



DMC

Smart People. Expert Solutions.®

LabVIEW User Group Meeting

New York Metro Area (Farmingdale)

2015-12-03

Agenda

1. DMC
2. Design Patterns – What? Why?
3. Basic Tools
4. Simple Patterns
5. Reference Architectures

Company Overview





Established in 1996, offices in New York,
Boston,
Chicago, Denver and Houston

75+

employees &
growing

Industries Served:

Automotive

Bio-medical

Chemical and Food Processing

Defense

Electronics/Semiconductor

Fuel Cells/Alternative Energy

Hydraulics

Laboratory Testing

Machine Tool

Material Handling

Medical Devices

Packaging

Pharmaceutical

Printing & Textiles

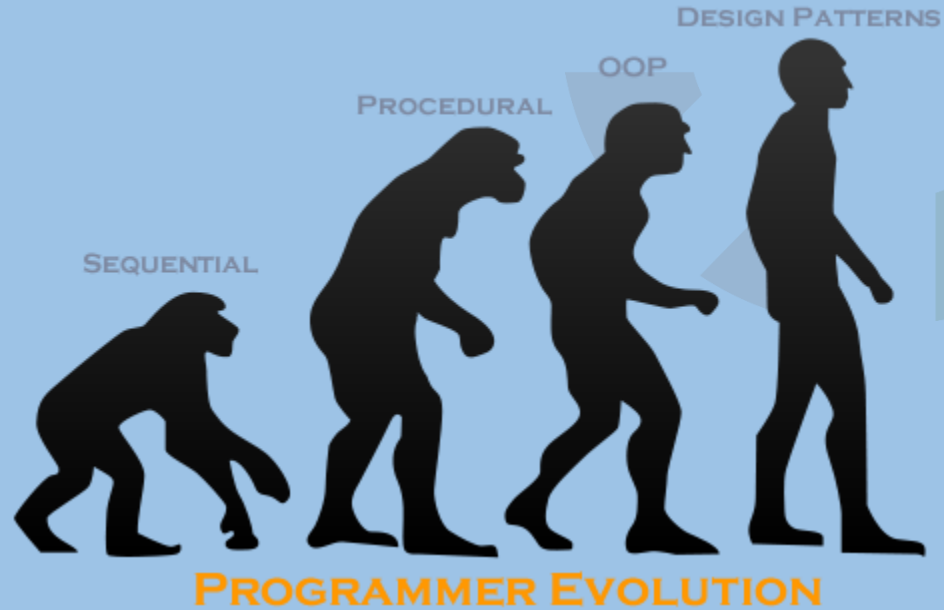


Smart People. Expert Solutions.®

Certifications



DESIGN PATTERNS



Smart People. Expert Solutions.®

Why should I use design patterns?

- Save development time
- Improve Modularity
- Increase Readability
- Take advantage of proven code/architecture
- SMORES



SMoRES – criteria for a well designed application

Scalable: extending to N+1 should be simple

Modular: the application is broken into well-defined components that can stand on their own

Reusable: the code structured in a way that it could be reused in different applications

Extensible: new features can be added easily

Simple: simplest solution that meets all of the requirements

SMoRES – criteria for a well designed application

Scalable: extending to N+1 should be simple

Modular: the application is broken into well-defined components that can stand on their own

Reusable: the code structured in a way that it could be reused in different applications

Extensible: new features can be added easily

Simple: simplest solution that meets all of the requirements

SMoRES – criteria for a well designed application

Scalable: extending to N+1 should be simple

Modular: the application is broken into well-defined components that can stand on their own

Reusable: the code structured in a way that it could be reused in different applications

Extensible: new features can be added easily

Simple: simplest solution that meets all of the requirements

SMoRES – criteria for a well designed application

Scalable: extending to N+1 should be simple

Modular: the application is broken into well-defined components that can stand on their own

Reusable: the code structured in a way that it could be reused in different applications

Extensible: new features can be added easily

Simple: simplest solution that meets all of the requirements

SMoRES – criteria for a well designed application

Scalable: extending to N+1 should be simple

Modular: the application is broken into well-defined components that can stand on their own

Reusable: the code structured in a way that it could be reused in different applications

Extensible: new features can be added easily

Simple: simplest solution that meets all of the requirements

Basic Tools

- For/While Loops
- Shift Registers
- Enums
- Case Structures
- Event Structures
- Queues



Smart People. Expert Solutions.®

Simple Patterns

- Functional Global Variables
- State Machine
- Event Driven User Interface



Smart People. Expert Solutions.®

Functional Global Variables

Need: share data across a large application

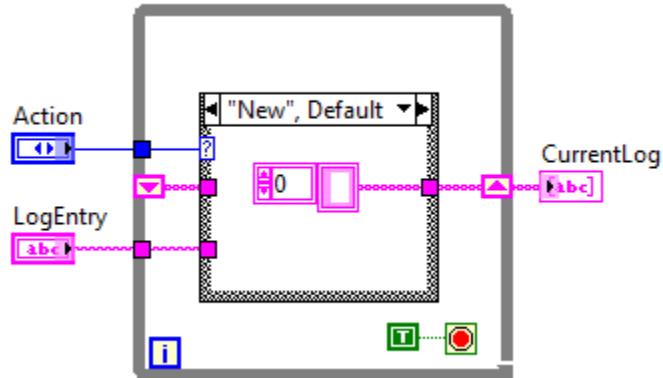
- Local Variables?
- Global Variables?



