LabVIEW Developer Days

Build Code. Form Communities. Gain Confidence.

NATIONAL INSTRUMENTS™

Decisions Behind the Design of the Queued Message Handler Template

Outline

QMH Philosophy

QMH Design

- Main VI organization
- Inter-loop communication (Queues)
- Inter-loop communication (User Events)
- Error Handling
- Project organization

A complete desktop application based on QMH

Potential areas of expansion and customization

QMH Philosophy - Why start here?

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

Good architecture for applications where

1. UI must always be responsive
2. Buffer overflows can be an issue due to acquiring, processing, logging data simultaneously
3. Communication with another application is required

QMH Philosophy - Values

Readability

- Non Class Based
- Static Number of Asynchronous Threads

Moderately Easy to Debug

- Can use highlight execution

Moderately Extensible

- With minimal effort can add new functionality at edit time

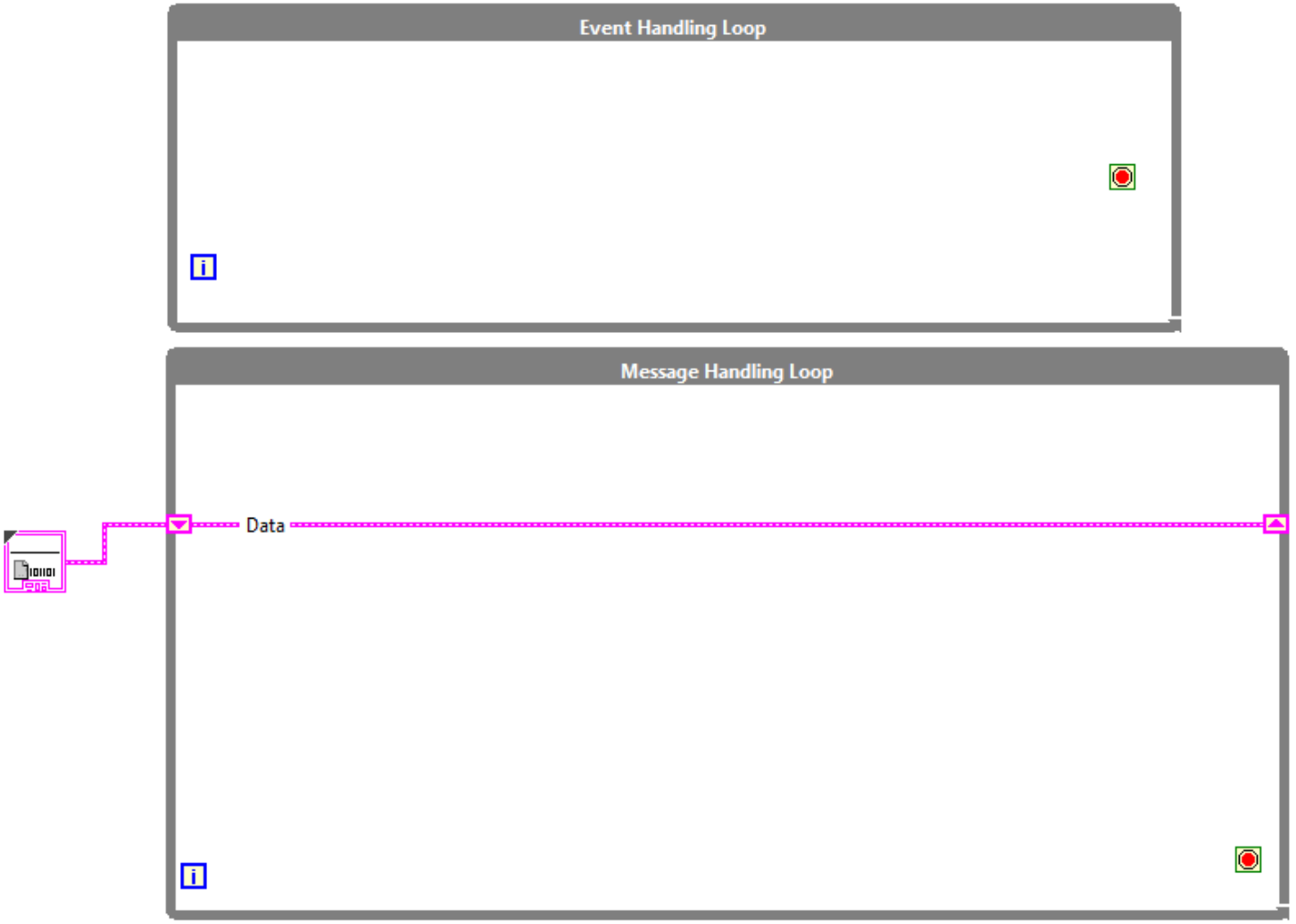
Modular

Demo

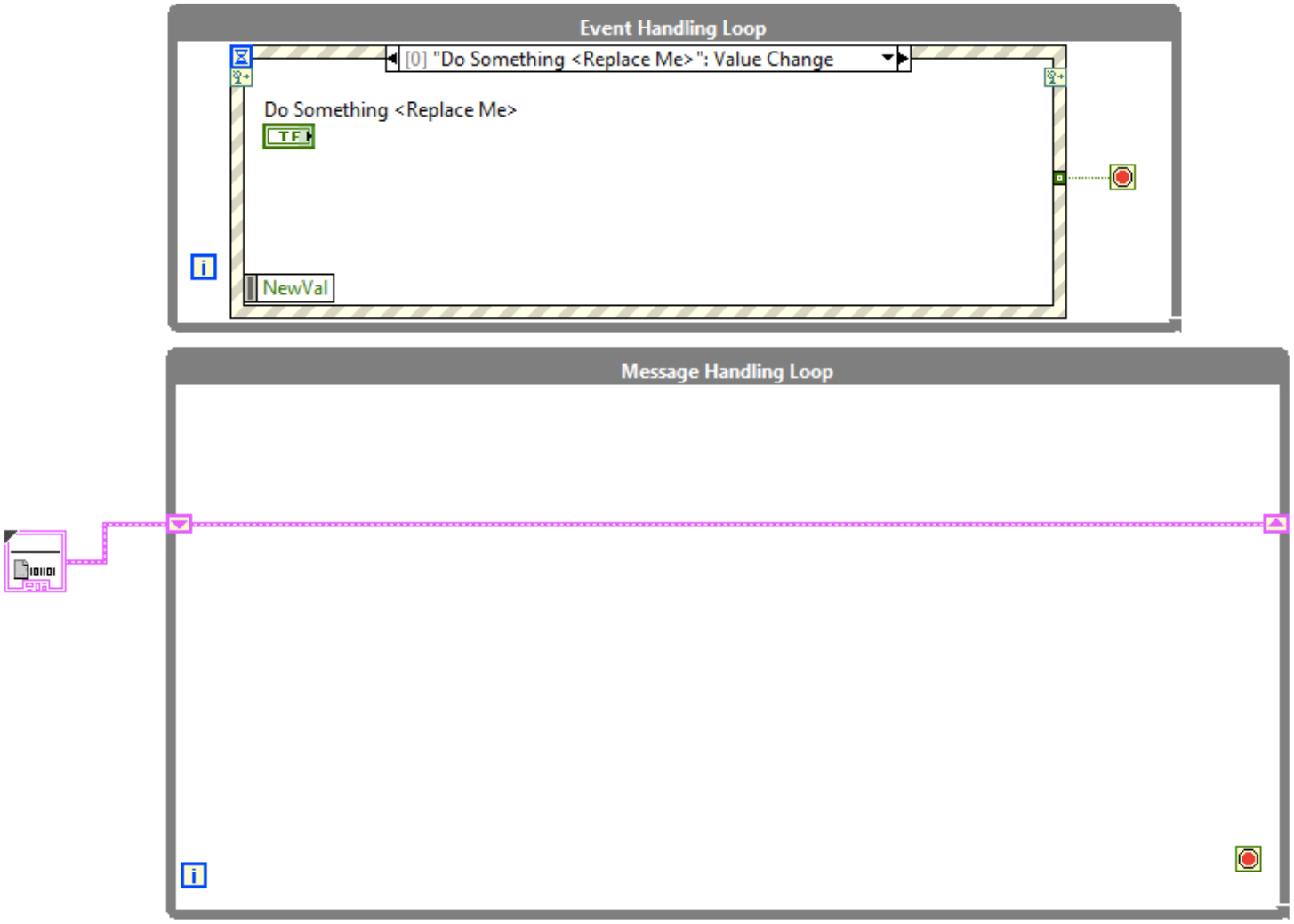
1. How to create an application from the template

