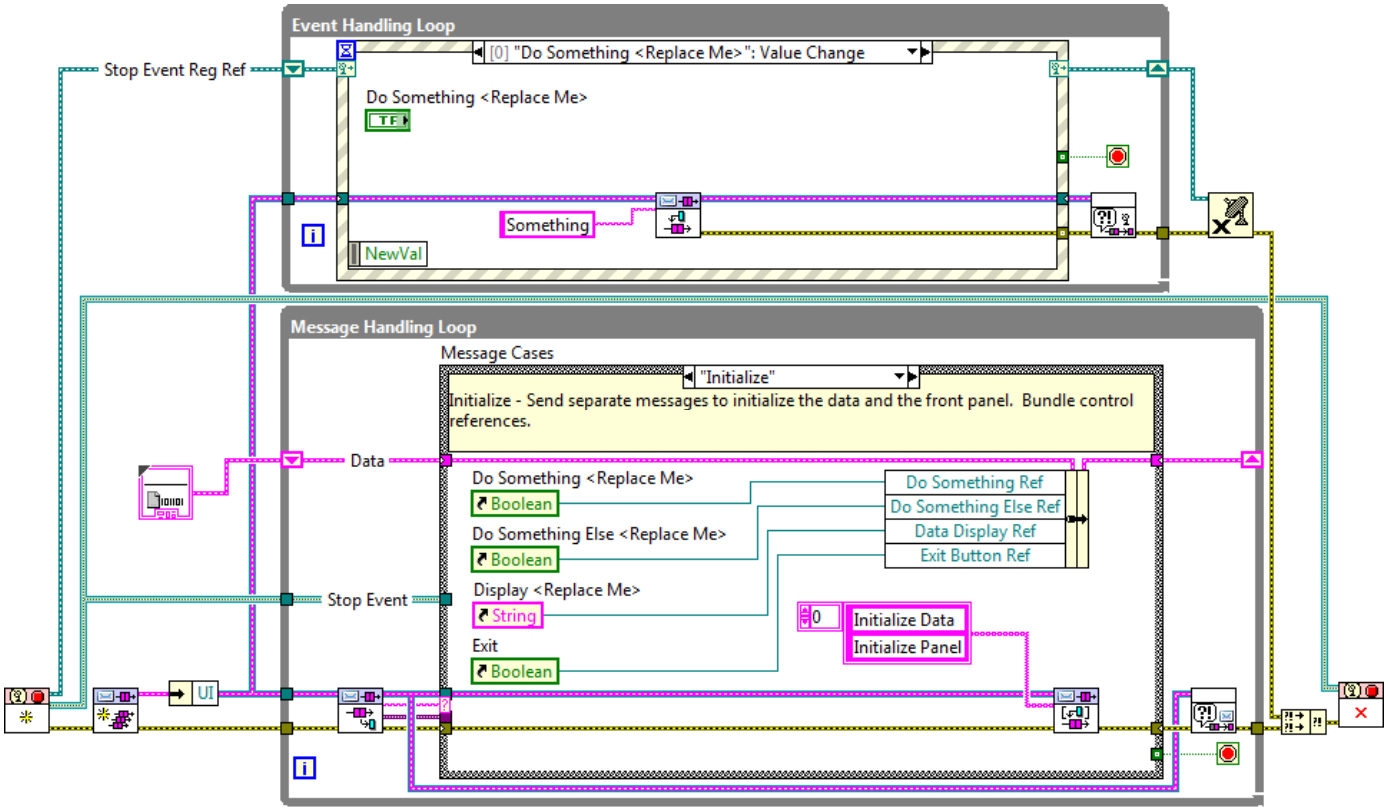
Queued Message Handler

Caveats and Design Decisions



Darren Nattinger

Senior Software Engineer, CLA, LabVIEW R&D

NATIONAL INSTRUMENTS

Outline

- Where QMH fits in with other patterns
- QMH Design
  - Main VI organization
  - Project organization
  - Messaging API
  - User Event - Stop
  - Error Handling
- A complete desktop application based on QMH
- Potential areas of expansion and customization

Desktop Project Templates

Simple State Machine *Templates*

Facilitates defining the execution sequence for sections of code.

Queued Message Handler *Templates*

Facilitates multiple sections of code running in parallel and sending data between them.