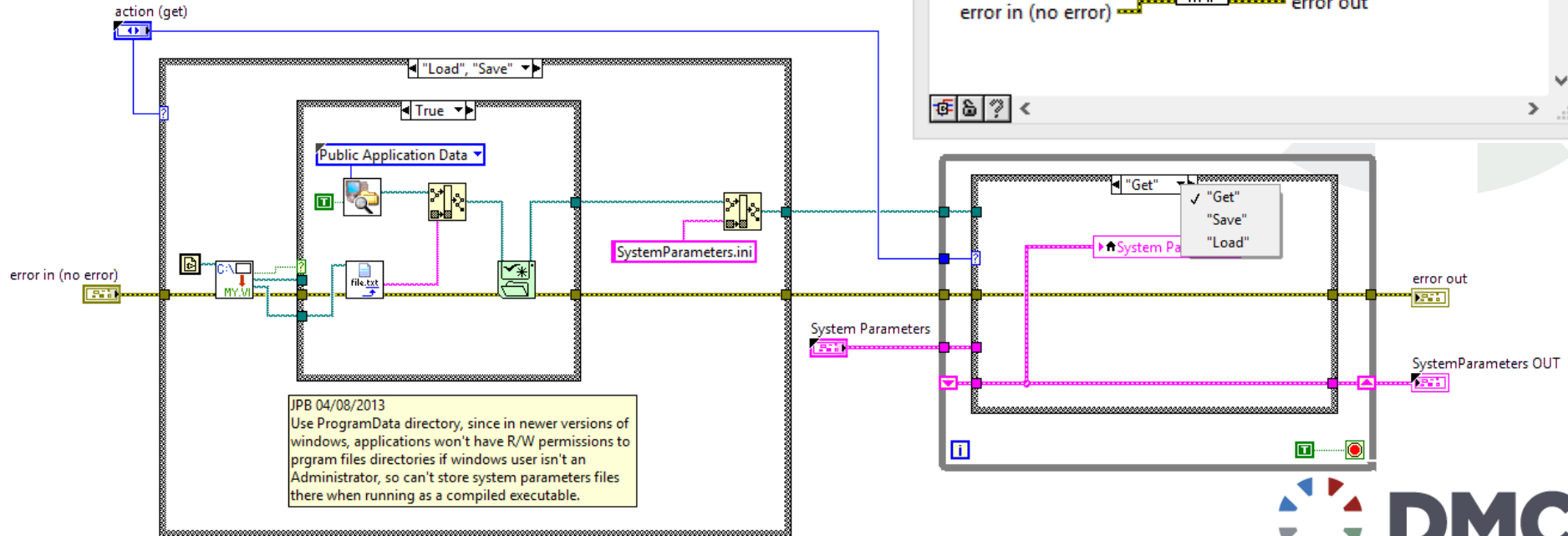
Functional Global Variables

How does it work?

1. Functional Global is a Non-Reentrant SubVI
2. Actions can be performed on data
3. Enumerator (input) selects action, case structure action engine
4. Stores data in un-initialed shift register
5. Loop only executes once

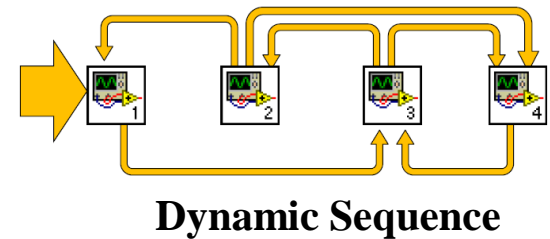
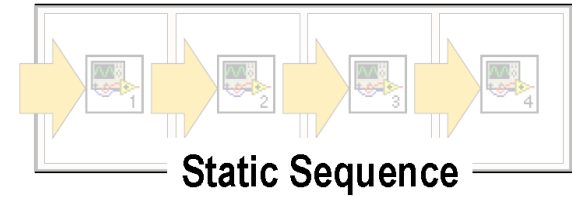
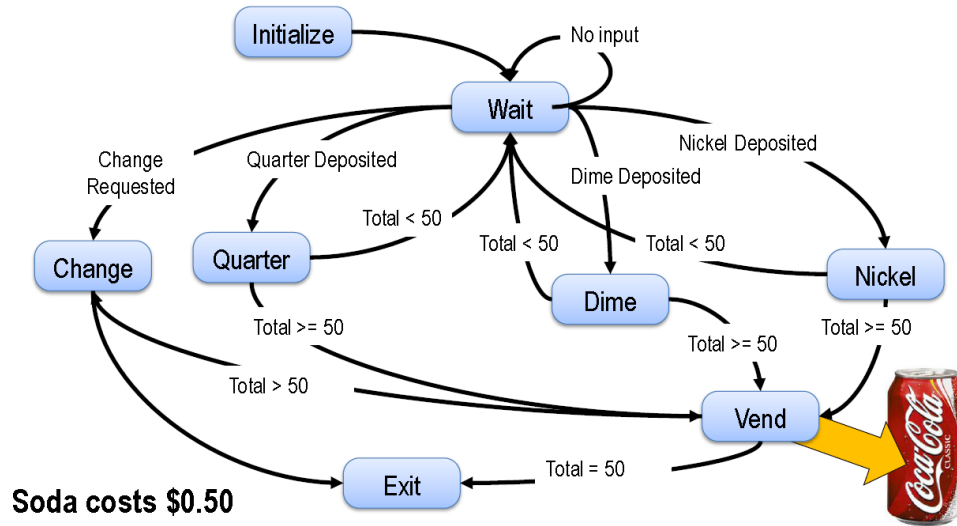


Functional Global Variables

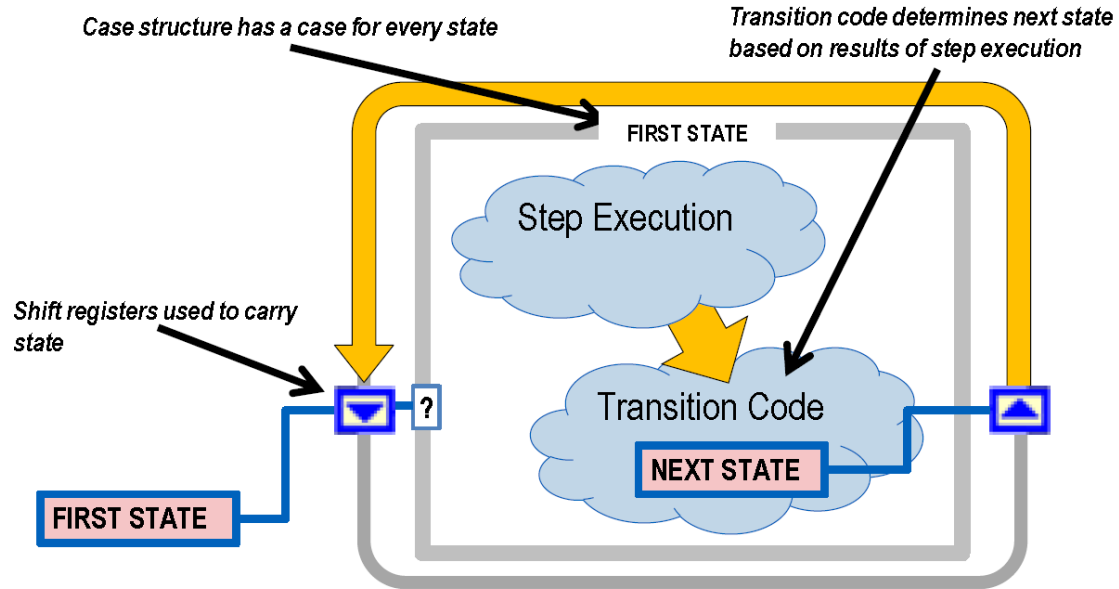


State Machine

Need: execute a sequence of events but the order is determined programmatically (usually user driven)