1. Template documentation

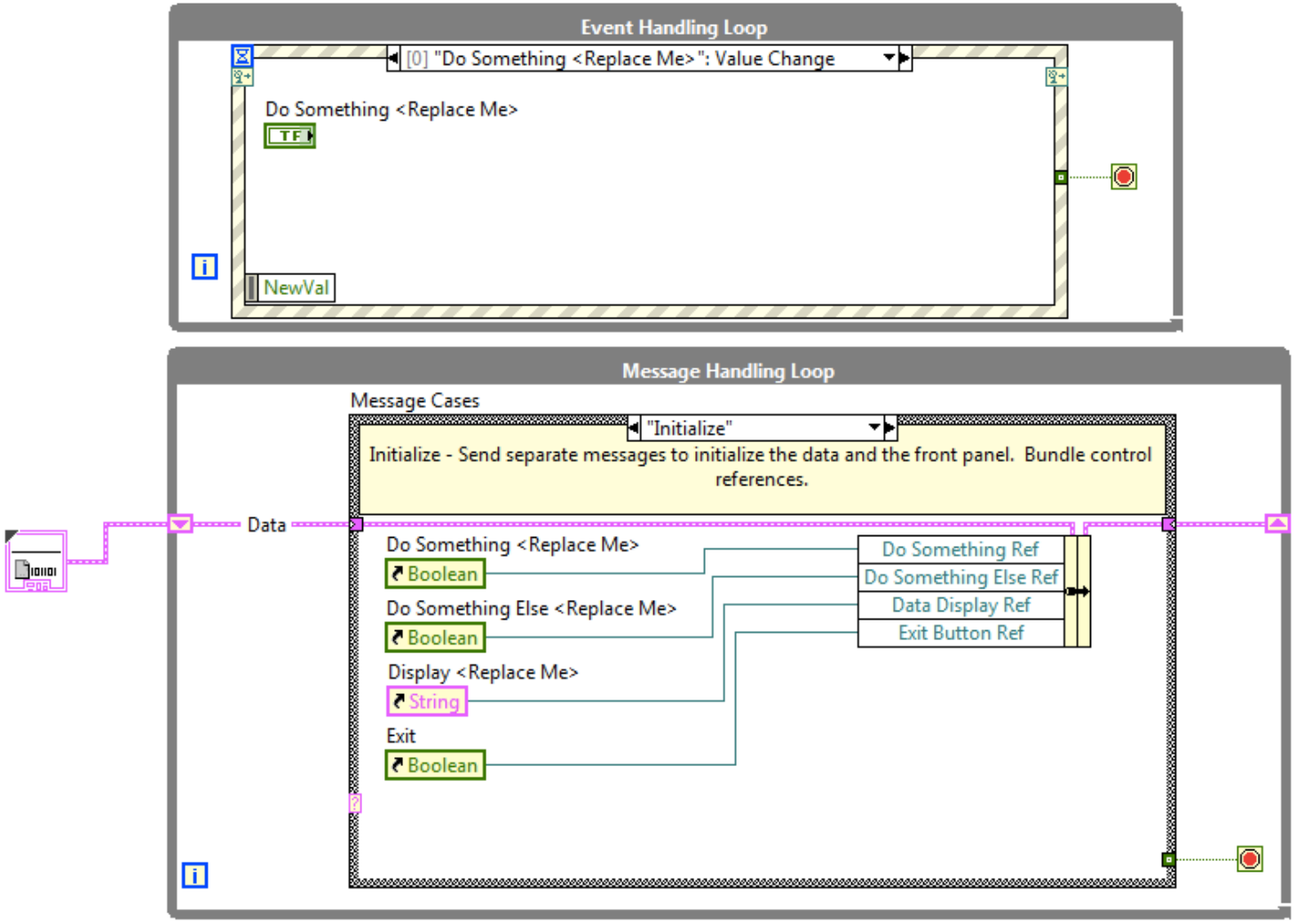
QMH Design – Main VI



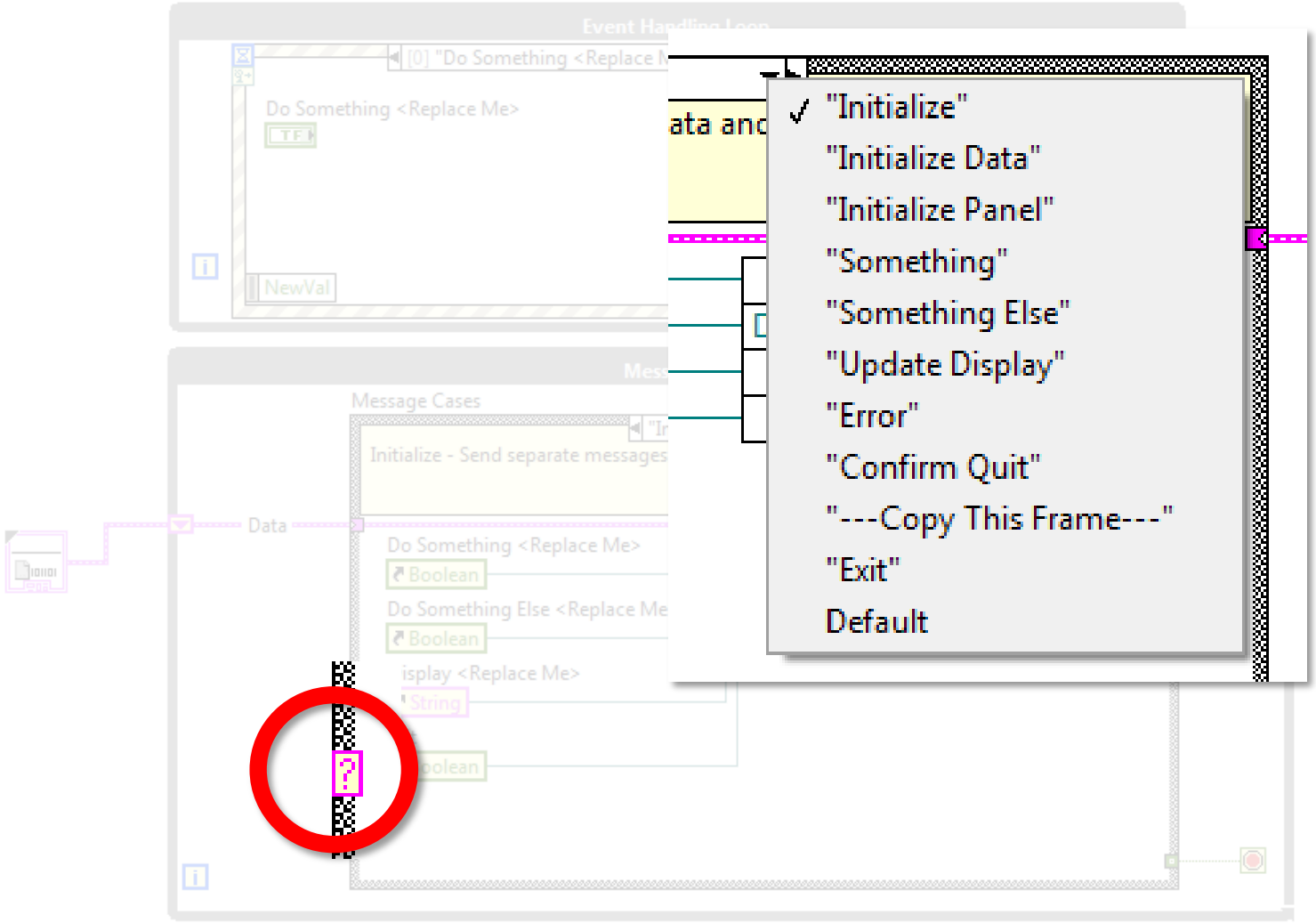
QMH Design – Main VI



QMH Design – Main VI



QMH Design – Main VI



QMH Design – String Type Rationale

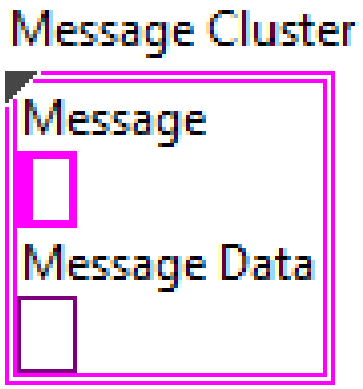
Why does the QMH use string-based messages instead of enum-based?

