CTI
Community Training Initiative

support:	forums.ni.com
add-ons:	vipm.io
resources:	gcentral.org

Understanding events

Version v1.0, 2025-08-15

Table of Contents

Overview

Course Objective

This course provides a practical and in-depth exploration of LabVIEW's Event-driven programming model. You'll learn how to build responsive, maintainable applications using Event Structures, User Events, and best practices for scalable architectures.

Target Audience

- Intermediate LabVIEW developers
- Engineers designing interactive UIs
- Developers transitioning from polling to event-driven architectures

Course Outline

Introduction to Events

Understand the motivation behind LabVIEW's event-driven programming. Compare polling-based and event-based interaction strategies.

Event Sources

List all kind of sources for events an Event Structure

Event Types

Understand the differnece between filtered and non-filtered events.

Memory Management

Understand how LabVIEW handle the memory when it comes to deal with events.

Learning Outcomes

After completing this course, you will be able to:

- Use Event Structures to build responsive UIs
- Register and generate custom User Events
- Troubleshoot and avoid common design pitfalls
- Architect scalable LabVIEW applications using Events

Set-up Instructions

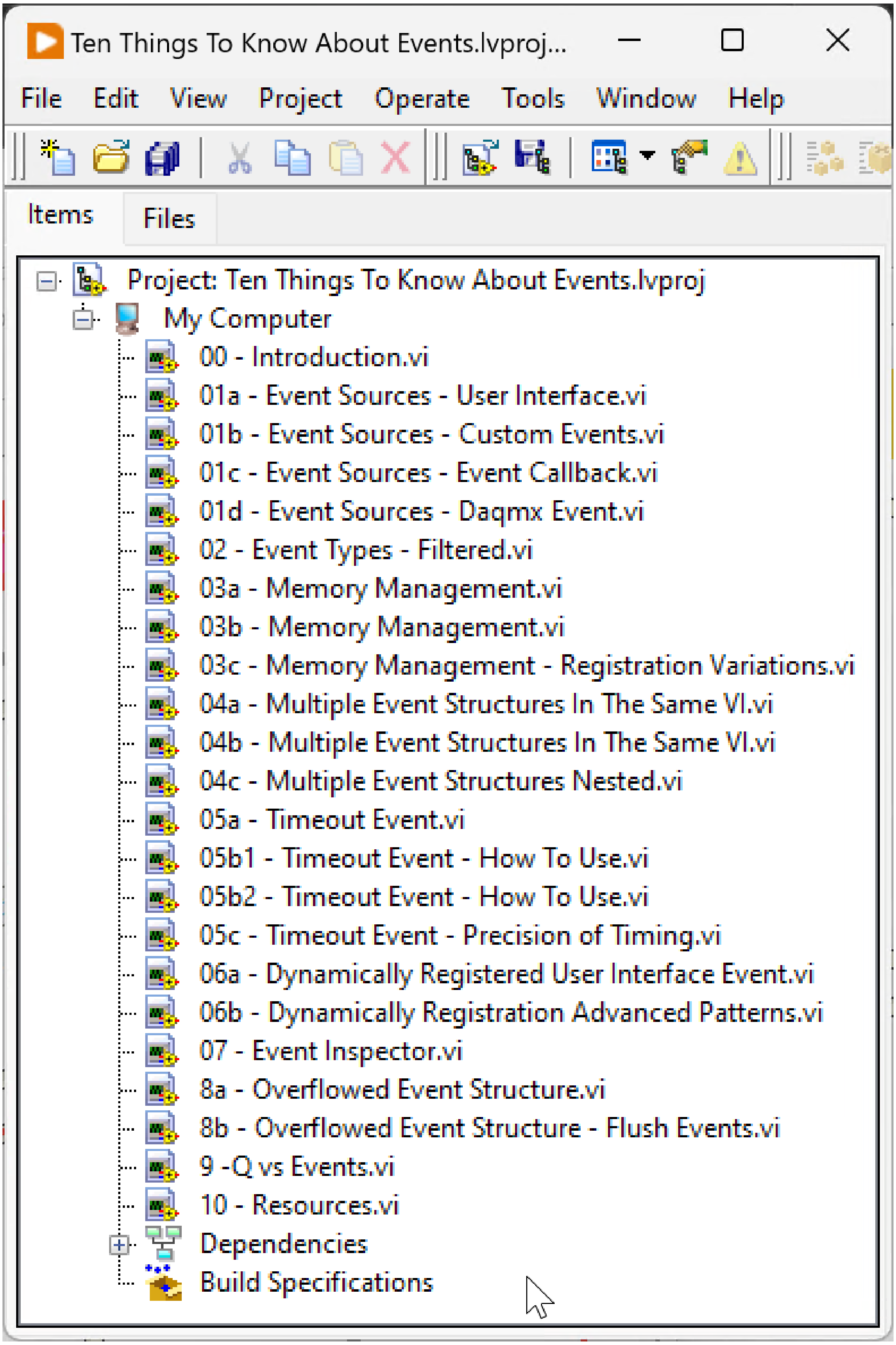
You need LabVIEW 2020 or later to open the examples provided in this course.

If you don’t have access to a LabVIEW™ version, see Set-up LabVIEW™ Virtual Machine Instructions to get a free LabVIEW™ environment running.

Download the course material

- Course material zip

The course material consists in a collection of examples code that helps you understand better how events work in LabVIEW™.



Project: Ten Things To Know About Events.lvproj
My Computer
00 - Introduction.vi
01a - Event Sources - User Interface.vi
01b - Event Sources - Custom Events.vi
01c - Event Sources - Event Callback.vi
01d - Event Sources - Daqmx Event.vi
02 - Event Types - Filtered.vi
03a - Memory Management.vi
03b - Memory Management.vi
03c - Memory Management - Registration Variations.vi
04a - Multiple Event Structures In The Same VI.vi
04b - Multiple Event Structures In The Same VI.vi
04c - Multiple Event Structures Nested.vi
05a - Timeout Event.vi
05b1 - Timeout Event - How To Use.vi
05b2 - Timeout Event - How To Use.vi
05c - Timeout Event - Precision of Timing.vi
06a - Dynamically Registered User Interface Event.vi
06b - Dynamically Registration Advanced Patterns.vi
07 - Event Inspector.vi
8a - Overflowed Event Structure.vi
8b - Overflowed Event Structure - Flush Events.vi
9 -Q vs Events.vi
10 - Resources.vi
Dependencies
Build Specifications

Figure 1. The example project

Video

The fisrt release of these examples was made during the 2022 GLA Summit. Video of the related presentation presenting the examples is available on YouTube.

▶ https://www.youtube.com/watch?v=ddXp1Itpq0M *(YouTube video)*

Introduction to Events

Events were introduced in LabVIEW™ 6.i. Before events, user interaction handling was largely dependent on polling controls inside a while loop, leading to inefficient and CPU-intensive designs. Event-driven programming enables your VI to react only when something important happens, such as a button click or value change, resulting in cleaner, more responsive UIs.

Events are handled with the Event Structure.

