LABVIEW: THIS OR THAT?

TERRY STRATOUDAKIS

March 15, 2016

AGENDA

- Introductions
- Background
- LabVIEW: This or That?
- Future Meetings/Topics?

Introductions

Terry Stratoudakis, P.E.

- LabVIEW user since 1998
- BSEE/MSEE NYU Polytechnic Institute
- CLA since 2010, CPI since 2008
- Taught over 110 weeks of LabVIEW courses
- ALE System Integration co-founder

ALE System Integration

- New York and Maryland offices
- Defense, energy, and research applications
- Advanced custom LabVIEW training
- Process driven
- NI Alliance Partner since 2004

Background

Recipe vs. Strategy

- Recipe Driven:
  - Write to a file with 1 second loop time
  - Write to a file that is 50Msamples/sec x 256 channels
- Strategy Driven:
  - Refactor a 1,000 VI project for making additions
  - Move parts of an existing test to run on an FPGA

Basis for Analysis

- Objectively review all views
- Review ni.com, lavag.org, and LabVIEW blogs
- Understand general computer programming perspective
- Identify examples
- Expand examples with scenarios

This or That?

Cast of Characters – Part 1

- timeout or no-timeout?
- one loop or two loops?
- tabs or subpanels?
- project folders: virtual or auto-populating?

Cast of Characters – Part 2

- VI Server or SubVIs
- lvlibs or lvlclasses
- strings or enums
- queues or events

Extra Credit: how much debug logging?

Prediction: we will run out of time!

Time-out or no Time-out?

Scenario: Events, Notifiers, Queues have timeout options

Timeout

- Pros – non-blocking, something can always run
- Cons – not a substitute for a loop

No Timeout

- Pros – no polling, execute only when needed
- Cons – blocking code can lock up program

Verdict – an escape path is always needed

One Loop or Two?