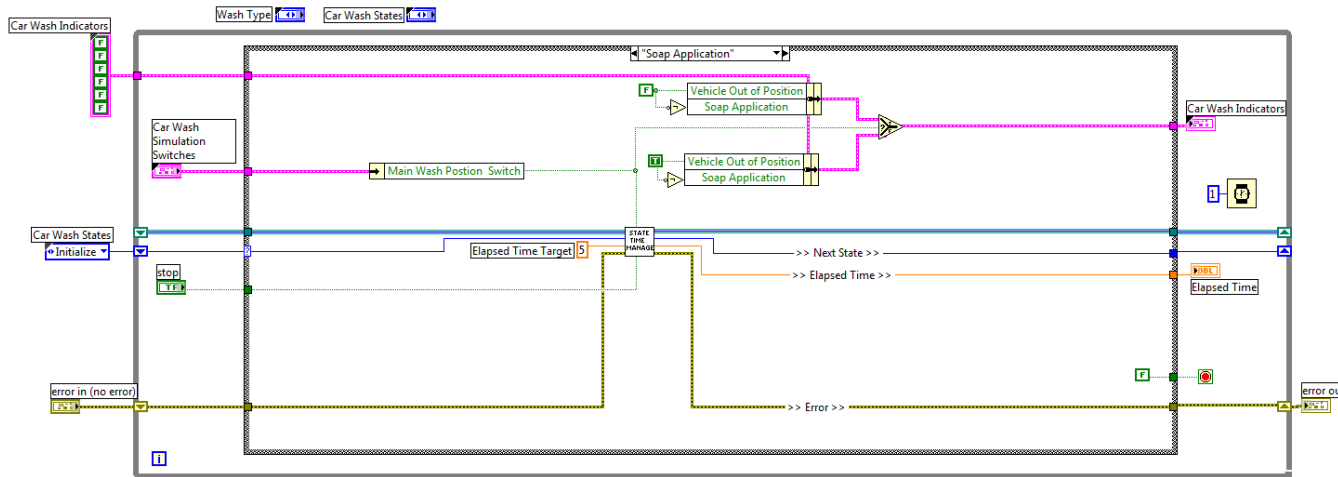
State Machine



State Machine

How does it work?

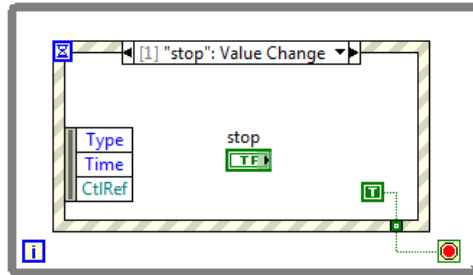
1. Case structure inside of a While-loop
2. Each state in the case structure holds code to be executed in its state
3. Each state has decision making code that determines next state
4. Enumerators (case selector) is used to pass next state via shift registers



Event Driven User Interface

Need: to detect user actions without slowing your application/missing them

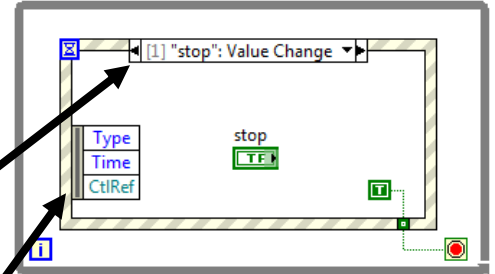
- Event structure within While-loop
- Blocking function until event is registered or timeout
- Event structure configured to choose which events are registered (button presses, mouse click, etc)



Event Driven User Interface

How does it work?

1. Operating system broadcasts system events (mouse click, key press, button click) to applications
2. Registered events are captured by event structure and executes appropriate case
3. Event structure returns information about events to case
4. Event structure enqueues events that occur while it's busy

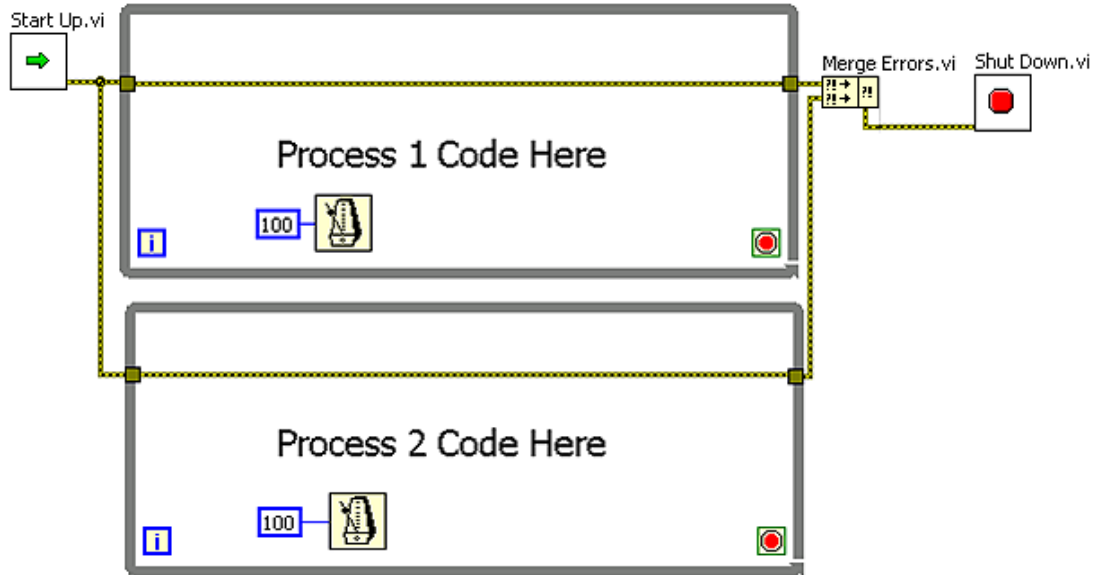


Reference Architectures

- Producer/Consumer or Master/Slave
- Queued State Machine & Event-driven User Interface
- Daemon
- SEA Monster
- ??? (Yours!)