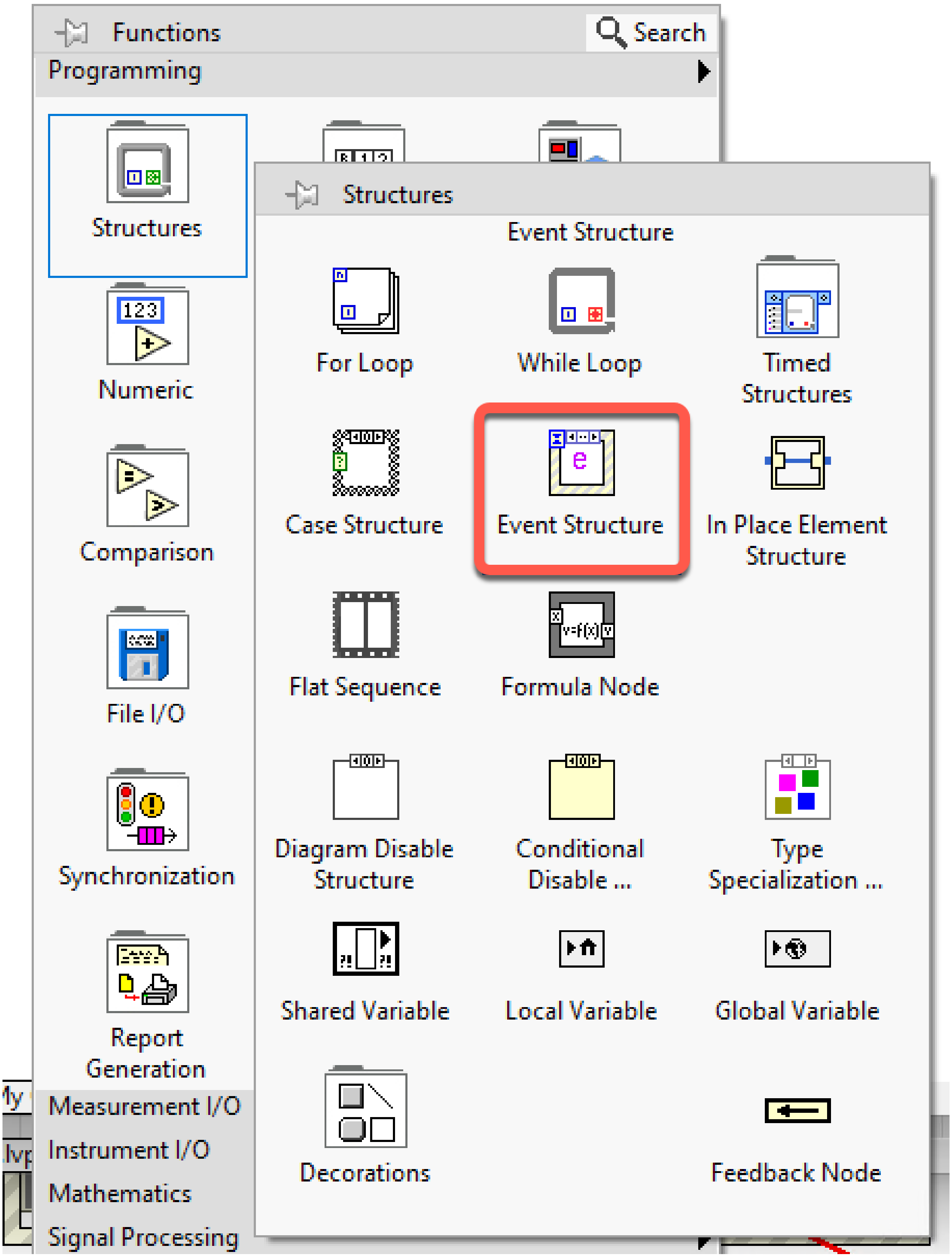


Figure 2. The Event Structure in the function palette

Event Sources

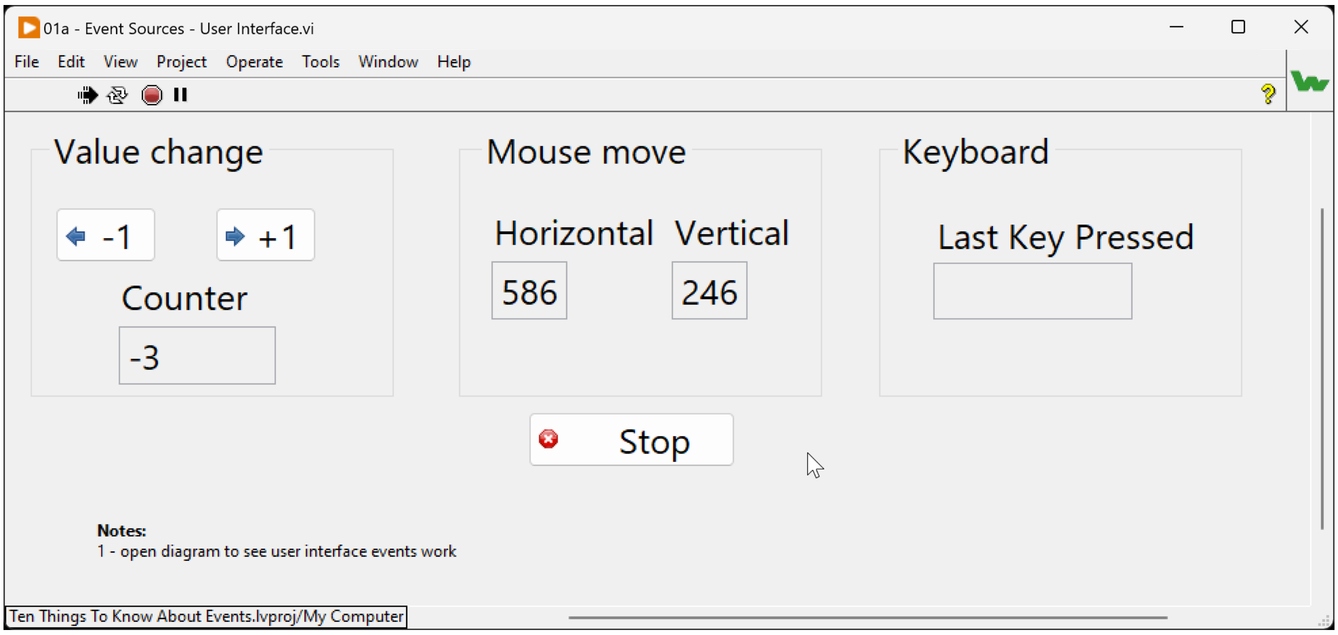
User Interface

The first source of events you come across when starting with LabVIEW™ are the events generated by the UI when a user of your application use the mouse or the keybard.

The Event Structure is the way to detect the event and execute the appropriate caode.

Before the introduction of the Event Structure (a long time ago), LabVIEW developpers had to periodically poll the controls to detect interactions with the User Interface .

Example code: **01a - Event Sources - User Interface.vi**


01a - Event Sources - User Interface.vi
File Edit View Project Operate Tools Window Help
Value change
-1
+1
Counter
-3
Mouse move
Horizontal
Vertical
586
246
Keyboard
Last Key Pressed
Stop
Ten Things To Know About Events.lvproj/My Computer