Scenarios: Producer Consumer, QMH

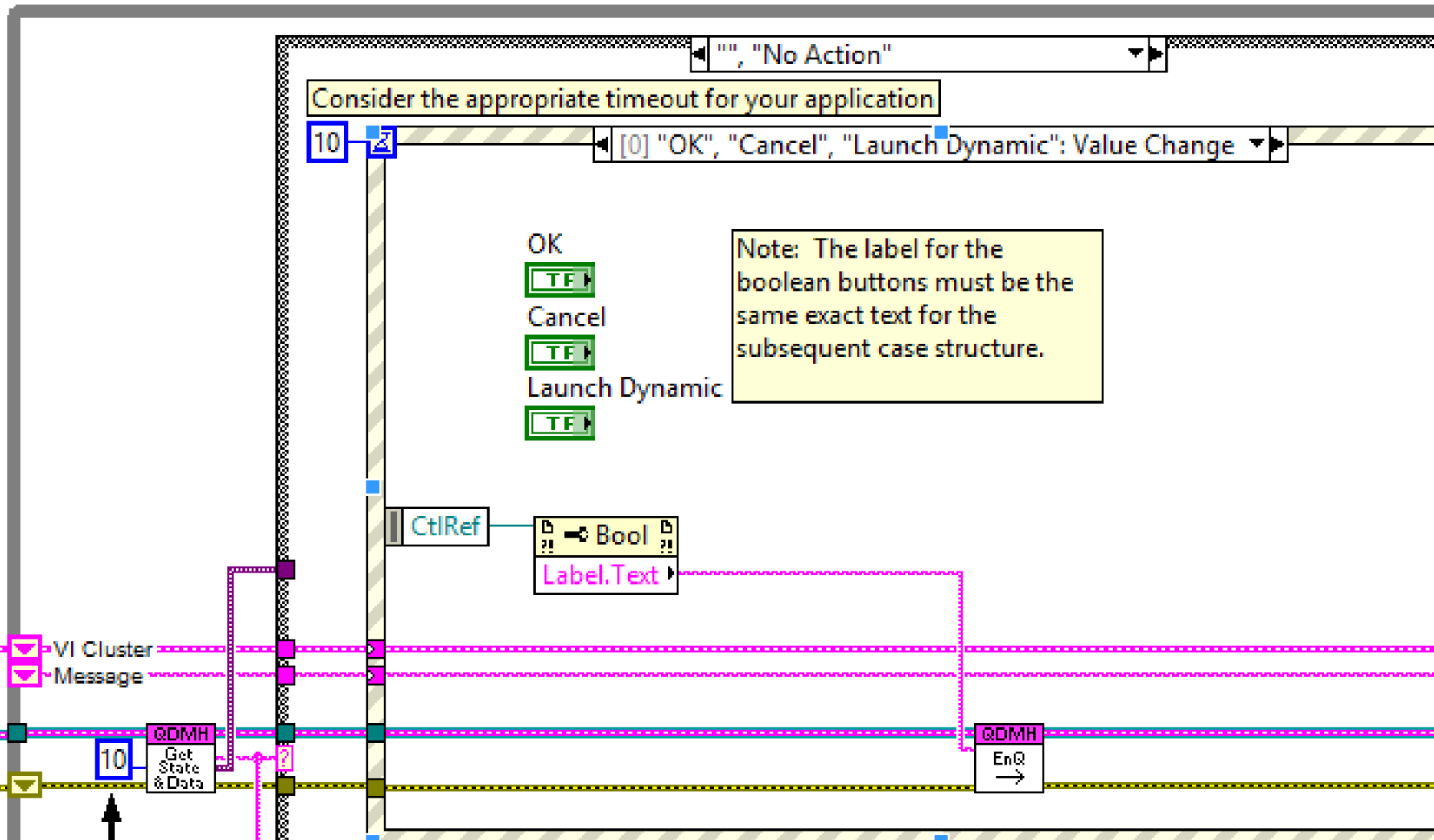
One While Loop

- Loop 1: State Machine; contains event structure

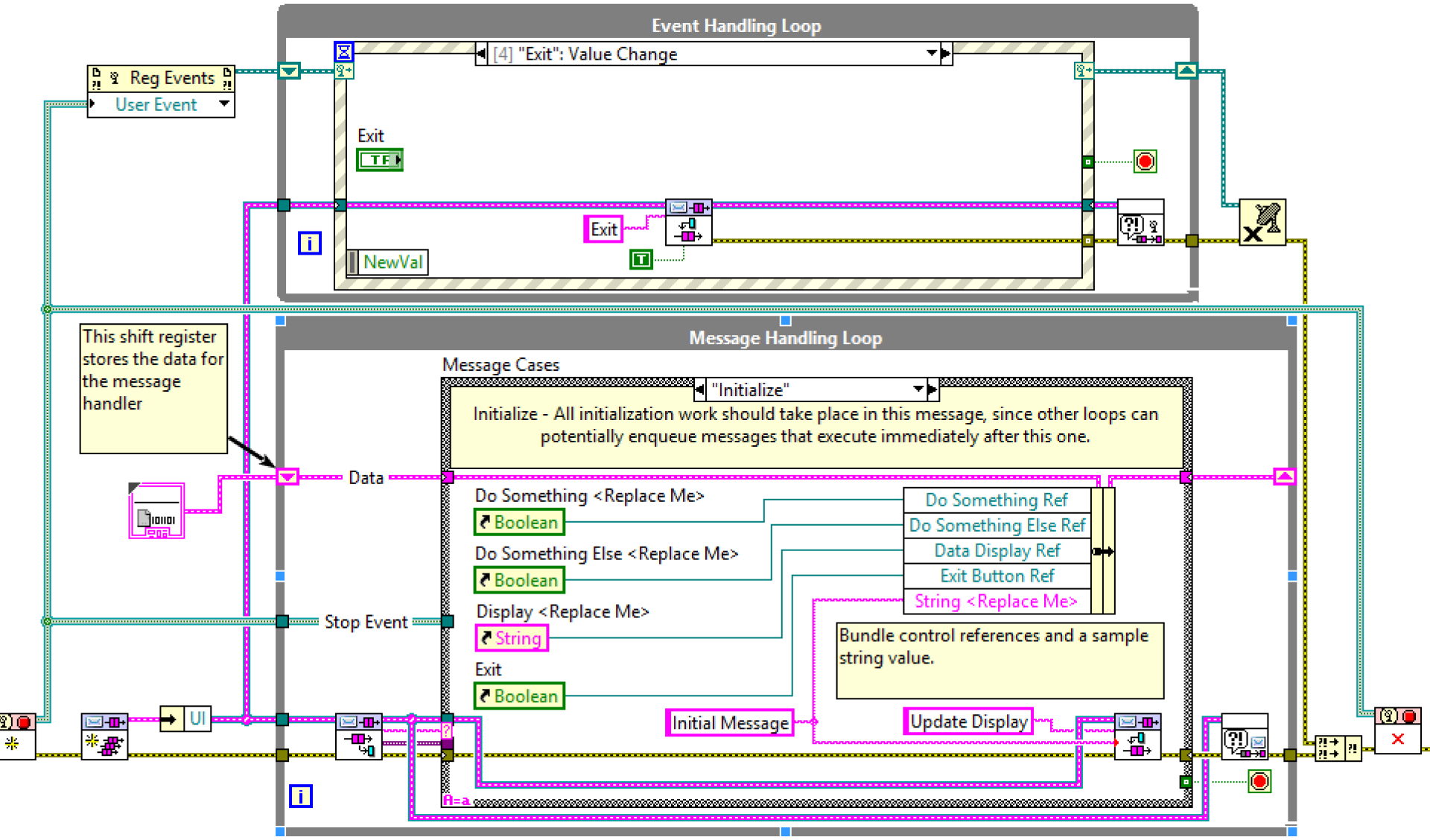
Two While Loops

- Loop 1: Event Structure
- Loop 2: State Machine

One Loop


"", "No Action"
Consider the appropriate timeout for your application
10
[0] "OK", "Cancel", "Launch Dynamic": Value Change
OK
TF
Cancel
TF
Launch Dynamic
TF
Note: The label for the boolean buttons must be the same exact text for the subsequent case structure.
CtlRef
Bool
Label.Text
VI Cluster
Message
10
QDMH
Get State & Data
QDMH
EnQ

Two Loops


Event Handling Loop
[4] "Exit": Value Change
Reg Events
User Event
Exit
TF
Exit
NewVal
This shift register stores the data for the message handler
Message Handling Loop
Message Cases
"Initialize"
Initialize - All initialization work should take place in this message, since other loops can potentially enqueue messages that execute immediately after this one.
Data
Stop Event
Do Something <Replace Me>
Boolean
Do Something Else <Replace Me>
Boolean
Display <Replace Me>
String
Exit
Boolean
Do Something Ref
Do Something Else Ref
Data Display Ref
Exit Button Ref
String <Replace Me>
Bundle control references and a sample string value.
Initial Message
Update Display
UI

One Loop or Two - Analysis

One Loop

- Loop: State Machine; contains event structure
- Pros – simpler to debug, one state at a time
- Cons – not multi-core, no choice in timeouts (required)

Two Loops -

- Loop 1: Event Structure, Loop 2: State Machine
- Pros – multi-core, event and queue handling in same VI
- Cons – need communication between loops

Tabs or Subpanels?

