

# Laboratorio Instrumentación Electrónica

---

**Herramientas Software para Adquisición de datos:  
Labview**

**José A. Jiménez**

**Curso 2016-2017**



## Índice

1. ¿Qué es Labview?.
2. Entorno Labview.
3. Elementos de Programación.





## ¿Qué es Labview?

- **Labview es un lenguaje de programación:**

- Distintos tipos de datos: enteros, reales, cadenas de caracteres y booleanos.
- Estructuras de programación: bucles, gestión de eventos, cases, etc.
- Librería de funciones: ficheros I/O, manipulación de Arrays y cadenas de caracteres, matemáticas, etc.
- Toolkits: visión, PID, Procesamiento de señal, radio frecuencia, bluetooth, FPGAs, arduino, etc.
- Compilación continua a código máquina del código desarrollado para mostrar errores de programación.
- Programación multihilo inherente.
- Permite gestión multicore.

- **Labview es un entorno de desarrollo:**

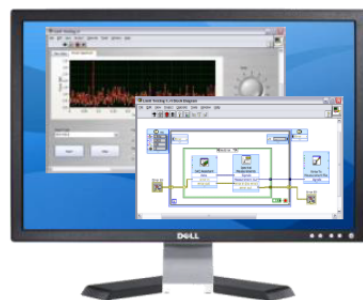
- Proporciona herramientas de depuración.
- Diseño de Interface de usuario.
- Detección automática de hardware.
- Herramientas de optimización: después de crear el programa se pueden utilizar herramientas como VI analyzer y VI Profiler para optimizar la apariencia y el comportamiento del código.



## ¿Qué es Labview?

- **Labview se conecta fácilmente a diferentes tipos de hardware de I/O:**

- Más de 8000 instrumentos de más de 250 fabricantes.
- PCI, PCIe, PXI, USB, Ethernet, serial, GPIB, buses de campo (CAN, ModBus).
- Hardware de adquisición de datos.
- Cámaras.
- PLC's.



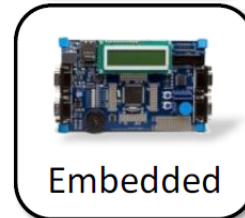
Software-defined  
behavior



Test



Industrial



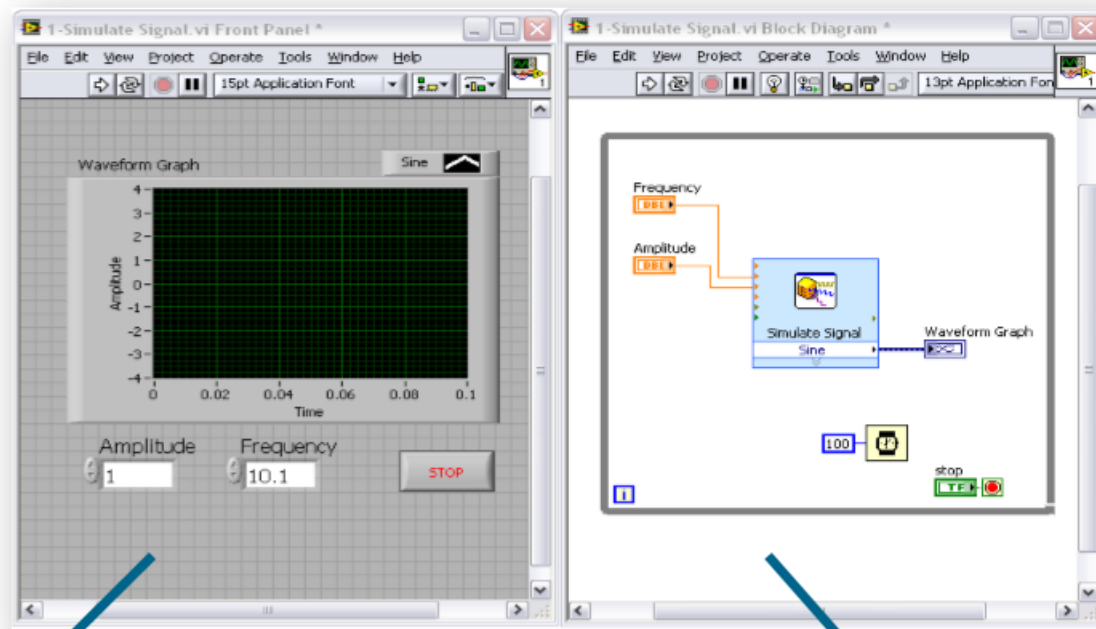
Embedded

Modular  
I/O



- Formado por: **Panel Frontal y Diagrama de Bloques con sus correspondientes paletas de objetos y funciones.**