Notes:
1 - open diagram to see user interface events work

Figure 3. Front Panel

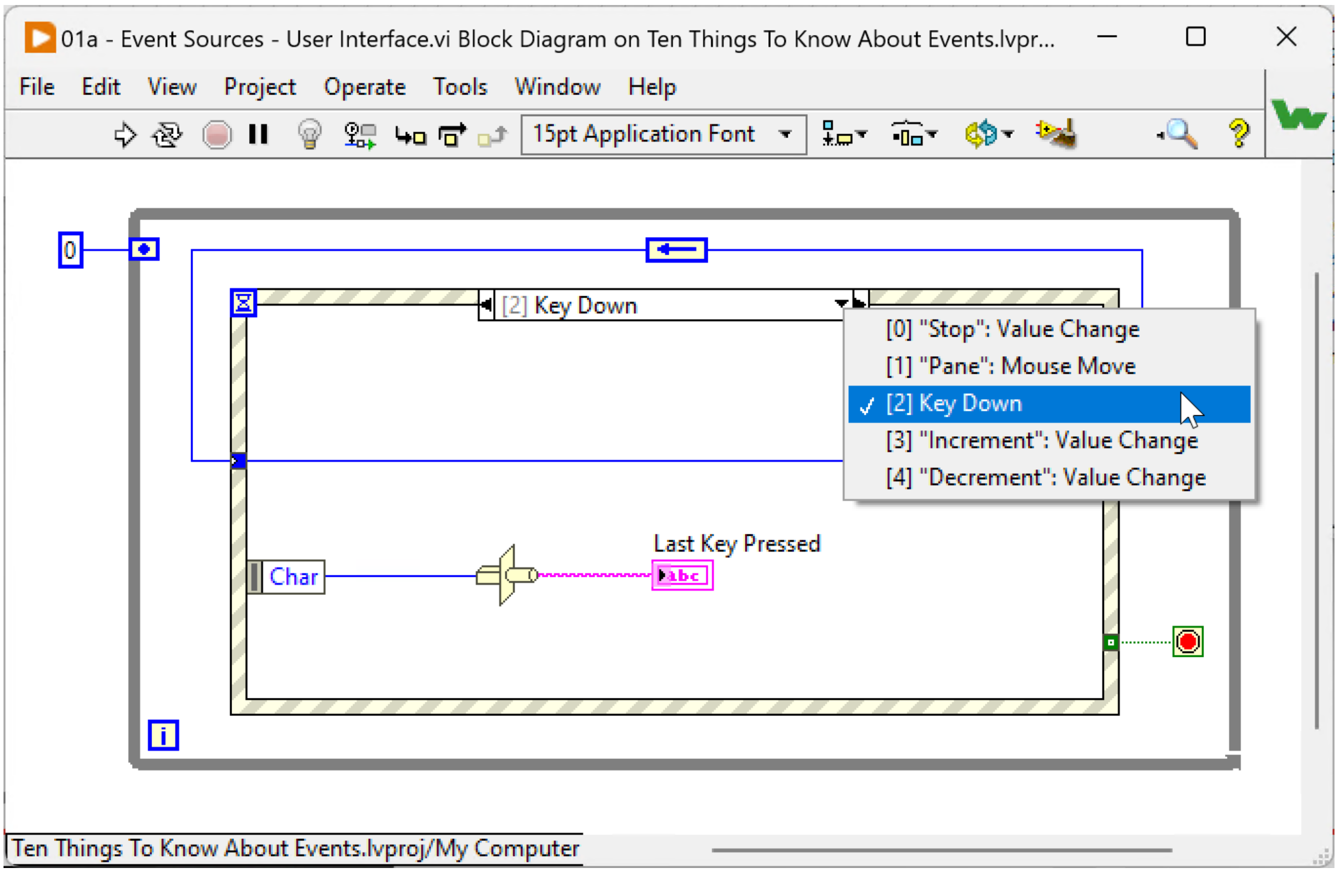

01a - Event Sources - User Interface.vi Block Diagram on Ten Things To Know About Events.lvpr...
File Edit View Project Operate Tools Window Help
15pt Application Font
[2] Key Down
[0] "Stop": Value Change
[1] "Pane": Mouse Move
✓ [2] Key Down
[3] "Increment": Value Change
[4] "Decrement": Value Change
Char
Last Key Pressed
abc
0
i
Ten Things To Know About Events.lvproj/My Computer

Figure 4. Diagram

User Events a.k.a Custom Events

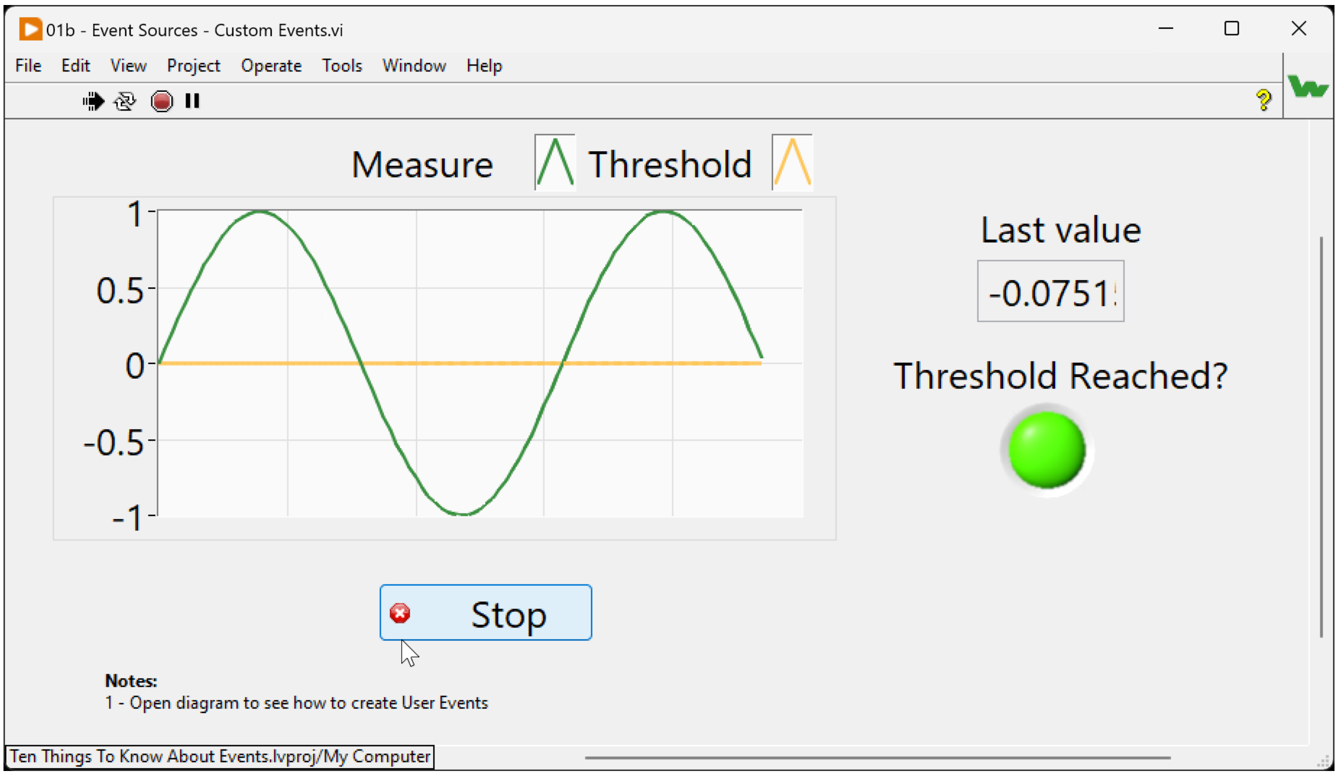
User Events, the worst named feature in LabVIEW™

— John Medland. MLUG October 2020

Behind this name, you'll find the way to create and generate events programmatically.

This source of events is the corner stones of many event driven architectures. The DQMH framework is probably one of the most popular framework based on the LabVIEW™ user event feature.

