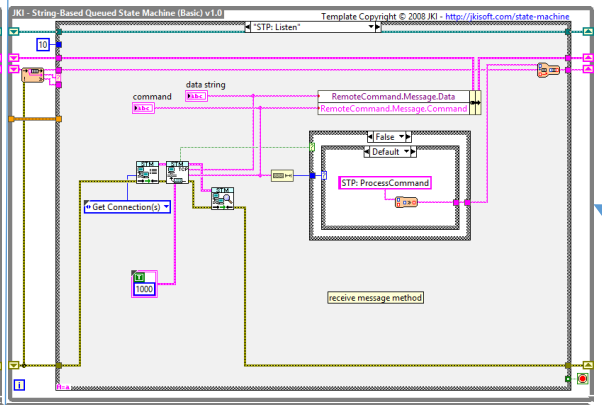
SMO.RC.lvclass

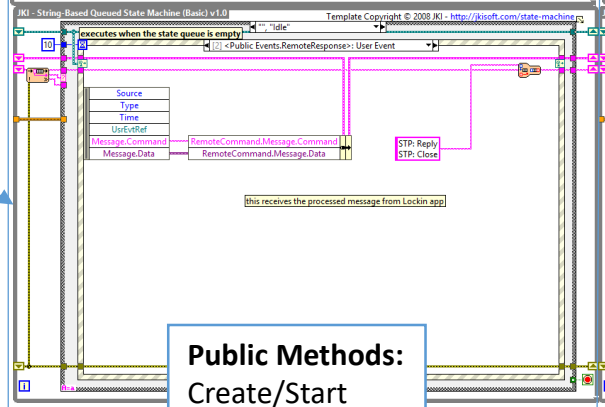


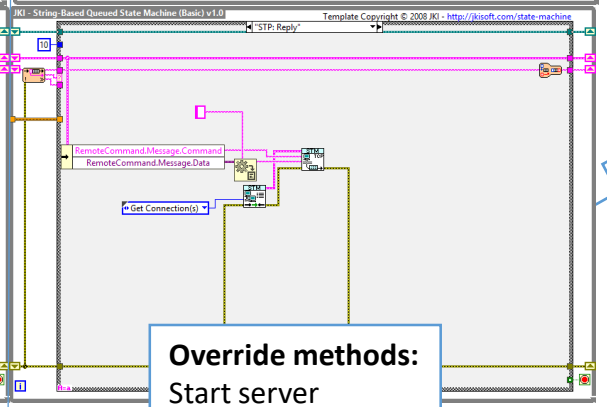
Does not change:

Changes according to comm protocol (TCP, ZMQ, etc):









INST HANDLE REMOTE Handle message

Send Message

Reply Message

Overidden by
SMO.LevyLab.System

Public Methods:
Create/Start
Config
Send Message
Reply Message
Stop/Destroy

Override methods:
Start server
Config Server
Read message
Write message
Stop server

Send Remote Message

Reply Remote Message

NETWORK

Remote Client

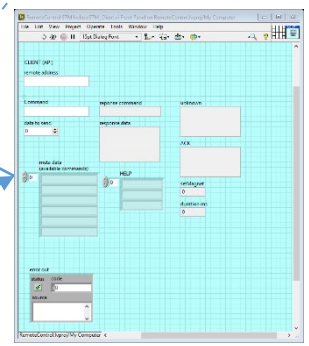
SMO.RC.lvclass provides a template?, example?, override? client

Open Connection

Configure(?)

Close Connection

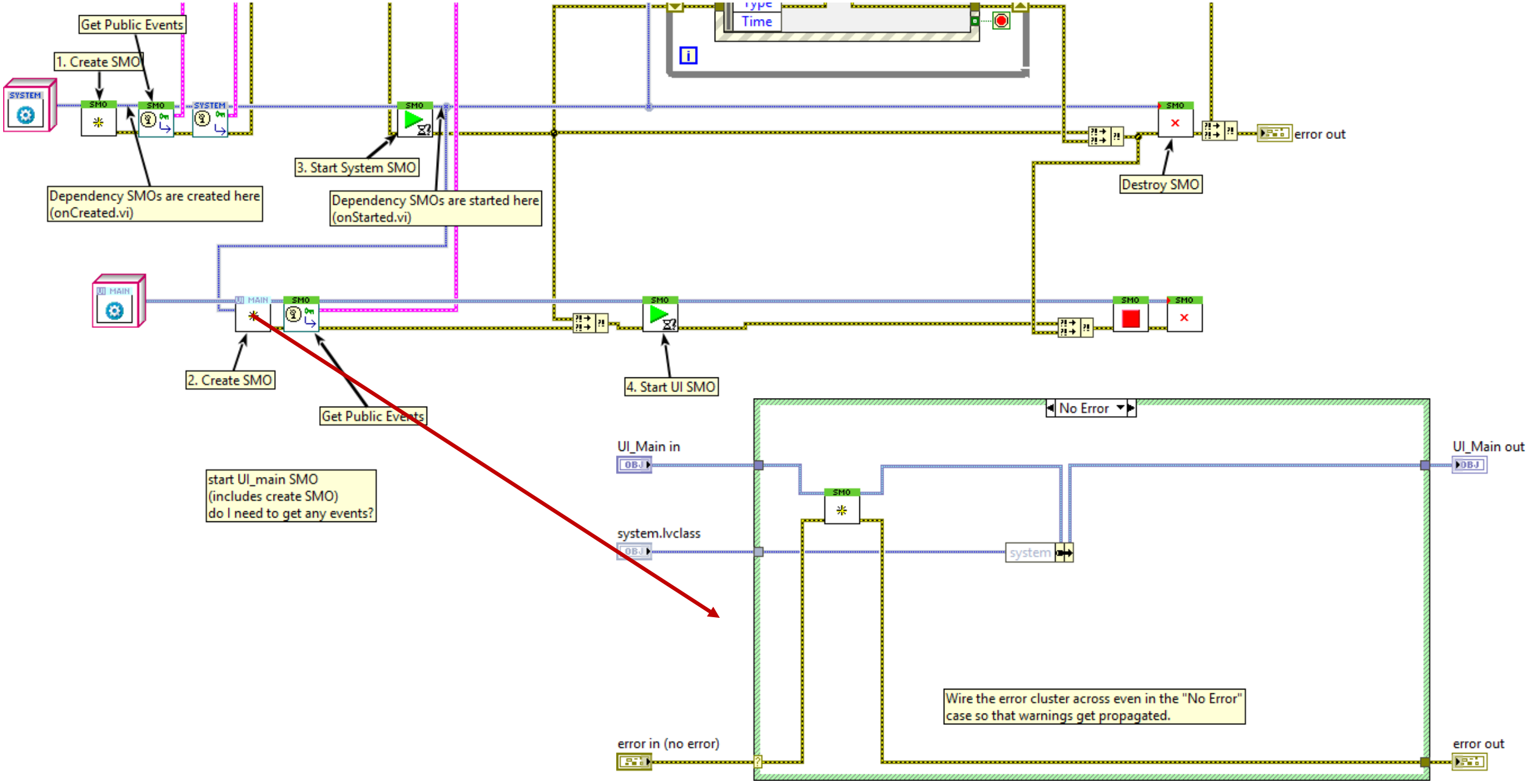
I think these are all override methods



Remote messages can come from python or labview

LabVIEW

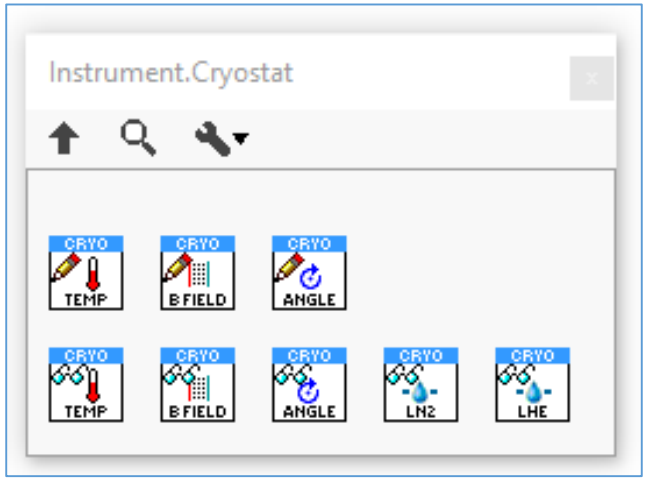
UI


Get Public Events
1. Create SMO
Dependency SMOs are created here
(onCreated.vi)
3. Start System SMO
Dependency SMOs are started here
(onStarted.vi)
Time
Destroy SMO
error out
2. Create SMO
Get Public Events
4. Start UI SMO
start UI_main SMO
(includes create SMO)
do I need to get any events?
No Error
UI_Main in
UI_Main out
system.lvclass
system
Wire the error cluster across even in the "No Error"
case so that warnings get propagated.
error in (no error)
error out

Instrument.Cryostat.lvclass: API Description

- Magnet
  - Set Magnet Field[Target Field, Target Rate, Mode, Axis]
    - Target Field: dbl (Teslas)
    - Target Rate: dbl (Teslas/minute)
    - Mode: enum: 0: Persistent, 1: Driven
    - Axis: enum: 0: Z (*Default*), 1: X, 2: Y
  - Get Magnet Field[Axis] → **Field, Status**
    - Axis: enum: 0: Z (*Default*), 1: X, 2: Y
    - **Field: dbl (Teslas)**
    - **Status: variant**
- Rotator
  - Set Angle[Angle, Axis]
    - Angle: dbl (°)
    - Axis: enum: 0, 1
  - Get Angle[Axis] → **Angle**
    - Axis: enum: 0, 1
    - **Angle: dbl (°)**
- Temperature
  - Set Temperature[Temperature, Rate, Channel]
    - Temperature: dbl (K)
    - Rate: dbl (K/min)
    - Channel: int
  - Get Temperature[Channel] → **Temperature, Status**
    - Channel: int
    - **Temperature: dbl (K)**
    - **Status: variant**
- Level
  - Get Helium Level → **Level**
    - **Level: dbl (%)**
  - Get Nitrogen Level → **Level**
    - **Level: dbl (%)**

- Command[argument, [optional argument]] → **Response**
- An Instrument does not need to override all methods
  - Example 1: PPMS/PPMS2 Model6000 handles all of these commands, so override them all in Instrument.Cryostat.PPMS2
  - Example 2: MNK system uses separate instruments for each:
    - Instrument.Cryostat.Attocube overrides:
      - Rotator
    - Instrument.Cryostat.OxfordITC overrides:
      - Temperature
    - Instrument.Cryostat.OxfordIPS overrides:
      - Level, Temperature, Magnet
    - Instrument.Cryostat.LeidenTC overrides:
      - Temperature
  - SweepControl.vi allows different instruments for each type (Magnet, Temperature, etc)



- There was a problem building PPMS2 Instrument.