“VI” = program or function



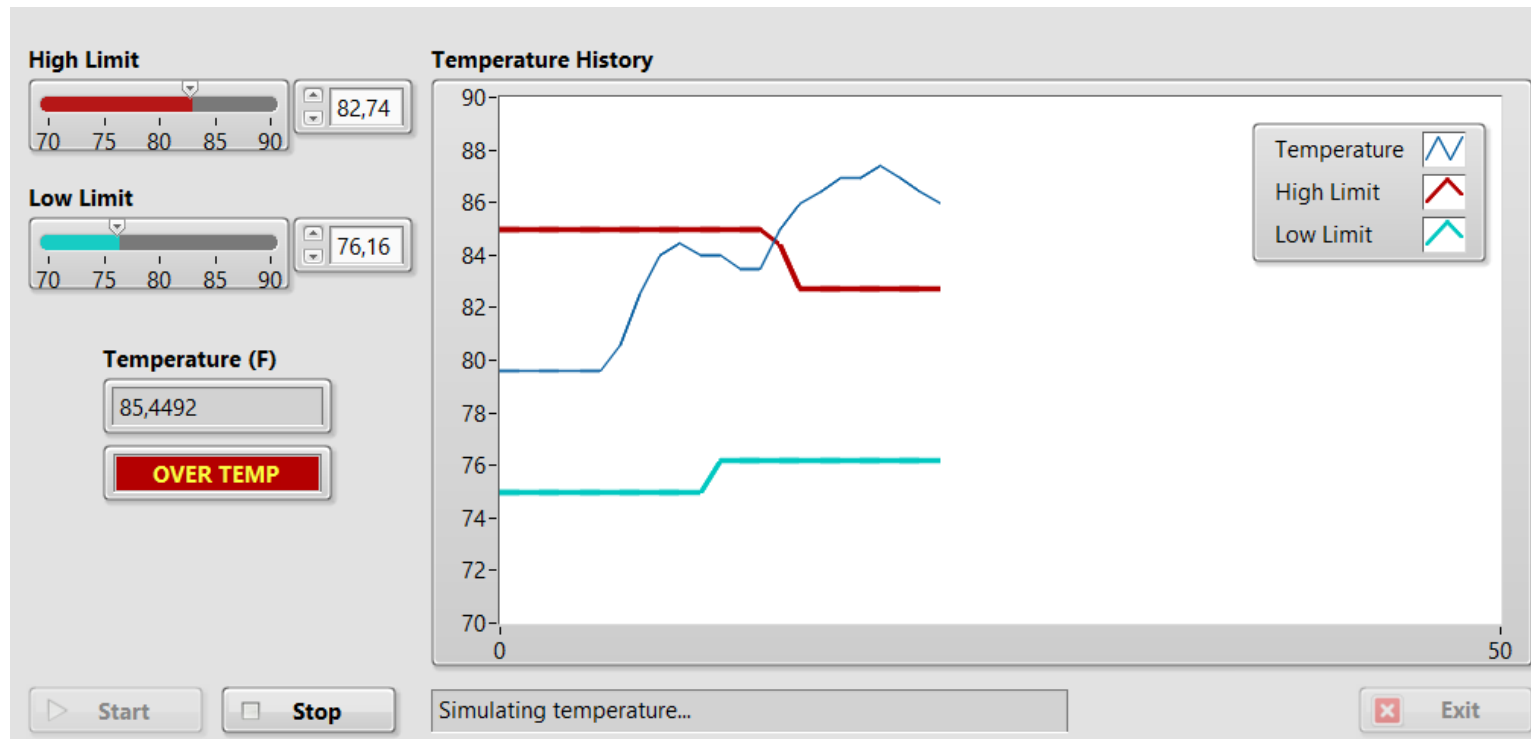
“Front Panel” = user interface

“Block Diagram” = code



### Panel Frontal:

- Es el panel de usuario de la aplicación que se está desarrollando.
- Labview proporciona los objetos (**controles e indicadores**) típicos en cualquier aplicación relacionada con sistemas electrónicos y de instrumentación: sistemas de adquisición de datos, instrumentación programable, comunicaciones industriales, etc.

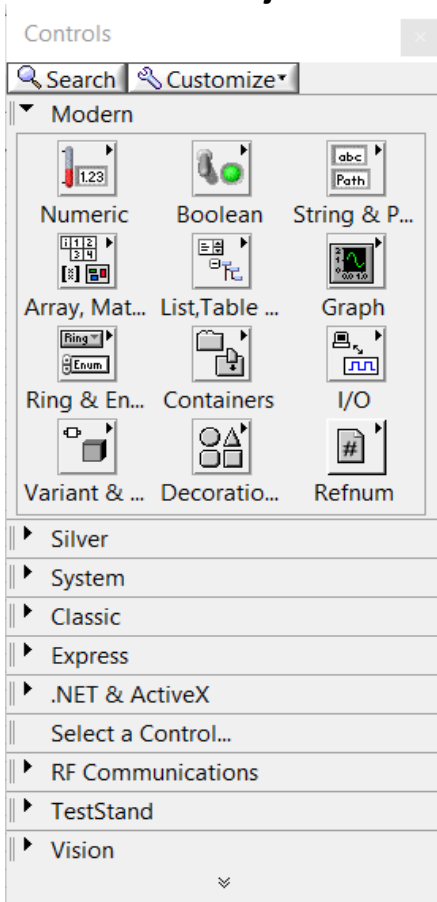




### Panel Frontal: paleta de objetos.

- Acceso a la paleta de objetos → *click* sobre el panel frontal con el botón derecho del ratón.