Example code: **01b - Event Sources - Custom Events.vi**


01b - Event Sources - Custom Events.vi
File Edit View Project Operate Tools Window Help
Measure
Threshold
1
0.5
0
-0.5
-1
Last value
-0.0751
Threshold Reached?
Stop

Notes:
1 - Open diagram to see how to create User Events

Ten Things To Know About Events.lvproj/My Computer

Figure 5. Front Panel

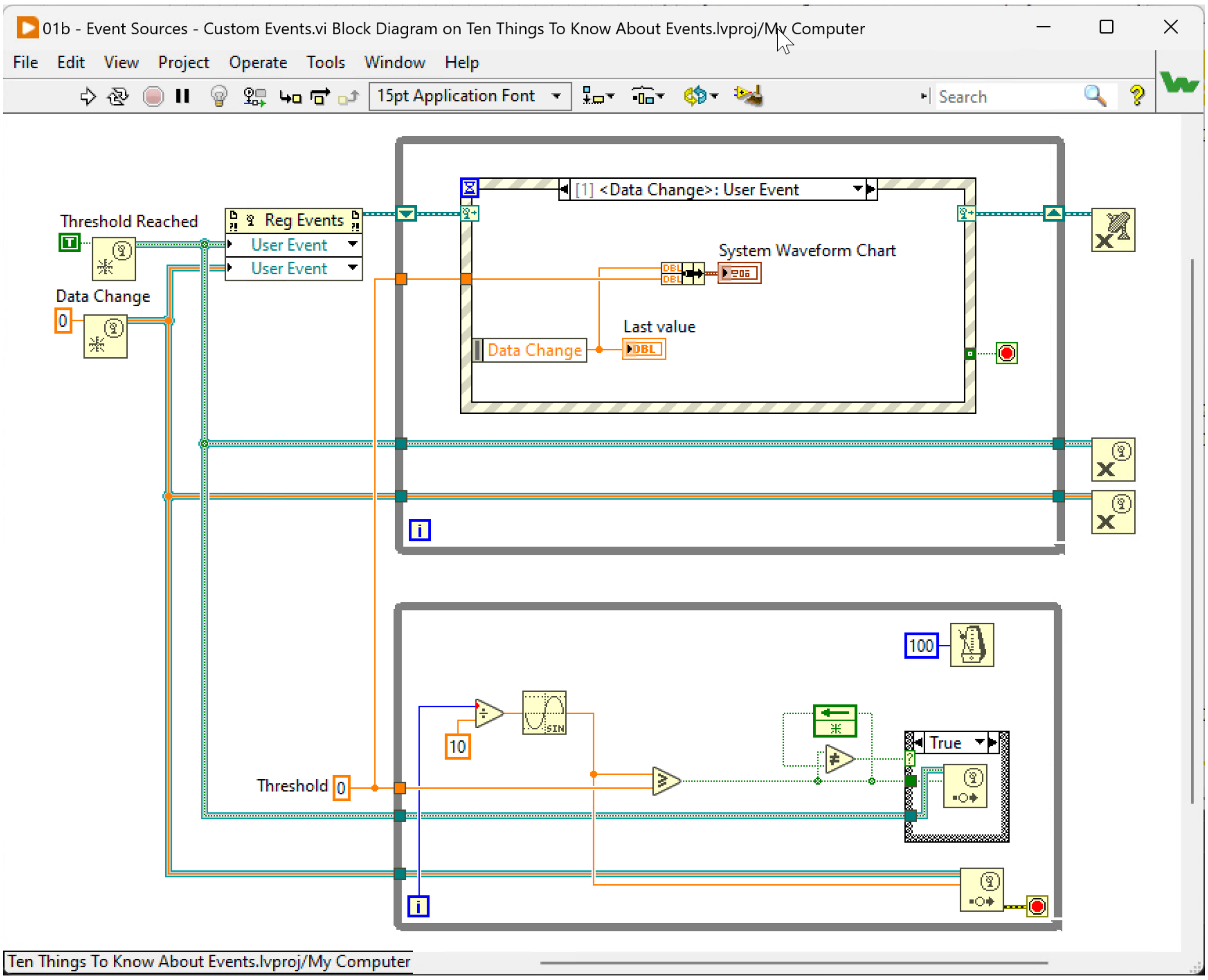


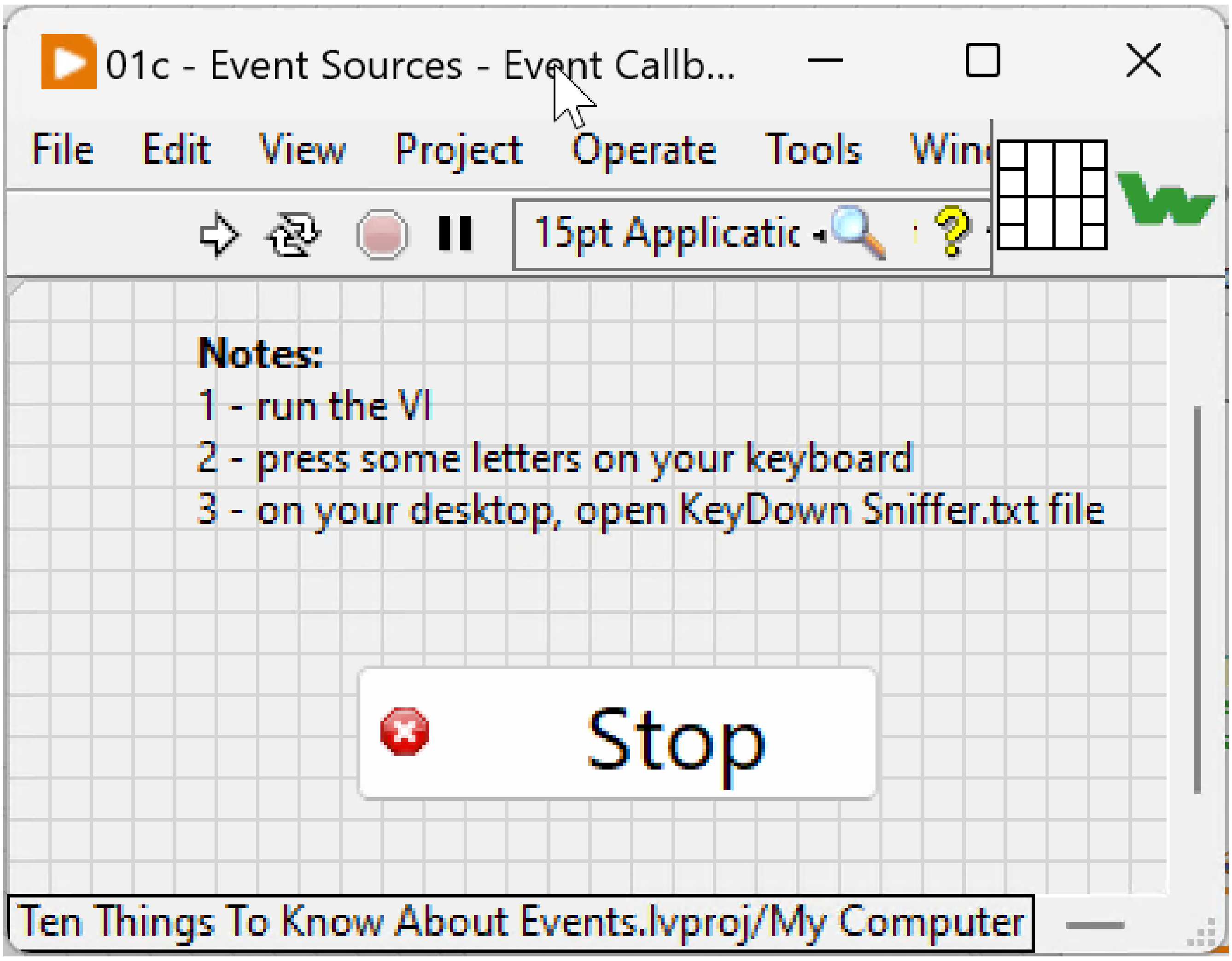
Figure 6. Diagram

Event Callback

This source of event doesn't involve the use of an Event Structure. Once registred to an event the Callback VI is executed each time the event occurs.