Producer/Consumer

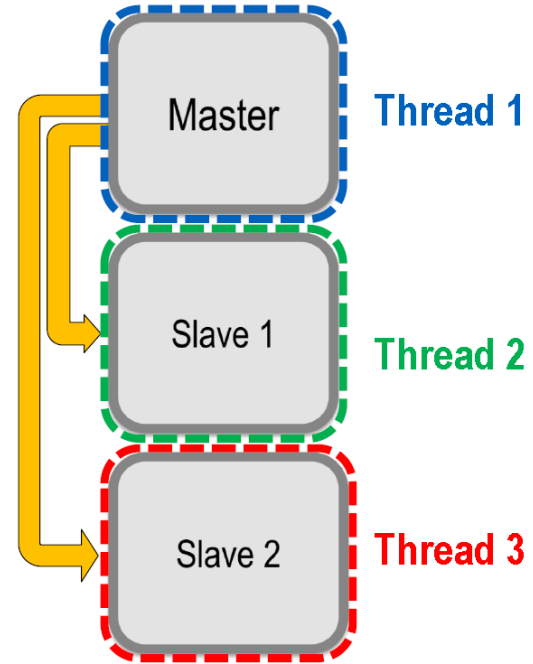
Need: to execute code in parallel and communicate between them



Producer/Consumer

How it works:

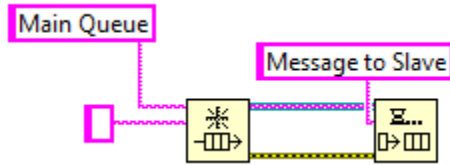
1. Master loop with one or more slave loops
2. Master loop controls execution of slaves
3. Allows for asynchronous execution of loops
4. Decouples processes to allow multi-threading
5. Communication between loops



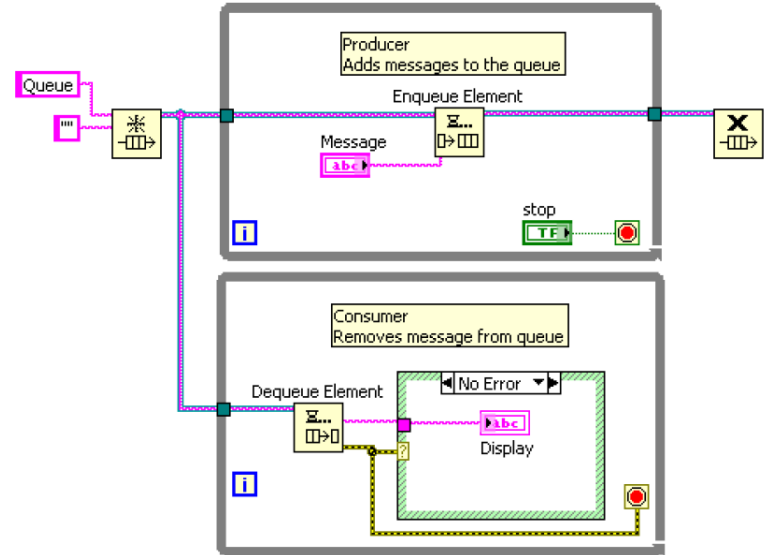
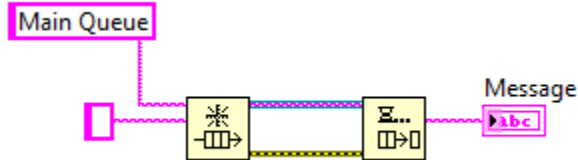
Producer/Consumer

Queue-based communication

- Adding elements to the queue



- Dequeueing elements from the queue



Queued State Machine & Event-Driven Producer/Consumer

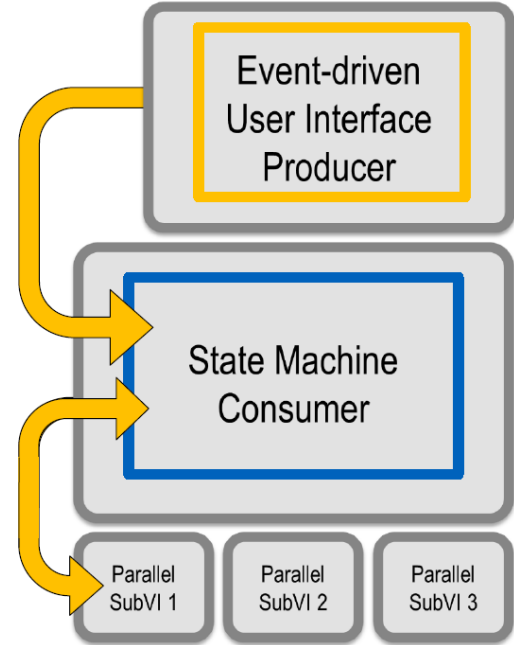
Need: to enqueue events from a user that control the sequence of events in a state machine

- Event-driven user interface as a producer loop
- State machine as a consumer loop
- Communication between the loops using an event queue

