An Overview of

Data Communication in LabVIEW

Brett Coetzee – Application Engineer

Certified LabVIEW Developer (CLD)
Data Communication Options in LabVIEW

1. TCP and UDP
2. Network Streams
3. Shared Variables
4. DMAs
5. Web Services
6. Peer-to-Peer Streaming
7. Queues
8. Dynamic Events
9. Functional Global Variables
10. RT FIFOs
11. Datasocket
12. Local Variables
13. Programmatic Front Panel Interface
14. Target-scoped FIFOs
15. Notifier
16. Simple TCP/IP Messaging (STM)
17. AMC
18. HTTP
19. FTP
20. Global variables

... just to name a few ...
Agenda

- Introduction of Data Communication
- Define Communication Types
- Identify Scope of Communication
  - Inter-process
  - Inter-target
- Next Steps

ni.com/largeapps
Demonstration
The pitfalls of local variables
Common Pitfalls of Data Communication

Race conditions- two requests made to the same shared resource

Deadlock- two or more depended processes are waiting for each other to release the same resource

Data loss- gaps or discontinuities when transferring data

Performance degradation- poor processing speed due to dependencies on shared resources

Buffer overflows- writing to a buffer faster than it is read from the buffer

Stale data- reading the same data point more than once
Scope of Communication

**Inter-process:** the exchange of data takes place within a single application context

**Inter-target:** communication between multiple physical targets, often over a network layer
Defining Inter-process Communication

- Communication on same PC or Target
- Communicate between parallel processes or loops
- Offload data logging or processing to another CPU/Core/Thread within same VI/executable
- Loops can vary in processing priority
- Used to communicate synchronously and asynchronously
Inter-process Communication Options

**Shared Variables**
Update GUI loop with latest value

**Queues**
Stream continuous data between loops on a non-deterministic target

**Dynamic Events**
Register Dynamic Events to execute sections of code

**Functional Global Variables (FGV)**
Use a non-reentrant subVI to protect critical data

**RT FIFOs**
Stream continuous data between time critical loops on a single RT target
Basic Actions

- Set the value of the shift register
Basic Actions

- Get the value currently stored in the shift register
Action Engine

- Perform an operation upon stored value and save result
- You can also output the new value
How It Works

1. Functional Global Variable is a **Non-Reentrant** SubVI
2. Actions can be performed upon data
3. Enumerator selects action
4. Stores result in uninitialised shift register
5. Loop only executes once
Benefits: Comparison

**Functional Global Variables**
- Prevent race conditions
- No copies of data
- Can behave like action engines
- Can handle error wires
- Take time to make

**Global and Local Variables**
- Can cause race conditions
- Create copies of data in memory
- Cannot perform actions on data
- Cannot handle error wires
- Drag and drop
While Loop

- Acquire: 10ms
- Analyse: 50ms
- Log: 250ms
- Present: 20ms

Doing Everything in One Loop Can Cause Problems

- One cycle takes at least 330 ms
- If the acquisition is reading from a buffer, it may fill up
- User interface can only be updated every 330 ms
Doing Everything in One Loop Can Cause Problems

- One cycle still takes at least 310 ms
- If the acquisition is reading from a buffer, it may fill up
- User interface can only be updated every 310 ms
Inter-Process Communication: ensures tasks run asynchronously and efficiently

How?

- Loops are running independently
- User interface can be updated every 20 ms
- Acquisition runs every 10 ms, helping to not overflow the buffer
- All while loops run entirely parallel of each other
Producer Consumer

Best Practices
1. One consumer per queue
2. Keep at least one reference to a named queue available at any time
3. Consumers can be their own producers
4. Do not use variables

Considerations
1. How do you stop all loops?
2. What data should the queue send?
LabVIEW FIFOs

- Queues
- RT FIFOs
- Network Streams
- DMAs
- User Events

In general, FIFOs are good if you need lossless communication that preserves historical information.
Queues

Adding Elements to the Queue

Select the data type the queue will hold

Reference to existing queue in memory

Dequeueing Elements

Dequeue will wait for data or time-out (defaults to -1)
Demonstration
Introduction to LabVIEW Queues
The Anatomy of Dynamic Events

- VI Gets Run on Event
- Defines Data Type
- Data Sent
- Dynamic Events Terminal
- Multiple Loops Can Register for Same Event
Using User Events

LabVIEW API for Managing User Events

Register User Events with Listeners
RT FIFOs vs. Queues

- Queues can handle string, variant, and other variable size data types, while RT FIFOs cannot.
- RT FIFOs are pre-determined in size, queues can grow as elements are added to them.
- Queues use blocking calls when reading/writing to a shared resource, RT FIFOs do not.
- RT FIFOs do not handle errors, but can produce and propagate them.

Key Takeaway:
RT FIFOs are more deterministic for the above reasons.
What is Determinism?