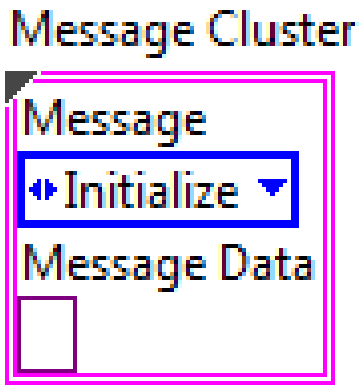
String-based Message

- 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



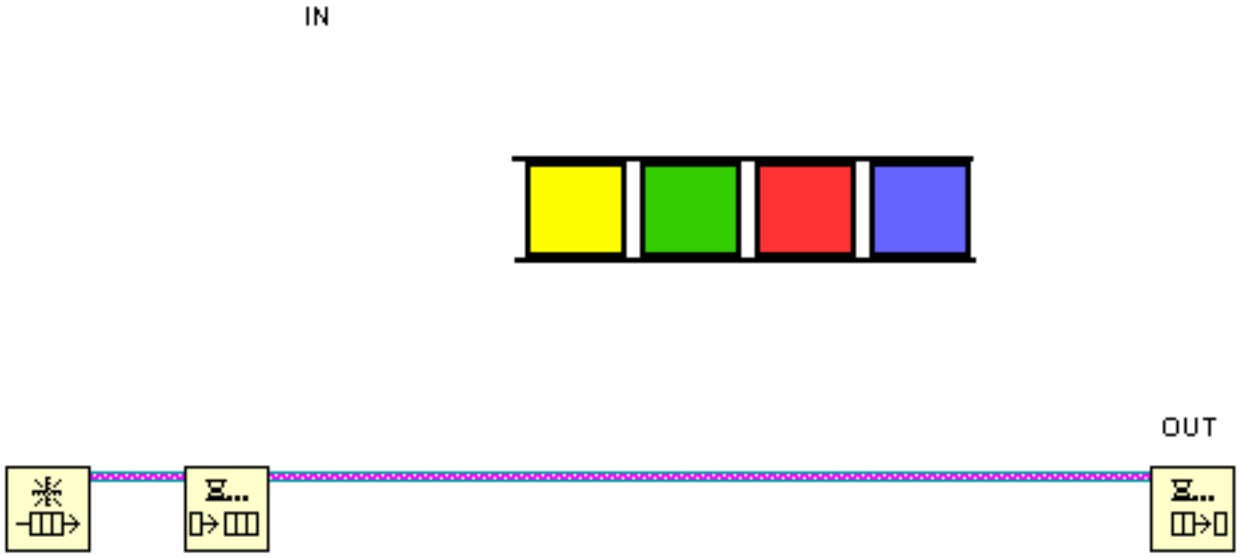


Quick Review - Queues

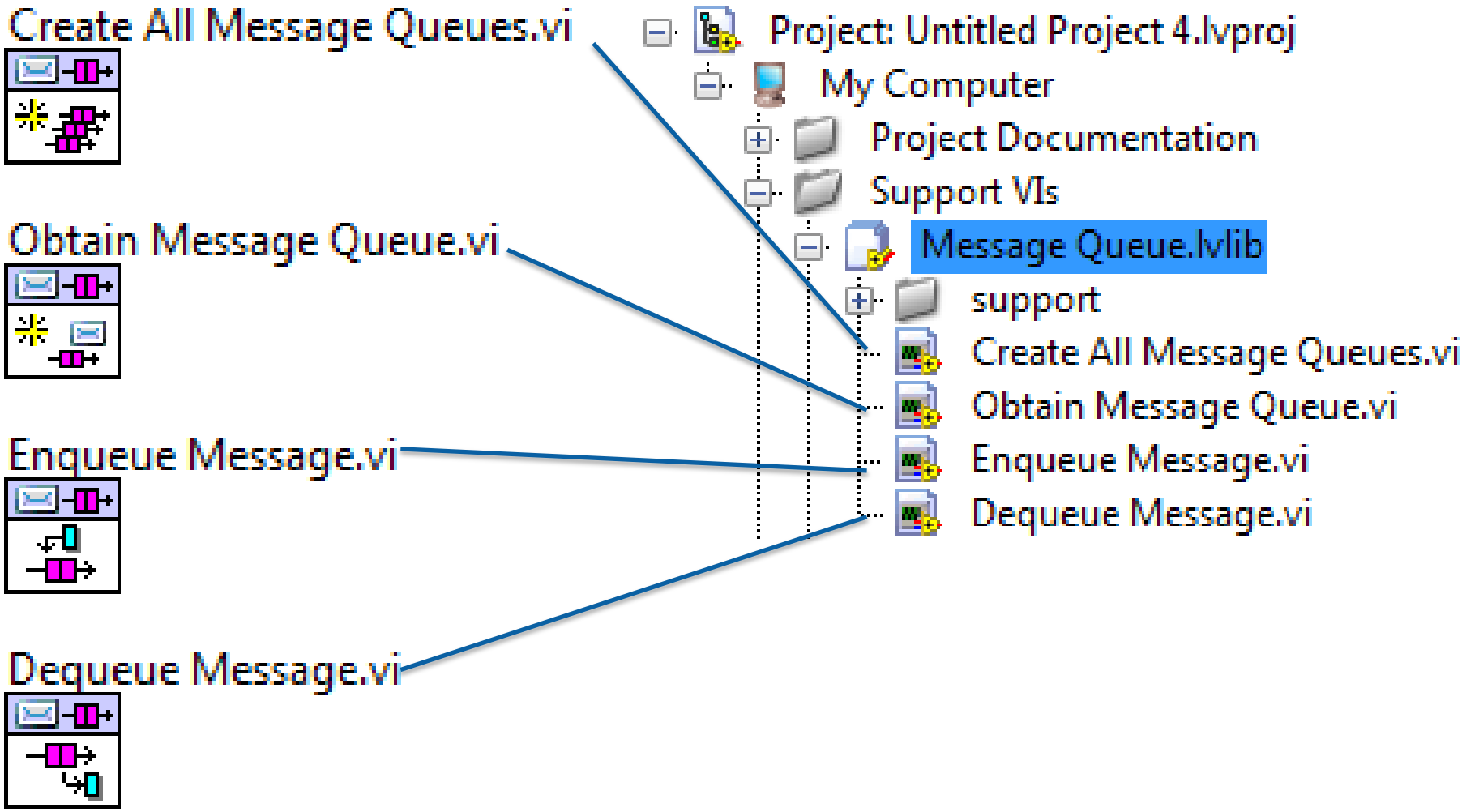
The queue is the heart of the QMH architecture

A queue is

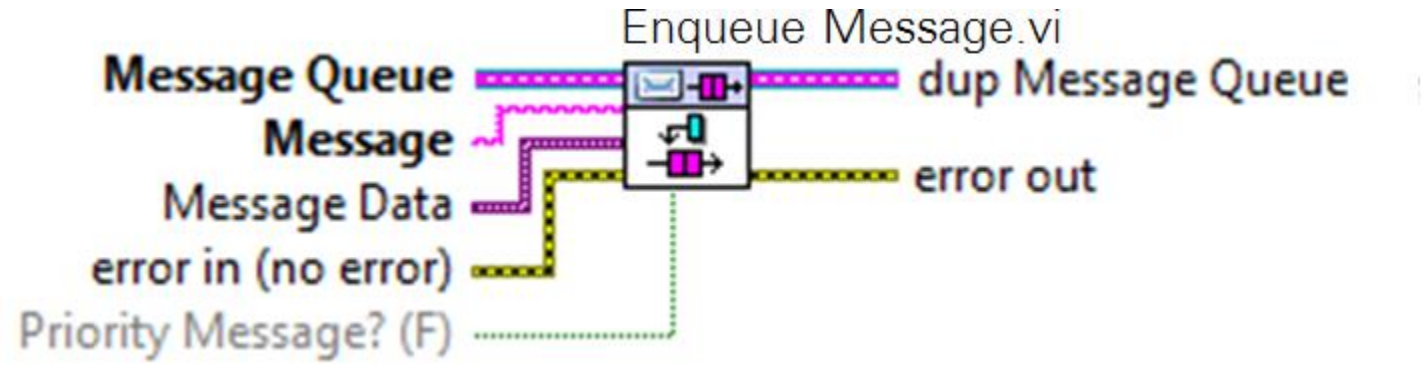
- A first in first out (FIFO) data transfer mechanism
- Lossless
- Reference based API



