Circle of Friends

Extending QMH with User Events

Fabiola De la Cueva

Delacor

LabVIEW Consultant's customers

1. Startup assistance
2. Firefighting
3. Tutoring
4. Refactoring
5. Complete Solution

Code type

1. Develop and maintain code for them
2. Customer keeps code and maintains it
3. Just give me an exe that works

My goals

- Customer calls back for more features or new projects, never to complain about bugs.

- If customer keeps code, they are able to maintain it.

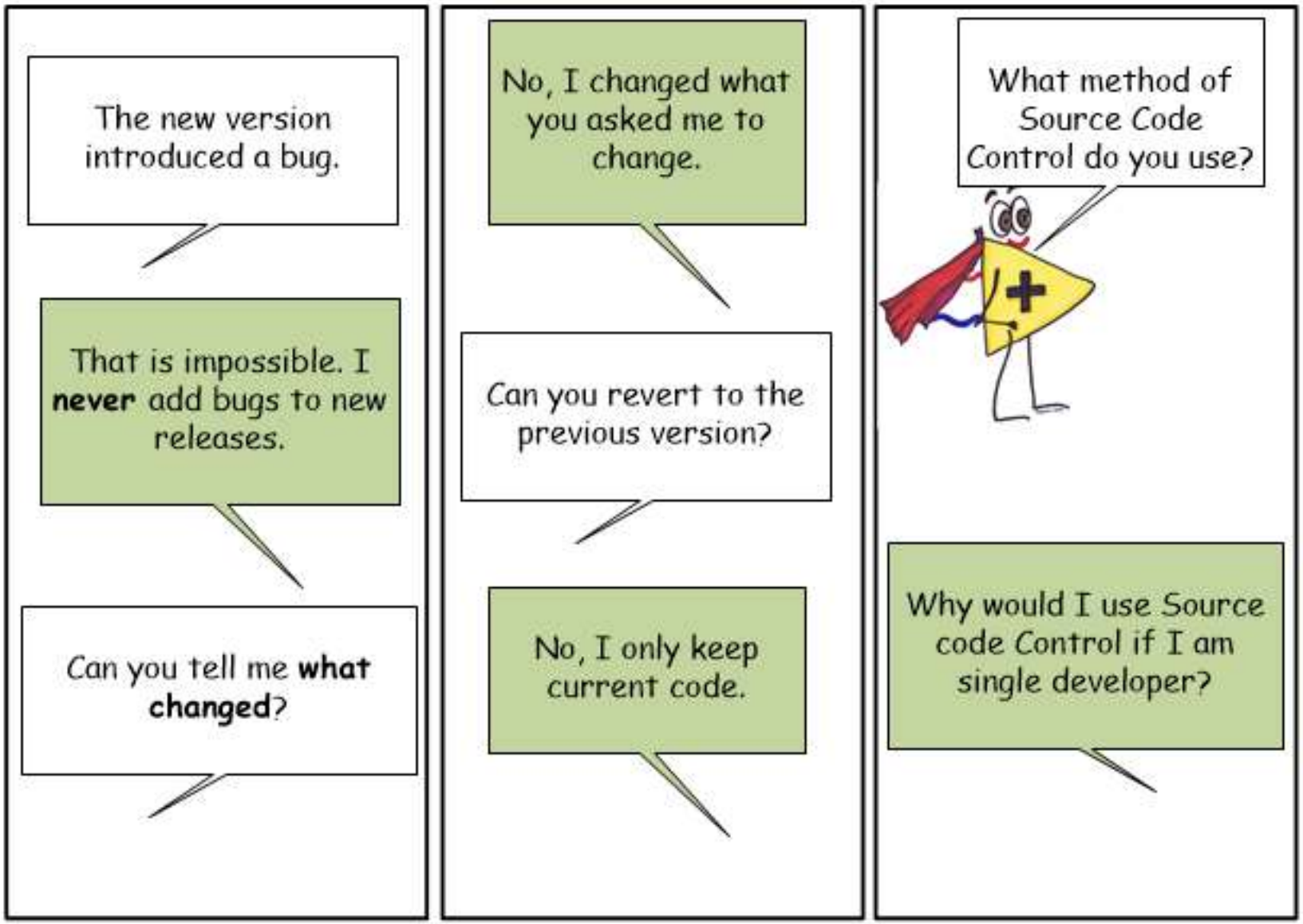
- I have seen the good, the bad and the ugly.

- I have not found a solution that fits all problems.

Aiming for best

Architecture Must Haves

- Have a system model on paper
- Determine solution components
- Determine “messages” needed between components
- Do not focus on implementation at this stage


The new version introduced a bug.
That is impossible. I **never** add bugs to new releases.
Can you tell me **what changed**?
No, I changed what you asked me to change.
Can you revert to the previous version?
No, I only keep current code.
What method of Source Code Control do you use?
Why would I use Source code Control if I am single developer?

Development Must Haves

- Use Source Code Control
- Even during crunch time, develop thinking of how to make debugging easier
  - Specially during crunch time !
- Beware of race conditions
  - They are not only introduced by global variables

Weren't you going to talk about QMH?