Actor Framework *Templates*

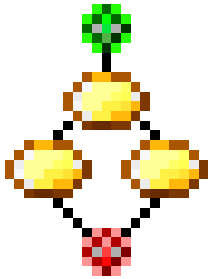
Creates an application that consists of multiple, independent tasks that communicate with each other. This template makes extensive use of LabVIEW classes.

How do these project templates compare to one another?

QMH vs. Simple State Machine

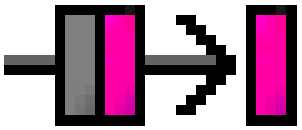
Simple State Machine

- Single Loop
- Sequential Operations
- Communication between iterations via state enum
- Good for applications that do not require parallel execution of tasks



Queued Message Handler

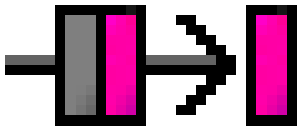
- Multiple Loops
- Parallel Operations
- Communication between iterations via queue
- Good for applications that execute multiple tasks in parallel



QMH vs. Actor Framework

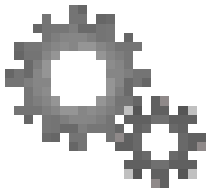
Queued Message Handler

- Messages are strings, message data is a variant
- Not based on LabVIEW classes
- Moderately extensible
- Easily debug parallel loop execution
- Static number of parallel tasks