#### Paleta Objetos



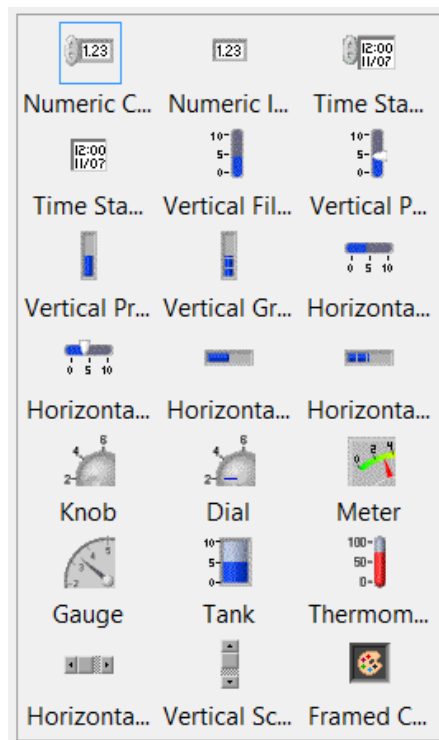
#### Características Sub-paletas.

Subpalette	Description
Modern	Extended collection of controls and indicators you can use to create most front panels.
Silver	Alternative extended collection of controls and indicators you can use to create most front panels. The silver controls change appearance depending on which platform you run the VI.
System	Collection of controls and indicators to use in dialog boxes you create. The system controls change appearance depending on which platform you run the VI.
Classic	Collection of controls and indicators to create VIs for low-color monitor settings.
Express	Subset of controls and indicators available on the Modern palette. Because the Express palette contains a more limited selection, you can locate the controls and indicators you need faster and build front panels quickly. Use the controls and indicators on the Modern palette if you need a wider selection.
.NET & ActiveX	Collection of controls and indicators to manipulate common .NET or ActiveX controls.
User Controls	Contains objects you add to the Controls palette. By default, the User Controls palette does not contain any objects.