QMH

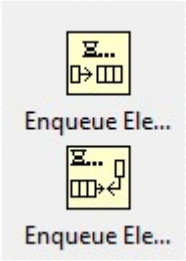
- Queue Message Handler

- Is what we were converting the old Producer Consumer Design Pattern into

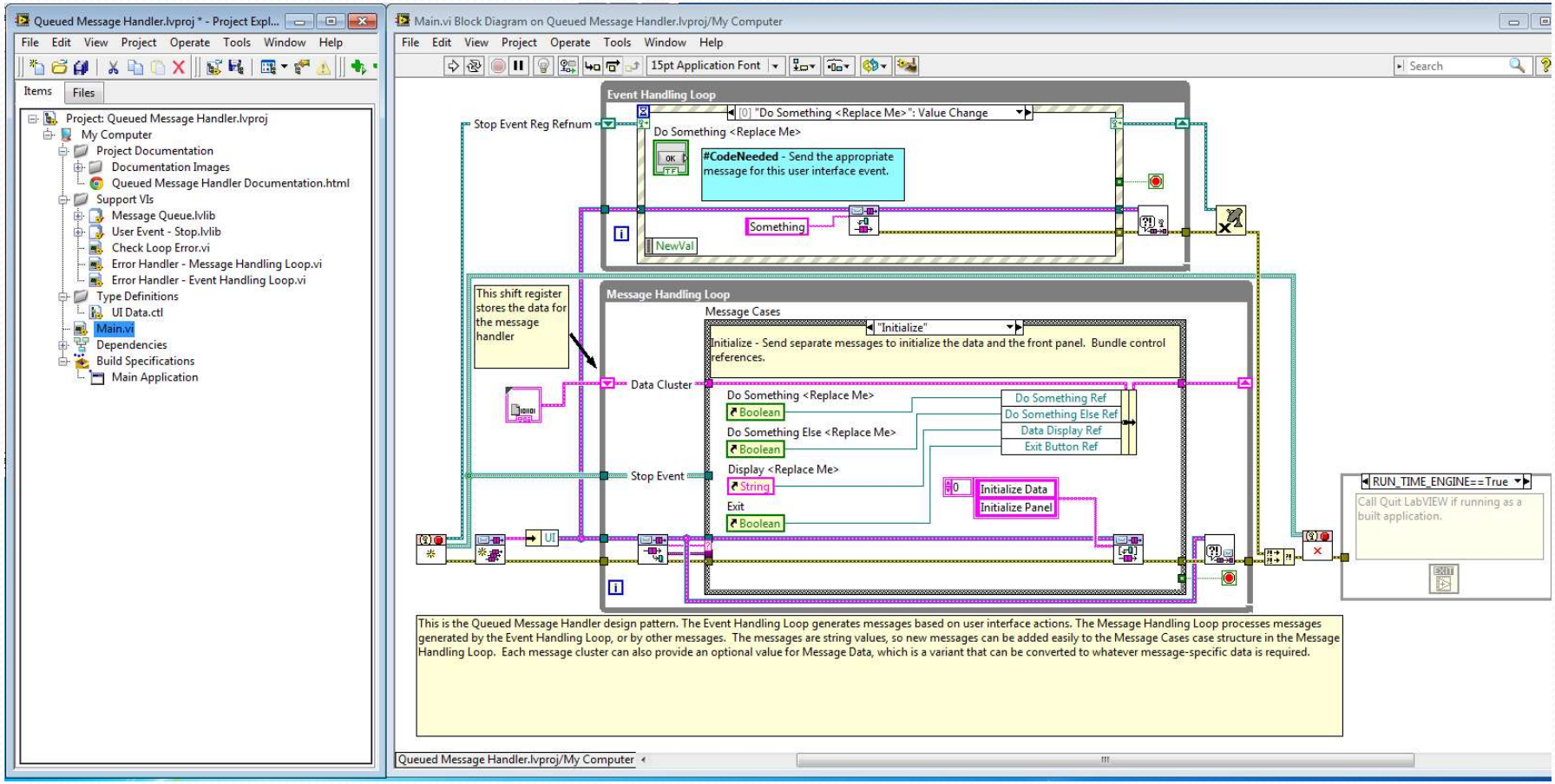
Why “QMH”?

- Well known throughout the LabVIEW community
- Uses components introduced early on LabVIEW training
- Starting with LabVIEW 2012 ships with LabVIEW
- Customer can call NI Support with questions
- Well documented

Why Not “QMH”?

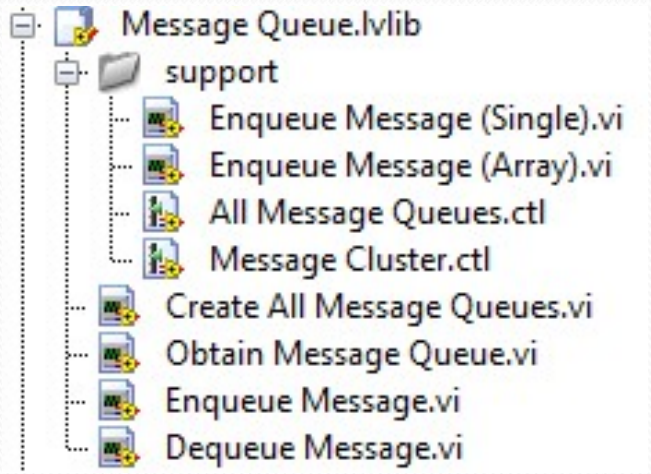


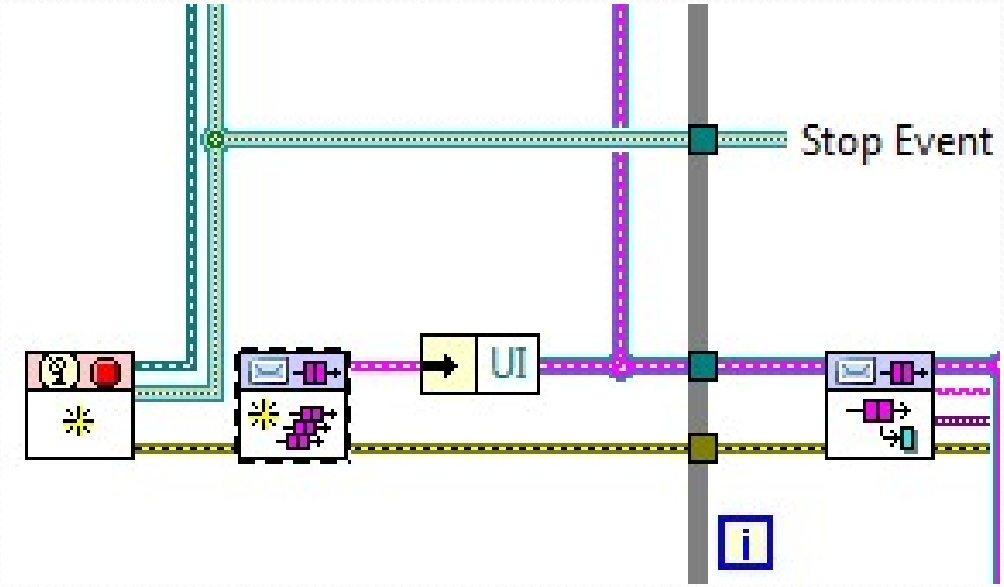
- Managing priority messages is hard
- Larger number of modules result in large block diagram
- Developer can introduce hard to troubleshoot bugs by not using API


Queued Message Handler.lvproj * - Project Expl...
File Edit View Project Operate Tools Window Help
Items
Files
Project: Queued Message Handler.lvproj
My Computer
Project Documentation
Documentation Images
Queued Message Handler Documentation.html
Support VIs
Message Queue.lvlib
User Event - Stop.lvlib
Check Loop Error.vi
Error Handler - Message Handling Loop.vi
Error Handler - Event Handling Loop.vi
Type Definitions
UI Data.ctl
Main.vi
Dependencies
Build Specifications
Main Application
Main.vi Block Diagram on Queued Message Handler.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Event Handling Loop
Stop Event Reg Refnum
[0] "Do Something <Replace Me>": Value Change
Do Something <Replace Me>
#CodeNeeded - Send the appropriate message for this user interface event.
Something
NewVal
This shift register stores the data for the message handler
Message Handling Loop
Message Cases
"Initialize"
Initialize - Send separate messages to initialize the data and the front panel. Bundle control references.
Data Cluster
Do Something <Replace Me>
Boolean
Do Something Else <Replace Me>
Boolean
Display <Replace Me>
String
Exit
Boolean
Stop Event
Do Something Ref
Do Something Else Ref
Data Display Ref
Exit Button Ref
0
Initialize Data
Initialize Panel
UI
RUN_TIME_ENGINE==True
Call Quit LabVIEW if running as a built application.
This is the Queued Message Handler design pattern. The Event Handling Loop generates messages based on user interface actions. The Message Handling Loop processes messages generated by the Event Handling Loop, or by other messages. The messages are string values, so new messages can be added easily to the Message Cases case structure in the Message Handling Loop. Each message cluster can also provide an optional value for Message Data, which is a variant that can be converted to whatever message-specific data is required.
Queued Message Handler.lvproj/My Computer