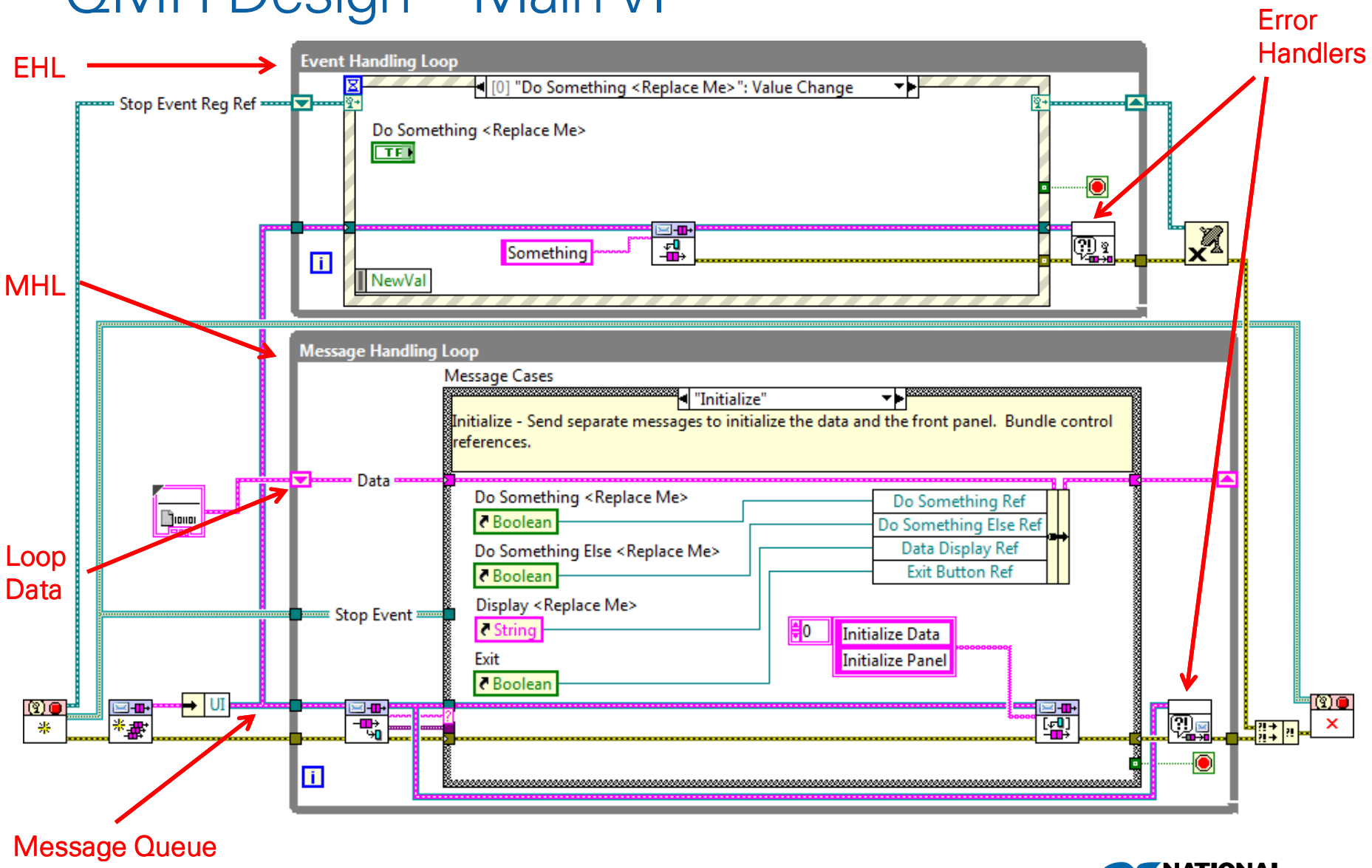


Actor Framework

- Message and message data combined in a class
- Entirely based on LabVIEW classes
- Very extensible and scalable
- More sophisticated debugging steps required
- Dynamic number of parallel tasks