Scenario: Not enough room on Front Panel

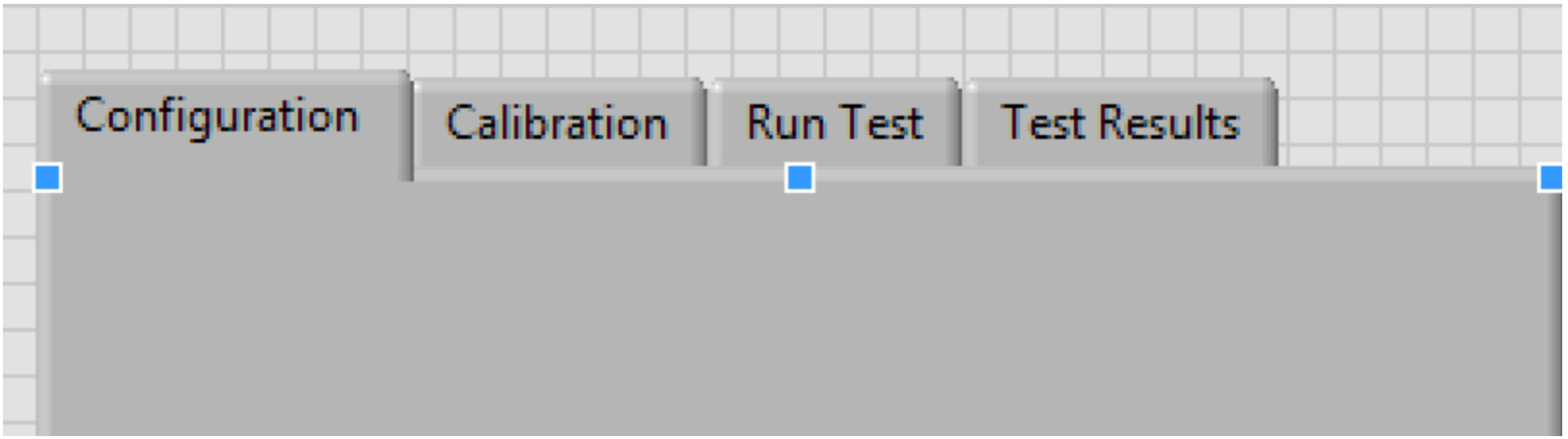
Test application

- Config, Calibration, Test, Report screens

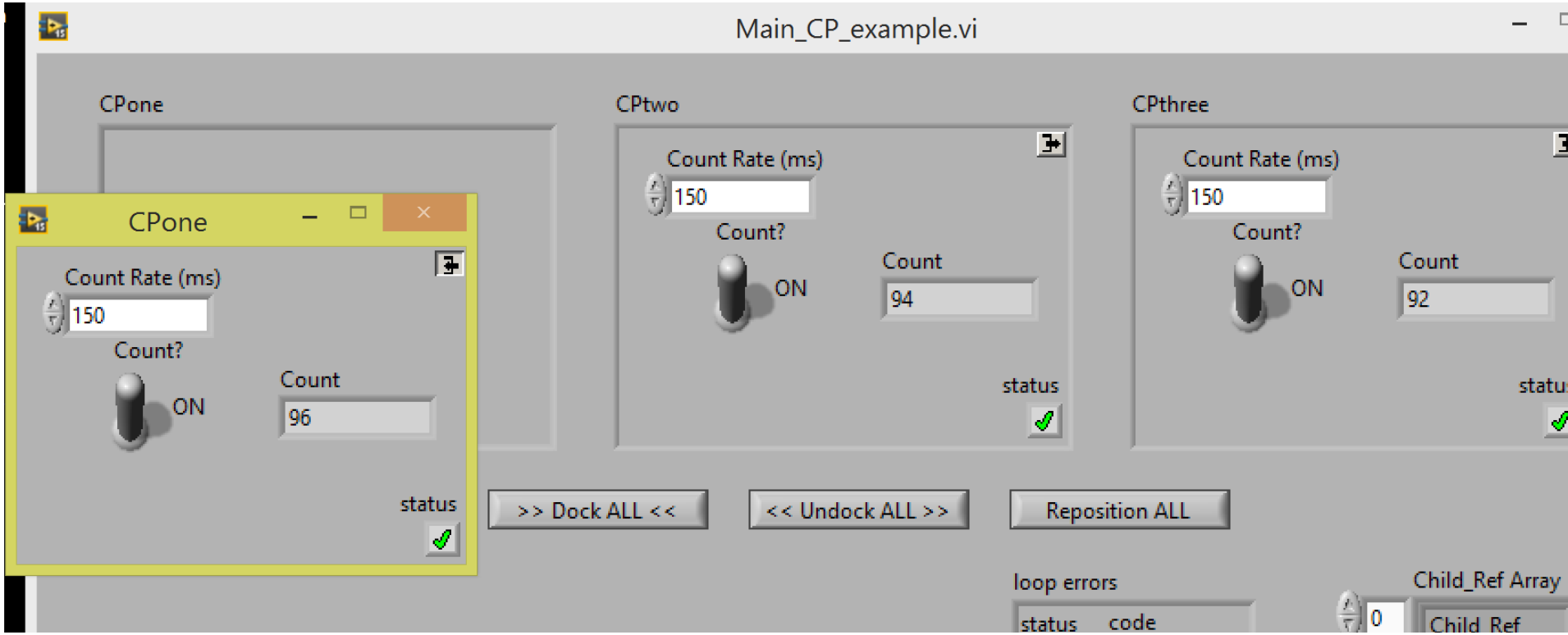
Control application

- Alarms, Monitoring, Dataviewer screens

Tab Control



Subpanels


Main_CP_example.vi
CPone
CPtwo
Count Rate (ms)
150
Count?
ON
Count
94
status
CPthree
Count Rate (ms)
150
Count?
ON
Count
92
CPone
Count Rate (ms)
150
Count?
ON
Count
96
status
>> Dock ALL <<
<< Undock ALL >>
Reposition ALL
loop errors
status
code
Child_Ref Array
0
Child_Ref