Promotes reusable code

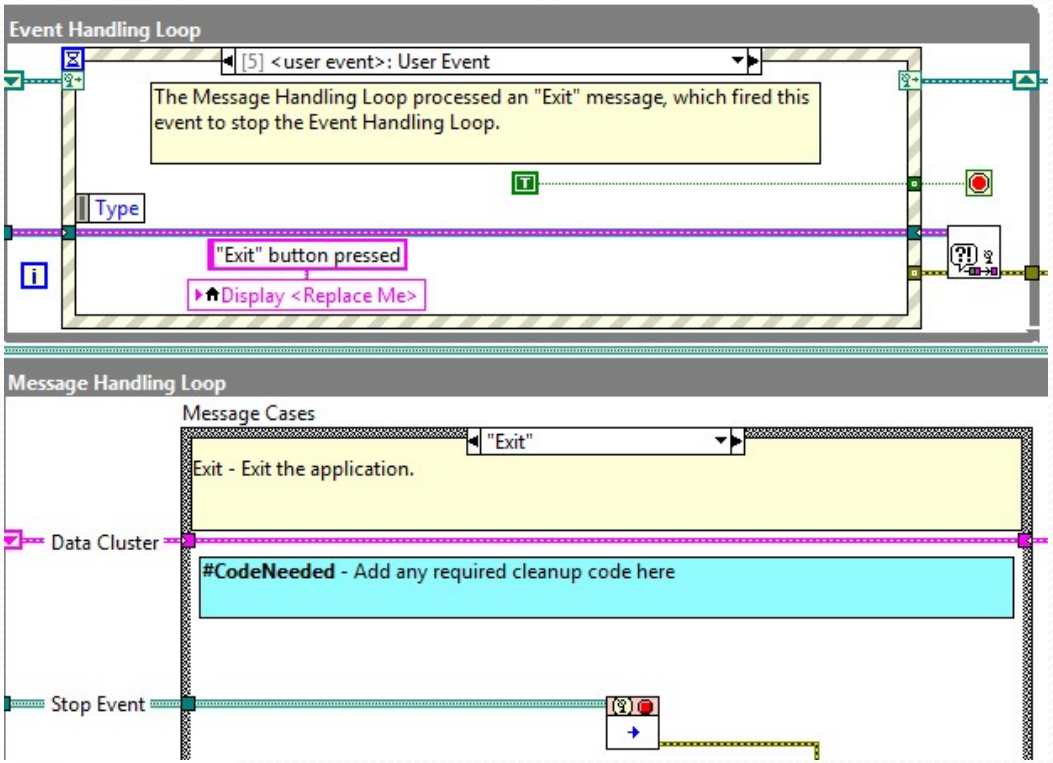
- The sample project creates your own copy of the Message Queue

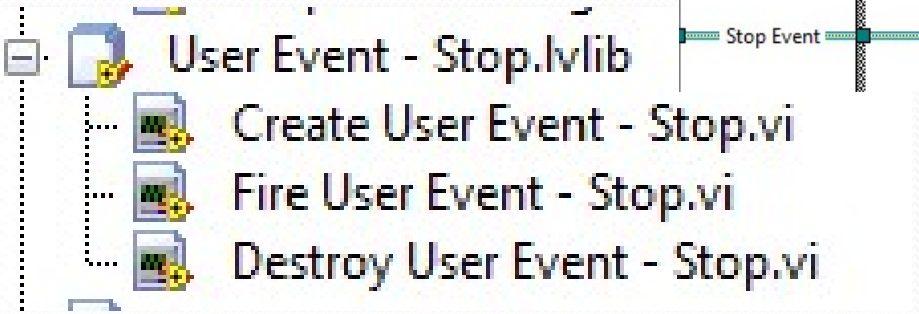




Local User Events use

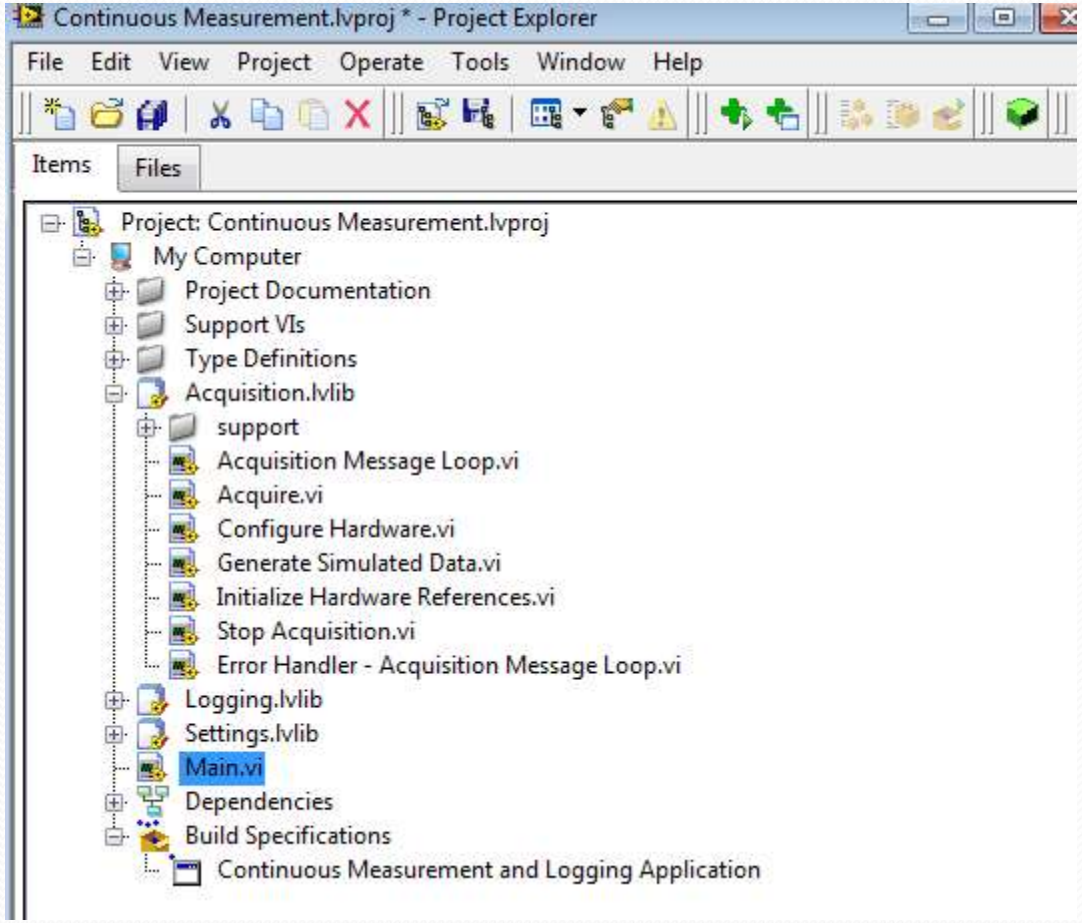
- Provides communication from lower loop to higher loop via User Events