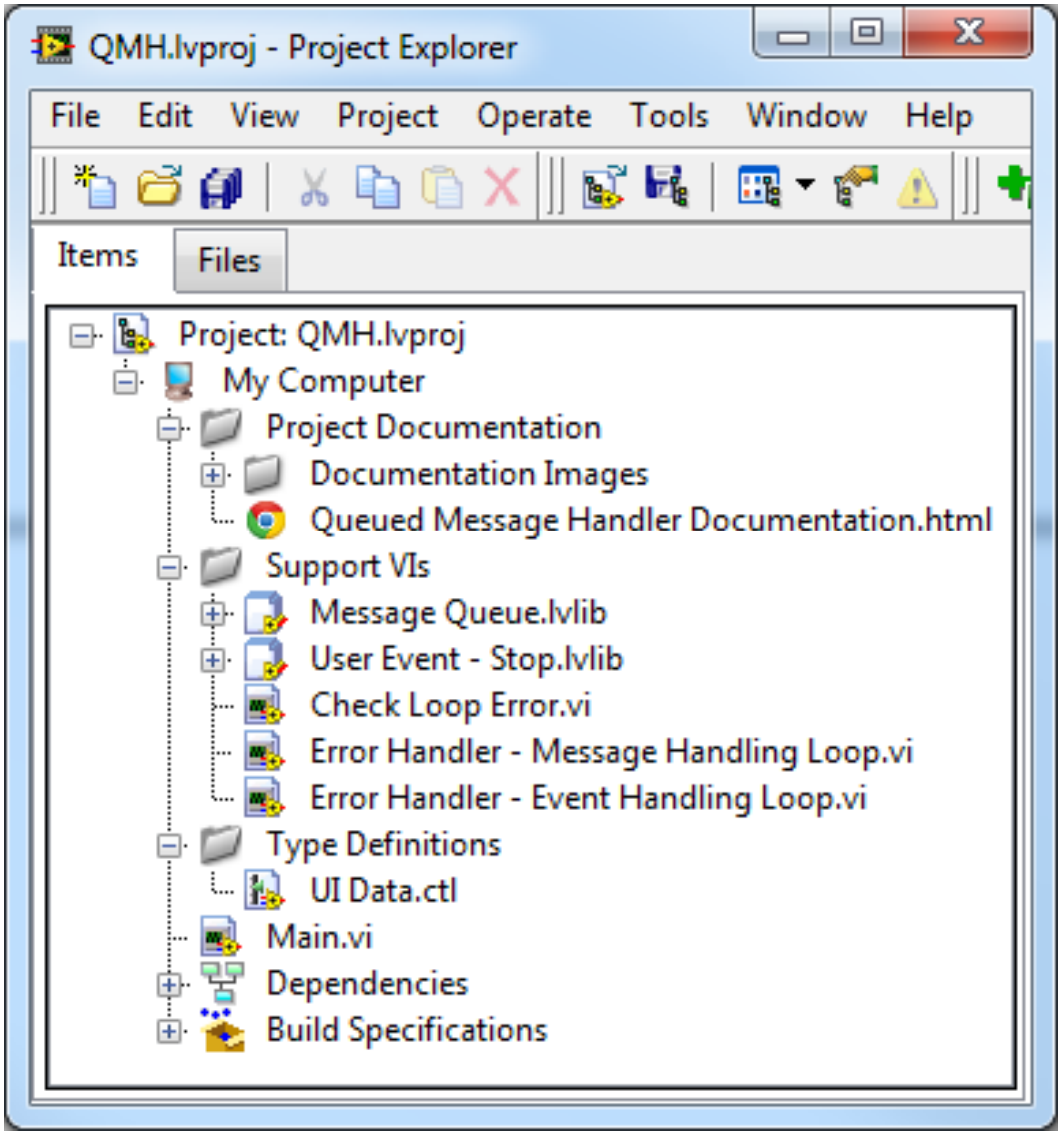


QMH Design – Main VI

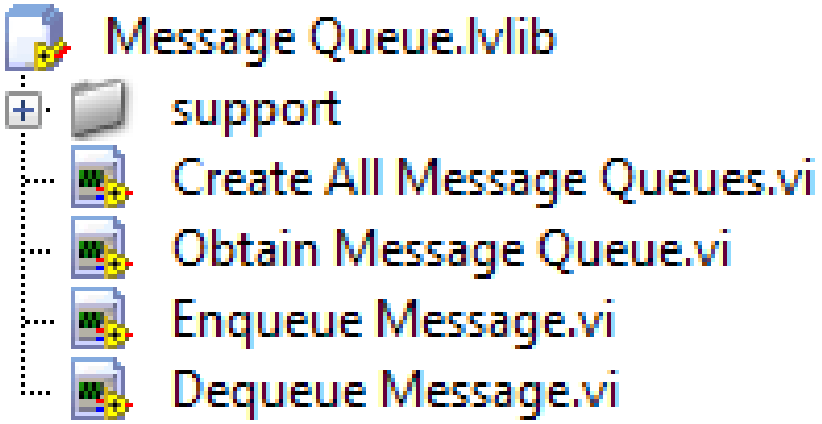


NATIONAL INSTRUMENTS

QMH Design – Project Organization



QMH Design – Message Queue.lvlib

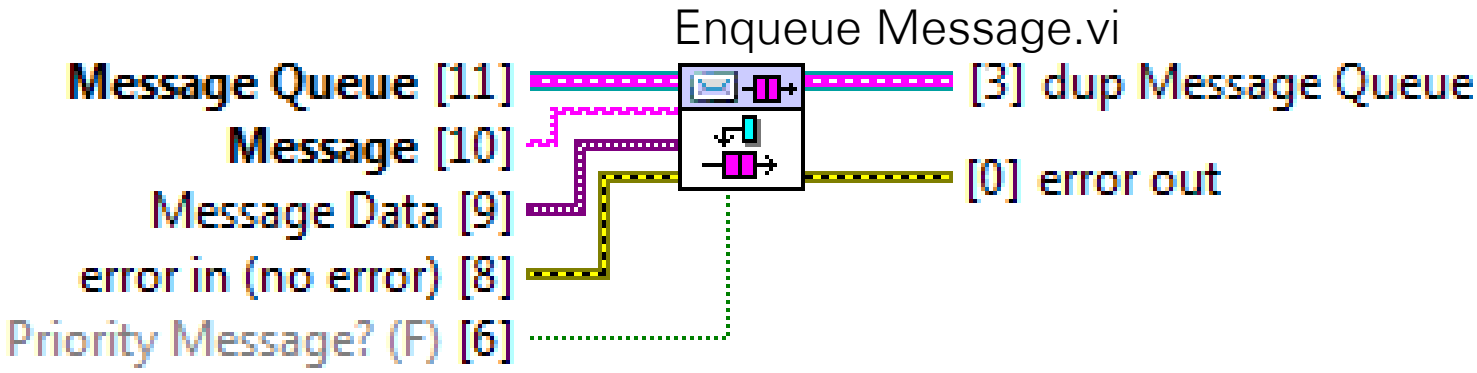


Code for passing messages between loops in the QMH application

- Create All Message Queues.vi – Extend functionality to create message queues for other Message Handling Loops
- Enqueue Message.vi – Call from any message in any loop* to send a message to a particular MHL
- Dequeue Message.vi – For a given message queue, should only be called in the MHL associated with that queue

* - any loop that you, as the app designer, have decided should have the ability to send messages on this particular message queue

QMH Design – Message Format



- A string specifies the Message type
- A variant optionally specifies the Message Data
- A message can be optionally placed at the front of the message queue as a Priority Message

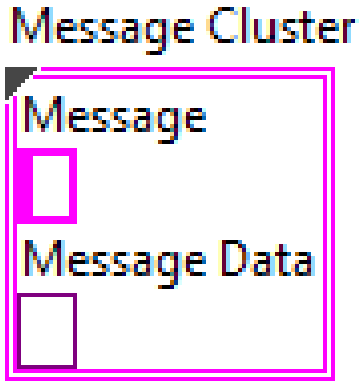
Note: This VI is polymorphic in LabVIEW 2013, accepting a single message or an array of messages.