Tab or Subpanels - Analysis

Tab Control

- Pros – simple, familiar, all controls in same VI
- Cons – not scalable, most code runs in UI thread, memory intensive, results in one massive top-level VI

Subpanels

- Pros
  - Load GUIs into memory as needed, plug-in
  - Modular – more simultaneous developers
  - abstract GUI from computing code
- Cons – increases complexity, more inter-VI messaging

What about Xcontrols?

Graphical User Interface

Sub panels

- Load GUIs into memory as needed
- Modular – more simultaneous developers

Tab controls

- Not scalable
- Every GUI is always in memory
  - Inefficient memory usage for large applications
- Results in one massive GUI VI

VI Server or SubVIs?*

Scenario: Not enough room in one Block Diagram

SubVIs

- Pros – easy to create and call
- Cons – always in memory**

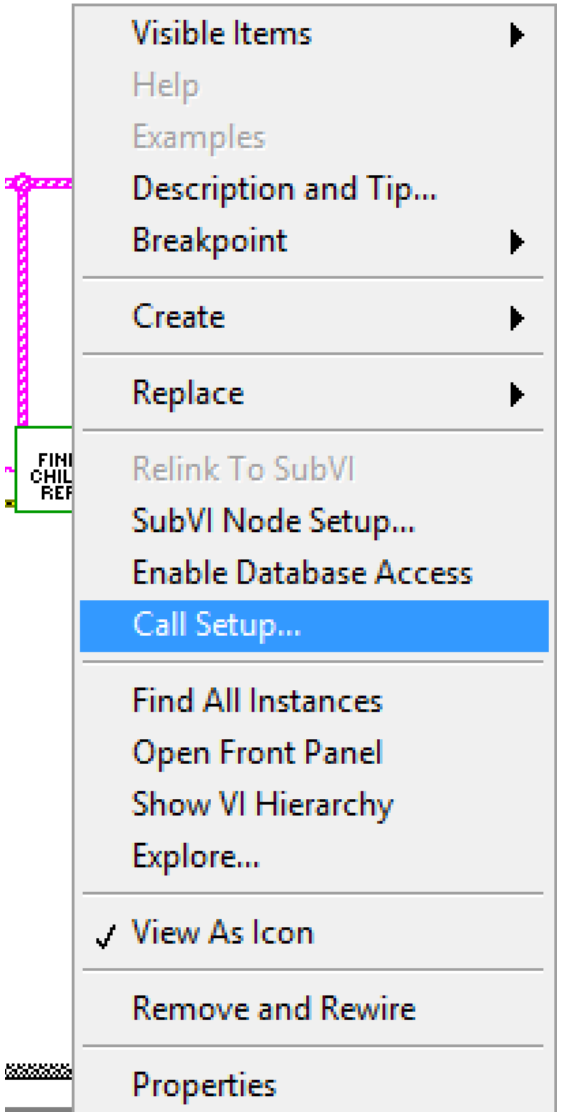
VI Server

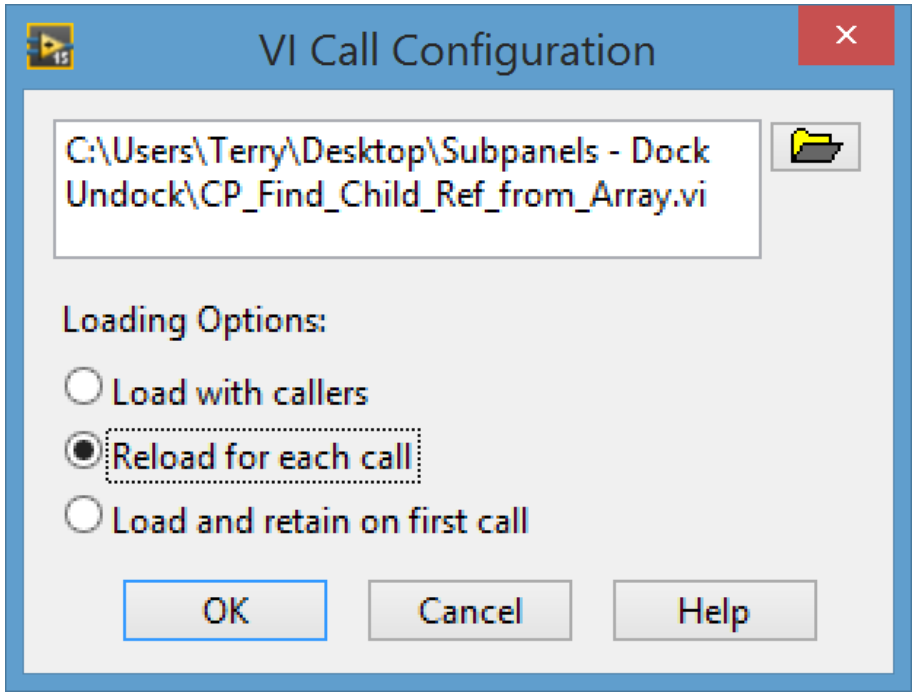
- Pros – plug-in architectures, load VIs as needed, less memory, remote access
- Cons – increases complexity, more inter-VI messaging

* *Similar to tab controls or subpanels discussion*

** Call Setup…

SubVIs: Call Node Setup





lvlib or lvlclass?

Scenario: Need LabVIEW Libraries (not .LLBs)

LVLIBs

- Pros – group code, easy to start using, scope settings, namespace
- Cons – no inheritance

LVCLASS

- Pros – OOP benefits
- Cons – complexity goes up

strings or enums?

Scenario: State Machine command datetype

Enums

- Pros – Strictly-typed, enforced at compile-time
- Cons – extra dependencies, cannot add without recompile

Strings

- Pros
  - can embed messages, low coupling
  - JKI State Machine, QDMH
- Cons – errors caught at run-time

Queues or Events?