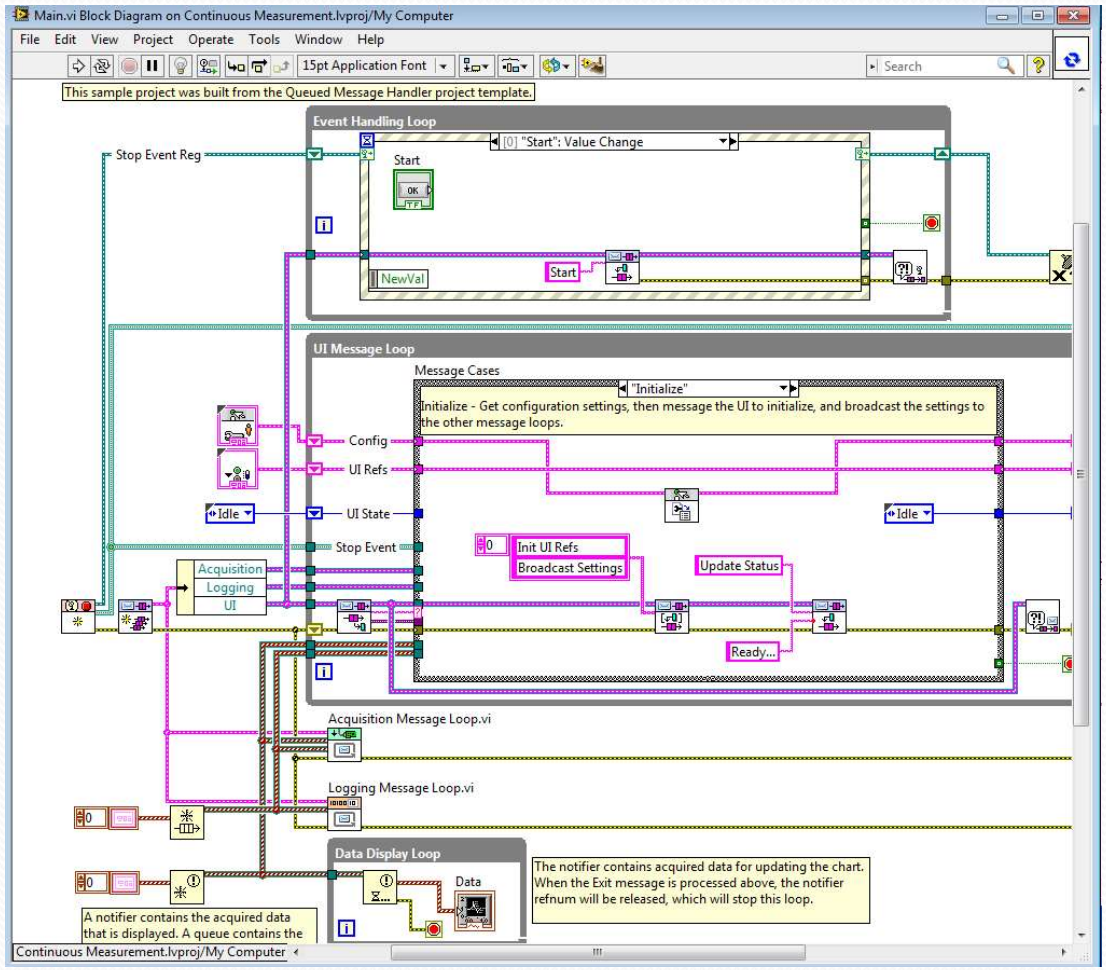


Continuous Measurement Sample Project

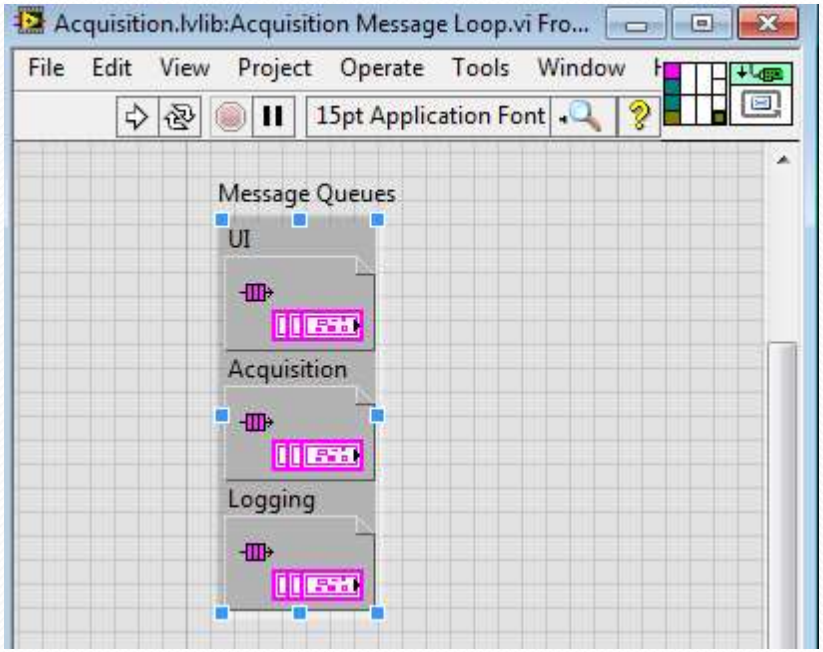
- Promotes modularity via Project Libraries