QMH Message Queue Module



QMH Design – Message Format



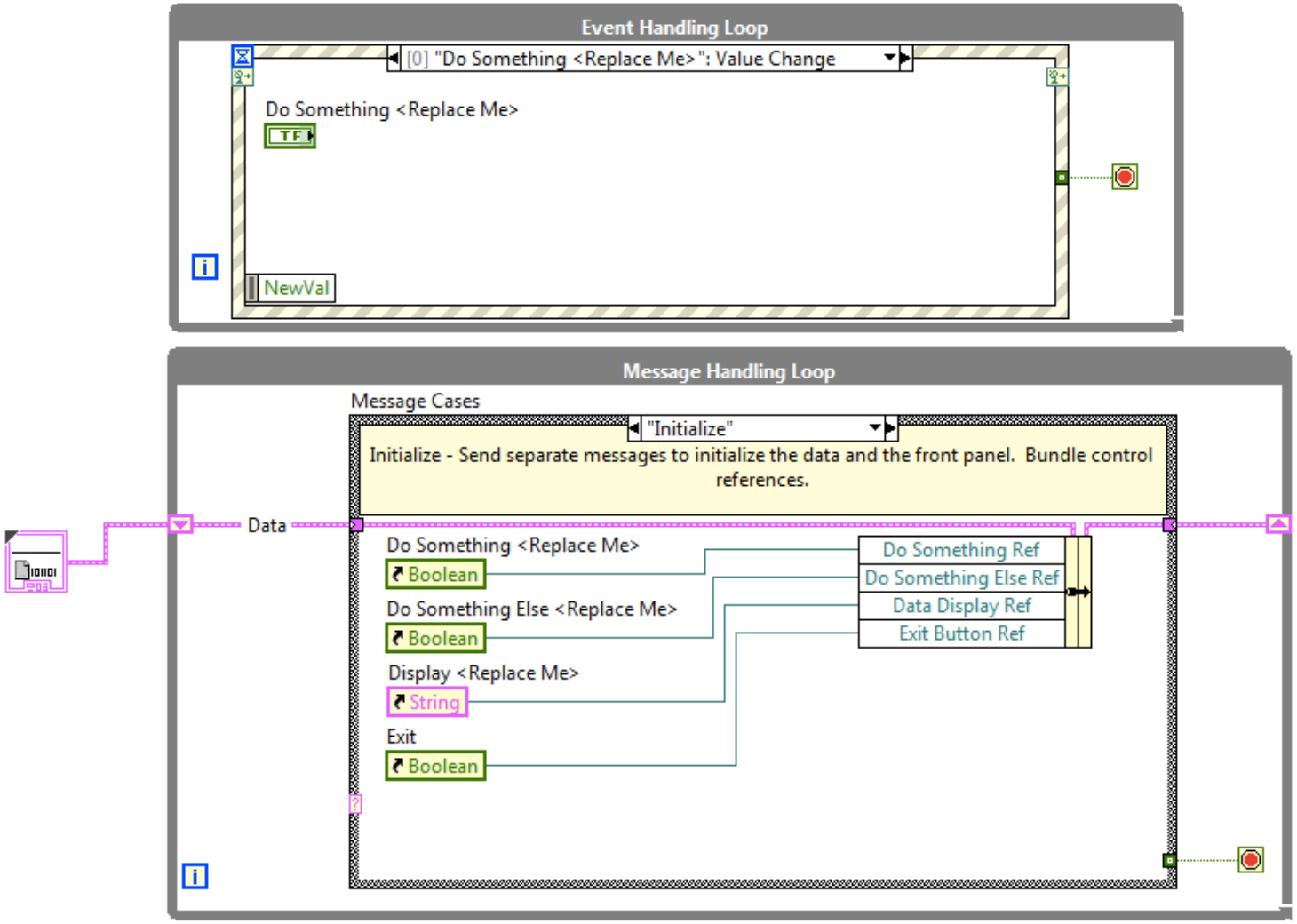
A string specifies the Message type

A variant optionally specifies the Message Data

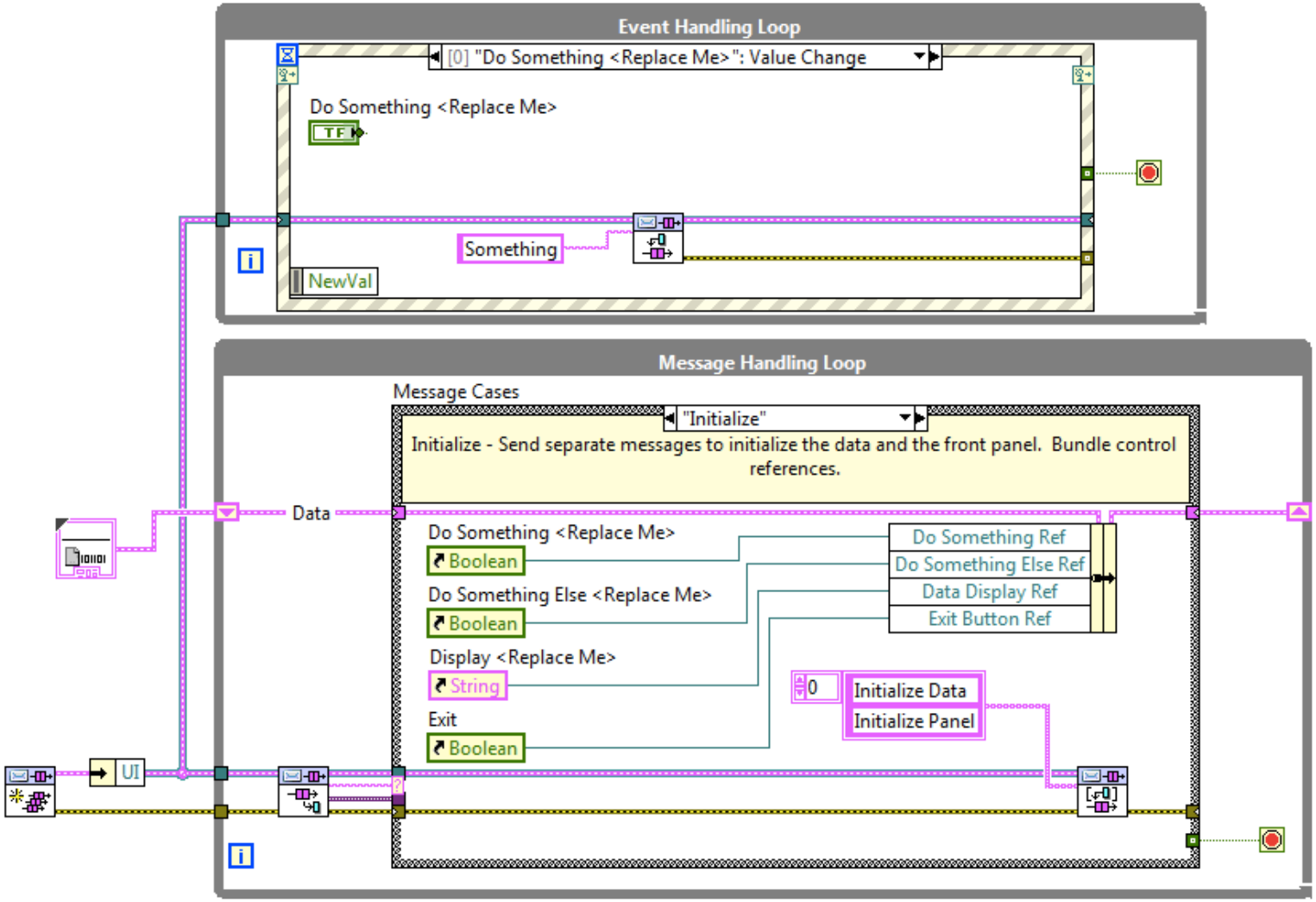
A message can be optionally placed at the front of the message queue as a Priority Message