QMH Design – String Type Rationale

Why does the QMH use string-based messages instead of enum-based?

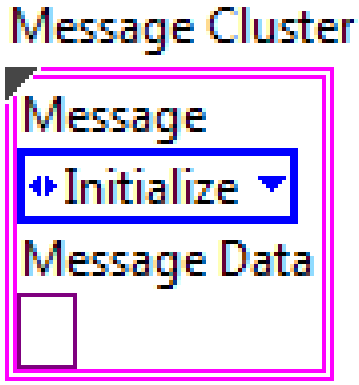
String-based Message

- If a message name is mis-typed, it must be detected at runtime
- Message type uses more memory (variable string length)
- Single message handling API for the entire application

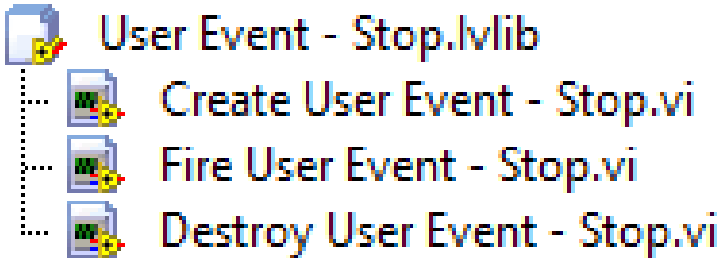


Enum-based Message

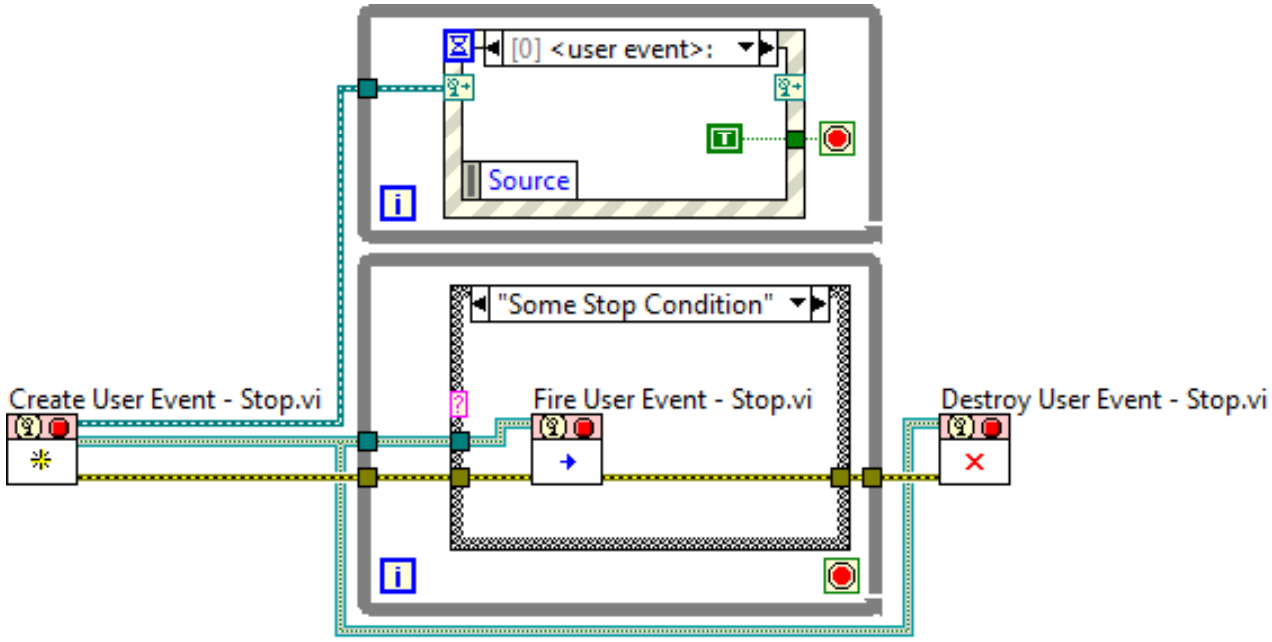
- No invalid messages…the editor forces valid selection
- Fixed memory size for Message values
- Unique message handling API required per handler