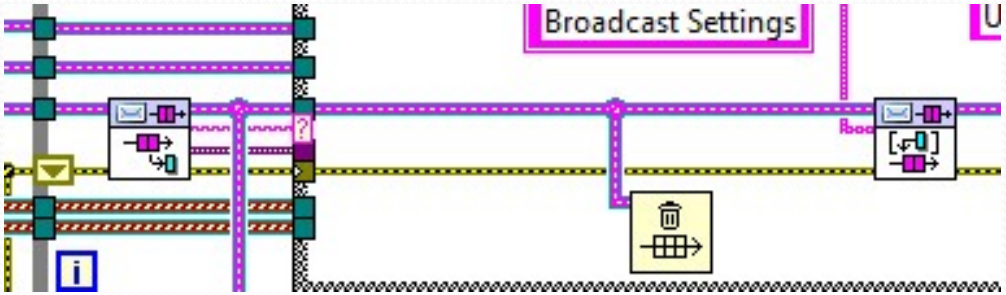
Inter Module Communication via Queues

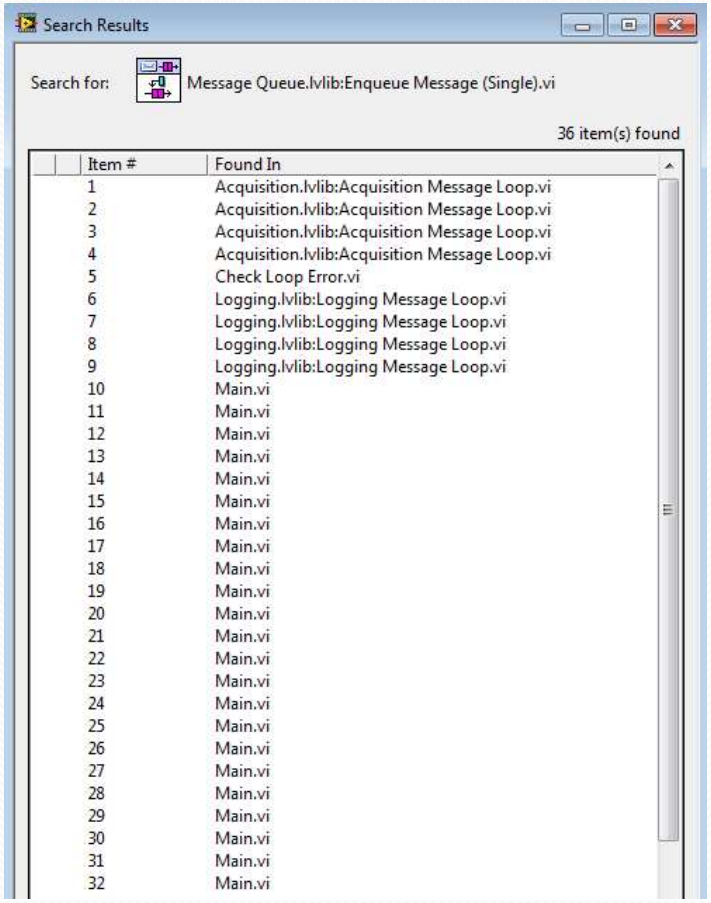


- Modules are not reusable:



- Can be difficult to debug
  - Anyone can enqueue messages
    - Race conditions
  - Developers could bypass using Message Queue.lvlib functions





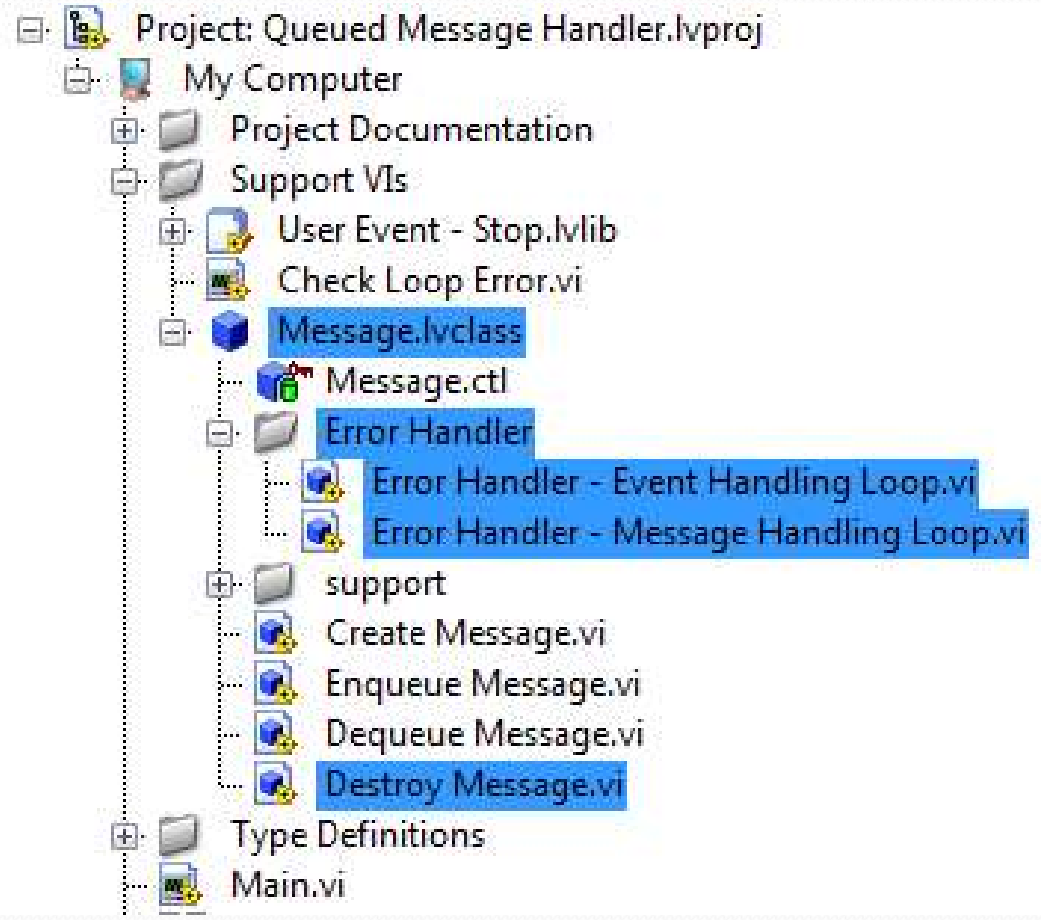
Some modifications for intermediate users