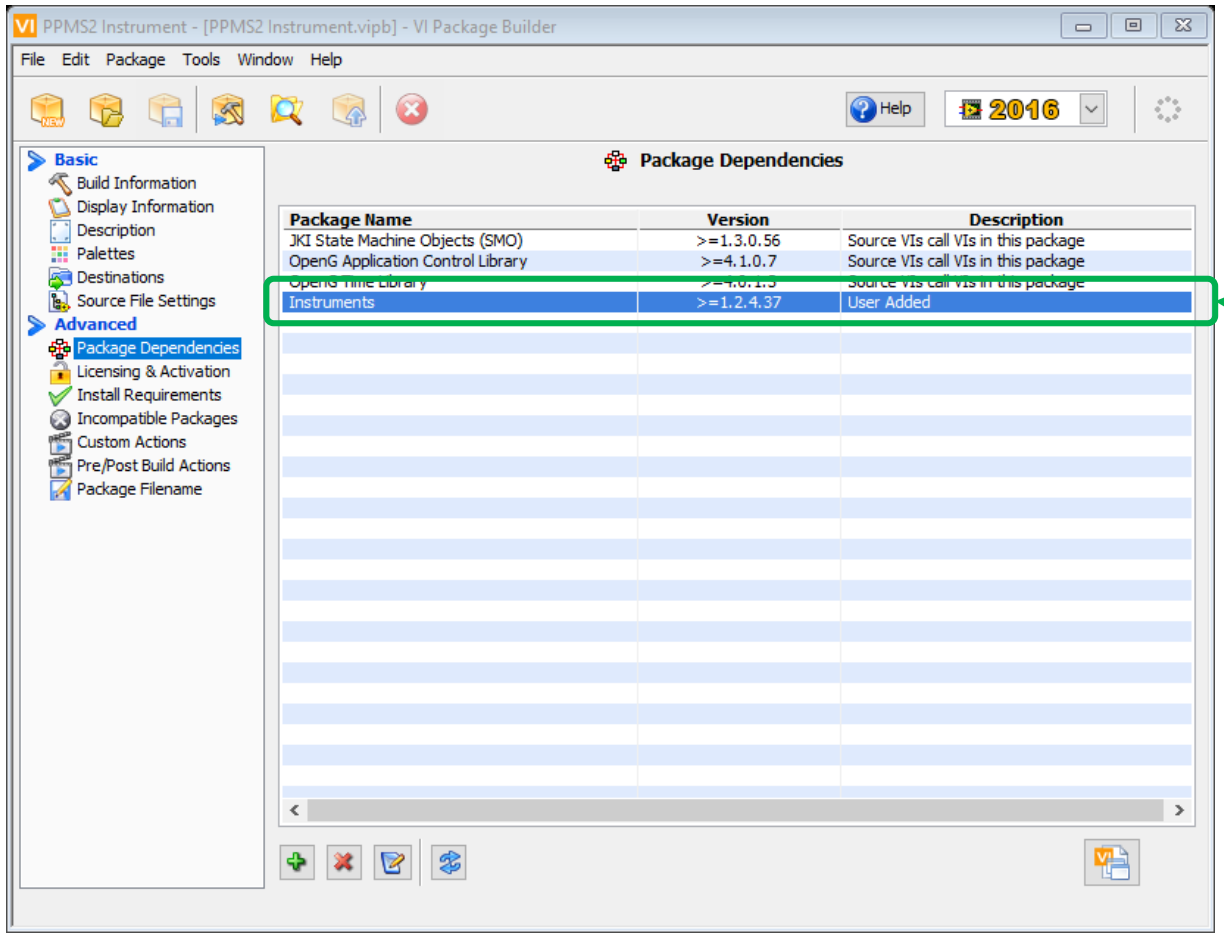
- Error Details:
- Code: 1357
- Source: Error 1357 occurred at Invoke Node in Save VIs to Destination__ogb.vi->Build Application__ogb.vi->Build Application from Build File__ogb.vi->Build Application from Build File__ogb.vi.ProxyCaller

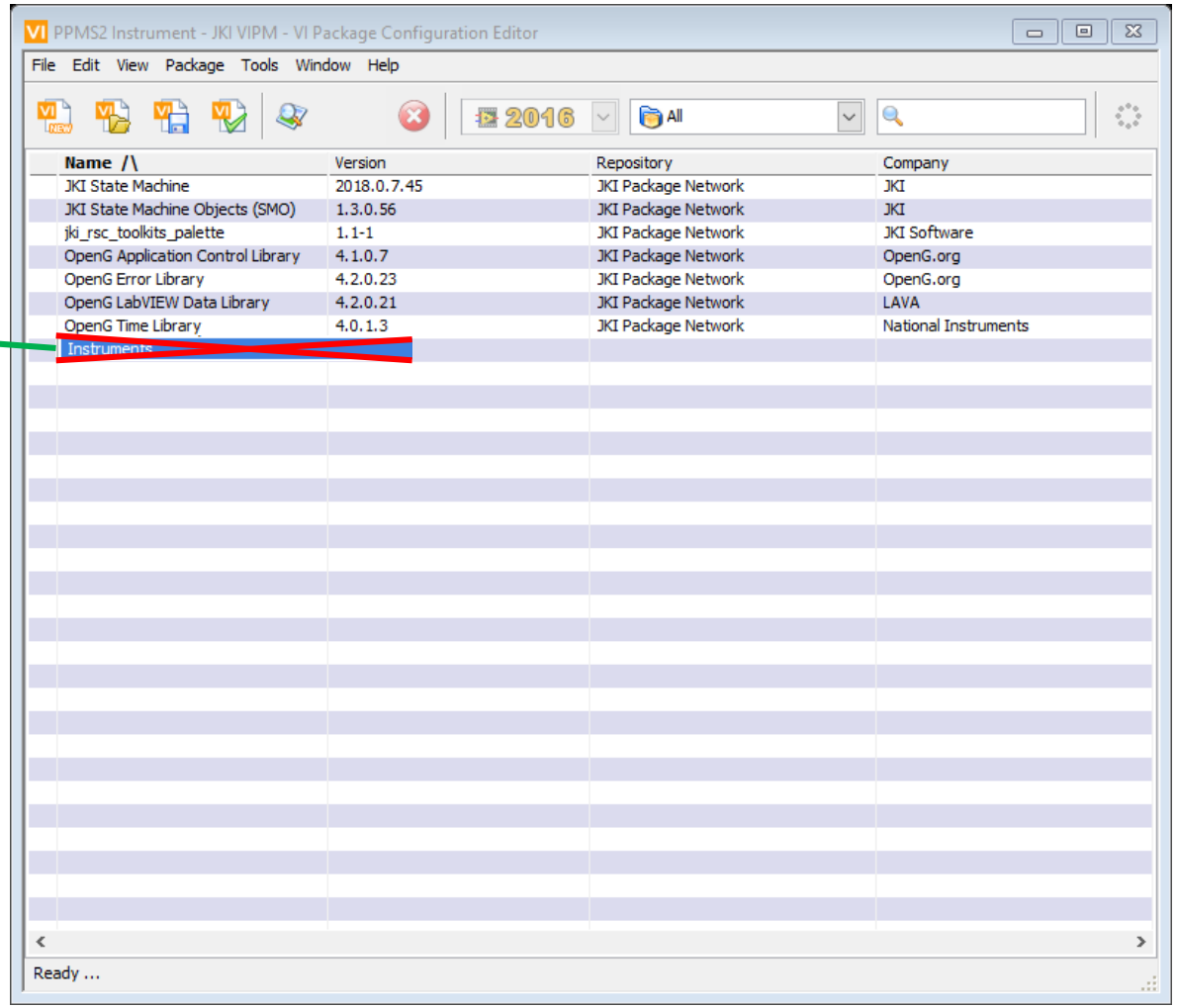
- Possible reason(s):

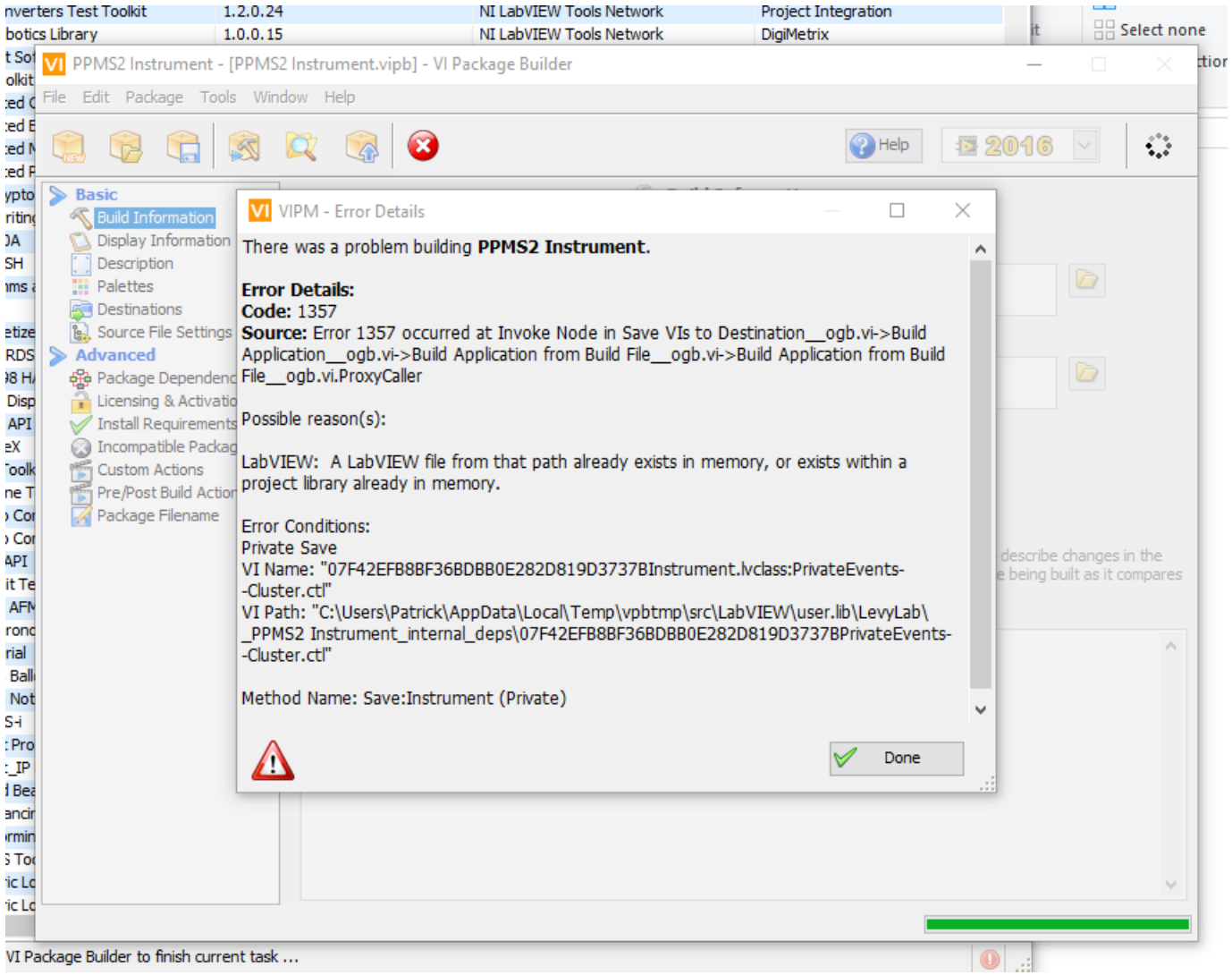
- LabVIEW: A LabVIEW file from that path already exists in memory, or exists within a project library already in memory.