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

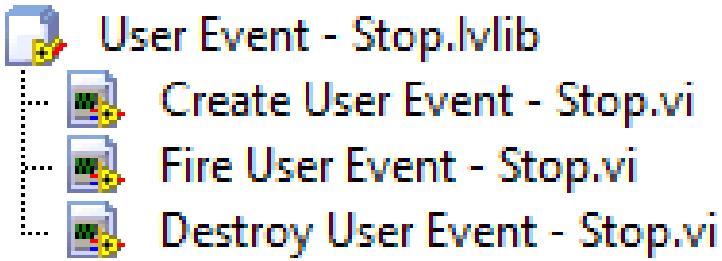
QMH Design – Main VI



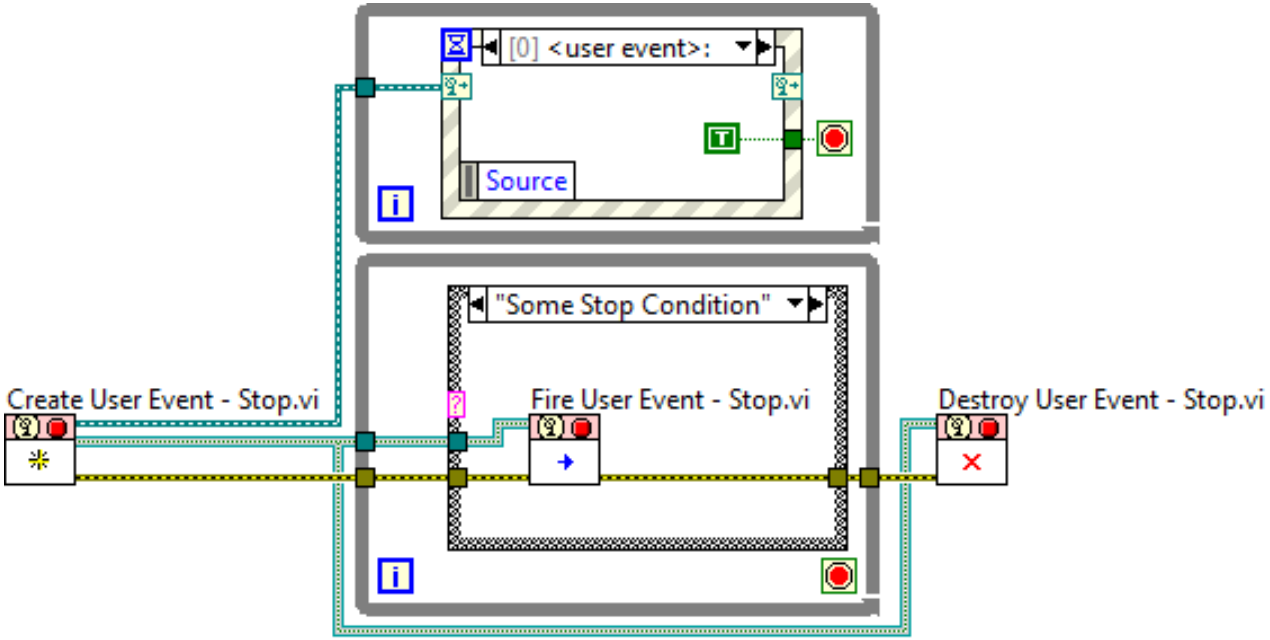
QMH Design – Main VI



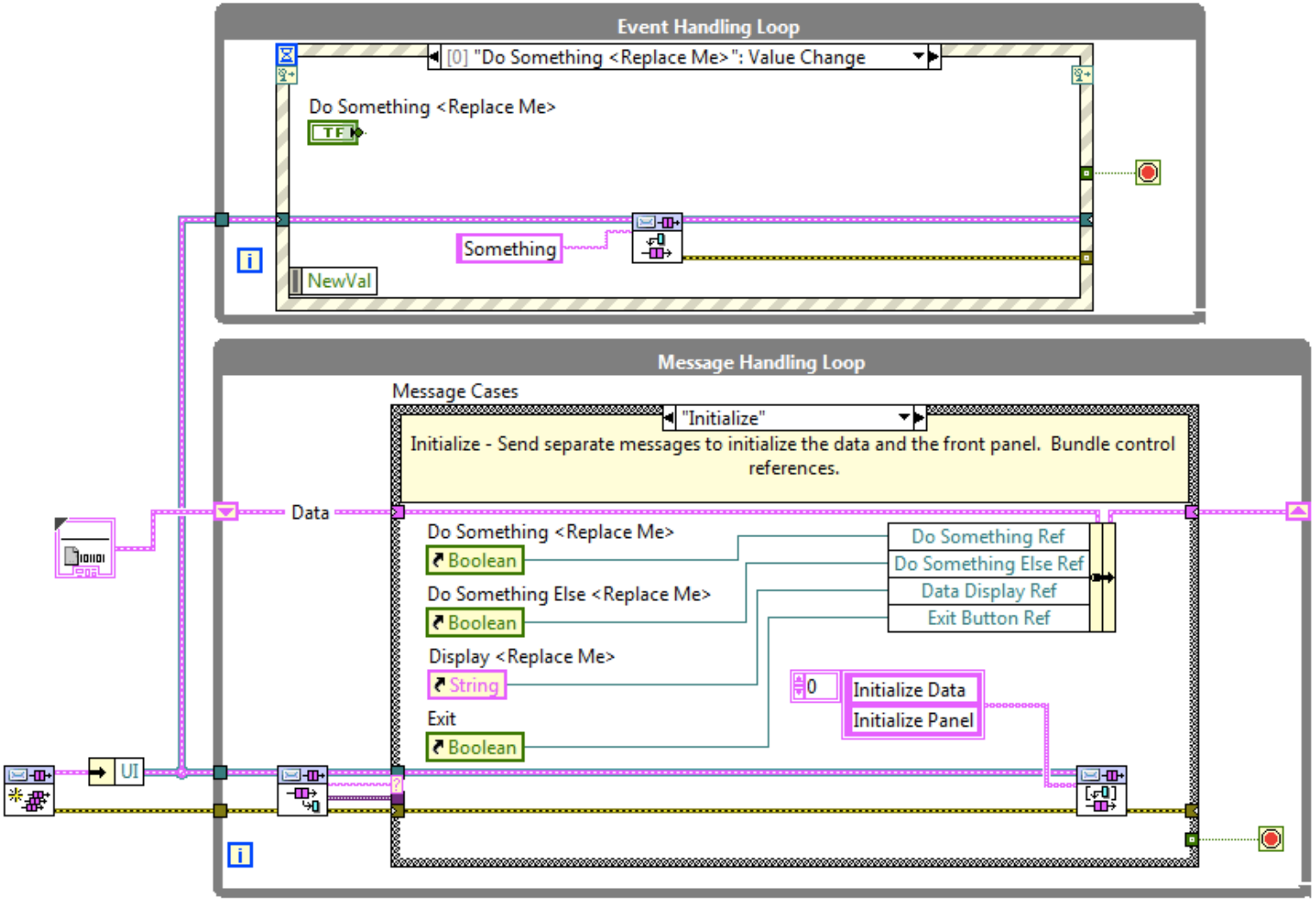
QMH Design – User Event – Stop.lvlib



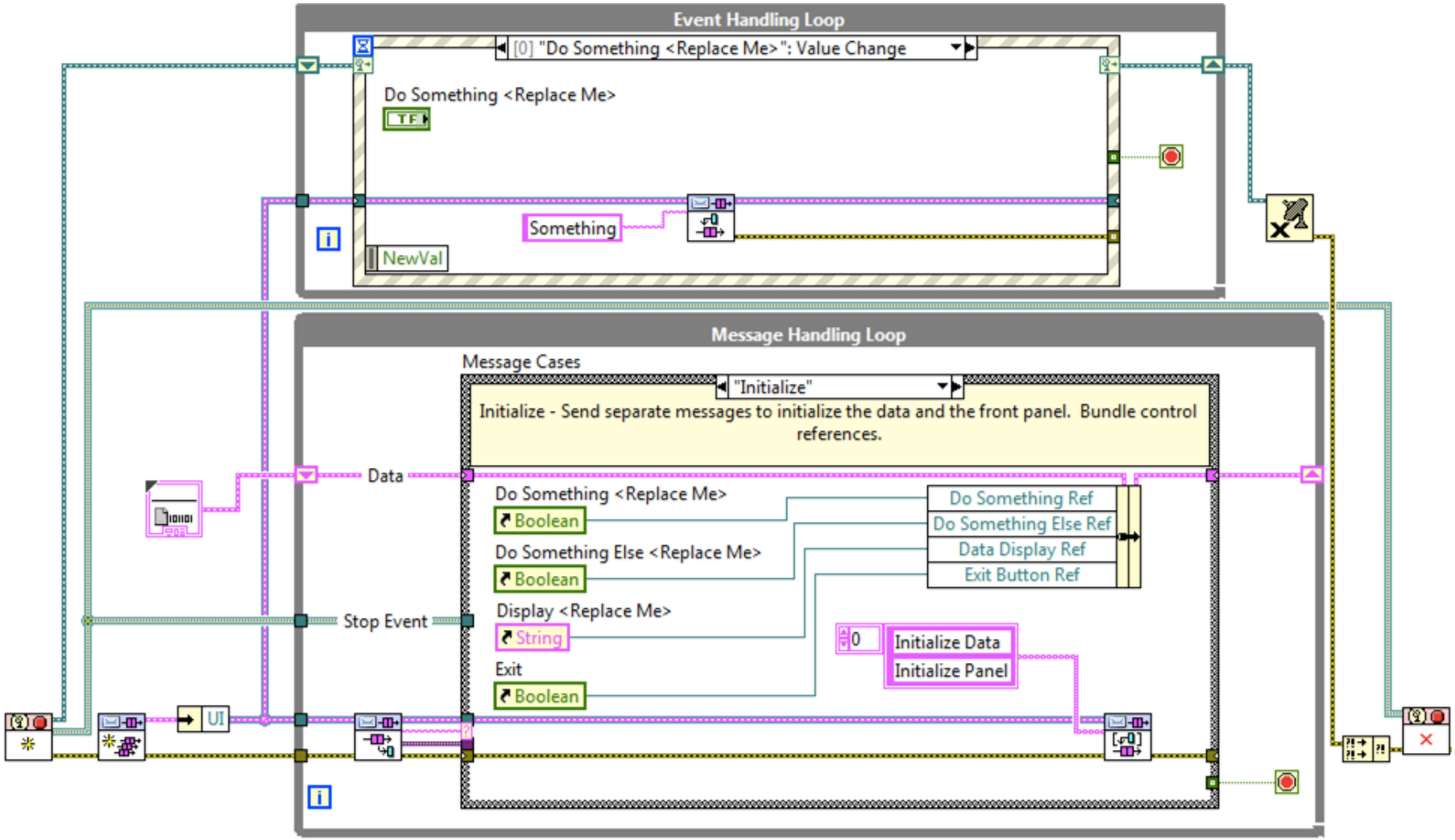
Code for stopping Event Handling Loop from Message Handling Loop