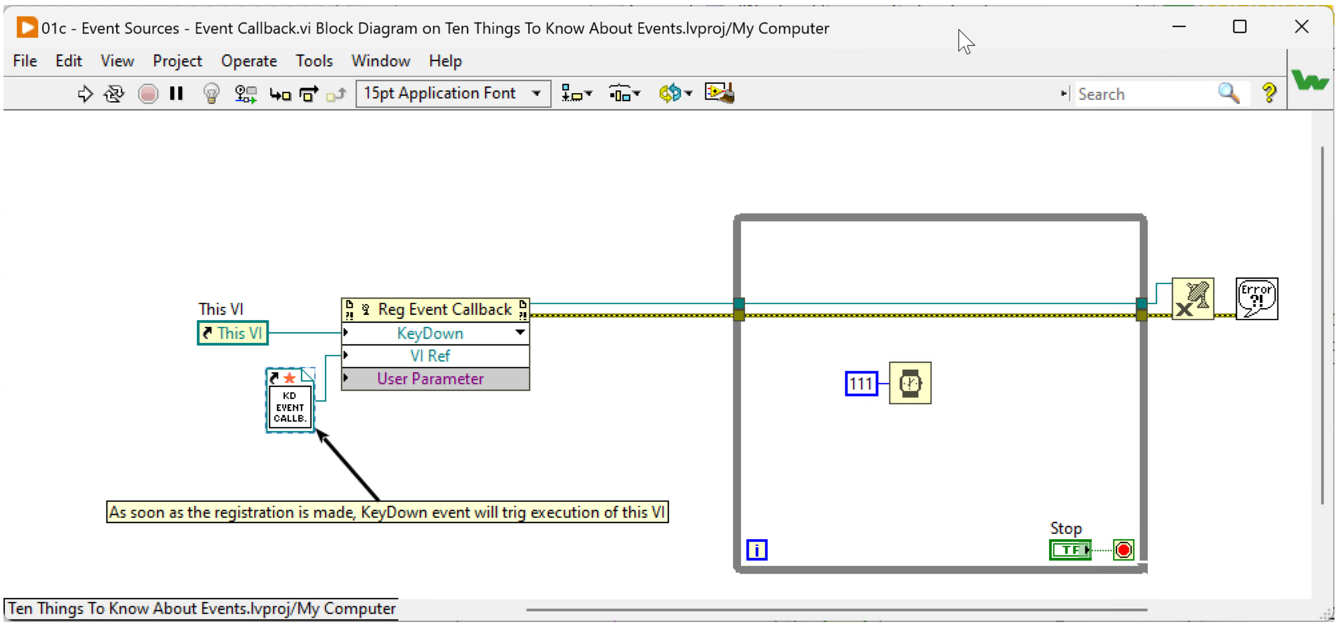
This kind of event breaks the usal data flow we are used to

Example code: **01c - Event Sources - Event Callback.vi**


01c - Event Sources - Event Callb...
File Edit View Project Operate Tools Win
15pt Applicatic
Stop
Ten Things To Know About Events.lvproj/My Computer

Notes:
1 - run the VI
2 - press some letters on your keyboard
3 - on your desktop, open KeyDown Sniffer.txt file

Figure 7. Front Panel


01c - Event Sources - Event Callback.vi Block Diagram on Ten Things To Know About Events.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
This VI
This VI
Reg Event Callback
KeyDown
VI Ref
User Parameter
KD EVENT CALLB.
111
Stop
TF
i
Ten Things To Know About Events.lvproj/My Computer

As soon as the registration is made, KeyDown event will trig execution of this VI

Figure 8. Diagram

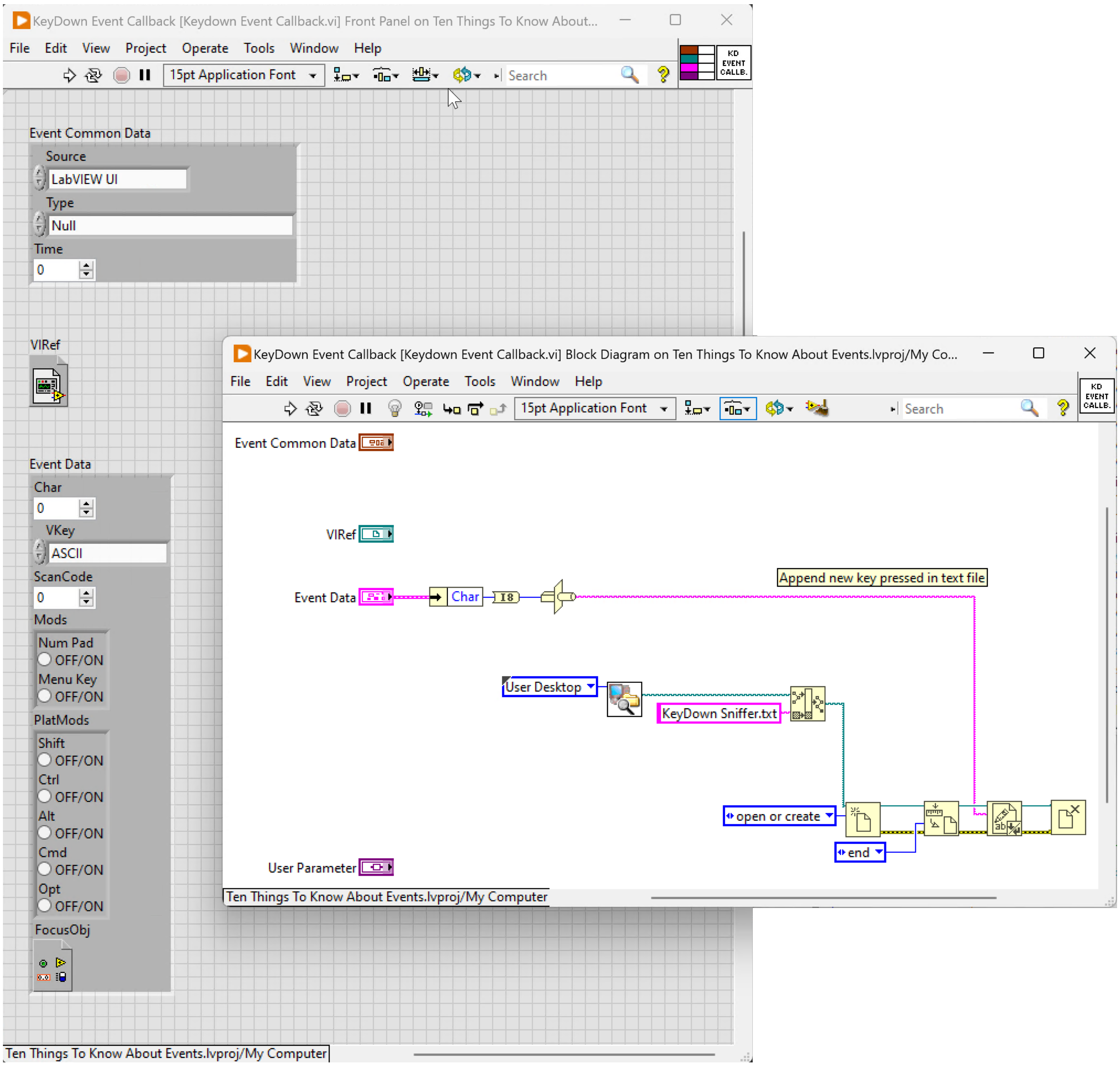

KeyDown Event Callback [Keydown Event Callback.vi] Front Panel on Ten Things To Know About...
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Event Common Data
Source
LabVIEW UI
Type
Null
Time
0
VIRef
Event Data
Char
0
VKey
ASCII
ScanCode
0
Mods
Num Pad
OFF/ON
Menu Key
OFF/ON
PlatMods
Shift
OFF/ON
Ctrl
OFF/ON
Alt
OFF/ON
Cmd
OFF/ON
Opt
OFF/ON
FocusObj
Ten Things To Know About Events.lvproj/My Computer
KeyDown Event Callback [Keydown Event Callback.vi] Block Diagram on Ten Things To Know About Events.lvproj/My Co...
Event Common Data
VIRef
Event Data
Char
I8
Append new key pressed in text file
User Desktop
KeyDown Sniffer.txt
open or create
end
User Parameter
Ten Things To Know About Events.lvproj/My Computer