- Error Conditions:
- Private Save
- VI Name: "7F5BF585CD3CEE1343C47873D09E4DD8Logger.DSC.lvclass:PrivateEvents--Cluster.ctl"
- VI Path: "C:\Users\Patrick\AppData\Local\Temp\vpbtmp\src\LabVIEW\user.lib\LevyLab_PPMS2 Instrument_internal_deps\7F5BF585CD3CEE1343C47873D09E4DD8PrivateEvents--Cluster.ctl"

- Method Name: Save:Instrument (Private)

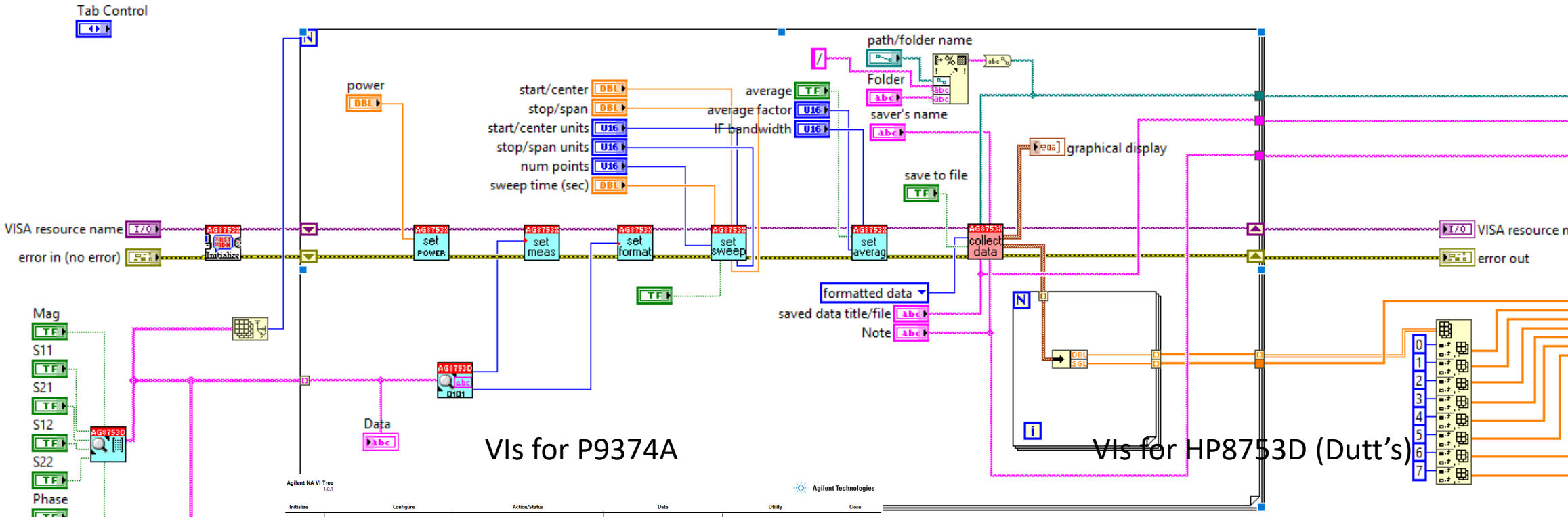

PPMS2 Instrument - [PPMS2 Instrument.vipb] - VI Package Builder
File Edit Package Tools Window Help
Help
2016
Basic
Build Information
Display Information
Description
Palettes
Destinations
Source File Settings
Advanced
Package Dependencies
Licensing & Activation
Install Requirements
Incompatible Packages
Custom Actions
Pre/Post Build Actions
Package Filename
Package Dependencies
Package Name
Version
Description
JKI State Machine Objects (SMO)
>=1.3.0.56
Source VIs call VIs in this package
OpenG Application Control Library
>=4.1.0.7
Source VIs call VIs in this package
Instruments
>=1.2.4.37
User Added


PPMS2 Instrument - JKI VIPM - VI Package Configuration Editor
File Edit View Package Tools Window Help
2016
All
Name /\
Version
Repository
Company
JKI State Machine
2018.0.7.45
JKI Package Network
JKI
JKI State Machine Objects (SMO)
1.3.0.56
JKI Package Network
JKI
jki_rsc_toolkits_palette
1.1-1
JKI Package Network
JKI Software
OpenG Application Control Library
4.1.0.7
JKI Package Network
OpenG.org
OpenG Error Library
4.2.0.23
JKI Package Network
OpenG.org
OpenG LabVIEW Data Library
4.2.0.21
JKI Package Network
LAVA
OpenG Time Library
4.0.1.3
JKI Package Network
National Instruments
Instruments
Ready ...


nverters Test Toolkit 1.2.0.24 NI LabVIEW Tools Network Project Integration
botics Library 1.0.0.15 NI LabVIEW Tools Network DigiMetrix
Select none
PPMS2 Instrument - [PPMS2 Instrument.vipb] - VI Package Builder
File Edit Package Tools Window Help
Help
2016
Basic
Build Information
Display Information
Description
Palettes
Destinations
Source File Settings
Advanced
Package Dependenc
Licensing & Activatio
Install Requirements
Incompatible Packag
Custom Actions
Pre/Post Build Actior
Package Filename
VIPM - Error Details
There was a problem building PPMS2 Instrument.
Error Details:
Code: 1357
Source: Error 1357 occurred at Invoke Node in Save VIs to Destination__ogb.vi->Build Application__ogb.vi->Build Application from Build File__ogb.vi->Build Application from Build File__ogb.vi.ProxyCaller
Possible reason(s):
LabVIEW: A LabVIEW file from that path already exists in memory, or exists within a project library already in memory.
Error Conditions:
Private Save
VI Name: "07F42EFB8BF36BDBB0E282D819D3737BInstrument.lvclass:PrivateEvents-Cluster.ctl"
VI Path: "C:\Users\Patrick\AppData\Local\Temp\vpbtmp\src\LabVIEW\user.lib\LevyLab_PPMS2 Instrument_internal_deps\07F42EFB8BF36BDBB0E282D819D3737BPrivateEvents-Cluster.ctl"
Method Name: Save:Instrument (Private)
Done
describe changes in the
e being built as it compares
VI Package Builder to finish current task ...

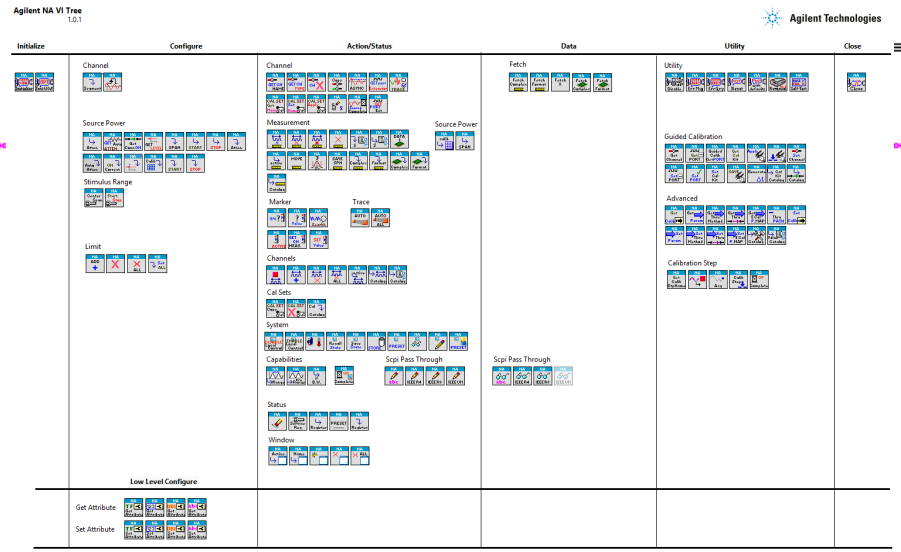
Instrument.VNA.lvclass: API Description

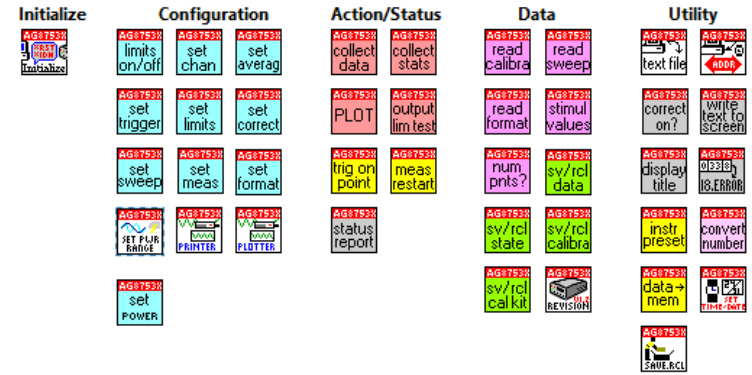
- Initializ
- Set Pov
- Set me
  -
- Set for
  -
- Set bia
  -
  -
  -
- Set swe
  -
  -
  -
  -
  -
- Set ave
  -
  -
  -
- Collect data
  - Save to file (T/F)
  - Data format
  - Frequency
  - Folder, file, title, note…



VIs for P9374A

VIs for HP8753D (Dutt's)

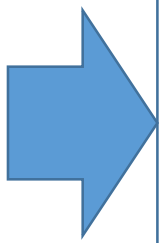




Instrument.VNA.lvclass: API Description

Step 1: Set VNA parameters
Directly give commands to VNA

- Initialize VNA
- Set Power
- Set measurement
  - S11, S21, S12, S22 (choose)
- Set format
  - Log Mag, Phase (choose)
- Set sweep
  - Start/stop or center/span (T/F)
  - Start/center frequency
  - Stop/span frequency
  - Number of points
  - Sweep time
- Set average
  - Average or not(T/F)
  - Average factor
  - IF bandwidth
- Collect data setting
  - Save to file (T/F)
  - Data format
  - Folder, file, title, note…



Step 2: Sweep control and take VNA data every sweep

sweep bias
or sweep Vbg
or sweep B field
or sweep temperature
or …
by Sweep control

Collect a sweep data from VNA (smallest loop)