Queued State Machine & Event Driven Producer/Consumer

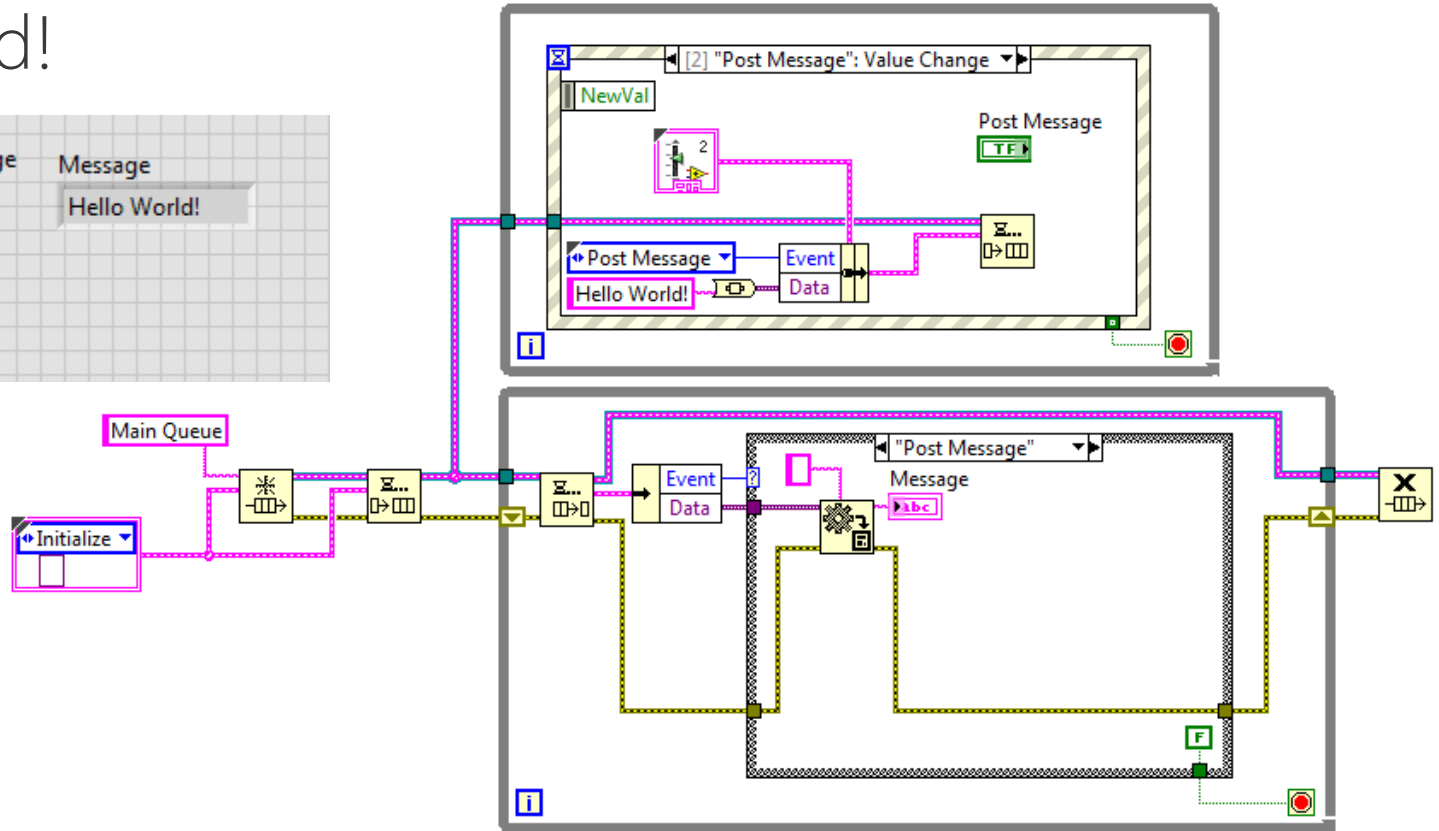
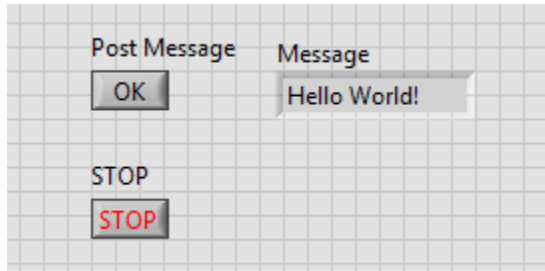
How it works:

1. Events are captured by producer loop
2. Producer places data on the queue
3. Consumer loop dequeues data
4. State machine in consumer loop executes on data
5. Parallel processes communicate with state machine using queue references



Queued State Machine & Event Driven Producer/Consumer

Hello World!



Daemon

What is a daemon?

- Used to create and launch applications that run invisibly in the background
 - Auto-save
 - TCP/UDP messaging
 - Garbage collection of temp files
- Perform low-priority monitoring and/or maintenance or communication based processes

Daemon

Self-launching:

- Daemon must keep an open reference to itself to keep from being purged

Standard Launched:

- Launcher must transfer responsibility for reference to daemon VI
- Launcher must not close reference to daemon



DMC SEA Monster

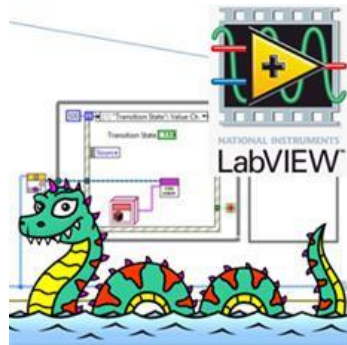
Need: a very powerful, flexible, prebuilt/proven architecture to speed up development time for complex applications

Solution: internally developed (over the last 15 years) architecture based on a queued state machine & event-driven user interface producer/consumer model.