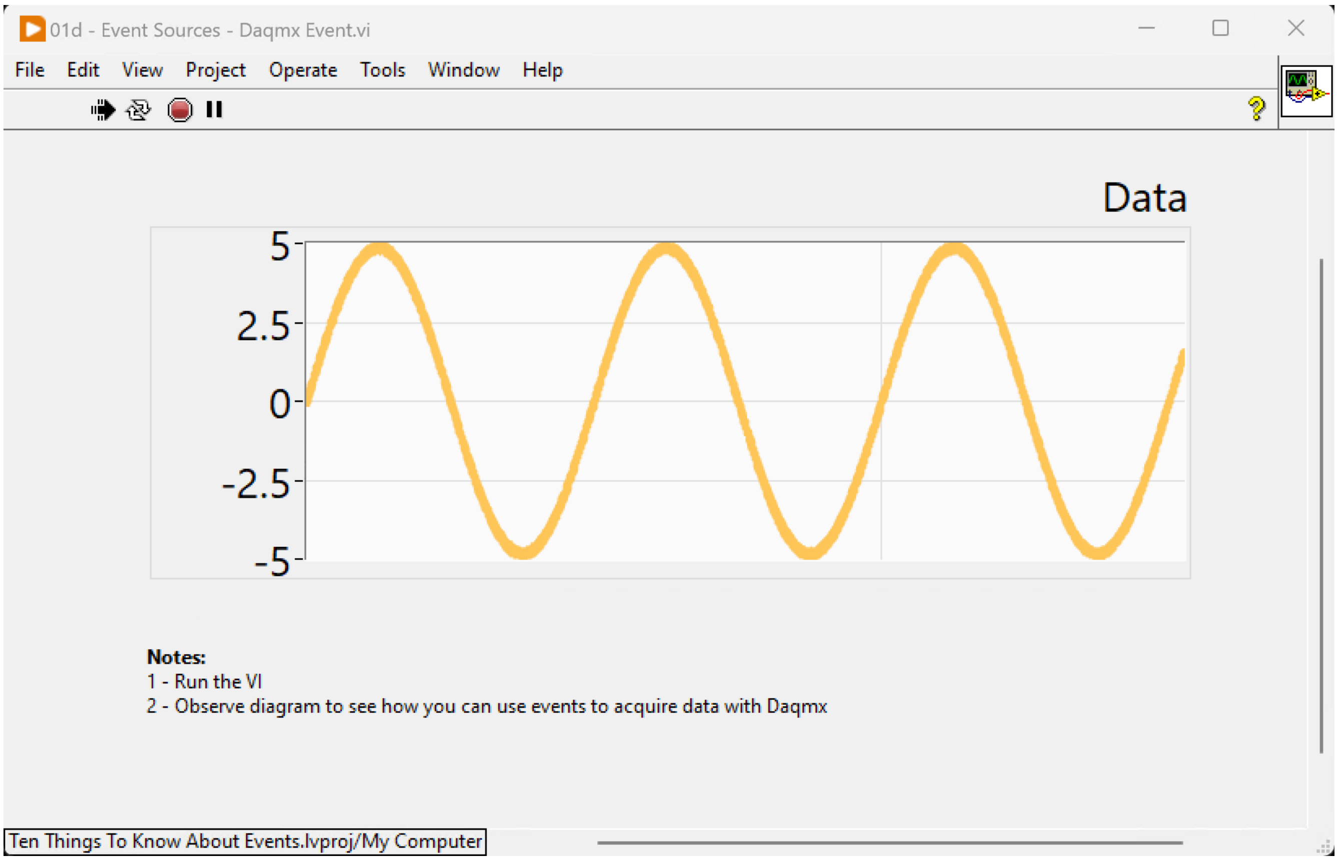
Figure 9. Callback VI diagram

Daqmx

This source of event is problably the less known. With it, you can handle acquistion event in the Event Structure. It makes straigtforward the use of the other sources of events in the loop that handle the acquistion.

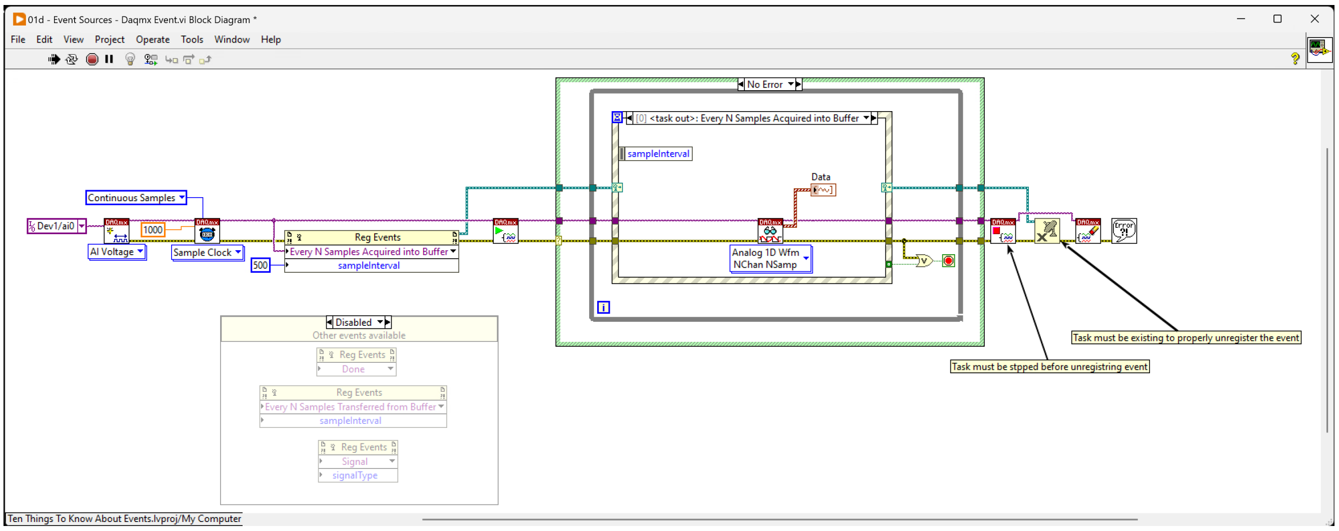
This example requieres the NI Daqmx driver to be executed.