- Frequency
- S parameters
- Log mag, phase

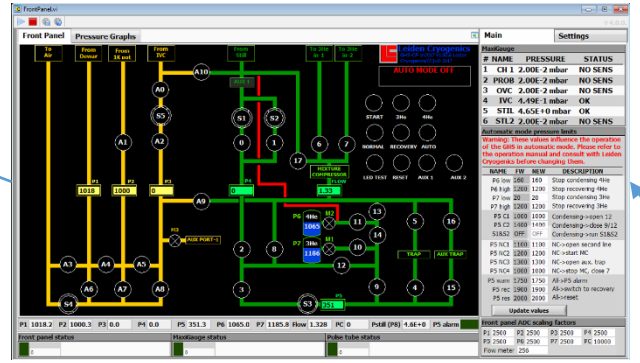
Instrument.VoltageUpdate.lvclass

- VoltageUpdate private data:
  - ~~+ gain in “configuration”~~
    - ~~Add a configuration window~~
  - ~~Voltage → DAQ Voltage & Amplified Voltage~~
- Constants
  - SMO Name-Constant.vi (overridable)
  - ~~SupportedCommands-Constant.vi (overridable)~~
- ~~Configure API Methods~~
- ~~VoltageUpdate UI~~
- DSC Process
  - Option 1: PPMS 2 VoltageUpdate
  - Option 2: VoltageUpdate_ComputerName (e.g. VoltageUpdate_BLACKPEARL)
  - Option 3: VoltageUpdate_S/N
- Instrument.lvclass:
  - Add accessor DSC process
  - Joe Albro: VI tags for scripting Instrument/Instrument UI.vi
  - Configuration stuff is a mess

Experiment 1: Hardware Overview

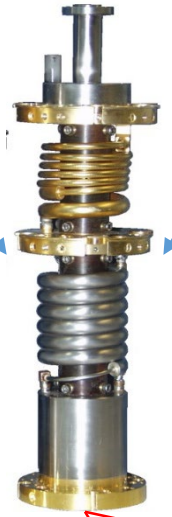
Citadel Historical Database

Front Panel (FP) Valve Controller



LabVIEW Instrument (FP)

Dilution Refrigerator (T=20mK)



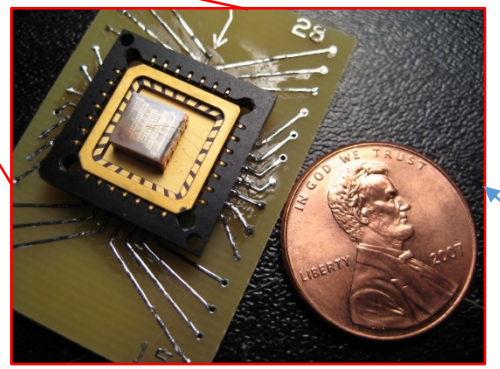
Temperature Controller (TC)

LabVIEW Instrument (TC)

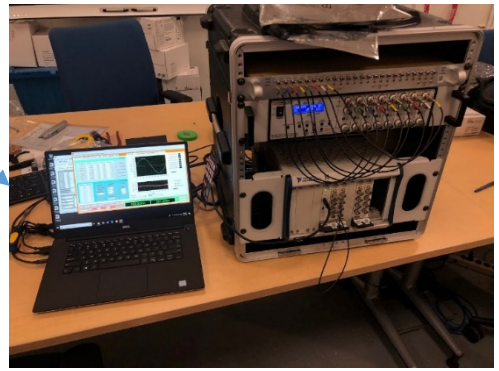
LabVIEW Instrument (PS)

Superconducting Magnet Power Supply (PS)

Superconducting Magnet



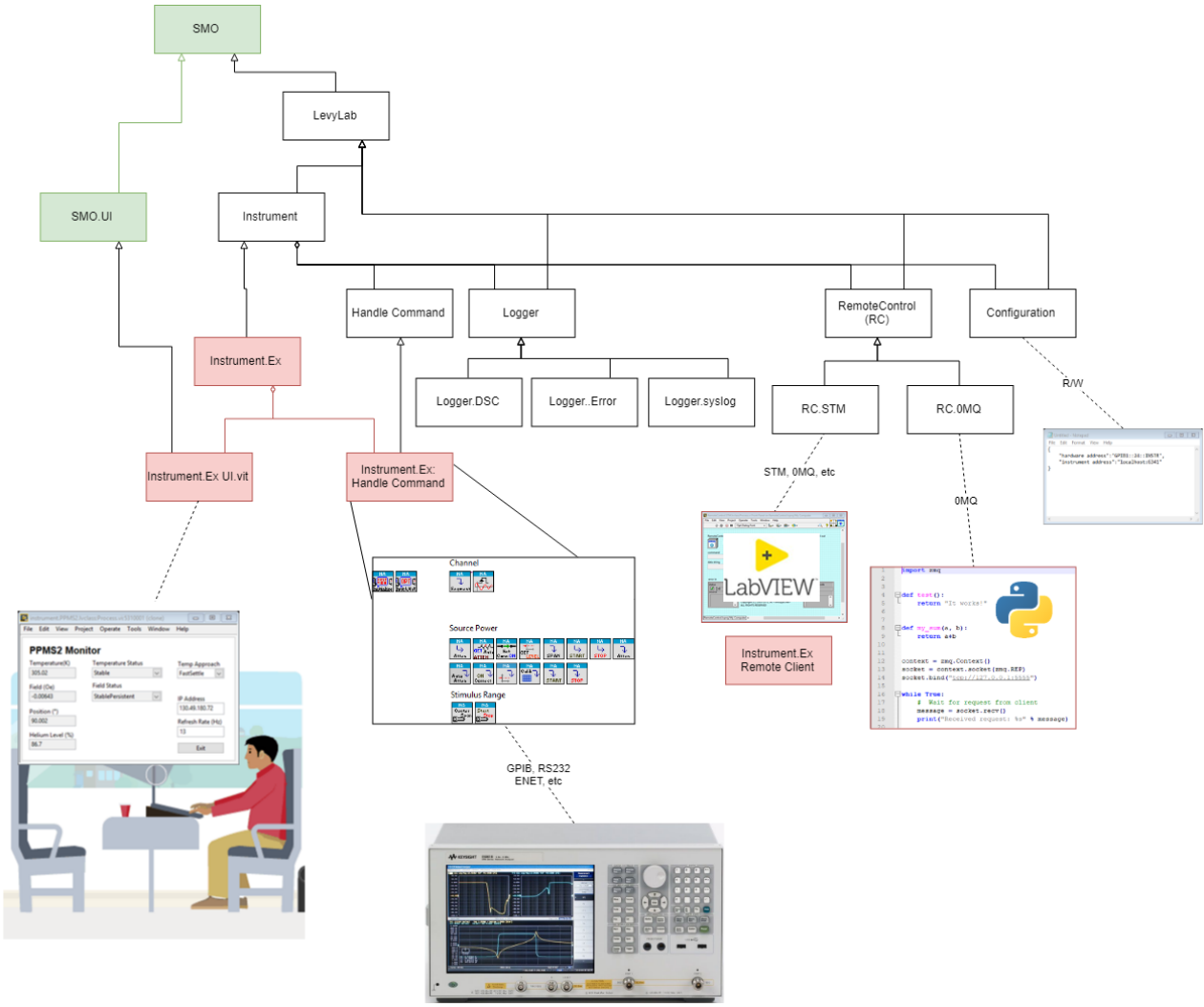
sample



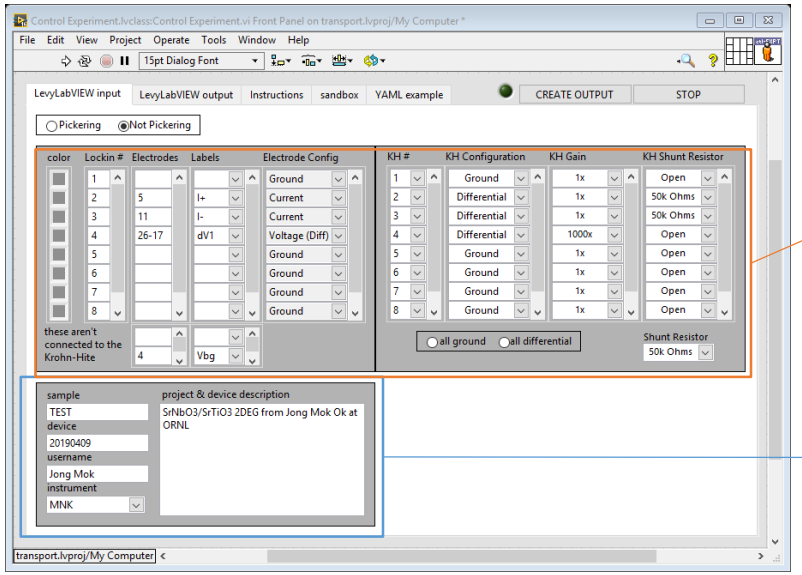
Data Acquisition (DAQ)

Experiment 2: LabVIEW Instrument

- A LabVIEW Instrument has the following responsibilities:
  - Know how to communicate with a piece of hardware (drivers)
  - Poll the instrument for its settings and log to a database
  - Open an API for external programs (e.g through cross-platform protocol such as 0MQ)
  - Provide a UI (optional but probably desirable)



Experiment 3: An “Experiment”