Scenario: Messaging between different VIs

Queues

- Pros – reduces copies of memory, code remains in same thread, complete API
- Cons – cannot broadcast, data can be intercepted if Queue name is known

Events

- Pros
  - Messaging between DLLs and LabVIEW
  - Broadcast (N:N
- Cons
  - API is not complete (i.e. no Get Event Status)

From LabVIEW R&D:

stos Queue #7

V R&D: I write C++/# so you don't have to.

Members

2,852 posts

Location: Austin, TX
Version: LabVIEW 2011
Since: 2000

Posted 01 December 2010 - 01:14 AM

POPULAR

Speaking as the guy who wrote the queue primitives...

The queues are meant to be the primary means of communicating data in LabVIEW between parallel chunks of code. They are completely thread safe, and they participate with LabVIEW's concurrency scheduling algorithms. They're designed to minimize both data copies and latency and have been stable for many years.

User events go through queues that share most of the code with the general queues, although they have a bit more overhead since at any time there might be multiple listeners for an event, which can (not necessarily "will", just "can") cause more data copies. User events are data broadcasters, where as general queues are data point-to-point transmitters. The general queues thus have less overhead, but it is a very small amount less overhead.

For me personally, I prefer the general queues instead of the user events only because of the arcane nodes and special terminals that are required to register dynamic user events. I use user events when I need to have code that sleeps on both UI events and data arrival. But it is really personal preference -- many programmers are successful using user events generally.