QMH Design – Main VI

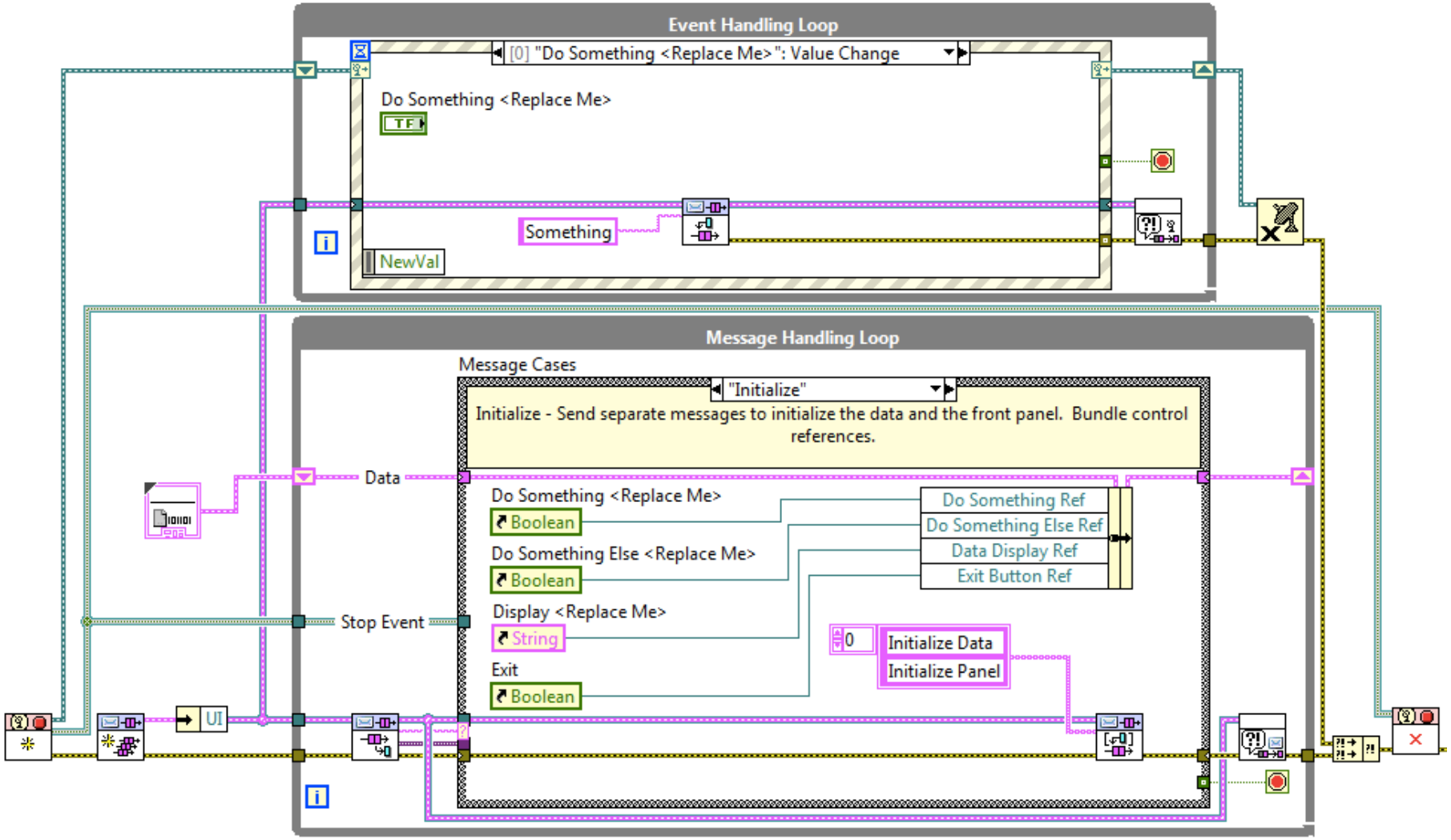


QMH Design – Main VI

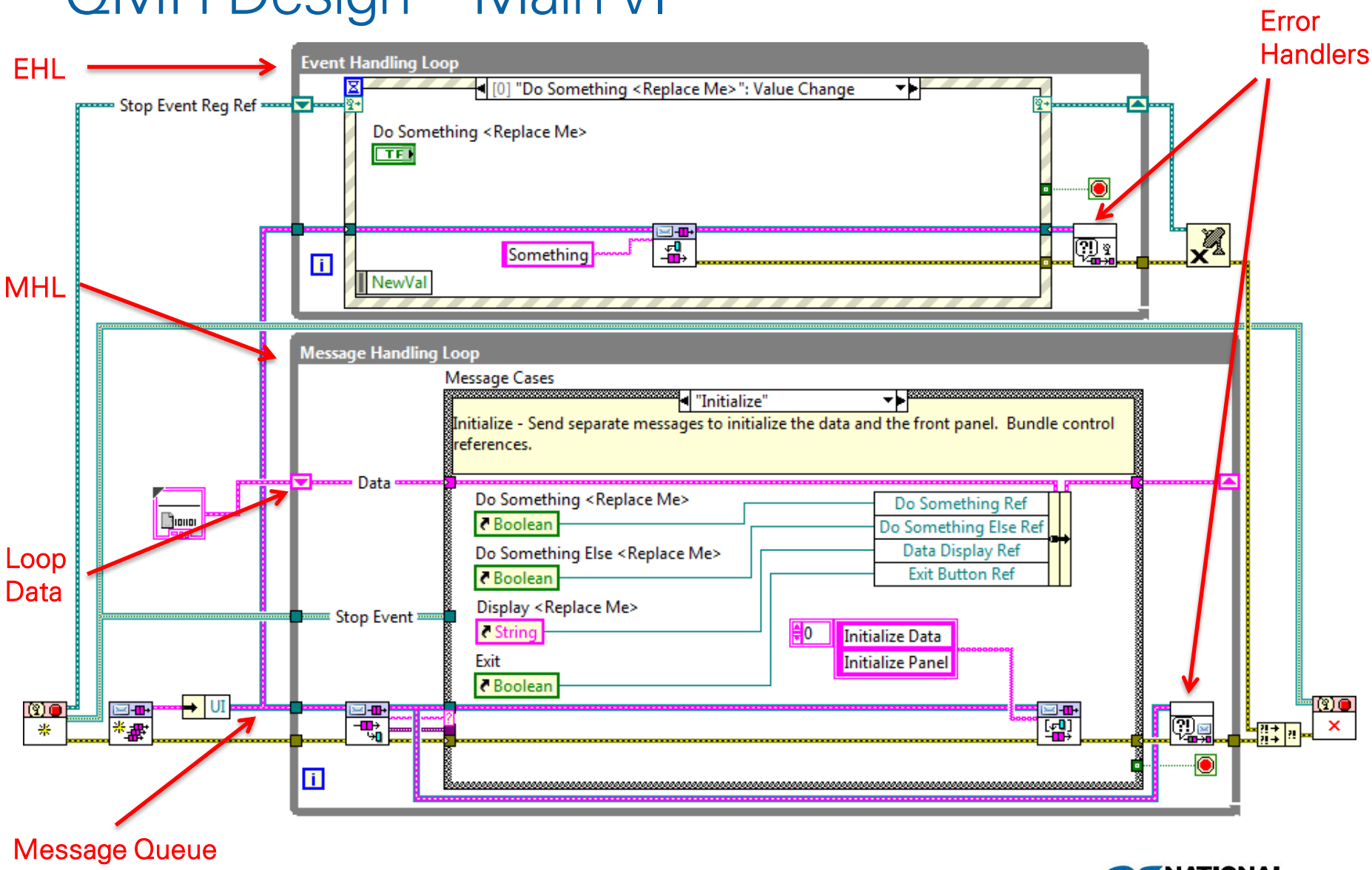


NATIONAL INSTRUMENTS

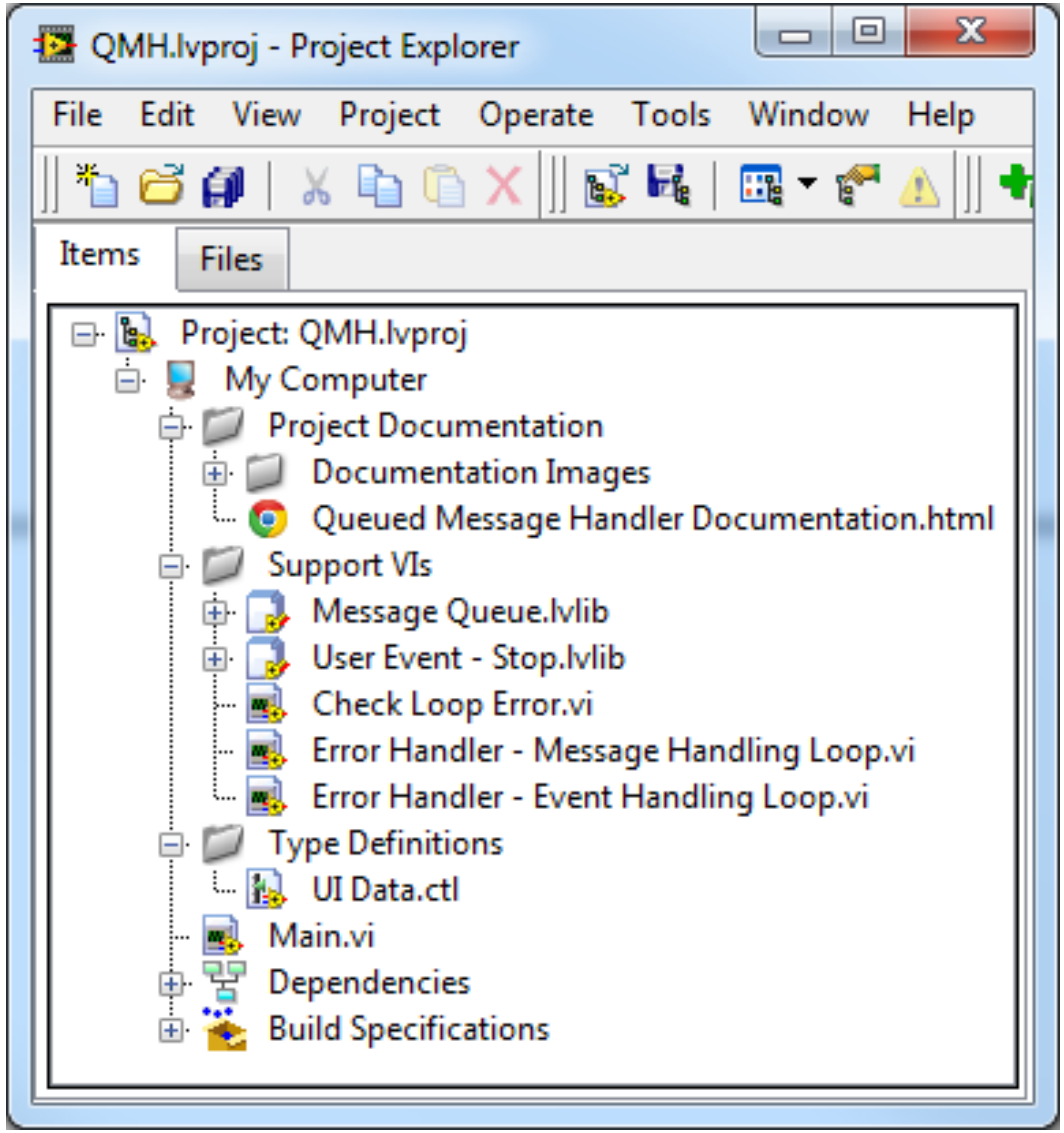
QMH Design – Main VI



QMH Design – Main VI



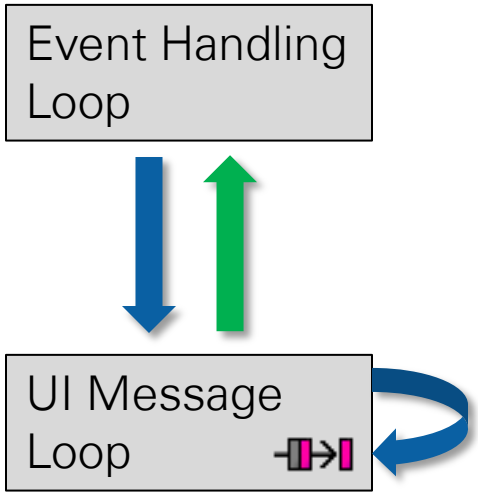
QMH Design – Project Organization



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

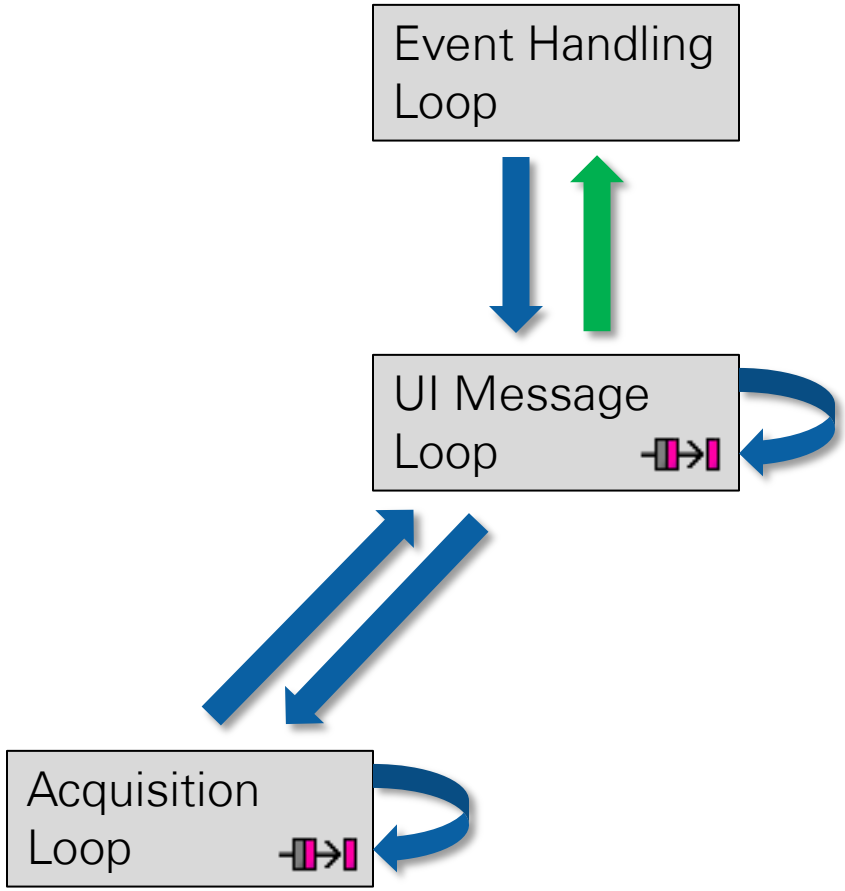
Cont. Meas. – Loop Communication



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

Same message handling API used across the entire application.