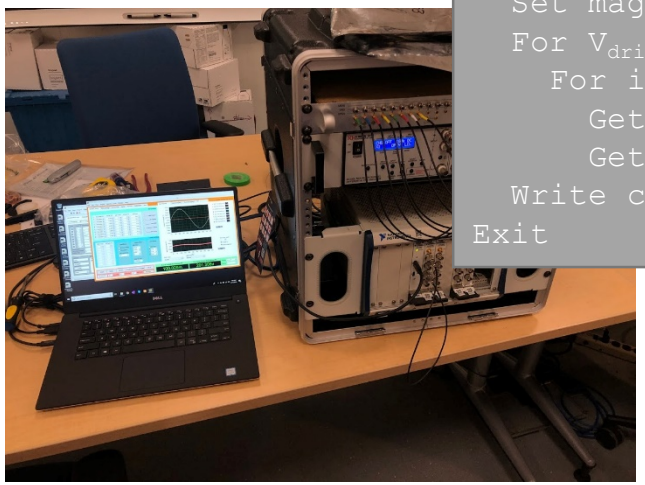


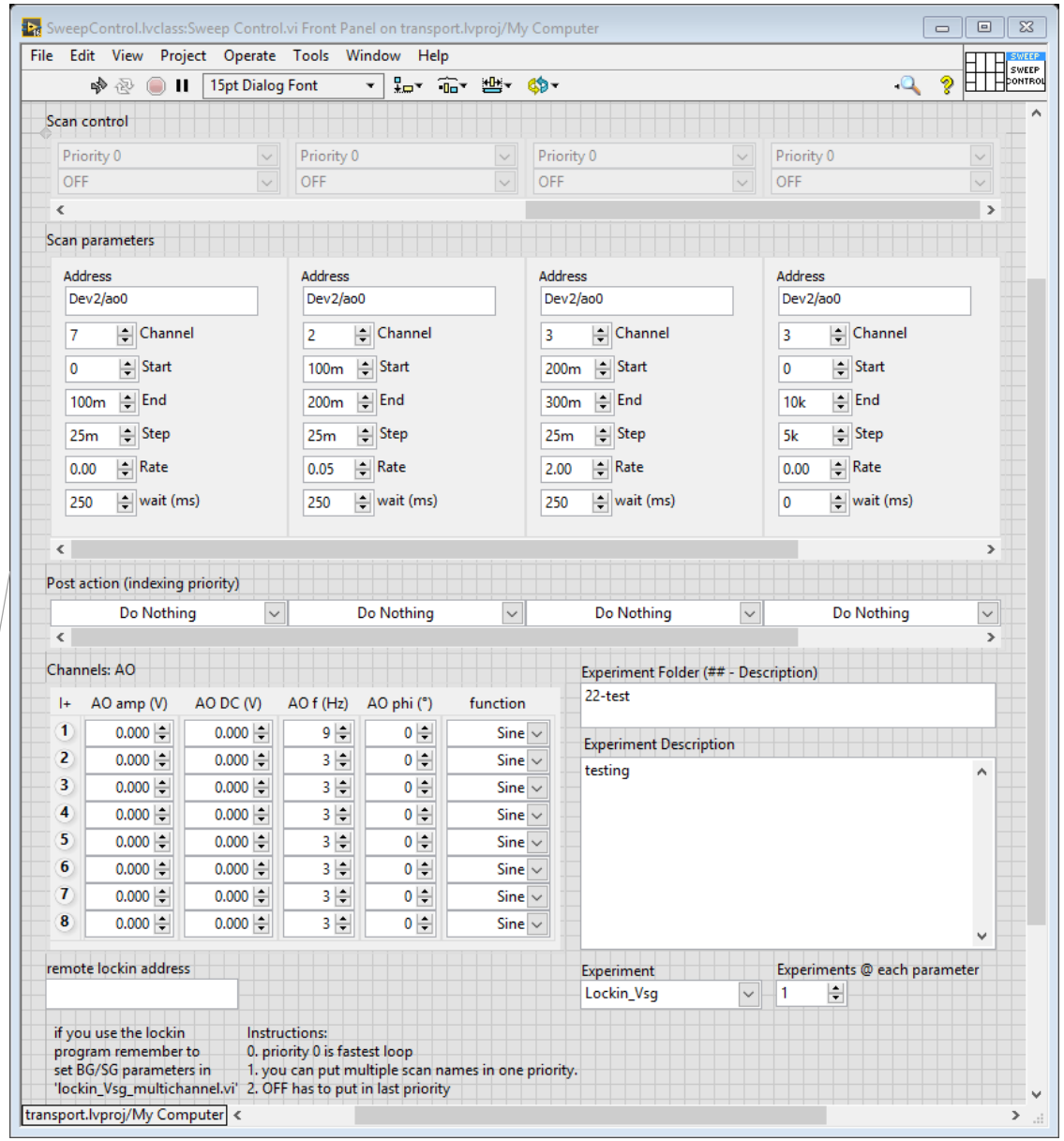
1. Configure wiring and voltage and current amplifiers

2. Define global settings such as sample name and description.

3. Define experiment

```
Initialize some stuff
For B = 0 to B_max:
  Set magnet = B
  For V_drive = 0 to V_drive,max:
    For i = 0 to N
      Get current_i
      Get voltage_i
  Write current and voltage to File_B.itx
Exit
```





Experiment 4: ITX (Igor Text File)* Description 1

```
IGOR
X// Date: 1/12/2017 8:40 PM
X// sweep B: 3.5T to 9.5T, 0.06 T/min
X// large axis: 147, small axis: 257 (20 deg)
X// sweep Vsg: 0mV to 120mV 250uV step
X// Vbg=0.3V
X//
X// tc=0.3s
X// order=4
X// 60 Hz filter OFF
X//
X// lockin:
X// 0: 1, voltage
X// 1: 2, current, 100uV AC 240uV DC, 13 Hz
X// 2: 3, voltage
X// 3: 4, voltage
X// 4: 5, current
X// 5: 7, voltage
X//
X// KH gian=100x, shurt resistor=50kOhm
```

- First line is always “IGOR”

- Metadata begins with X//
- This section should describe the experiment that was done
- This one describes that 1. B will be swept from 3.5 Tesla to 9.5 Tesla at a rate of 0.06 Tesla/minute. 2. Vsg will be swept from 0 mV to 120 mV in 250 uV steps.
- Each file will be one Vsg sweep. The folder will contain data between B = 3.5 T and 9.5 T. And as many Vsg sweeps that could be taken while B was changing.

- This section describes how the lockin was configured.
- The lockin measures current or voltage on the sample in response to some source voltage stimulus.
- There are typically 8 lockin channels (not all need to be used) and there are X and Y for each, which are in-phase and out-of-phase, respectively, with the source voltage.
- The information here means:
  - Lockin0 (connected to electrode 1) measures a voltage
  - Lockin1 (connected to electrode 2) sources a current using a 100 uV source voltage at 13 Hz
  - …
  - Lockin5 (connected to electrode 7) measures a voltage

*Described in more detail on p.II-131 here: https://www.wavemetrics.net/doc/igorman/II-09%20Data%20Import%20Export.pdf

Experiment 4: ITX Description 2

```
21
22  WAVES/D/N=(481) 'time.SA02836H.20170110.000002'
23  BEGIN
24  3.567116343758267E+9
25  3.567116353762839E+9
26  3.567116353912848E+9
27  3.567116354064857E+9
28  3.567116354215865E+9
29  3.567116354440878E+9
30  3.567116354590887E+9
31  3.567116354818900E+9
```

- Here starts a wave of size 481. This one is the time
- (seconds since epoch time (1904 I think))

- Sample code: SANNNNNL.YYYYMMDD
  - SA: Sample
  - NNNN: 5 digit sample number
  - L: one of 16 “canvases” on a particular sample
  - YYYY: Year
  - MM: Month
  - DD: Day

```
497  3.567116453997572E+9
498  3.567116454222585E+9
499  3.567116454298590E+9
500  3.567116454448598E+9
501  3.567116454599607E+9
502  3.567116454824620E+9
503  3.567116454986629E+9
504  3.567116455136638E+9
505  END
```

- End of the wave

```
1477  WAVES/D/N=(1) 'B.SA02836H.20170110.000002'
1478  BEGIN
1479  3.801700000000000E+0
1480  END
1481
```

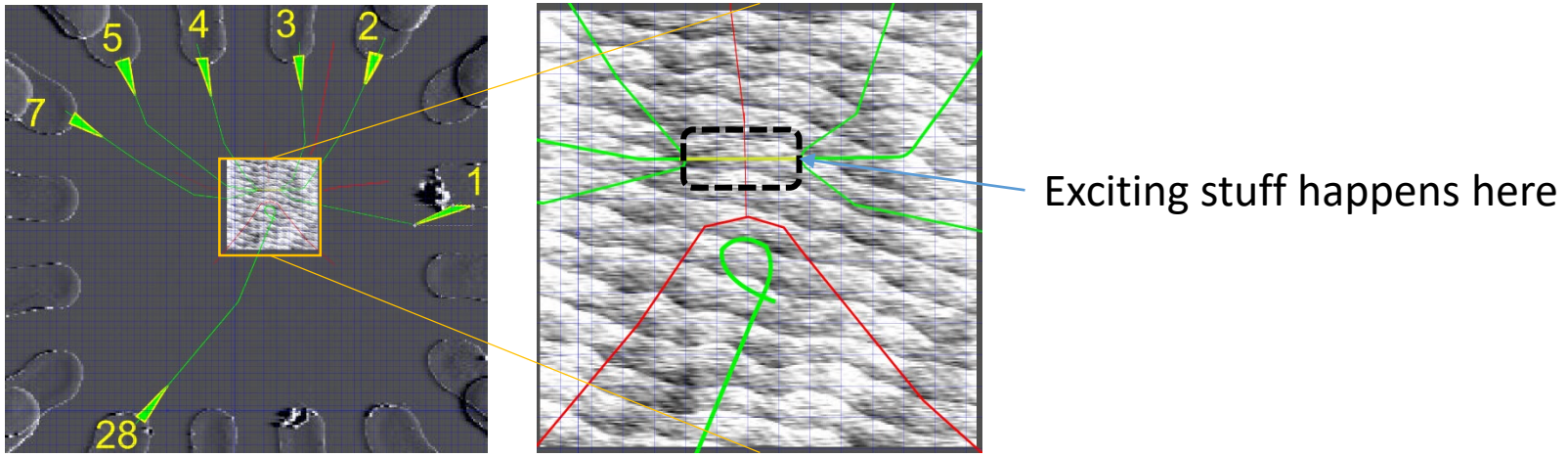
- Wave of size 1 showing the B field for THIS file

```
1482  WAVES/D/N=(481) 'X0.SA02836H.20170110.000002'
1483  BEGIN
1484  100.586757640980660E-6
1485  100.562881961642580E-6
1486  100.554882945805970E-6
1487  100.548795103440390E-6
1488  100.541729931379820E-6
1489  100.540689133795410E-6
1490  100.545901069673990E-6
1491  100.563244064805010E-6
1492  100.589700051043400E-6
```

- Wave for In-phase (X) Lockin0

Experiment 5: Data Processing 1

1. Look at experiment notebook for sample SA02836H.20170110. There should be a description & diagram of the device:



2. We want to plot the Four terminal resistance of the exciting section. So we need a current and voltage on each side of the device.

2a. The metadata tells us electrode 2 and 5 were current source and drain. Then pick a voltage on each side, e.g. 3 and 4, 1 and 7, 3 and 7, or 1 and 4.

2b. Now go back to the metadata again and decipher which lockins those correspond to*. Let's choose electrodes 1(V), 2(I), 4(V), 5(I), which are lockins 0(V), 1(I), 3(V), 4(I).

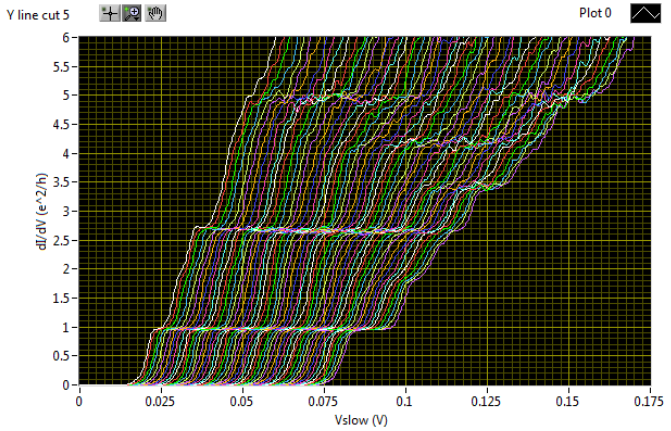
3. The 1st dependent variable is V_{sg}. The 2nd is B

4. Read the waves X0.*, X1.*, X3.*, X4.*, Vsg.*, B.* from each file. Calculate $R_{4T} = \frac{V_1 - V_2}{I} = \frac{\mathrm{X3} - \mathrm{X0}}{\mathrm{X4}}$. You now have R_{4T} vs V_{sg} for each file, each of which is one value of B.