S – States

E – Events

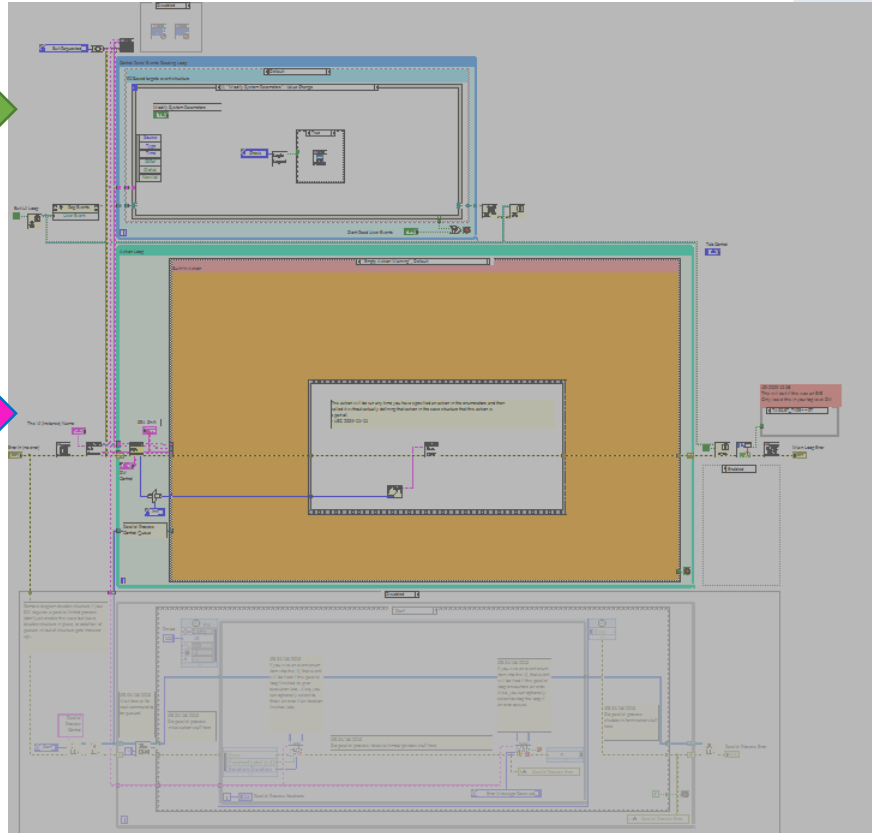
A – Actions



DMC SEA Monster

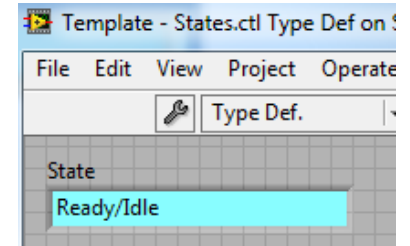
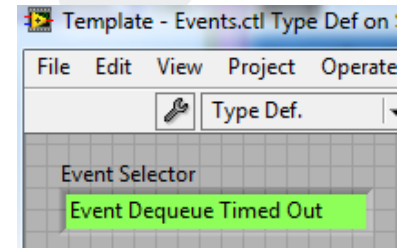
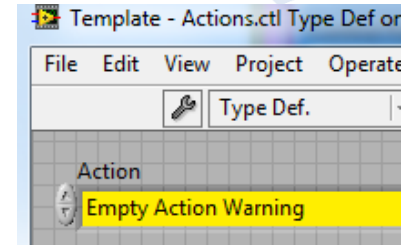
Event Scheduler
Lives Here

Event Scheduler
asynchronously “injects”
events into Event Queue



DMC SEA Monster

	StateA	StateB	StateC
Event1	Actions: ActX, ActY	Actions: ActZ Next State: StateC	
Event2			Actions: ActN, ActW, ActV
Event3	Actions: ActZ, ActX		Actions: ActV, ActY
Event4	Actions: ActM, ActN Next State: StateB	Actions: ActN	
Event5		Actions: ActJ, ActK, ActL	Actions: ActK, ActZ Next State: StateA
Event6		Actions: ActZ, ActY	
Event7	Actions: ActK, ActM Next State: StateC		Actions: ActW, ActX



DMC SEA Monster Logic Editor

The screenshot displays the DMC SEA Monster Logic Editor interface. At the top, there are tabs for 'SEA Misc Info (Read Only)', 'State Matrix (Treat as Read only)', 'Initialization', 'Messaging', 'Event Matrix', 'On Enter/Exit', 'SEA Viewer', and 'Additional Options'. The 'Event Matrix' tab is active.

The 'Logic File Name' field shows the path: `C:\Projects\DMC_SEA_Monster_Development\Source\State Machines\Template\Template.vi.logic12`.

On the left, a list of 'Events' is shown, with '1 - Reset Error Requested' selected. The list includes:

- 0 - Exit Requested
- 1 - Reset Error Requested
- 2 - Event Dequeue Timed Out
- 3 - Reset Error Message Received
- 4 - Exit Message Received
- 5 - Error Message Received
- 6 - Toggle UI Lock Requested

The main area shows the configuration for the selected event 'Reset Error Requested'. A note at the top right states: 'Note: Right Click on Parametric String to choose from defined parametric actions in "Parametric Cases" structure'.

The configuration includes:

- Applies To:** Selected States (cyan bar)
- State:** Error, Ready/Idle (cyan bars)
- Comment:** Sending to global is optional... (text area)
- Event Scheduling Method:** When Event Occurs - Strict Deadlines
- Period (sec):** 0
- # of Occurrences (-1=infinite):** 0
- Enqueue Method:** Back of Queue
- Actions:** Send to Global [para] (yellow bar), RESET ERROR (yellow bar), Empty Action Warning (yellow bars)
- Next State Mode:** Specific State
- Next state:** Ready/Idle (cyan bar)

At the bottom left, there are buttons for 'Load Matrices', 'Power Scripting' (checked), 'FTP Put', 'Save Matrices', and 'EXIT'.

DMC SEA Monster Logic Editor

The screenshot displays the DMC SEA Monster Logic Editor interface. At the top, there are several tabs: "SEA Misc Info (Read Only)", "State Matrix (Treat as Read only)", "Initialization", "Messaging", "Event Matrix", "On Enter/Exit", "SEA Viewer", and "Additional Options". The "Initialization" tab is currently selected.

Below the tabs, the "Logic File Name" is set to "C:\Projects\DMC\SEA_Monster_Development\Source\State Machines\Template\Template.vi.logic12".

On the left side, there are three sections for state configuration:

- Initial State:** A dropdown menu showing "Ready/Idle" (highlighted in cyan).
- Error State:** A dropdown menu showing "Error" (highlighted in cyan).
- Event Dequeue Timeout:** A dropdown menu showing "Event Dequeue Timed Out" (highlighted in green).

The main area is divided into two sections:

- Initialization Actions:** A list of actions with a scroll bar on the right. The actions are:
 - Set UI [para] → Lock
 - Go To Tab [Tab # or Tab Name] → Main
 - Init →
 - Communication [Start Global, Stop Global, etc] → Start Global
 - Communication [Start Global, Stop Global, etc] → Start TCP Daemon
 - Communication [Start Global, Stop Global, etc] → Start UDP Daemon
 - Global Queue Connection [ADD, REMOVE] → ADD
- Error Actions:** A list of actions with a scroll bar on the right. All actions are "Empty Action Warning".

DMC SEA Monster Logic Editor

The screenshot displays the DMC SEA Monster Logic Editor interface. At the top, there are several tabs: SEA Misc Info (Read Only), State Matrix (Treat as Read Only), Initialization, Messaging, Event Matrix, On Enter/Exit, SEA Viewer, and Additional Options. The main window title is "Logic File Name" with the path "C:\Projects\DMC_SEA_Monster_Development\Source\State Machines\Template\Template.vi.logic12".

The central area is titled "SEA Tree" and contains a table with the following columns: "States, Events, Actions" and "Next State (if different), or Parametric String". The table lists various states and actions, such as "Ready/Idle", "Exit Requested", and "Error".