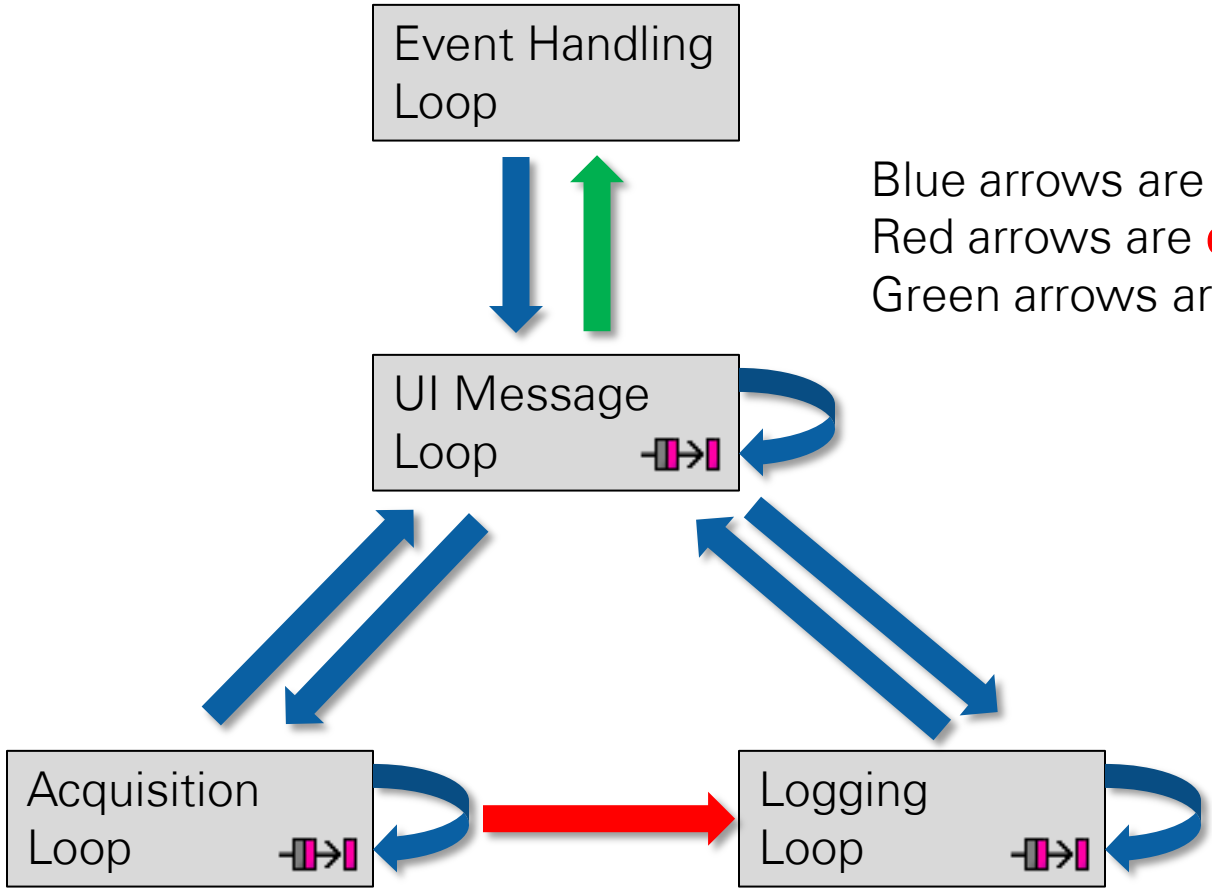
Cont. Meas. – Loop Communication



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

Same message handling API used across the entire application.

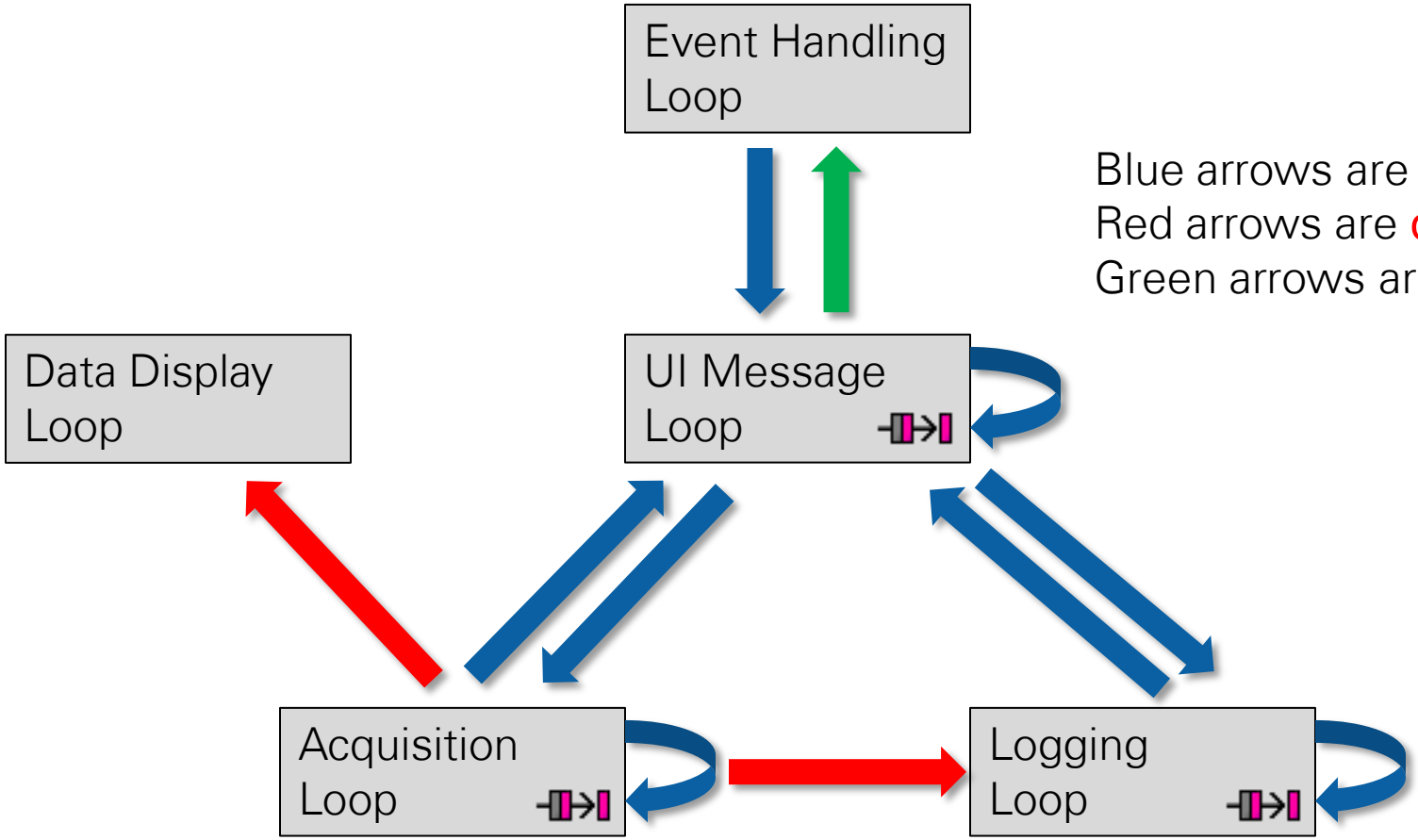
Cont. Meas. – Loop Communication



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

Same message handling API used across the entire application.

Cont. Meas. – Loop Communication



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

Same message handling API used across the entire application.