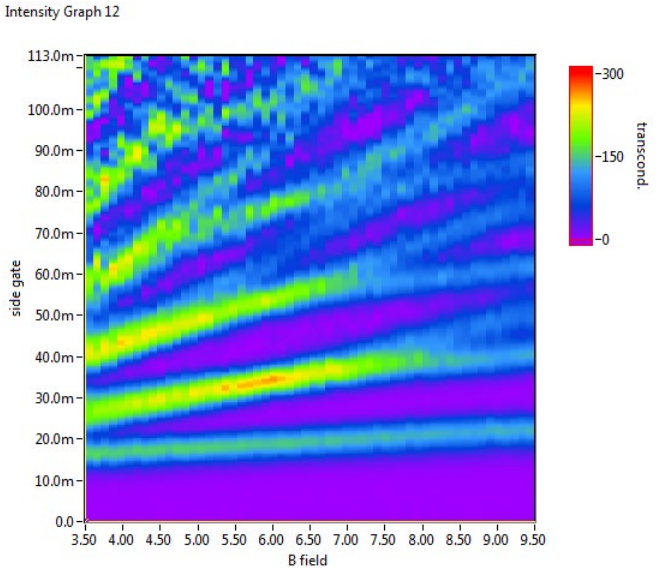
**I will not be offended if you roll your eyes at this point*

Experiment 5: Data Processing 2

5. We also typically calculate four terminal conductance $\frac{dI}{dV} = G_{4T} = \frac{1}{R_{4T}}$ and expressed in units of e^2/h (electron charge squared/Planck constant). Each line in the next graph is data from one file:

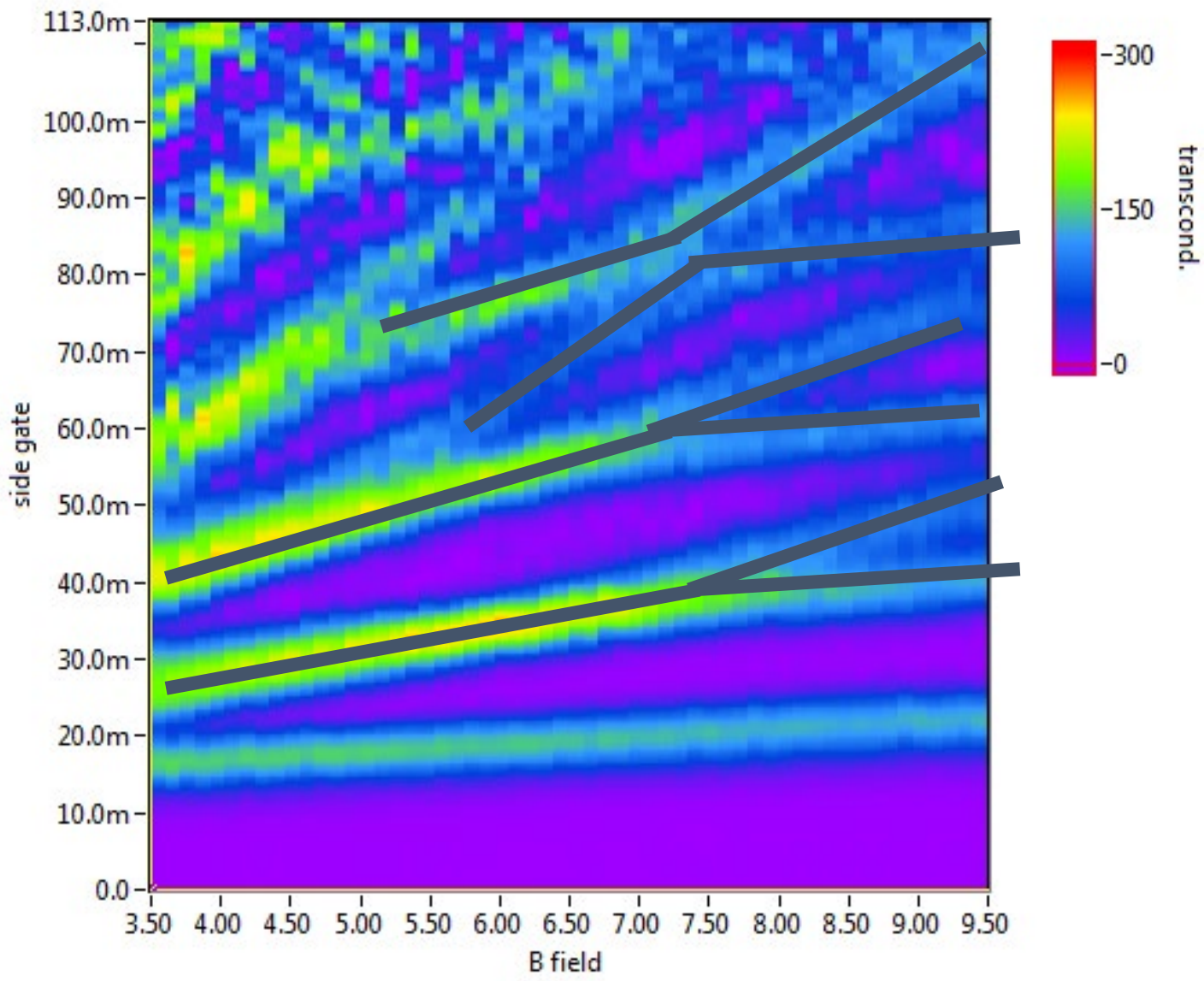


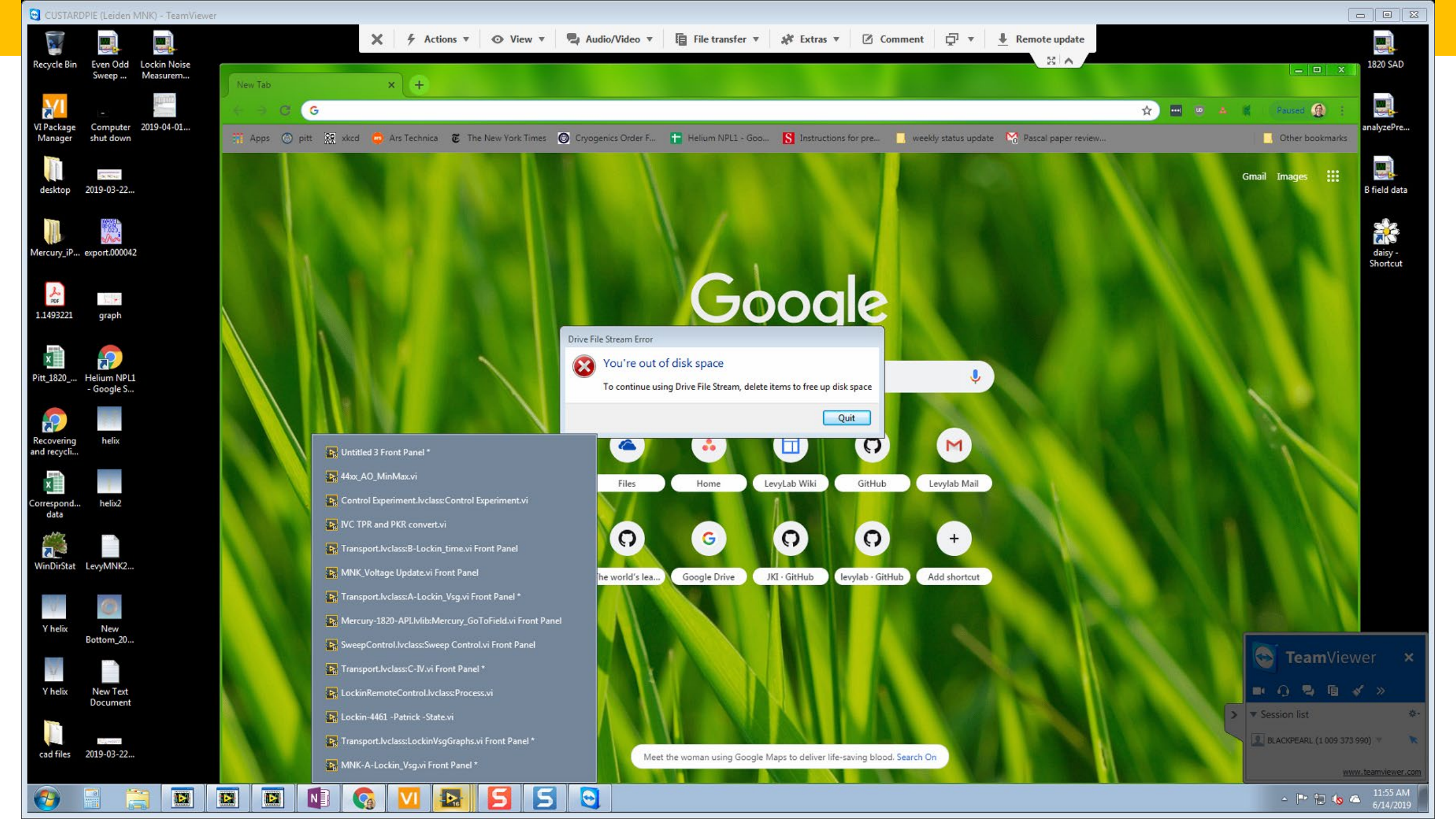
6. Calculate the transconductance $\frac{dG_{4T}}{dV_{sg}}$ by numerically differentiating each curve. We typically plot this as an intensity plot vs Vsg and B, with color representing the transconductance.