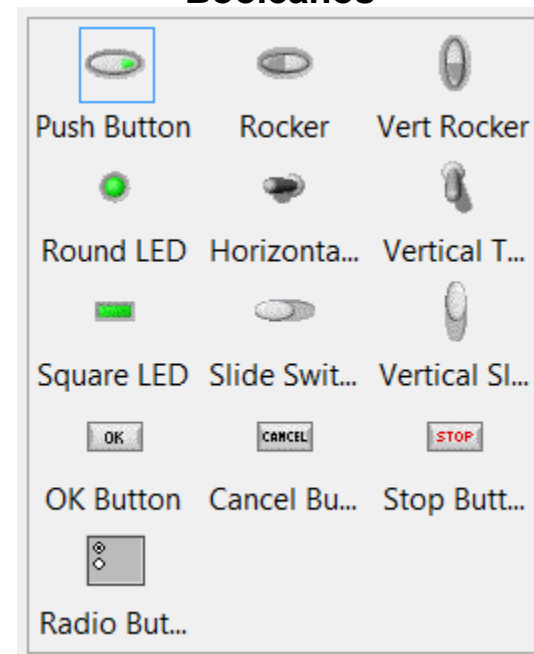


### Panel Frontal: paleta de objetos.

#### Numéricos



#### Booleanos

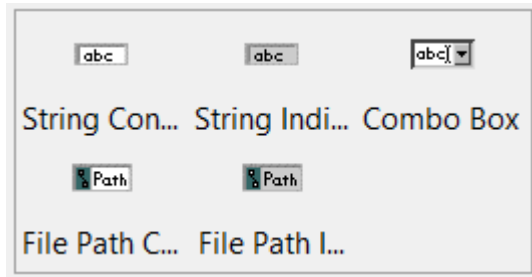




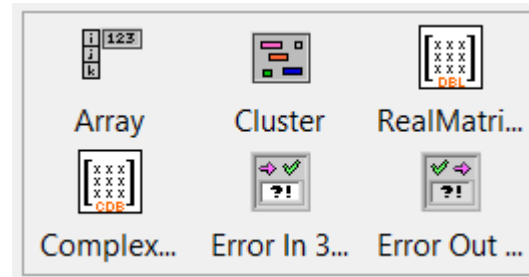


### Panel Frontal: paleta de objetos.

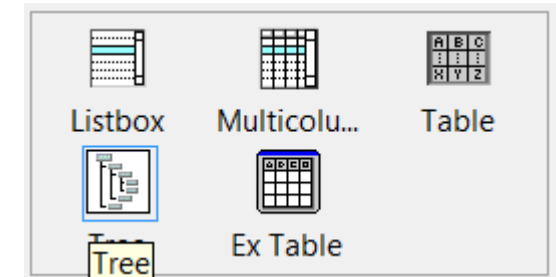
#### String y Paths



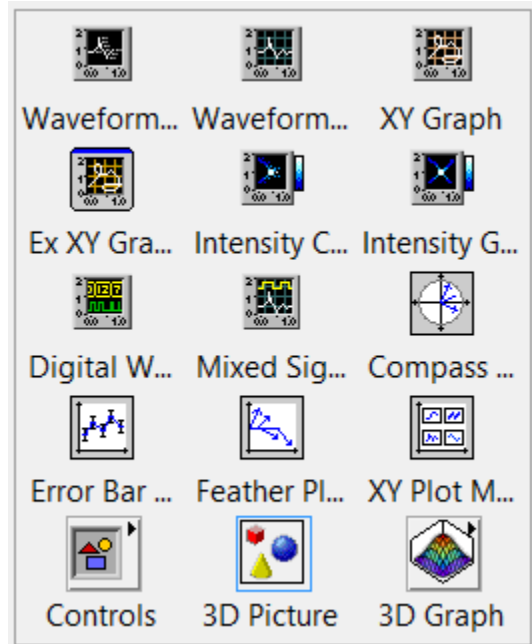
#### Arrays y Matrices



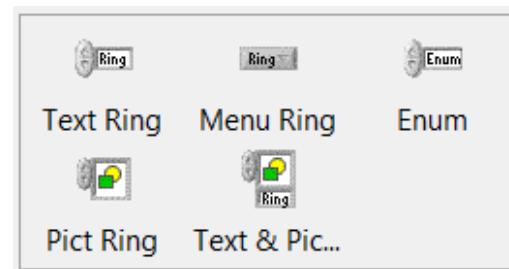
#### Listas y Tablas



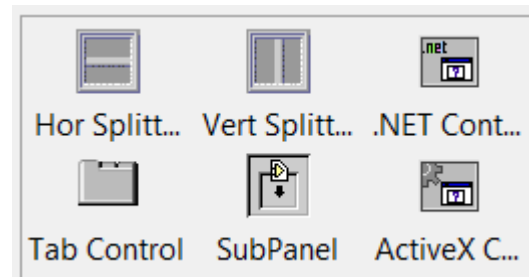