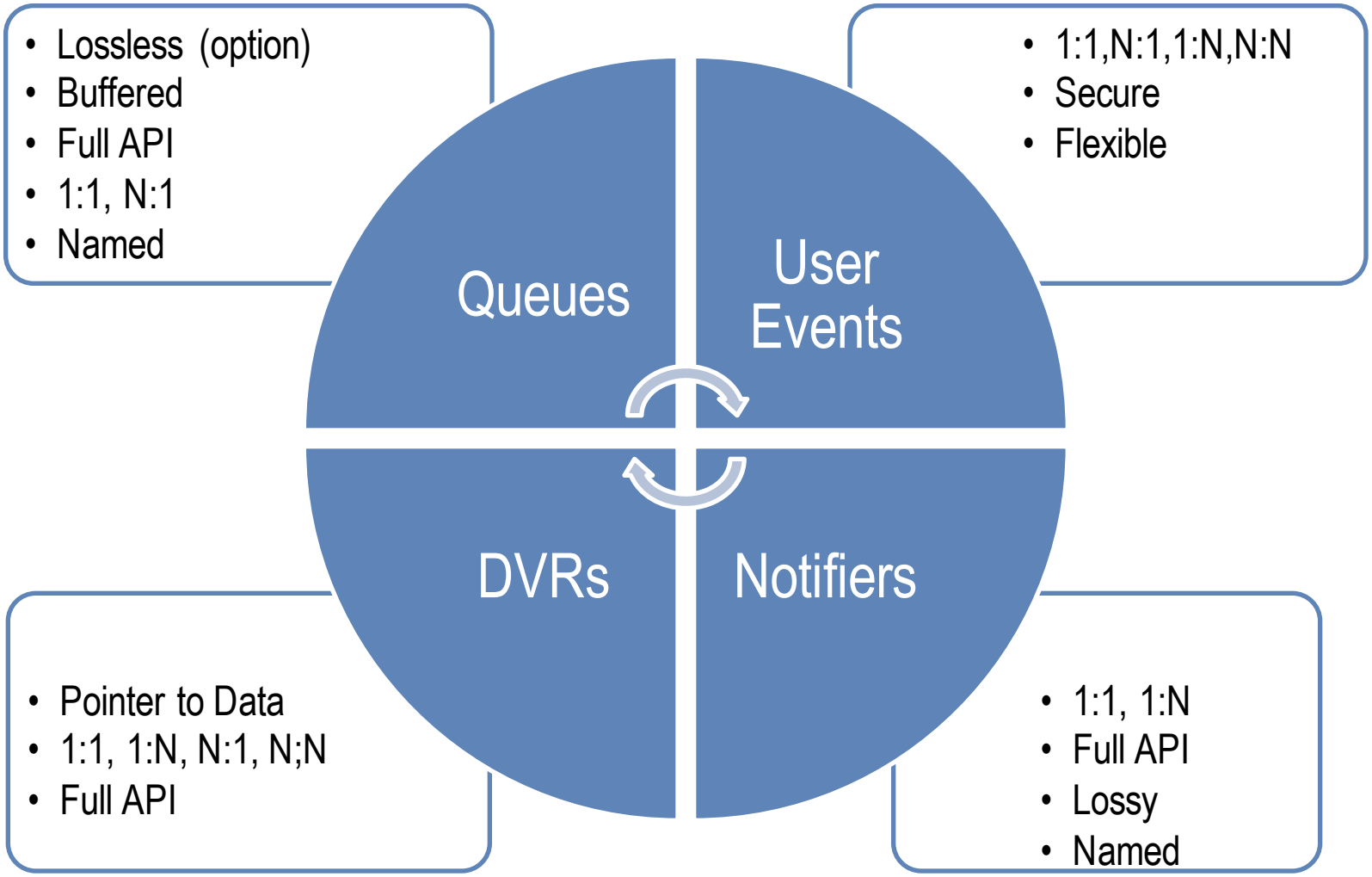
Various Inter-process Communication Methods

	Same target Same application instance	Same target, different application instances OR Different targets on network
Storing - Current Value	• Single-process shared variables • Local and global variables • FGV, SEQ, DVR • CVT • Notifiers (Get Notifier)	• Network-published shared variables (single-element) • CCC
Sending Message	• Queues (N:1) • User events (N:N) • Notifiers (1:N) • User Events	• TCP, UDP • Network Streams (1:1) • AMC (N:1) • STM (1:1)
Streaming	• Queues	• Network Streams • TCP

Example Native LabVIEW APIs

and more (RT, FPGA, etc)…

Foundational Native LabVIEW APIs for Messaging



Questions? Comments?

CONTACT INFO

terry@aleconsultants.com

www.aleconsultants.com

THANK YOU

DHPC Technologies – hosting

National Instruments – food & refreshments