Communication Within a Module

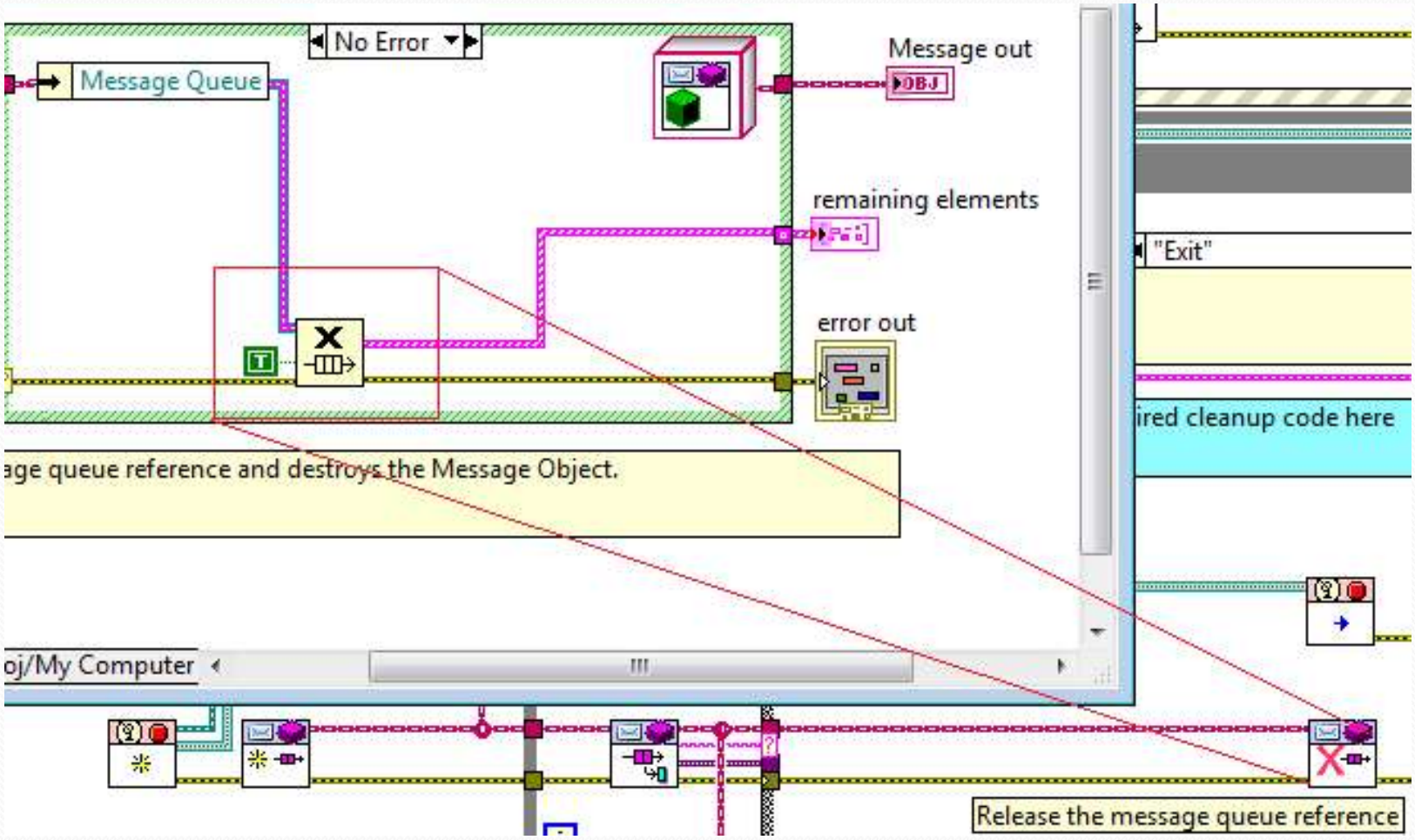
- Make the Queue for each module private to that module
  - Convert the Message Queue.lvlib into a Message Queue class.
  - The developers will have to use the class methods, because the reference won't be naked
    - This improves debugging, because you can find all the places where the queue is being accessed.
  - Any queue function will have to be added as a method to the class

Convert Message Queue.lvlib into a class

- Cover your naked Queue reference



Developers Use Methods to Access Q



CLASummit2013-SVN revision 21

Inter Module Communication

- We have seen that User Events are more powerful than just to send a message from the Message Handling loop to the Event Handling loop
- See Jim Kring's "Private and Public Events Framework" presentation
- See Jack Dunaway's "User Events Tips, Tricks, and Sundry"

Extending QMH with User Events

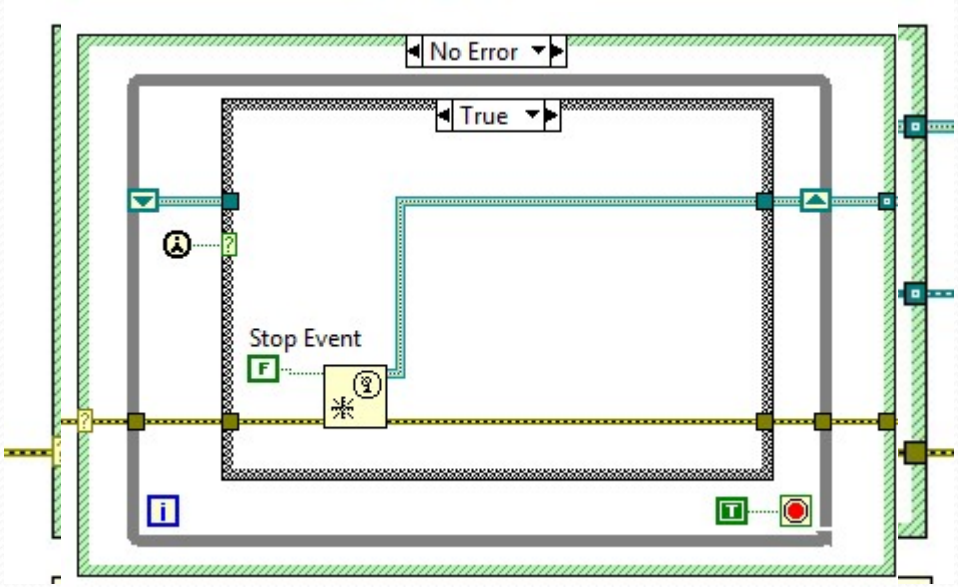
1. Move the User Event registration out of the User Event creation

2. Keep user event reference in an uninitialized shift register

3. Have Modules register for the events they want to listen to

Move User Event registration out

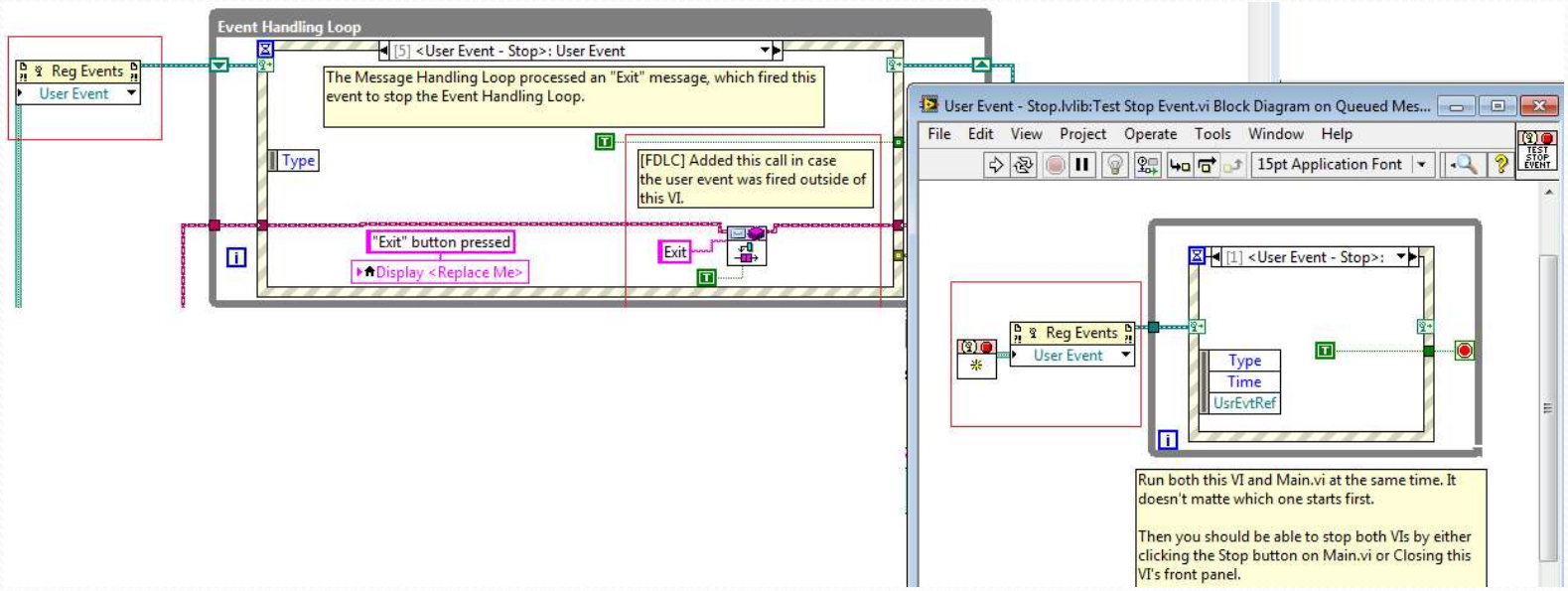
- This will allow multiple Loops to be able to register for the same event.