Experiment 6: Next-Level Analysis

7. Use R magic to analyze locations of splittings/crossing/avoided crossings as a function of B and Vsg

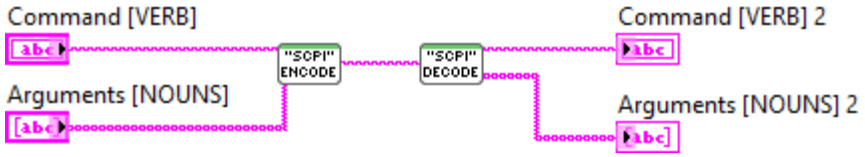



CUSTARDPIE (Leiden MNK) - TeamViewer
Actions
View
Audio/Video
File transfer
Extras
Comment
Remote update
Recycle Bin
Even Odd Sweep ...
Lockin Noise Measurem...
VI Package Manager
Computer shut down
2019-04-01...
desktop
2019-03-22...
Mercury_iP...
export.000042
1.1493221
graph
Pitt_1820_...
Helium NPL1 - Google S...
Recovering and recycli...
helix
Correspond... data
helix2
WinDirStat
LevyMNK2...
Y helix
New Bottom_20...
Y helix
New Text Document
cad files
2019-03-22...
1820 SAD
analyzePre...
B field data
daisy - Shortcut
New Tab
Apps
pitt
xkcd
Ars Technica
The New York Times
Cryogenics Order F...
Helium NPL1 - Goo...
Instructions for pre...
weekly status update
Pascal paper review...
Other bookmarks
Paused
Gmail
Images
Google
Drive File Stream Error
You're out of disk space
To continue using Drive File Stream, delete items to free up disk space
Quit
Files
Home
LevyLab Wiki
GitHub
Levylab Mail
he world's lea...
Google Drive
JKI · GitHub
levylab · GitHub
Add shortcut
Untitled 3 Front Panel *
44xx_AO_MinMax.vi
Control Experiment.lvclass:Control Experiment.vi
IVC TPR and PKR convert.vi
Transport.lvclass:B-Lockin_time.vi Front Panel
MNK_Voltage Update.vi Front Panel
Transport.lvclass:A-Lockin_Vsg.vi Front Panel *
Mercury-1820-API.lvlib:Mercury_GoToField.vi Front Panel
SweepControl.lvclass:Sweep Control.vi Front Panel
Transport.lvclass:C-IV.vi Front Panel *
LockinRemoteControl.lvclass:Process.vi
Lockin-4461 -Patrick -State.vi
Transport.lvclass:LockinVsgGraphs.vi Front Panel *
MNK-A-Lockin_Vsg.vi Front Panel *
Meet the woman using Google Maps to deliver life-saving blood. Search On
TeamViewer
Session list
BLACKPEARL (1 009 373 990)
www.teamviewer.com
11:55 AM
6/14/2019

Pseudo SCPI Message String for chaining arbitrary number of arguments 2

Example:
setConfig >> Sys >> Time >> hh:mm:ss
setConfig >> Sys >> Date >> yyyy:mm:dd
setSignal >> Magnet >> Target Current >> float (-170 to +1

Inspiration: SCPI:
<VERB>:<NOUN>:<NOUN>



IEEE Mandated Commands

All SCPI instruments shall implement all the common commands declared mandatory by *IEEE 488.2*.

Mnemonic	Name	488.2 Section
*CLS	Clear Status Command	10.3
*ESE	Standard Event Status Enable Command	10.10
*ESE?	Standard Event Status Enable Query	10.11
*ESR?	Standard Event Status Register Query	10.12
*IDN?	Identification Query	10.14
*OPC	Operation Complete Command	10.18
*OPC?	Operation Complete Query	10.19
*RST	Reset Command	10.32
*SRE	Service Request Enable Command	10.34
*SRE?	Service Request Enable Query	10.35
*STB?	Read Status Byte Query	10.36
*TST?	Self-Test Query	10.38
*WAI	Wait-to-Continue Command	10.39

SCPI Requirements

IEEE 488.2 describes the syntax of programming and device behavior to a certain level. Further refinement of the syntax and addition rules are imposed by SCPI in order to give instruments a common "look and feel". The "Syntax and Style" volume of this standard describes the concepts behind SCPI and sets guidelines for originating new commands. A SCPI device shall follow these guidelines and style requirements.

Required Commands

The following commands are required in all SCPI instruments:

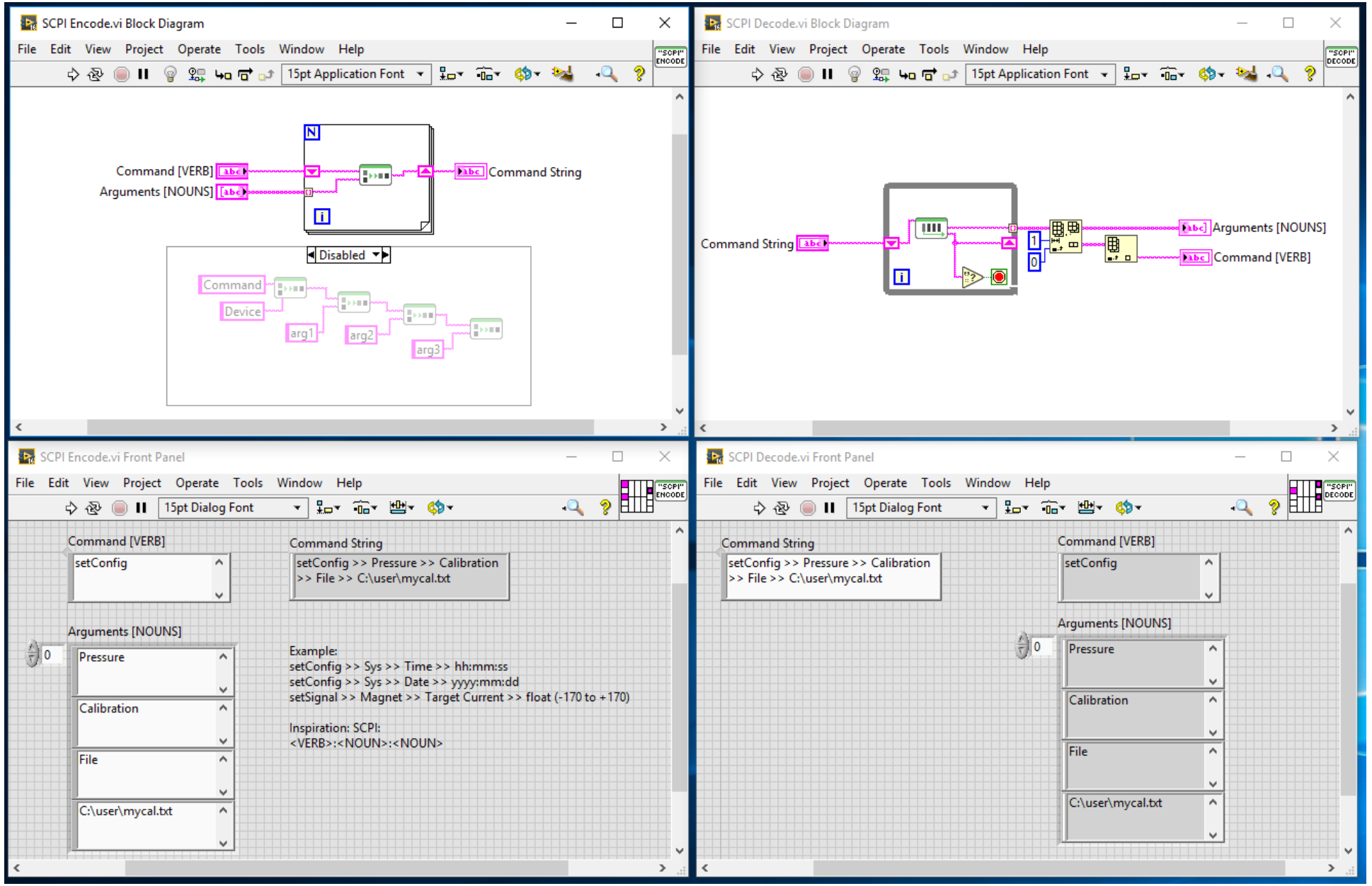
Mnemonic	Command Reference Section	Syntax and Style Section
:SYSTem		
:ERRor	21.8	
[:NEXT]?	21.8.3e (see Note 1)	1996
:VERSion?	19.16 (see Note 2)	1991
:STATus	18	5
:OPERation		
[:EVENt]?		
:CONDition?		
:ENABle		
:ENABle?		
:QUEStionable		
[:EVENt]?		
:CONDition?		
:ENABle		
:ENABle?		
:PRESet		

https://forums.ni.com/t5/Example-Program-Drafts/Using-Existing-C-Code-or-a-DLL-in-LabVIEW/ta-p/3499233

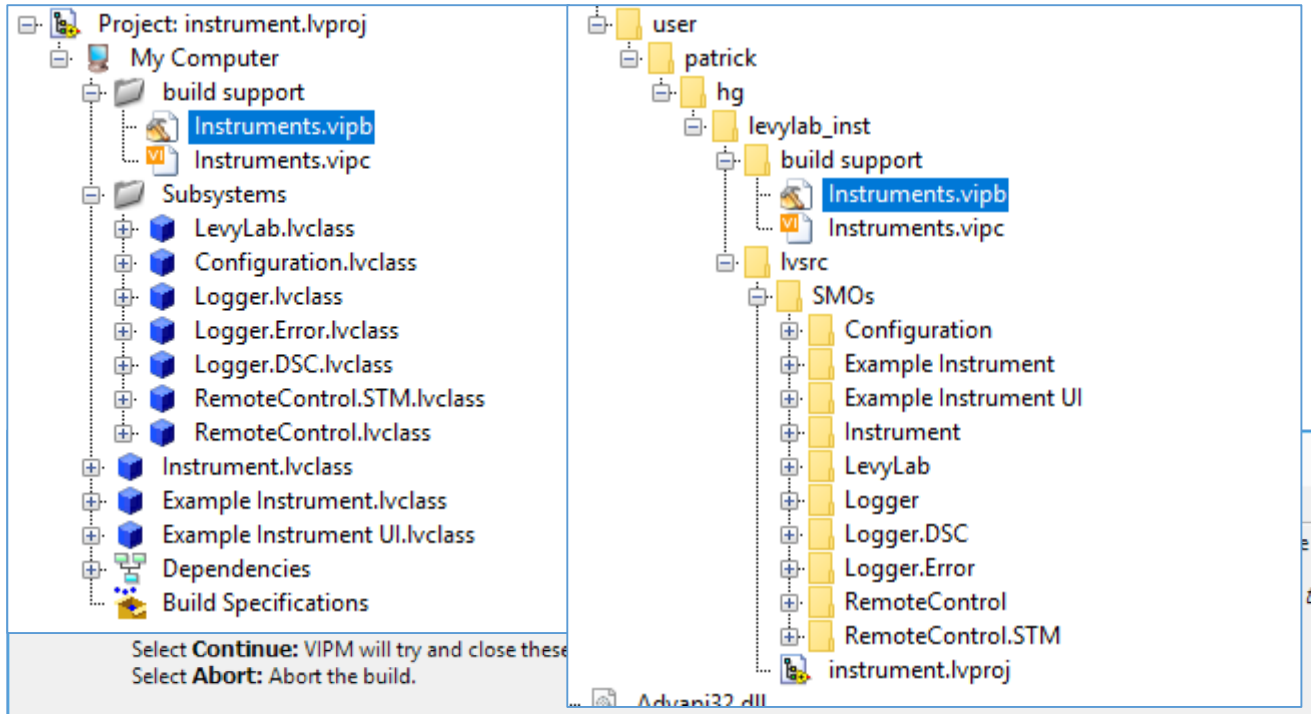
http://www.the-control-freak.com/SCPI/SCPI.htm