States, Events, Actions	Next State (if different), or Parametric String
Logic	
Ready/Idle	
On State Enter	
Go To Tab [Tab # or Tab Name]	Main
Panel UpdateByState	
Event Dequeue Timed Out	
Wait [ms]	0
Exit Requested	
Send to Global [para]	EXIT
Communication [Start Global, Stop Global, etc]	Stop Global
Communication [Start Global, Stop Global, etc]	Stop TCP Daemon
Communication [Start Global, Stop Global, etc]	Stop UDP Daemon
Set UI [para]	Unlock
Close	
Exit Message Received	
Set UI [para]	Unlock
Close	
Error Message Received	
Extract [para]	ERROR
Reset Error Requested	
Send to Global [para]	RESET ERROR
Toggle UI Lock Requested	
Set UI [para]	Toggle
Error	
On State Enter	
Send to Global [para]	ERROR
Go To Tab [Tab # or Tab Name]	Error
Display Processed Error	
Panel UpdateByState	
Exit Requested	

On the left side, there are buttons for Refresh, Print Data, Send Report to: (Printer), Search, Home, and Recolor. A search term input field shows "items found: 59".

On the right side, there are three panels: "Events Potentially Fired By Selected Action" (containing "Error Accessing State Machine v"), "Unused States", "Unused Events", and "Unused Actions" (containing "Empty Action Warning", "Set Event Dequeue Timeout [ms]", and "Set Cursor [Busy, Unbusy]"). A note below the first panel reads: "Note: Alt+Double Click on an action to view the code for that action in the state machine".

At the bottom, there are buttons for Load Matrices, Power Scripting (checked), FTP Put, Save Matrices, and EXIT.

??? (Yours!)



References

Too many to list - many readily available NI whitepapers

Questions?