QMH Design – User Event – Stop.lvlib



Code for stopping Event Handling Loop from Message Handling Loop



QMH Design – User Event Rationale

- User Events are the **cleanest approach** to communicating with an event handling loop from other parts of the code
- If necessary, the approach taken in the QMH can be easily extended to include more event data and/or more events
- Code is part of the template and does not require editing

Rejected Alternatives

- Adding a 'timeout' case that polls a local/global variable turns the event structure into a polling loop
- Adding a 'Value(Signaling)' property call to fire the Stop Button (or a hidden button) event results in a clunky UI

QMH Design – Error Handling

LabVIEW 2012

- The Dequeue Message.vi generates an 'Exit' message if it encounters an unknown error
- Other loops perform their own error checking and send an 'Exit' message accordingly

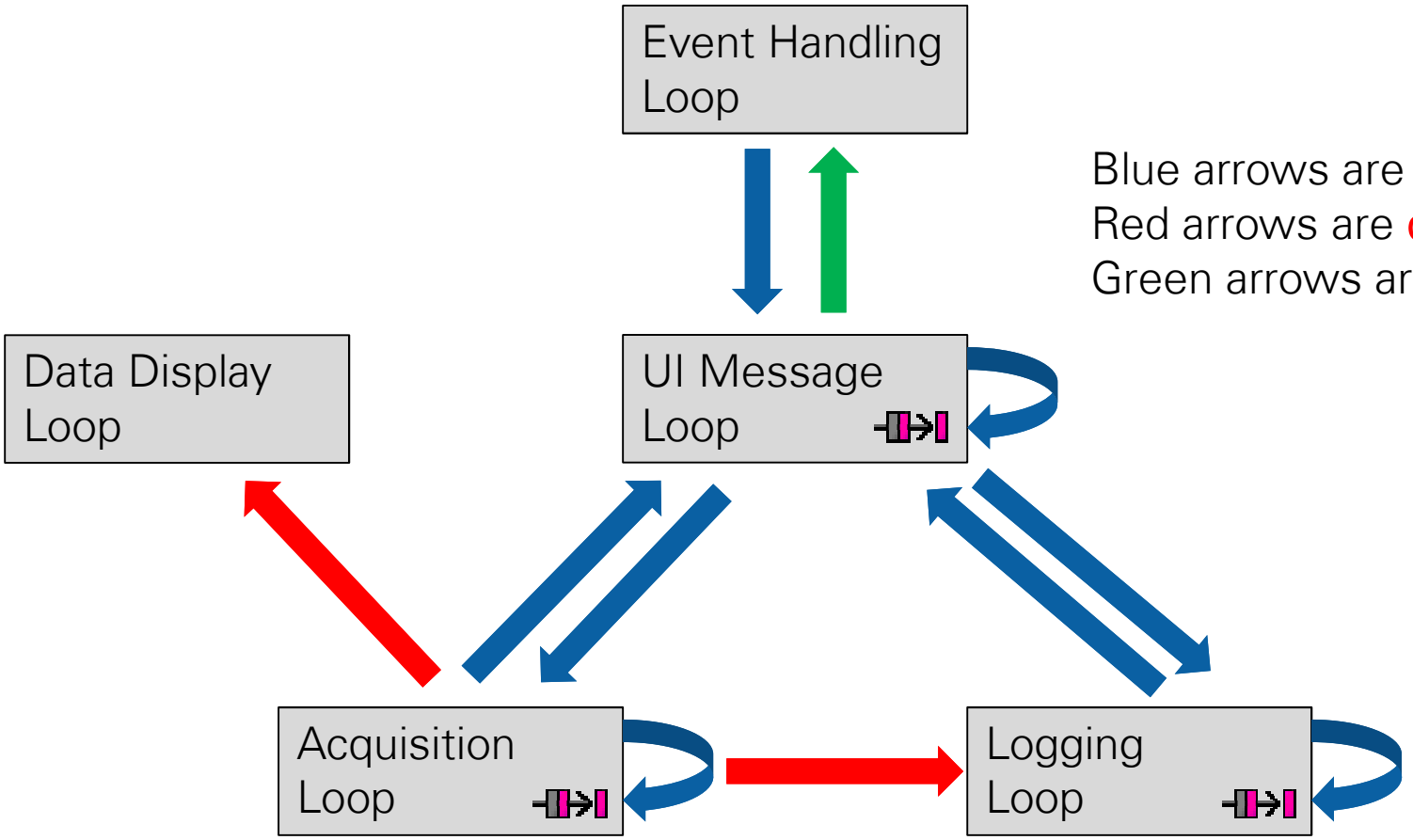
LabVIEW 2013

- Error logic removed from Dequeue Message.vi
- Each loop contains its own error handling subVI that can make loop-specific decisions on how to handle errors
- The MHL has an "Error" message to perform custom actions when any loop's error handler subVI processes a legitimate error

Continuous Measurement and Logging

- The Continuous Measurement and Logging sample project is a complete application based on the QMH
- It has three message handlers…the primary MHL, an acquisition MHL, and a logging MHL
- The core version of this sample project uses simulated acquisition data
- The NI-DAQmx version of this sample project uses real acquisition data

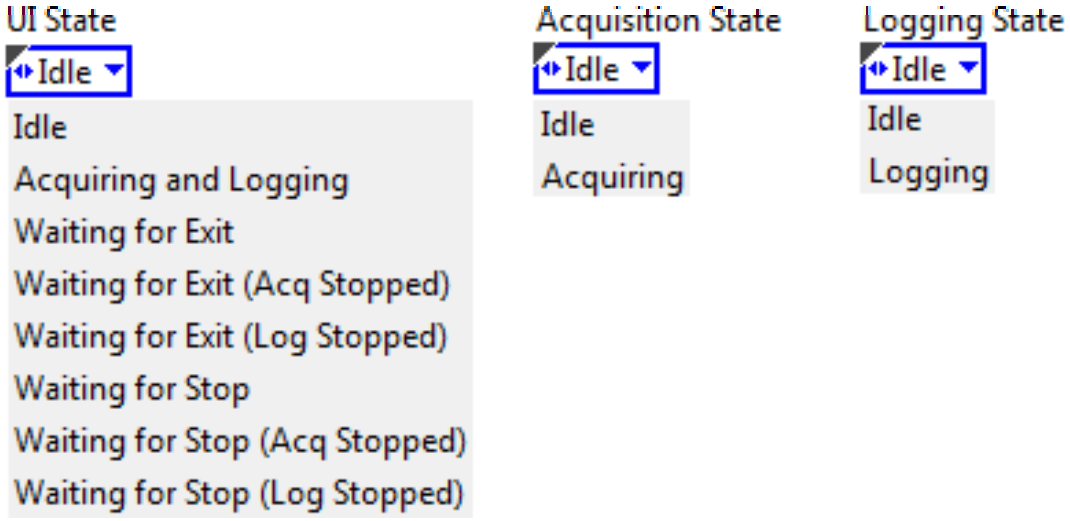
Cont. Meas. – Loop Communication



Blue arrows are messages.
Red arrows are data.
Green arrows are events.

Cont Meas. – Messages + State Checking

- In LabVIEW 2013, state checking was added to all three Message Handling Loops
- State checking helps with:
  - Performing simple handshaking between messages
  - Safeguarding against bogus messages



Customizing the QMH

- Replace the message data with a LabVIEW class
  - Allows for custom handling of message data via dynamic dispatch vs. having to cast to a specific data type
- Add a maximum Message Queue size
  - Avoids having an unbounded resource on more tightly-bounded systems like RT
- Add a timeout to Dequeue Message.vi
  - Allows for periodic updates or other operations from the MHL
- Implement more sophisticated communication between parallel loops via User Events
  - Potential use of new 'high priority' user events in LabVIEW 2013
- Provide additional error handling functionality
  - Error logging
  - Error classification (ignore, log only, non critical, critical)

Questions?

Stay Connected During and After NIWeek

ni.com/niweekcommunity


facebook.com/NIWEEK


twitter.com/#!/niweek

youtube.com/niglobal