**Determinism:** An application (or critical piece of an application) that runs on a hard real-time operating system is referred to as deterministic if its timing can be guaranteed within a certain margin of error.
LabVIEW Real-Time Hardware Targets

- CompactRIO
- PXI
- Desktop or Industrial PC
- Vision Systems
- Single-Board RIO
RT FIFOs

Write Data to the RT FIFO

Select the data type the RT FIFO will hold

Reference to existing RT FIFO in memory

Read Data from the RT FIFO

Read/Write wait for data or time-out (defaults to 0)
Write can overwrite data on a timeout condition
Defining Inter-target Communication

- PC, RT, FPGA, Mobile Device
- Offload data logging and data processing to another target
- Multi-target/device application
- Network based
Common Network Transfer Policies

“Latest Value” or “Network Publishing”

• Making the current value of a data item available on the network to one or many clients

• Examples
  – I/O variables publishing to an HMI for monitoring
  – Logging temperature values on a remote PC

• Values persist until over written by a new value

• Lossy – client only cares about the latest value
# Latest Value Communication

<table>
<thead>
<tr>
<th>API</th>
<th>Type</th>
<th>Performance</th>
<th>Ease of Use</th>
<th>Supported Configurations</th>
<th>3rd Party APIs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Variable*</td>
<td>LabVIEW Feature</td>
<td><img src="1" alt="Performance" /></td>
<td><img src="1" alt="Ease of Use" /></td>
<td>1:1, 1:N, N:1</td>
<td>Yes (TCP/IP)</td>
</tr>
<tr>
<td>CCC (CVT)</td>
<td>Ref. Arch. <em>Publishes the CVT</em></td>
<td><img src="1" alt="Performance" /></td>
<td><img src="1" alt="Ease of Use" /></td>
<td>1:1</td>
<td>Yes (TCP/IP)</td>
</tr>
<tr>
<td>UDP</td>
<td>LabVIEW Prim.</td>
<td><img src="1" alt="Performance" /></td>
<td><img src="1" alt="Ease of Use" /></td>
<td>1:1, 1:N, N:1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Network buffering should be disabled*
Using Shared Variables Effectively

Programming Best Practices:
• Initialise shared variables
• Serialise shared variable execution
• Avoid reading stale shared variable data
Common Network Transfer Policies

“Streaming”

• Sending a lossless stream of information
• Examples
  – Offloading waveform data from cRIO to remote PC for intensive processing
  – Sending waveform data over the network for remote storage
• Values don’t persist (reads are destructive)
• Lossless – client must receive all of the data
• High-throughput required (latency not important)
Streaming Lossless Data

<table>
<thead>
<tr>
<th>API</th>
<th>Type</th>
<th>Performance</th>
<th>Ease of Use</th>
<th>Supported Configurations</th>
<th>3rd Party APIs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Streams</td>
<td>LabVIEW Feature</td>
<td>🥶</td>
<td>🥶</td>
<td>1:1</td>
<td>Not this year</td>
</tr>
<tr>
<td>NEW!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STM</td>
<td>Ref. Arch.</td>
<td>🥶</td>
<td>🥶</td>
<td>1:1</td>
<td>Yes (TCP/IP)</td>
</tr>
</tbody>
</table>

What about the shared variable with buffering enabled?

NO!
Pitfalls of Streaming with Variables

- Lack of flow control can result in data loss
- Data may be lost if the TCP/IP connection is dropped
- Data loss does not result in an error, only a warning
Network Streams

- Lossless transfer, even in connection loss*
- Can be tuned for high-throughput (streaming) or low-latency (messaging)
- Unidirectional, P2P, LabVIEW only
- Not deterministic
Network Streams

Writing Elements to the Stream

Select the data type the queue will hold

Reference to reader URL

Reading Elements from Stream

Read will wait for data or time-out (defaults to -1)
Network Streams **NEW!**
Network Streams in Action

Use Streams!
Demonstration

Inter-target Communication Using Network Streams
LabVIEW Web Services

Application Architecture:

LabVIEW Application → LabVIEW Web Service → Client

Request → Response

Sending Requests via URL:

http://localhost/Calculator/Sum/3/6

- Physical Location of Server
- Name of Web Service
- Mapping to a VI
- Terminal Inputs (Optional)
Web Services in LabVIEW

Web Server

Windows and Real-Time
Custom web clients
No runtime engine needed
Standard http protocol
Firewall friendly

Any Client

LabVIEW

Thin Client
Inter-Target Communication Options

TCP/IP and UDP
Define low-level communication protocols to optimise throughput and latency

Shared Variables
Access latest value for a network published variable

Network Streams
Point to Point streaming in LabVIEW with high throughput and minimal coding

Web UI Builder
Create a thin client to communicate with a LabVIEW Web Service

DMAs
Direct memory access between different components of a system