#### Gráficos



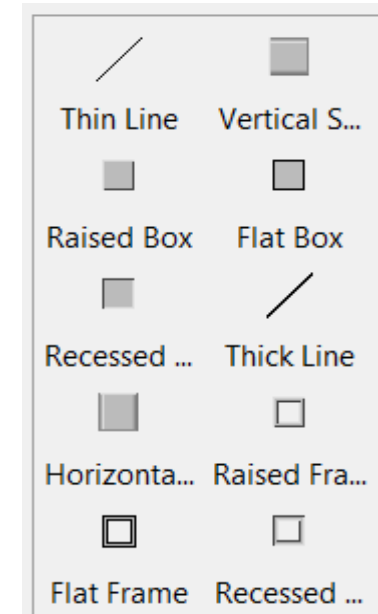
#### Rings



#### Contenedores



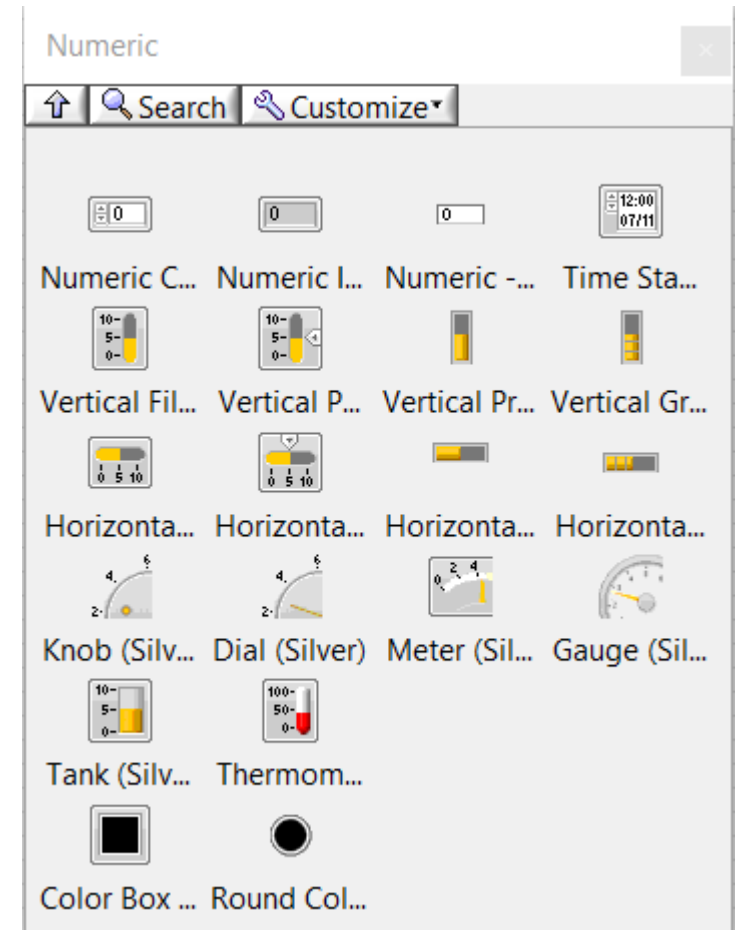
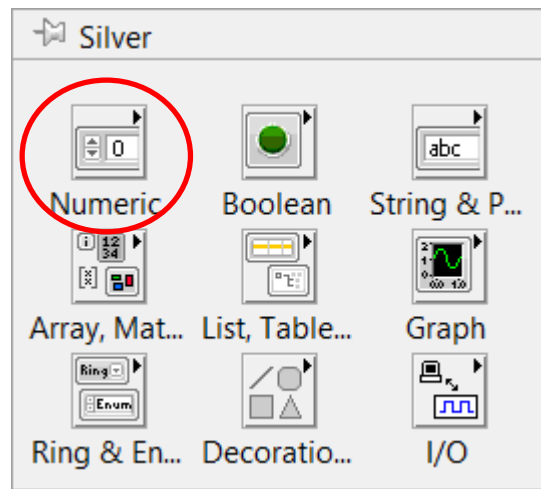
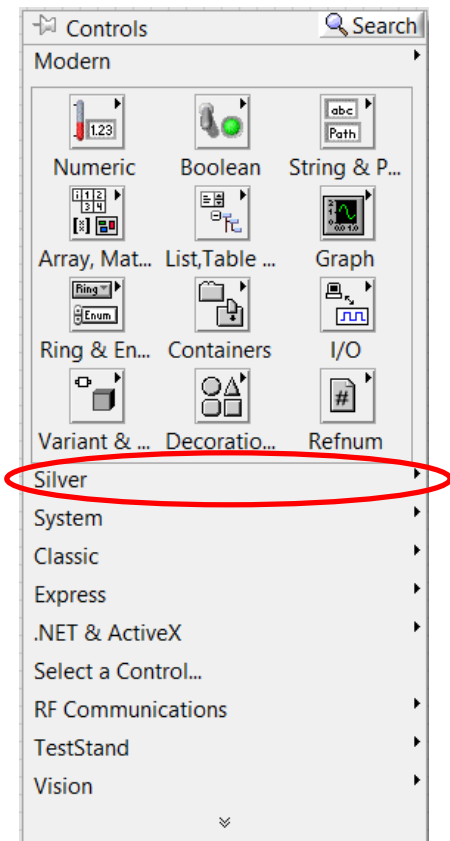
#### Decoraciones





### Panel Frontal: paleta de objetos.

Controles de aspecto más actual y que cambian de aspecto en función de la plataforma sobre la que se ejecuta la aplicación → **Controles Silver (a partir de Labview 2012).**