CLASummit2013-SVN revision 24

VIs Register For Event

- "Test Stop Event.vi" registers for Event



CLASummit2013-SVN revision 24

Converting Shipping Sample Project

- Continuous Measurement Project uses Queues to communicate between modules
- It takes about 10 steps to convert this communication to use User Events

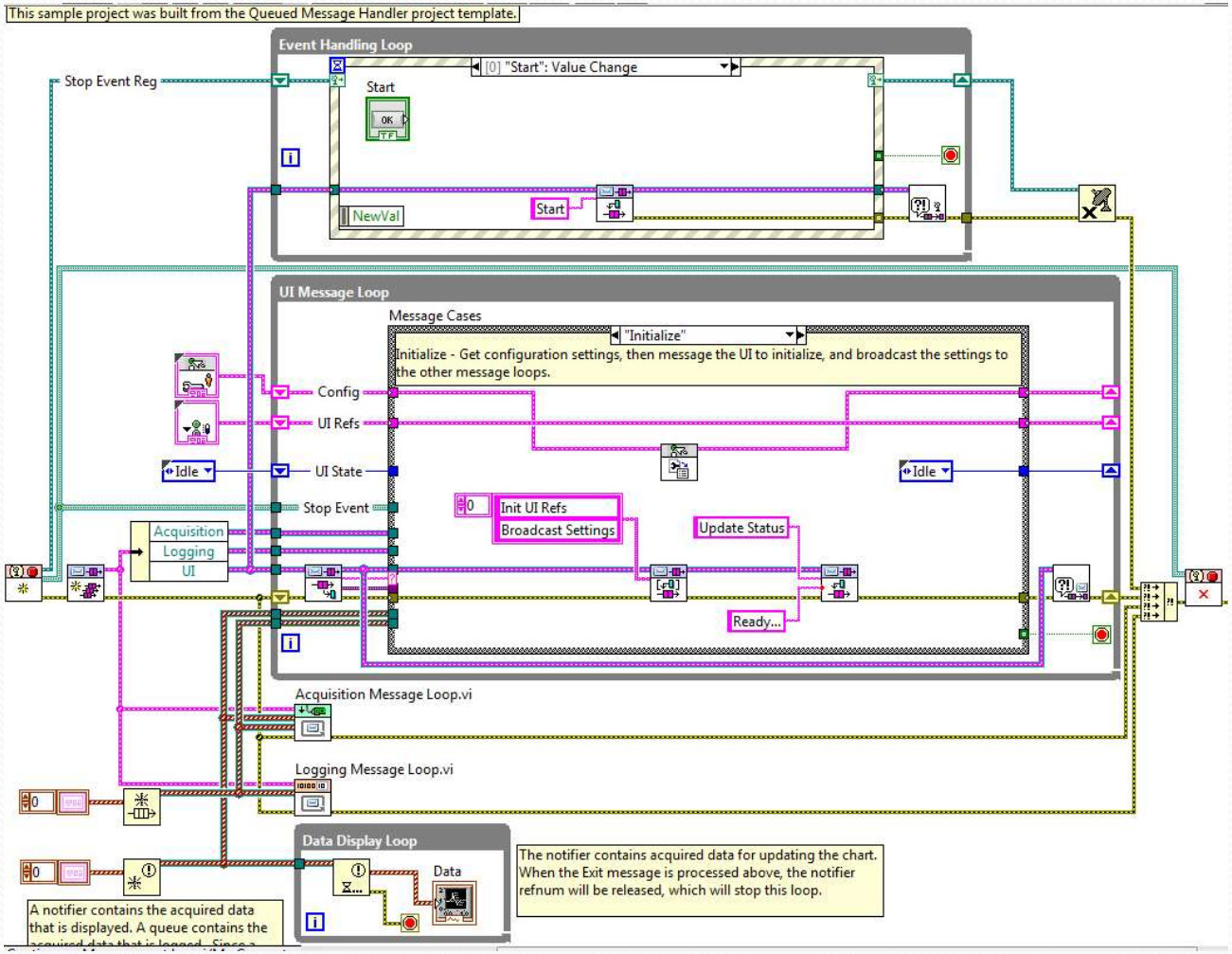
Continuous Measurement Sample Project

1. Replaced Queue Message Library with class
   - SVN rev 31
2. Replaced User Event Stop with our version
   - SVN rev 32
3. Removed Enqueue Exit message to other modules
   - Use Stop Event instead. SVN rev 33
4. Replaced Messages to Acquisition Loop with Events
   - SVN rev 34
5. Replaced Messages to Logging Loop with Events
   - SVN rev36
6. Replaced Error messages to Main with Events
   - SNV rev 37