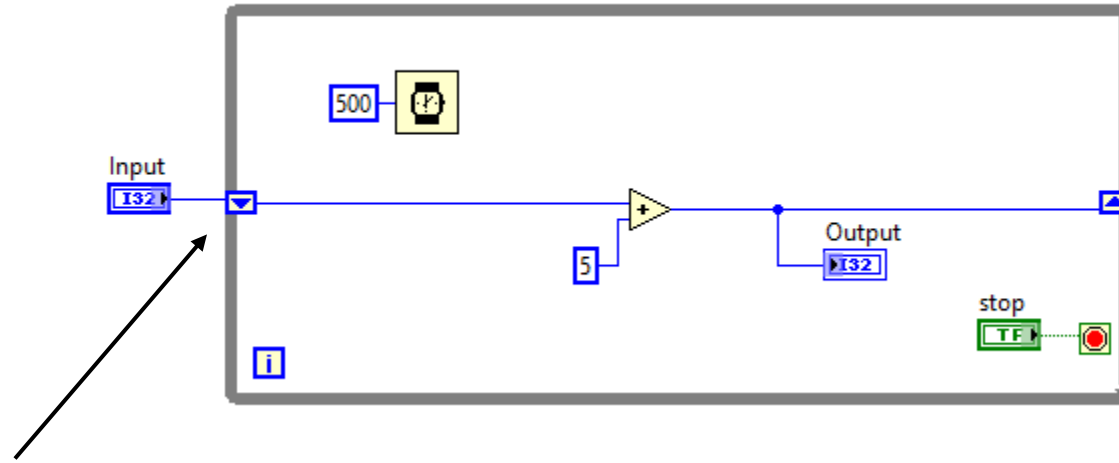
DMC

Smart People. Expert Solutions.®

End Presentation

Appendix - Tools

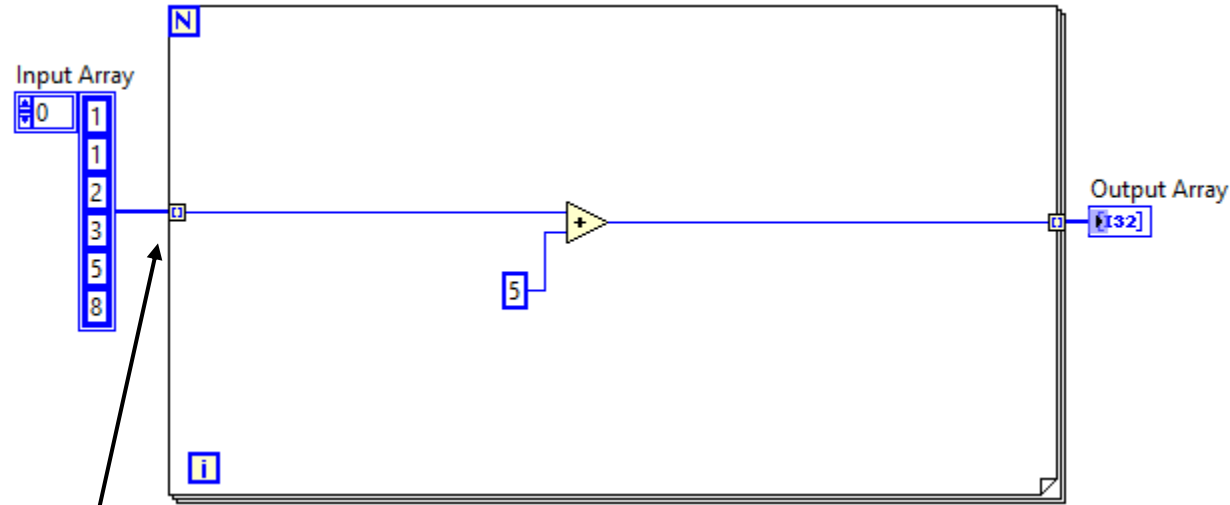
While Loop



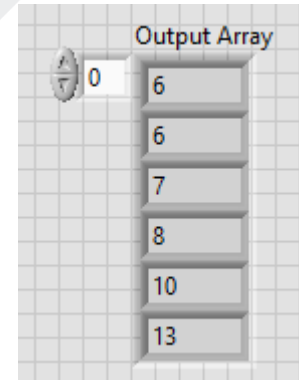
Shift Register

Appendix - Tools

For Loop

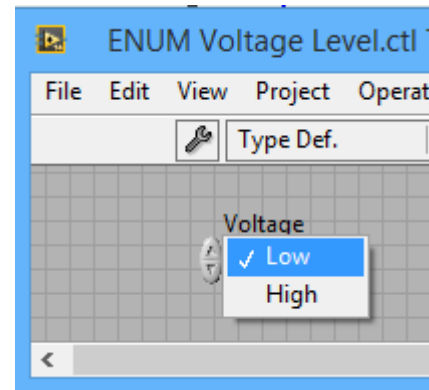
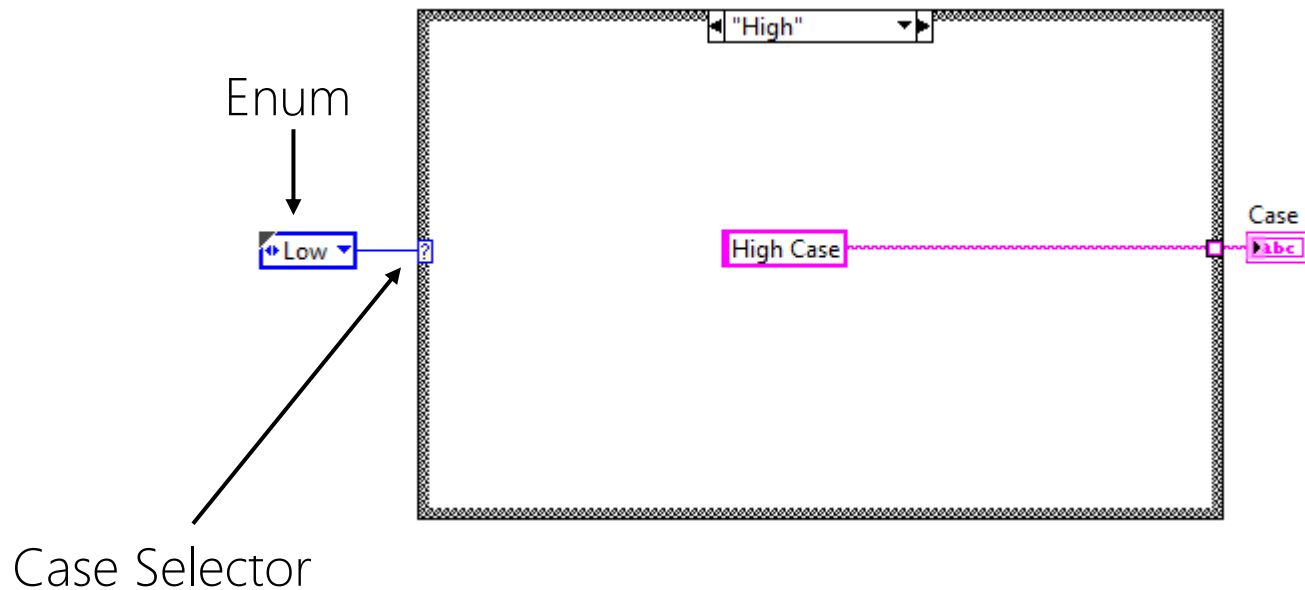


Tunnel



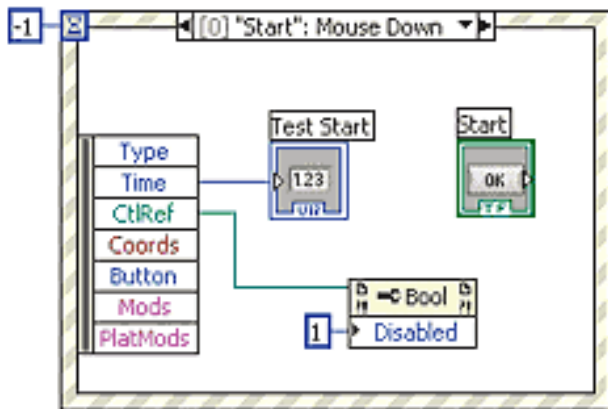
Appendix - Tools

Enums & Case Structures



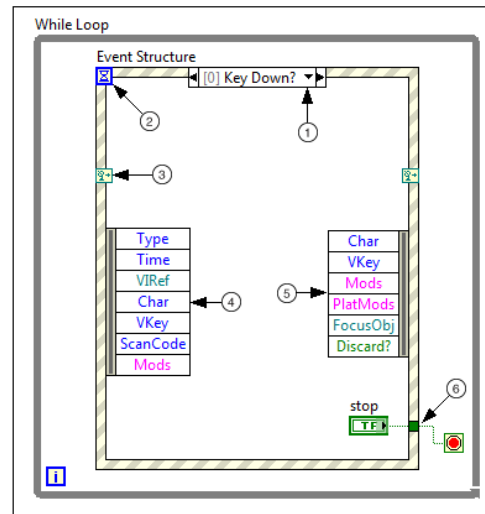
Appendix - Tools


Event Structures



Event Structure Components

The following example shows an Event structure with the Key Down? event case.



- 1 The event selector label specifies which events cause the currently displayed case to execute. To view other event cases, click the down arrow next to the case name.
 - 2 The Timeout terminal specifies the number of milliseconds to wait for an event before timing out. If you wire a value to the Timeout terminal, you must provide a **Timeout** event case to avoid an error.
 - 3 The dynamic event terminals accept an event registration refnum or a cluster of event registration refnums for **dynamic event registration**. If you wire the inside right terminal, that terminal no longer carries the same data as the left terminal. You can wire the event registration refnum or cluster of event registration refnums to the inside right terminal through a **Register For Events** function and **modify the event dynamically**. Depending on the palette from which you select the Event structure, the dynamic event terminals might not appear by default. To display these terminals, right-click the Event structure and select **Show Dynamic Event Terminals** from the shortcut menu.
 - 4 The Event Data Node identifies the data LabVIEW returns when an event occurs. Like the **Unbundle By Name** function, you can resize the node vertically and select the items you need. Use the Event Data Node to access event data elements, such as **Type** and **Time**, which are common to all events. Other event data elements, like **Char** and **VKey** for example, vary based on the event you configure.
-  **Note** For more information about event data elements, click the **Details** link in the event descriptions for the **Control**, **Application**, **Pane**, and **VI** event class topics.
- 5 The Event Filter Node identifies the event data you can modify before the user interface can process that data. This node appears in Event structure cases that handle **filter events**. If you want to change event data, you can wire and modify data items from the Event Data Node to the Event Filter Node. You also can change the event data by wiring new values to the node terminals. To completely discard an event, wire a TRUE value to the **Discard?** terminal. If you do not wire a value to a data item of the Event Filter Node, that data item remains unchanged.
 - 6 Like a **Case** structure, the Event structure supports tunnels. However, by default you do not have to wire Event structure output tunnels in every case. All unwired tunnels use the **default value** for the tunnel data type. Right-click a tunnel and deselect **Use Default If Unwired** from the shortcut menu to revert to the default Case structure behavior where tunnels must be wired in all cases. You also can **configure the tunnels** to wire the input and output tunnels automatically in unwired cases.