### Menú Panel Frontal.

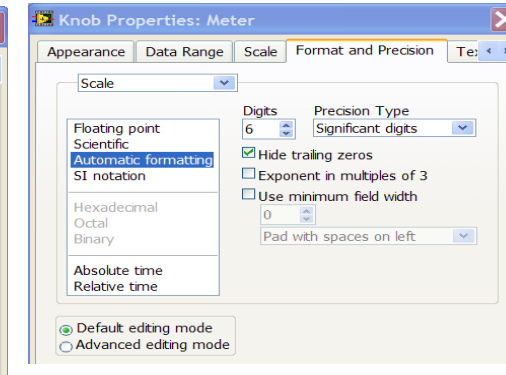
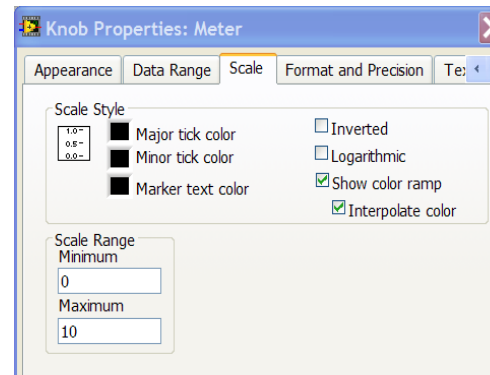
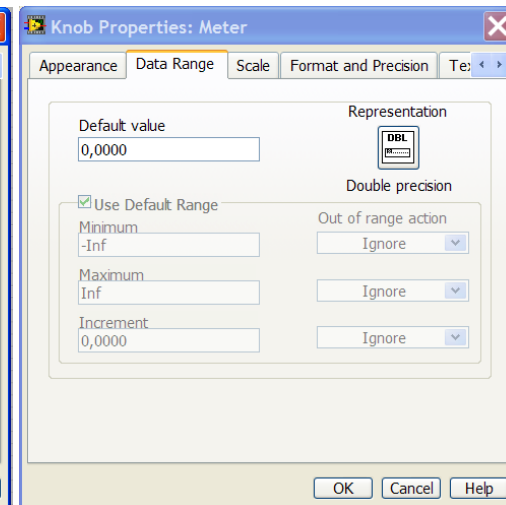
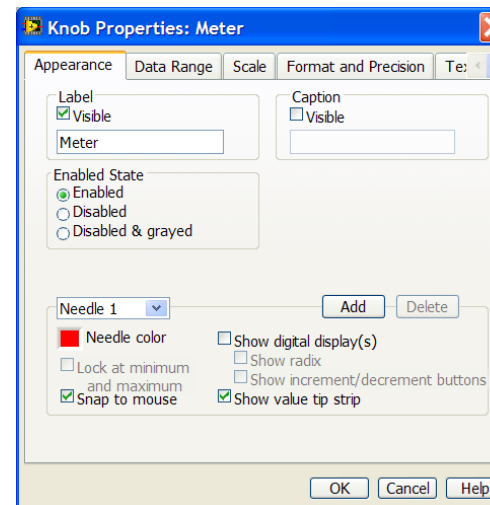
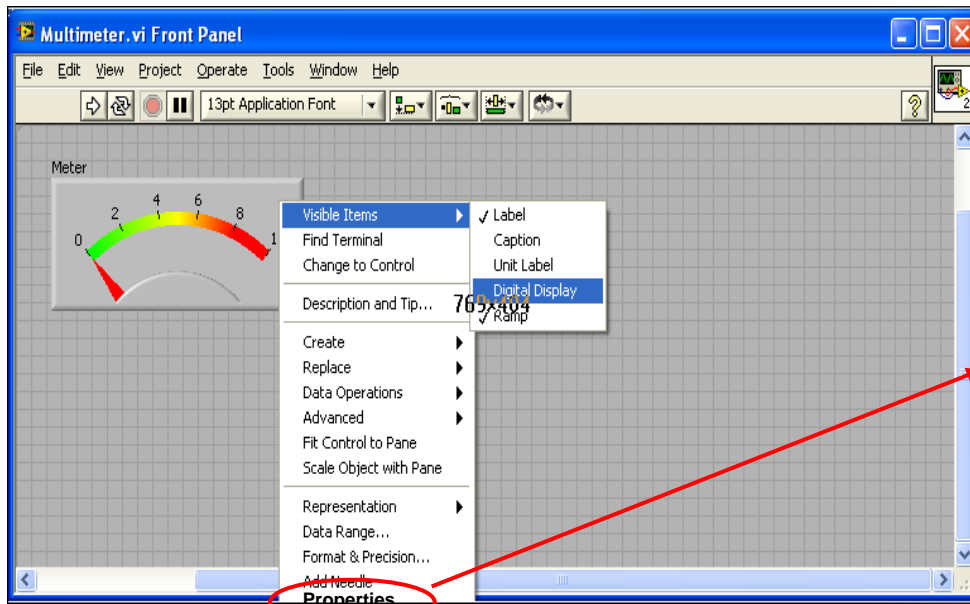
### Ayuda contextual

	Runs a VI. LabVIEW compiles the VI, if necessary. You can run a VI if the Run button appears as a solid white arrow, which indicates that you can use the VI as a subVI if you create a connector pane for the VI.
	Appears when a top-level VI is running.
	Appears when a subVI is running.
	Appears when the VI you are creating or editing contains errors. If this button appears even after you finish wiring the block diagram select this button to display the Error list window, which lists all errors and warnings.



### Panel Frontal: propiedades de los objetos.

- Todos los objetos tienen un menú flotante asociado que permite modificar sus propiedades.
- A través de este menú se accede a las **propiedades** de los objetos.