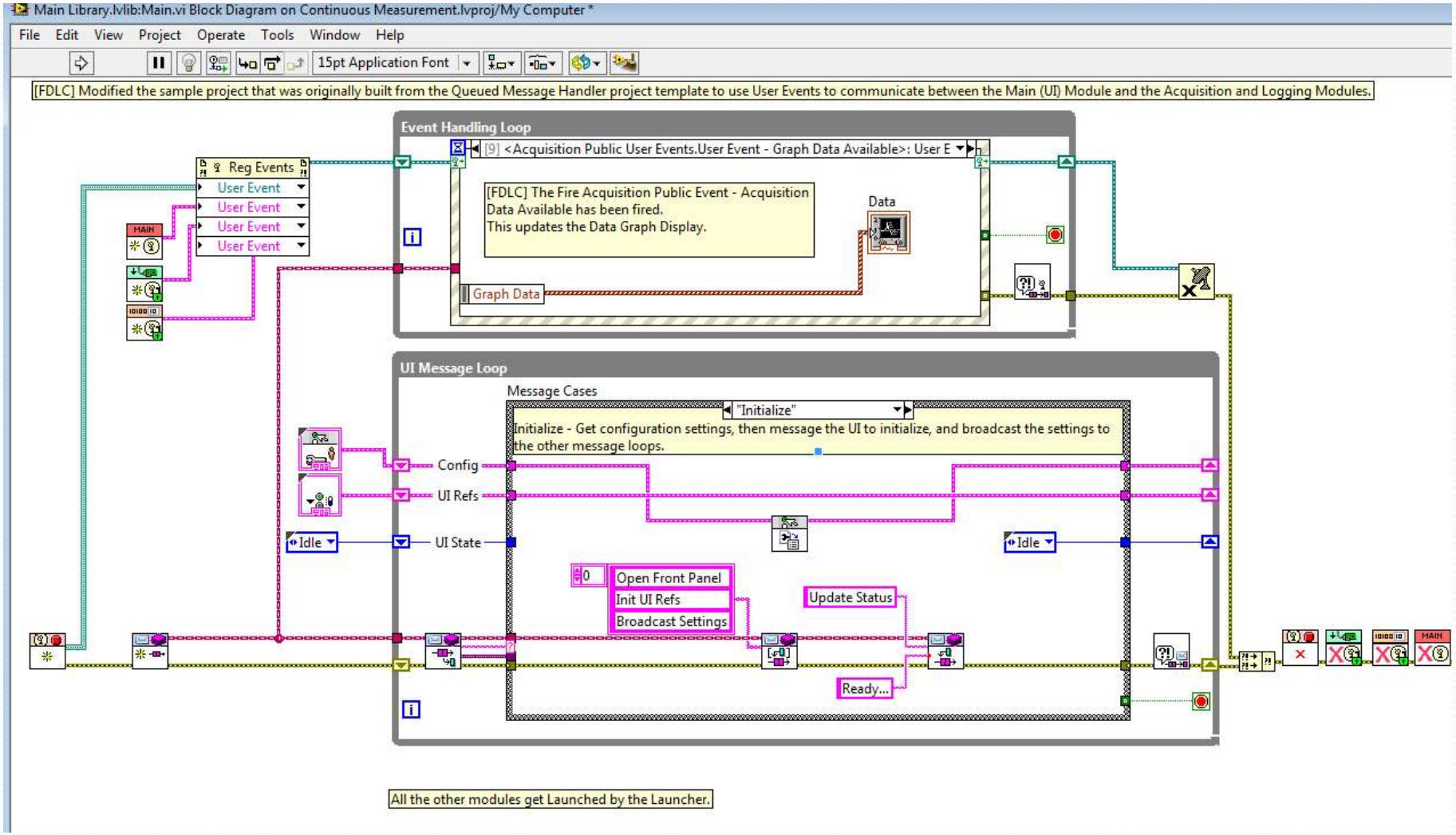
Continuous Measurement Sample Project

7. Replaced Data Queue and Data notifier with a Public Data Available User Event and had Main and Logging register to listen to it.
   - SVN rev 39
8. Created Launchers for each module, removed them from Main.vi and created Application Launcher.
   - SVN rev 41
9. Final touches and editing Build specification and first exe
   - SVN rev 42

Before


This sample project was built from the Queued Message Handler project template.
Event Handling Loop
Stop Event Reg
[0] "Start": Value Change
Start
OK
TF
NewVal
Start
UI Message Loop
Message Cases
"Initialize"
Initialize - Get configuration settings, then message the UI to initialize, and broadcast the settings to the other message loops.
Config
UI Refs
Idle
UI State
Stop Event
Acquisition
Logging
UI
0
Init UI Refs
Broadcast Settings
Update Status
Idle
Ready...
Acquisition Message Loop.vi
Logging Message Loop.vi
0
0
Data Display Loop
Data
The notifier contains acquired data for updating the chart. When the Exit message is processed above, the notifier refnum will be released, which will stop this loop.
A notifier contains the acquired data that is displayed. A queue contains the

After

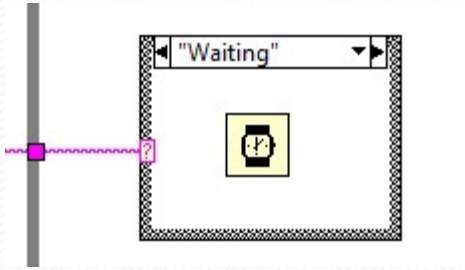

Main Library.lvlib:Main.vi Block Diagram on Continuous Measurement.lvproj/My Computer *
File Edit View Project Operate Tools Window Help
15pt Application Font
[FDLC] Modified the sample project that was originally built from the Queued Message Handler project template to use User Events to communicate between the Main (UI) Module and the Acquisition and Logging Modules.
Event Handling Loop
[9] <Acquisition Public User Events.User Event - Graph Data Available>: User E
Reg Events
User Event
User Event
User Event
User Event
[FDLC] The Fire Acquisition Public Event - Acquisition Data Available has been fired.
This updates the Data Graph Display.
Data
Graph Data
UI Message Loop
Message Cases
"Initialize"
Initialize - Get configuration settings, then message the UI to initialize, and broadcast the settings to the other message loops.
Config
UI Refs
UI State
Idle
Idle
0
Open Front Panel
Init UI Refs
Broadcast Settings
Update Status
Ready...
All the other modules get Launched by the Launcher.

Advantages of Using User Events

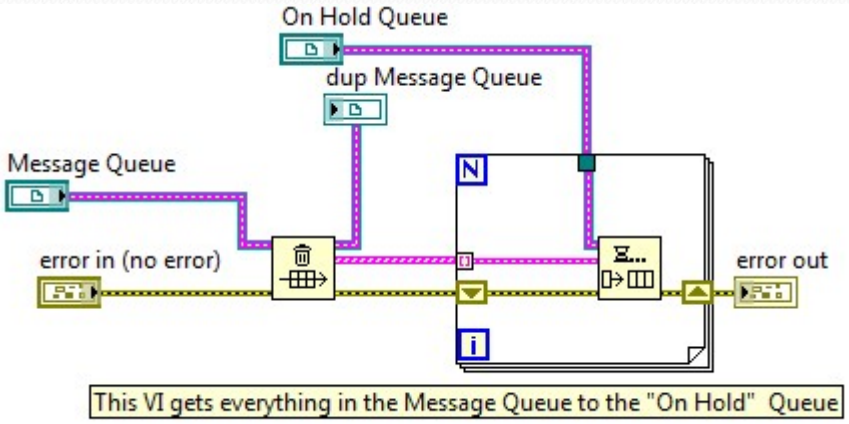
- Modules could be executed independently
  - Useful for testing
- Modules are reusable
  - New application needs to register for messages it cares
  - Use API to communicate with module
- Improves odds that the application will shutdown when expected
  - Unless the developer held the loop hostage

Waits and On Hold Queues

- Do not hold loops hostage



- If system is not ready, enqueue to try again later
- Use an “On Hold Queue” if you want to probe messages put on Hold



Module State Information

- Avoid race conditions by maintaining state information
  - Initialized
  - Ready
- When communicating between modules, use handshake
  - Request Action/ Send Reply
  - Fire API Event / Change state when getting broadcasted Event
  - Include a Notifier with your Request Action

Other thoughts

- Cover your naked references
  - Consider wrapping your User Events references in a class as well
- Rename Message.lvclass to Messenger.lvclass
  - Consider creating a Message class instead of statically defining the name and the payload as we are.

Want to Explore More

- Actor Framework implements Modularization
  - ni.com/actorframework
  - Inter modules communication is done via Queues
  - Mitigates risk of deadlocks/race conditions
- JKI Beyond State Machines (see Jim Kring's presentation)
  - Uses public and private events
  - Handles initialization of modules via rendez-vous

Thanks