Example code: **01d - Event Sources - Daqmx Event.vi**


01d - Event Sources - Daqmx Event.vi
File
Edit
View
Project
Operate
Tools
Window
Help
Data
5
2.5
0
-2.5
-5
Ten Things To Know About Events.lvproj/My Computer

Notes:
1 - Run the VI
2 - Observe diagram to see how you can use events to acquire data with Daqmx

Figure 10. Front Panel


01d - Event Sources - Daqmx Event.vi Block Diagram *
File
Edit
View
Project
Operate
Tools
Window
Help
No Error
[0] <task out>: Every N Samples Acquired into Buffer
sampleInterval
Data
Continuous Samples
Dev1/ai0
1000
AI Voltage
Sample Clock
500
Reg Events
Every N Samples Acquired into Buffer
sampleInterval
Analog 1D Wfm NChan NSamp
Disabled
Other events available
Reg Events
Done
Reg Events
Every N Samples Transferred from Buffer
sampleInterval
Reg Events
Signal
signalType
Ten Things To Know About Events.lvproj/My Computer

Task must be existing to properly unregister the event

Task must be stpped before unregistring event

Figure 11. Diagram

Event Types

Static events used to handle the User Interface events can be of two types:

Notify events

Those events trigger their case in the Event Structure once the event has happened. You can't modify it.

They have a green arrow in the Edit Events configuration dialog.

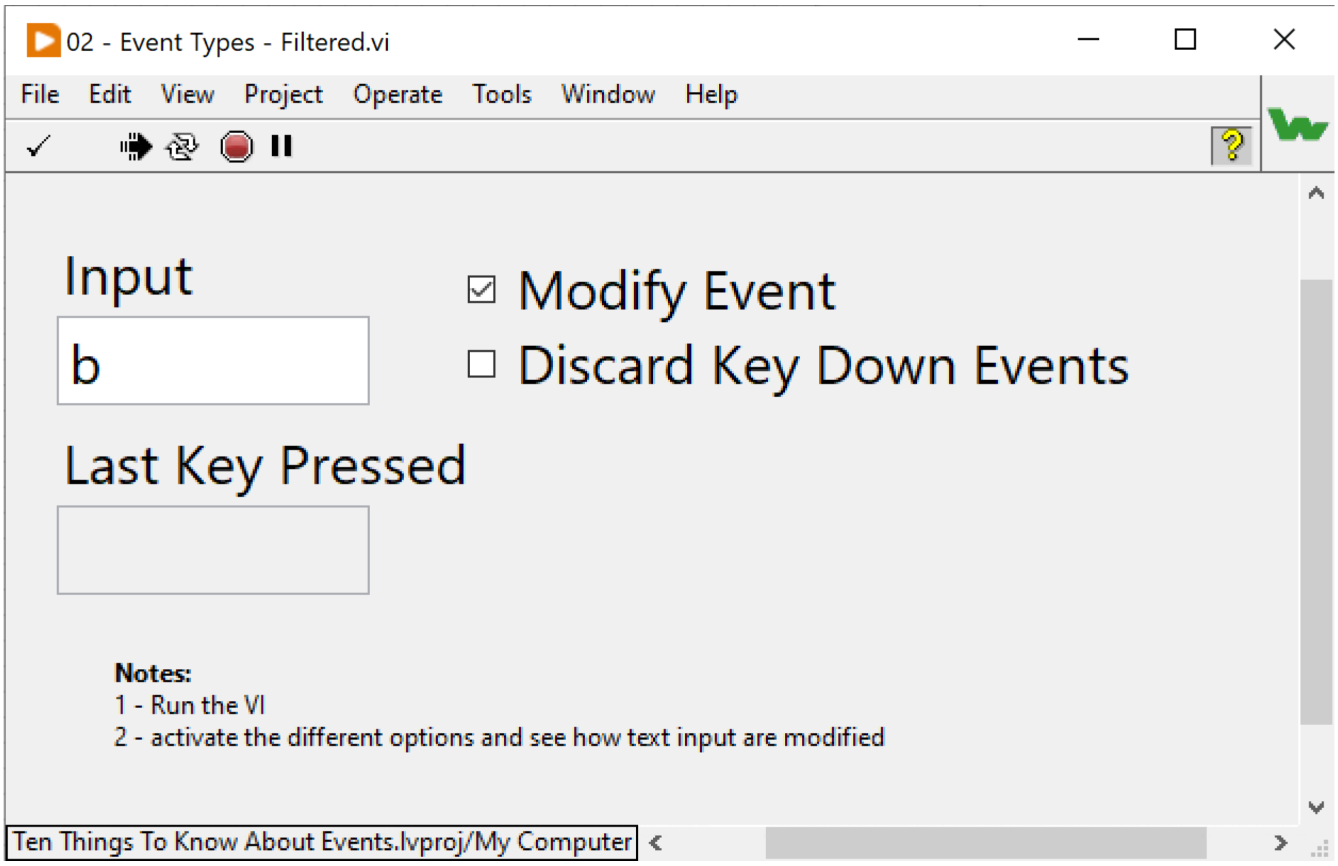
Filter events

Those events trigger their case in the Event Structure before the event is actually processed in the User Interface. This allows you to modify or even discard the event programatically.

They have a red arrow in the Edit Events configuration dialog and their names end with a ?.

In the following example, you can see different ways to interact with an event before actually processing it.

Example code: **02 - Event Types - Filtered.vi**


02 - Event Types - Filtered.vi
File Edit View Project Operate Tools Window Help
Ten Things To Know About Events.lvproj/My Computer

Input

b

☑ Modify Event

☐ Discard Key Down Events

Last Key Pressed

Notes:
1 - Run the VI
2 - activate the different options and see how text input are modified