http://j123b567.github.io/scpi-parser/

Pseudo SCPI Message String for chaining arbitrary number of arguments 2

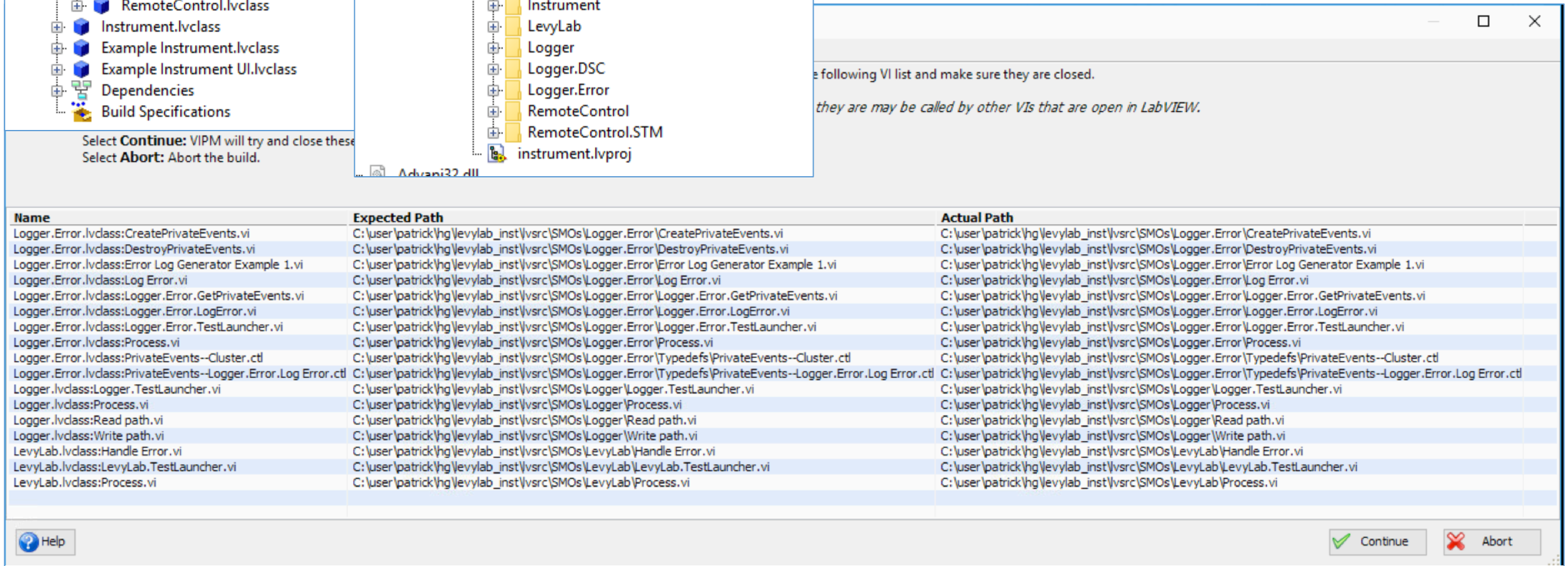


Packaging Stuff (2/2-2/3/2019)



Still have error after LV restart and system restart

https://support.jki.net/hc/en-us/articles/214135643



Troubleshooting Package Building Problems

Sometimes VIPM reports an error during the package building step. Issues that may cause package building to fail and possible solutions are listed below:

•One or more of your source files are missing dependencies. In other words, a VI is calling another VI which is missing. This can happen if your code depends on an add-on toolkit that is not installed with LabVIEW.

•You have LabVIEW files in your source with duplicate names. All your source filenames should be unique. To work around this issue, you can place VIs in libraries (*.LVLIB files) which provide name-spacing. Having VIs in separate classes also resolves this issue, since classes provide the same name-spacing as *.LVLIBs.

•You cannot have some of your source files located beneath LabVIEW. In other words, you cannot build your package, if the source is located beneath LabVIEW. VIPM supports Symbolic Paths so you don't need to locate your source beneath LabVIEW. When VIPM builds your package it replaces your absolute references to the LabVIEW installation location with symbolic paths. This allows your package to be installed under any LabVIEW version installed in any location under Windows.

•You cannot have some of your source files cross-linking to locations outside your source folder. For example, you have a VI named ***test.vi*** in your source and you are also calling ***test.vi*** inside of LabVIEW. This typically happens when you build your package and then install it. Now, when you try to open the source VIs, LabVIEW links to the version which is installed, not your source. - you should try uninstalling your package from LabVIEW before building a new one.

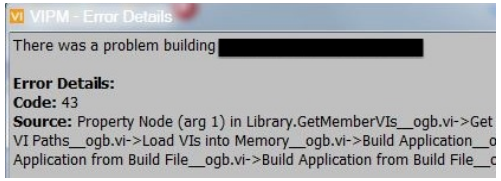
If you need to keep the package installed while you develop your source code, then you should consider adding a prefix to your VIs during build. This can be done automatically by VI Package Builder in the 'Source Files Settings' page. Note, that the installed package VIs must be the renamed version.

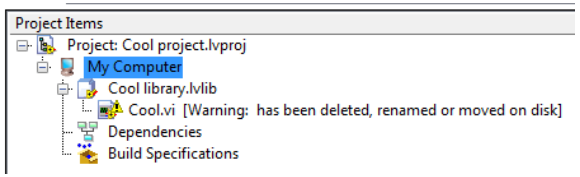
•You cannot have VIs open (in memory) while you perform the build.

•Do not include images, text files or other non labview files inside of LVLIBs or Classes. There have been reported issues with this.

•If you have several VIs in separate folders and you are trying to build a single package that includes all of those VIs, then you must select the common folder path as your source folder. For example, if you have 'c:\source\component1' and 'c:\source\component2'. Then your source folder must be: 'c:\source'

•If you are getting Error Code 43 (see image). The problem is that you have a library (lvlib) that has missing VIs inside it. Open the project and navigate to your lvlib project items. Look for instances of VIs that are missing on disk but are still referenced in the lvlib. If you remove these VIs from your library then the package building will complete correctly.





If you are trying to build packages of a class inheritance then there are special considerations:

•When building a package of a child class separate from the parent class (one package for each class) - the child class VIs (in your source) must call the **installed** parent package not the parent class in the source.

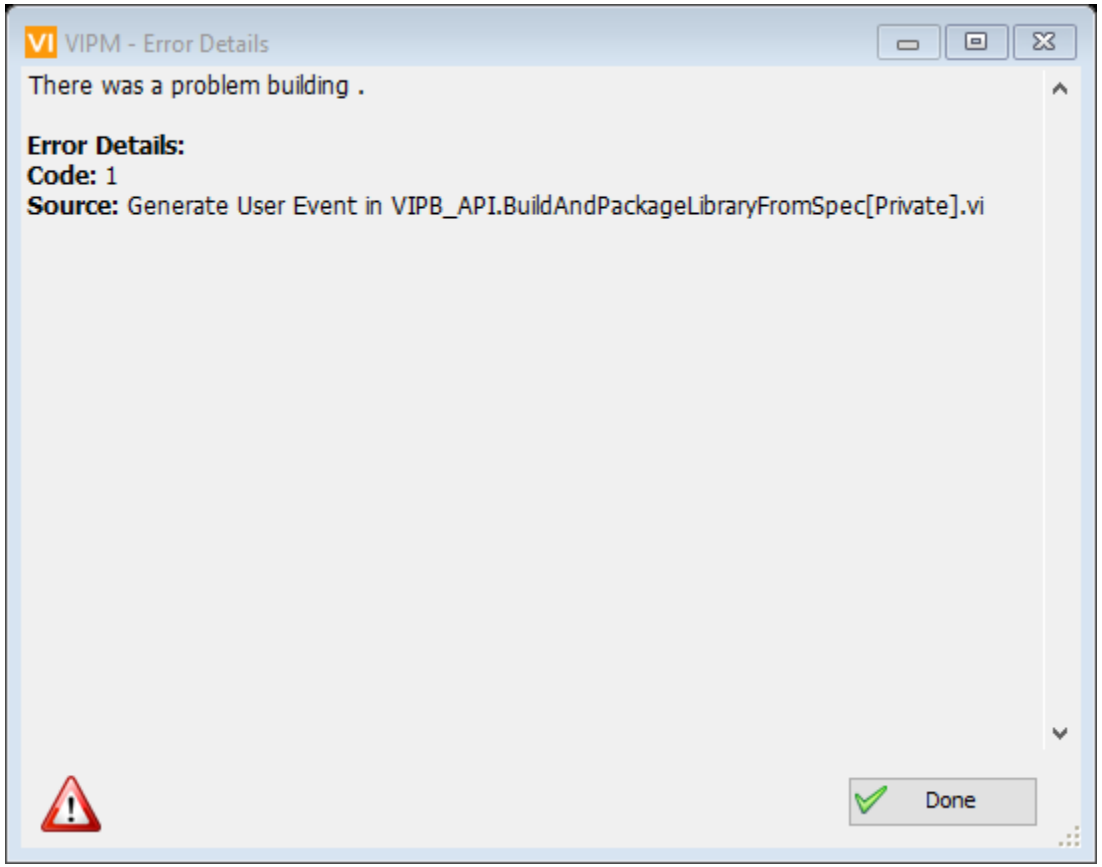
•If you have several classes in separate folders and want to build a single package of all classes, then you must select the common folder path as your source folder. For example, if you have 'c:\source\class1' and 'c:\source\class2'. Then your source folder must be: 'c:\source'

To debug package install issues do the following:

•Uninstall all other packages in the the LabVIEW version you are trying to install under. This will provide a clean slate to test your new package.

https://support.jki.net/hc/en-us/articles/214135643

Switch Build to 2016??



Rename RemoteControl.lvclass:

Rename RemoteControl.lvclass → RemoteControl.v2.lvclass. Get following error:

The file at
'C:\user\patrick\hg\levylab_inst\.lvsrc\SMOs\RemoteControl\RemoteControl.v2.lvclass
RemoteControl.v2.ctl' was expected to have the qualified name
'RemoteControl.v2.lvclass:RemoteControl.v2.ctl', but has the qualified name
'RemoteControl.lvclass:RemoteControl.v2.ctl'.

Rename RemoteControl.v2.lvclass → RemoteControl.lvclass. Get following error:

The file at
'C:\user\patrick\hg\levylab_inst\.lvsrc\SMOs\RemoteControl\RemoteControl.lvclass\Re
moteControl.ctl' was expected to have the qualified name
'RemoteControl.lvclass:RemoteControl.ctl', but has the qualified name
'RemoteControl.v2.lvclass:RemoteControl.ctl'.
🙄

Rename again RemoteControl.lvclass → RemoteControl.lvclass. Seems to work now

Instruments.vipb

Experimental Build
[1.1.0.4]

includes:
LevyLab.lvclass
Instrument.lvclass
Configuration.lvclass
Logger.lvclass
Logger.DSC.lvclass
RemoteControl.lvclass
RemoteControl.STM.lvclass

not included:
Logger.syslog.lvclass
RemoteControl.ZMQ.lvclass
Example Instrument.lvclass
Example Instrument UI.lvclass
Instrument's API (override) folder (Go to Angle, etc.)

Dependencies

Joe Albro — 9:08PM on Feb 3

•Patrick Irvin just in case this helps, all of the dependencies that I needed for the SR7270 Example included:

Remote Repository

•The Instrument Repository

•The GUID Repository

•And I had to install:

Structure Error Handler Package

•DSC (I have this one on other computers but not mine since I reinstalled LabVIEW)

•Syslog Package

•I already had but others may not have:

MGI Error Handler

•JKI JSON

I hope that this may help when it comes to making a package/example project.