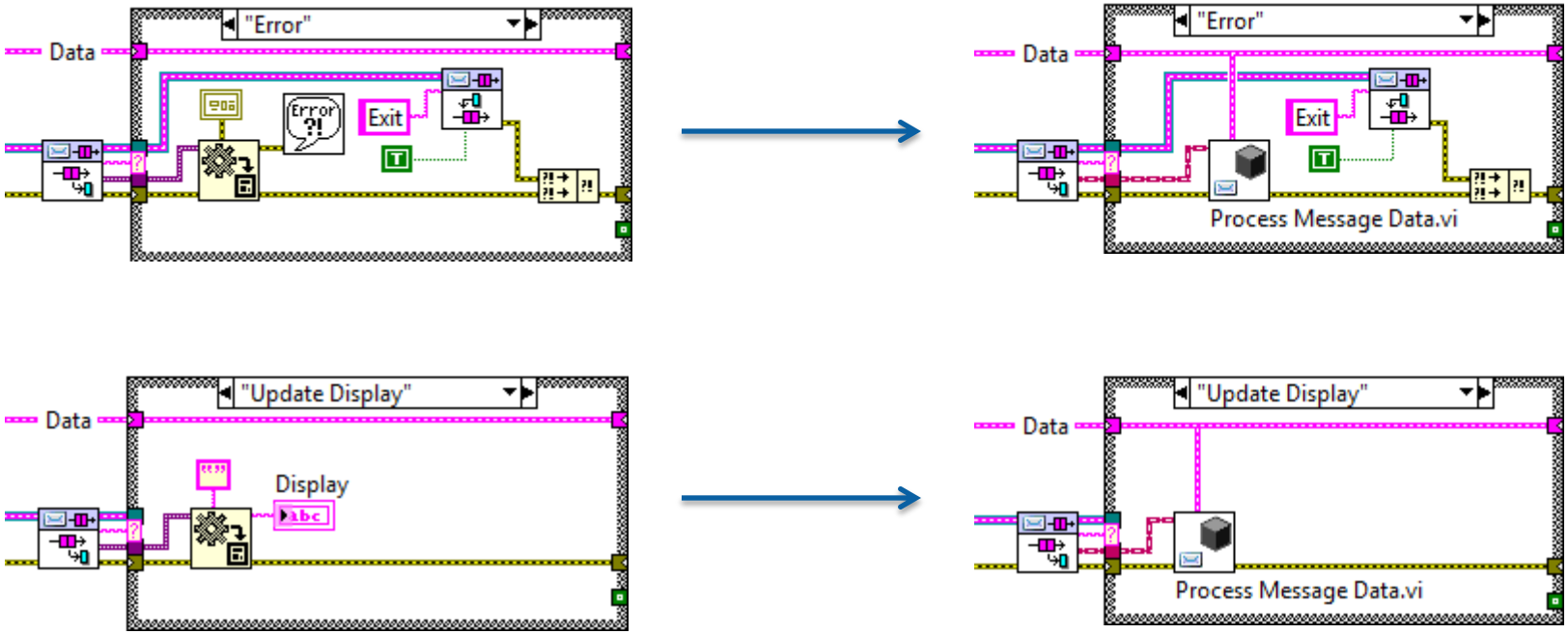
Customizing the QMH

Since you get your own copy of the support libraries when you create an instance of the QMH template, you can customize the code any way you wish. Here are some of the more common modifications that have been made by users for their QMH-based applications.

Customizing the QMH

Replace the message data variant with a LabVIEW class

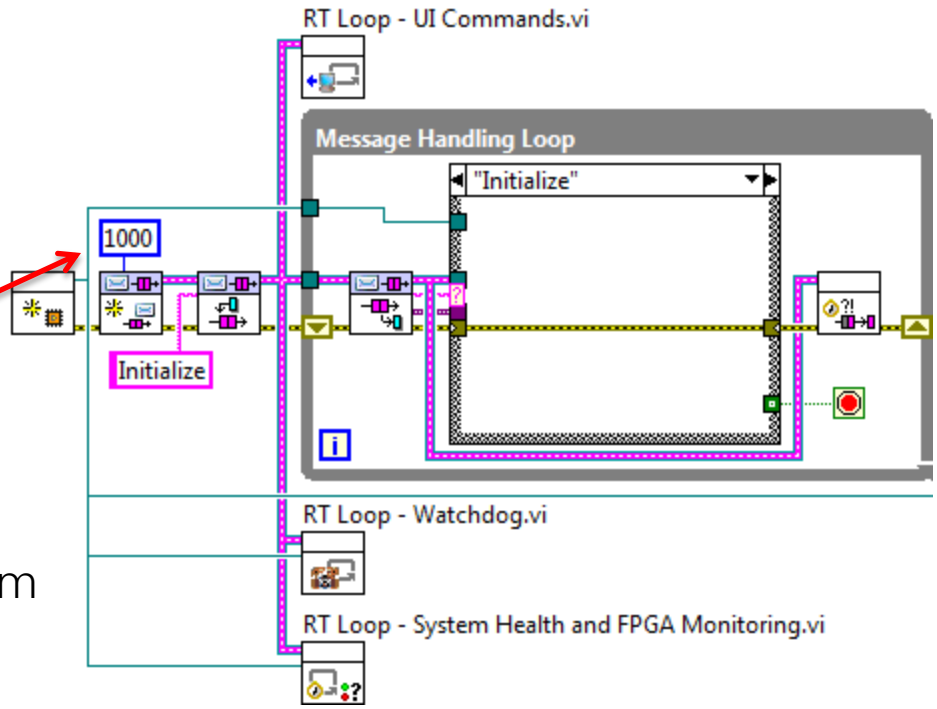
Allows for custom handling of message data via dynamic dispatch vs. having to cast the variant to a specific data type



Customizing the QMH

Add a maximum Message Queue size

Avoids having an unbounded resource on more tightly-bounded systems like RT

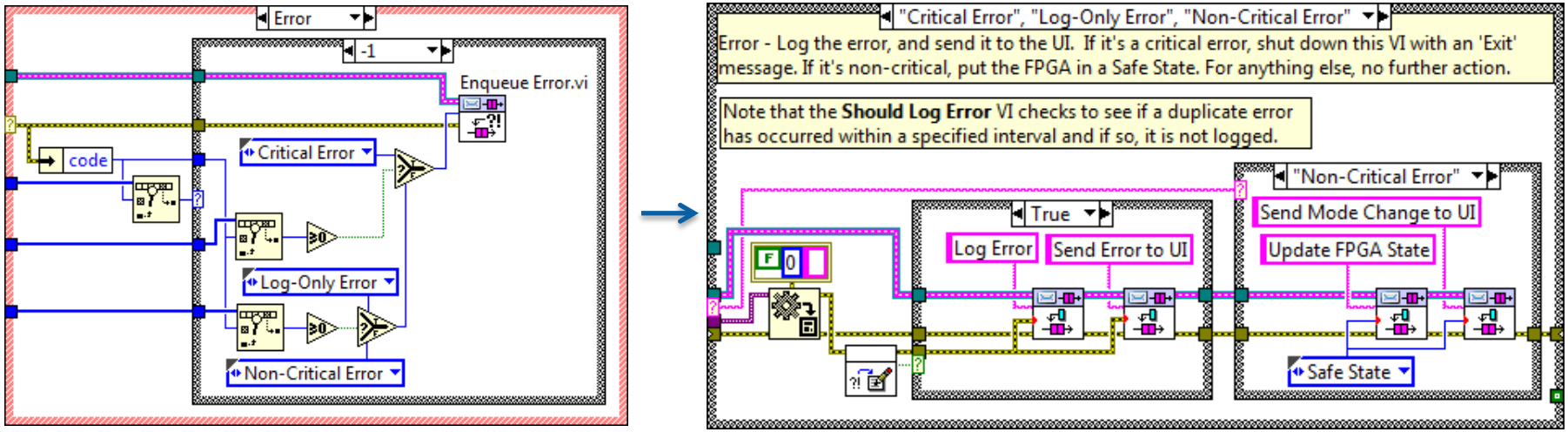


All of the CompactRIO sample projects shipped with NI products set a maximum size on their message queues.

Customizing the QMH

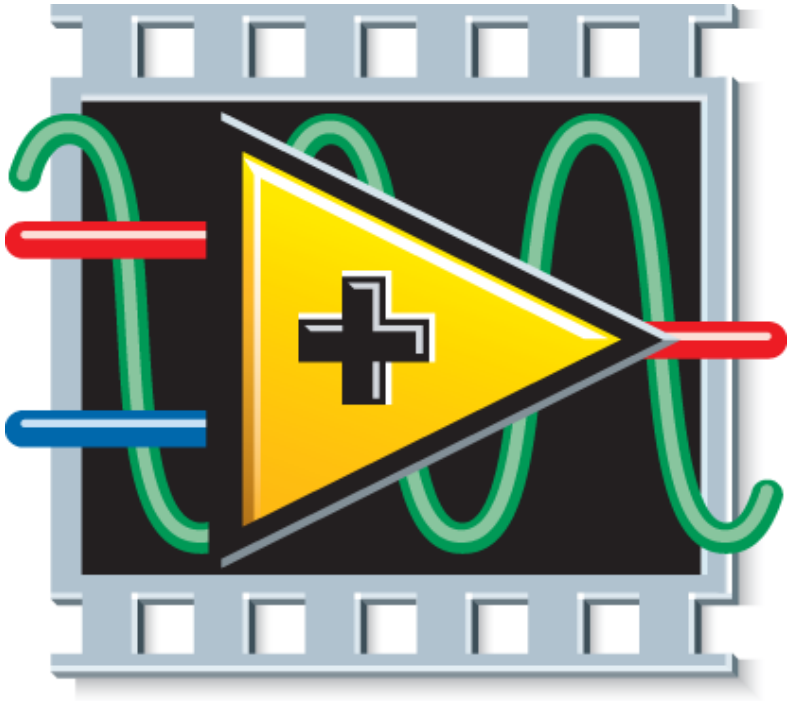
Provide additional error handling functionality

- Error logging
- Error classification (ignore, log only, non critical, critical)



All of the CompactRIO sample projects shipped with NI products implement this more advanced error handling functionality.

Thanks for attending!



NATIONAL INSTRUMENTS™

LabVIEW™

Already CLAD Certified?

You're immediately eligible to take the **Certified LabVIEW Developer** exam. Start preparing now!

- Join a local user group (**ni.com/usergroups**)
- Prepare using resources on Developer Zone **ni.com/training/certification_prep**
- Time yourself during practice exams

Note: CLAD certification must be current to take the CLD exam

Email certification@ni.com to register for an exam near you.