Figure 12. Front Panel

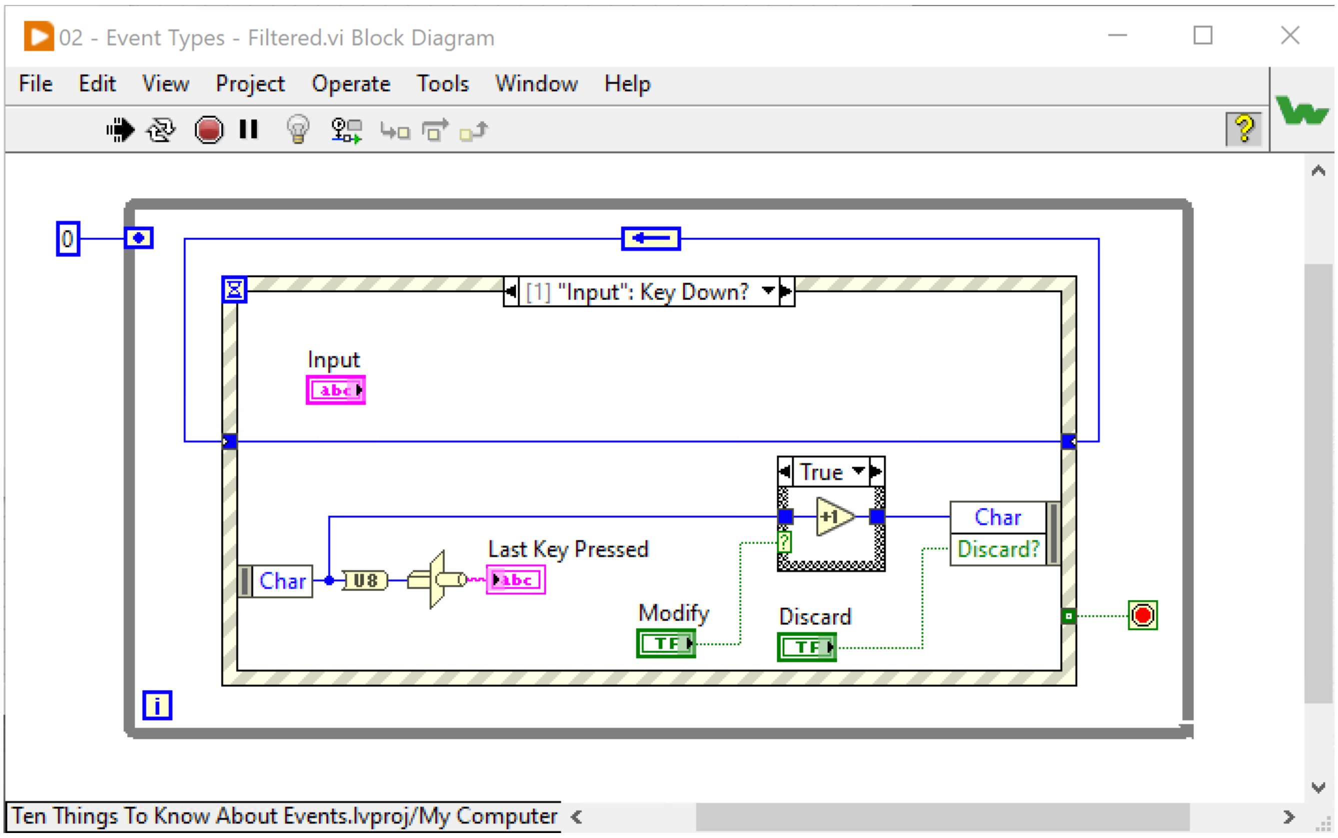

02 - Event Types - Filtered.vi Block Diagram
File Edit View Project Operate Tools Window Help
0
[1] "Input": Key Down?
Input
True
Char
Discard?
Last Key Pressed
Char
U8
Modify
Discard
i
Ten Things To Know About Events.lvproj/My Computer

Figure 13. Diagram

One of the most usually used filter events is the `Panel Close?` event. By discarding you can handle the stop of the program without using a **[Stop]** button in your User Interface.

Memory management

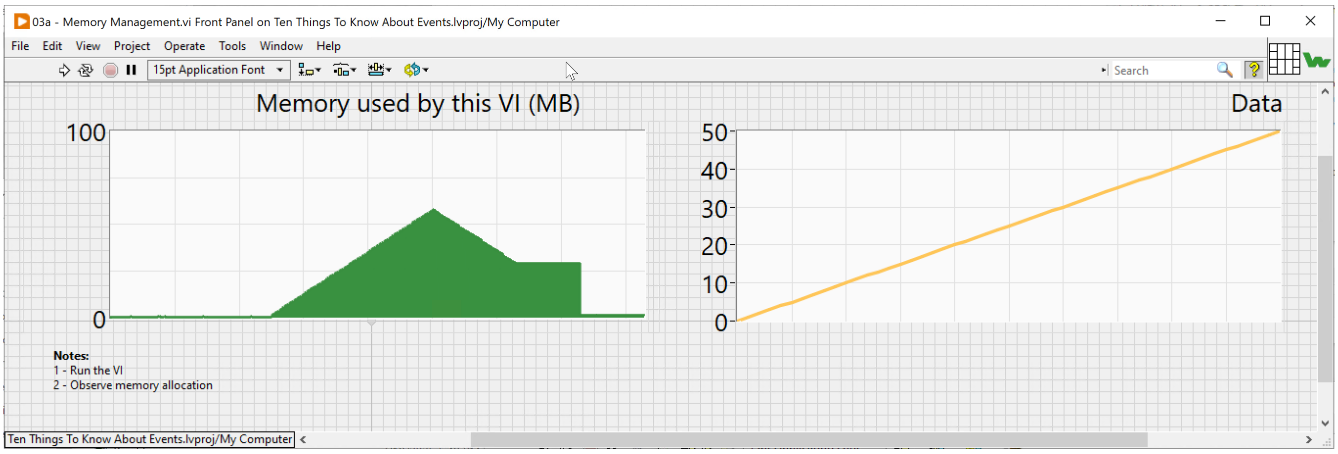
Memory management is one of the most misunderstood concepts and a source of concerns for LabVIEW™ developers.

This section gathers examples to understand when memory is allocated and deallocated.

Basic behavior

In this example, we placed breakpoints at strategic points to see how the memory allocation changes as the code runs.

Example code: **03a - Memory Management.vi**


03a - Memory Management.vi Front Panel on Ten Things To Know About Events.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Memory used by this VI (MB)
100
0
Data
50
40
30
20
10
0
Ten Things To Know About Events.lvproj/My Computer

Notes:
1 - Run the VI
2 - Observe memory allocation

Figure 14. Front Panel

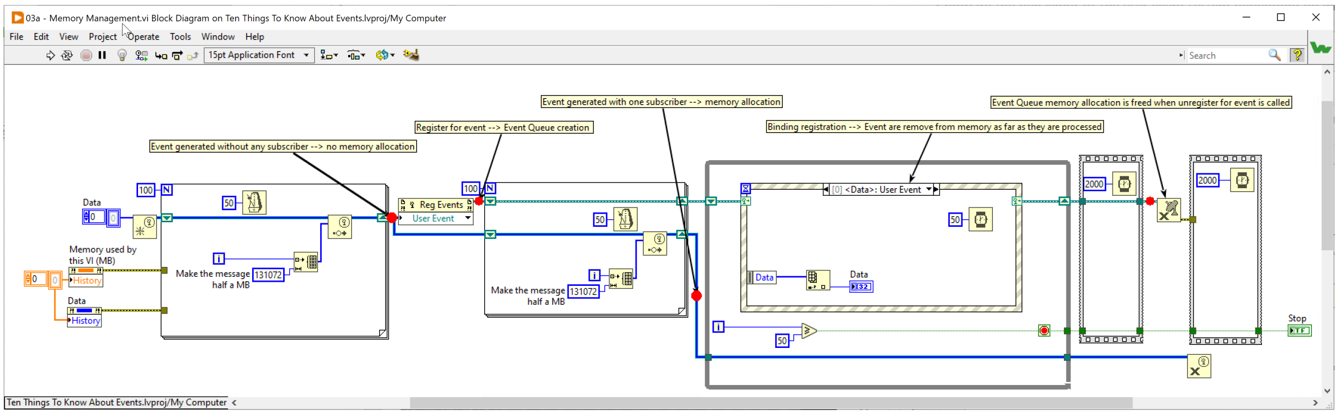

03a - Memory Management.vi Block Diagram on Ten Things To Know About Events.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Event generated with one subscriber --> memory allocation
Event Queue memory allocation is freed when unregister for event is called
Register for event --> Event Queue creation
Binding registration --> Event are remove from memory as far as they are processed
Event generated without any subscriber --> no memory allocation
Data
100
N
50
Memory used by this VI (MB)
History
Data
History
Make the message half a MB
131072
Reg Events
User Event
[0] <Data>: User Event
Data
2000
Stop
Ten Things To Know About Events.lvproj/My Computer

Figure 15. Diagram

The video below shows you the code in execution with the evolution of the memory.

▶ event-memory-allocation.mp4 *(video)*

Here are the key points to remember:

1. User Events generated before the User Event is registered are not kept in memory ▯ no memory leak.
2. User Events generated after the registration are kept in memory ▯ you should be very careful binding

your Registred Events ref to an Event Structure

3. When event are consumed by the Event Structure the memory is deallocated ▯ to prevent memory issues in your code, you must ensure that Event Structure is capable of consuming all generated events it is handling.
4. Events not consumed by the Event Structure are deallocated when calling the `Unregister For Events` function ▯ always call this function when you exit an Event Structure.