LabVIEW Design Patterns & Reference Architectures
Agenda

• Design Patterns

• Reference architecture and tools for developers
Design Patterns in NI LabVIEW
What Is a Design Pattern?

• Based on LabVIEW code template or framework
• Widely accepted and well known
• Easily recognizable
Successful LabVIEW development Practices

- **Scalable** — Easy to add functionality to a VI without completely redesigning the VI
- **Readable** — Easy to visually inspect the design of a VI and understand its purpose and functionality
- **Maintainable** — Easy to change a VI without affecting the intent of the original VI
Scalable VIs

- Scalable design easily allows modifications to original design

Data Acquisition VI that acquires data from one thermocouple

Data Acquisition VI that acquires data from any number of thermocouples
Readable VIs – Poorly Designed Code
Readable VIs – Well Designed Code

A version of the Stop Light VI that uses an Event Structure. Data is passed from the Event Structure via Notifiers and Queues.

1. Note that the Stop Boolean notifier in the Light Control While Loop performs double duty. It has a 25ms timeout, so no Wait Function is necessary.

2. The use of a Variant is notable in that only attributes are passed. This provides a degree of freedom in not requiring data to be predefined.

Event Handler While Loop

South Sensor

North Sensor

West Sensor

East Sensor

Send a Boolean TRUE to the Light Control While Loop
Maintainable VIs

• Allow you to add new features without completely rewriting the application
• Use good program design and architecture
• Are designed with the understanding that other programmers may need to use and modify your code
Getting Started: How Do I Choose?

• Identify the most important aspect of your application:
  ▪ Processes that require decoupling
  ▪ Clean, easy-to-read code
  ▪ Mission-critical components

• Select a template based on potential to improve
Event-Driven User Interface

I am polling for user actions, which is slowing my application down, and sometimes I do not detect them.
Background

Procedural-driven programming
• Performs a set of instructions in sequence
• Requires polling to capture events
• Cannot determine order of multiple events

Event-driven programming
• Determines execution at run time
• Waits for events to occur without consuming CPU
• Remembers order of multiple events
Event-Driven Simple Event

DEMO
Recommendations

Use Cases

• UI: Conserve CPU usage
• UI: Ensure you never miss an event
• Drive slave processes

Considerations

• Event structures eliminate determinism
• Avoid placing two Event structures in one loop
• Remember to read the terminal of a latched Boolean control in its Value Change event case
I have two processes that need to execute at the same time. I need to make sure one cannot slow the other down.
C. Multiple Loop Design Patterns

- Parallel Loop
How It Works

• Master loop tells one or more slave loops when they can run
• Allows for asynchronous execution of loops
• Data independence breaks data flow and permits multithreading
• Decouples processes
Loop Communication

- Variables
- Occurrences
- Notifier
- Queues
- Semaphores
- Rendezvous
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)
Caution

You can needlessly complicate your application if you use an unnecessarily complex design pattern.

Do not forget the most common design pattern of all – **data flow!**
Find LabVIEW Add-ons
Download
Questions