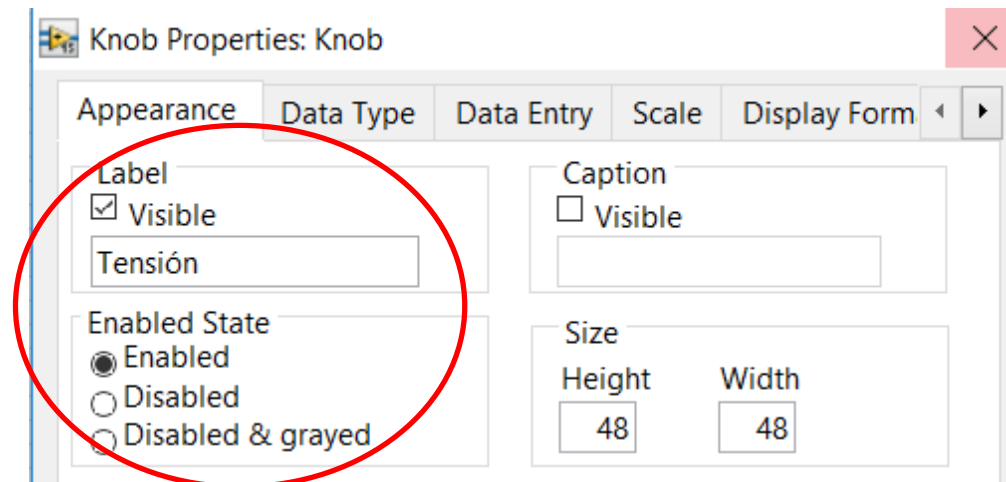




### Panel Frontal: propiedades de los objetos.

- Propiedad **Label**:

- Es imprescindible que todos los objetos que se añaden al panel frontal contengan una Label ya que esta representa el nombre de ese objeto dentro del panel frontal y es lo que lo distingue del resto de objetos. Es el equivalente al nombre de la variable en los lenguajes de programación basados en texto.
- Se puede hacer que no sea visible, pero todo objeto debe llevar una Label asociada.
- La label identifica a los objetos en el código del diagrama de bloques: terminales, variables locales, nodos de atributos.
- Si se cambia la Label el vi debe recompilarse.

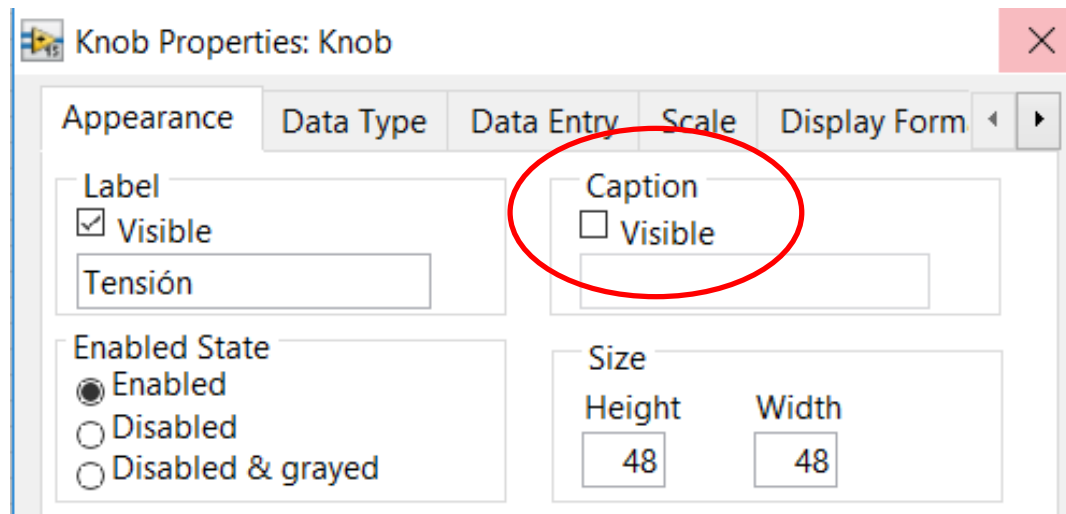




### Panel Frontal: propiedades de los objetos.

- Propiedad **Caption**:

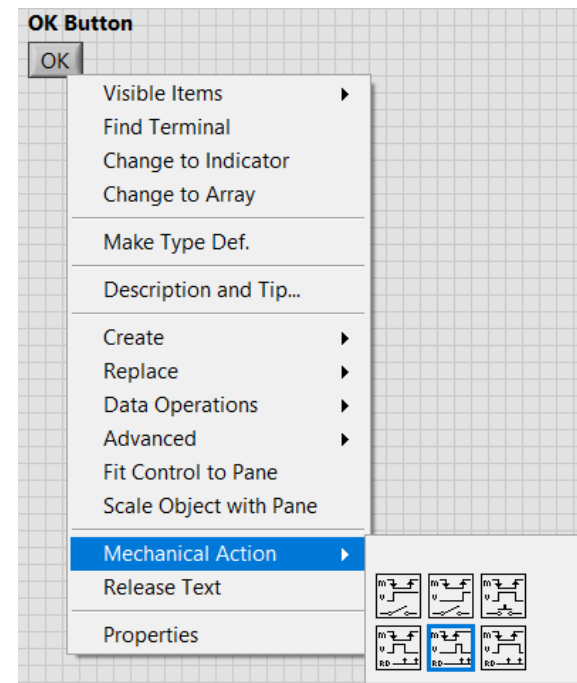
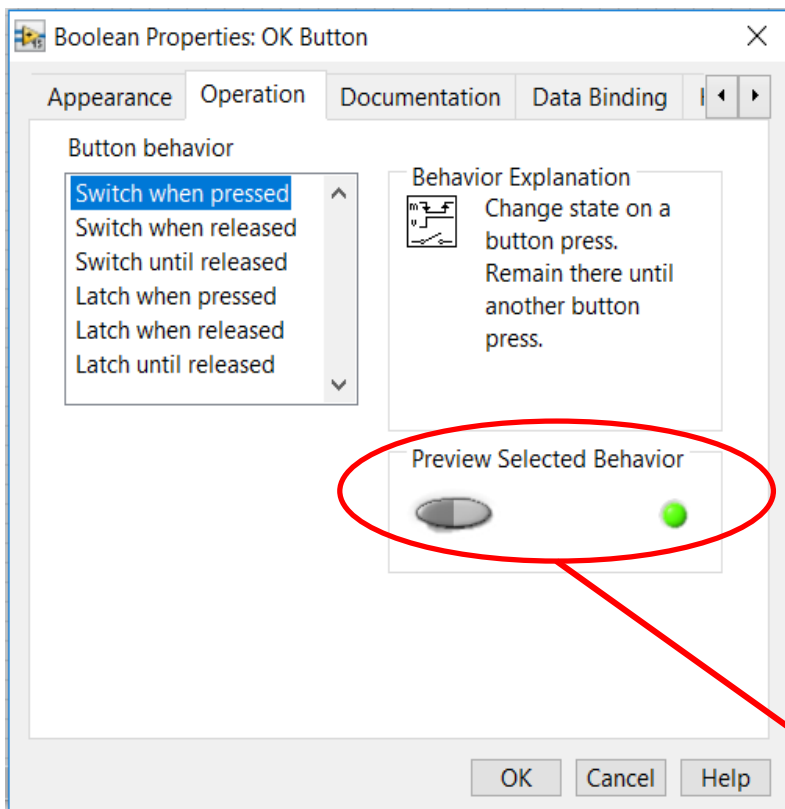
- A diferencia de la Label, esta propiedad solo aparece en el panel frontal.
- Su utilidad es añadir información adicional sobre los objetos.
- Su uso es opcional.
- Obviamente, Labview no debe recompilar el vi cuando se cambia la Caption.





### Panel Frontal: propiedades de los objetos.

- Propiedad **Operation (acción mecánica)** de controles booleanos (botones/switch): a través de esta propiedad los controles booleanos pueden configurarse para tener distintos comportamientos:
- A esta propiedad se puede acceder también mediante la opción **Mechanical Action** del menú flotante.



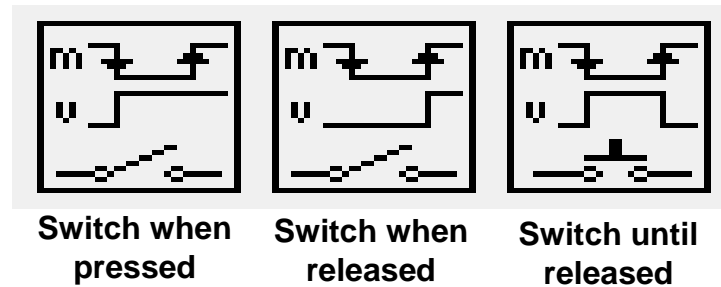
- A través del botón y el led que aparecen en el panel **Operation** se puede comprobar/chequear el comportamiento del botón para cada una de las opciones disponibles.



### Panel Frontal: propiedades de los objetos.

- Propiedad **Operation (acción mecánica)** de controles booleanos (botones/switch): existen dos tipos de acciones mecánicas **switch** y **latch**.

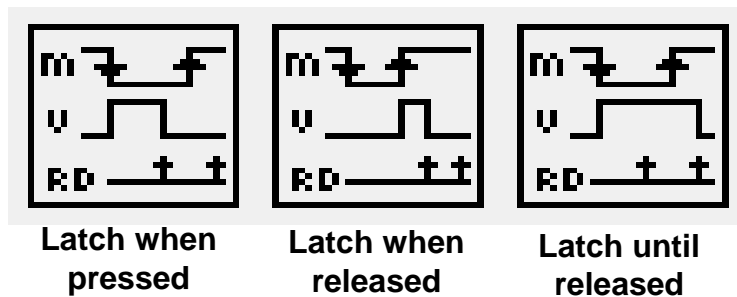
- **Switch:** con este comportamiento los controles booleanos retornan al estado inicial cuando el usuario lo decida en función de tipo elegido:



En este caso los controles booleanos se comportan como interruptores.

La variable solo cambia de estado por acciones del usuario.

- **Latch:** con este comportamiento los controles booleanos cambian de valor cuando el usuario lo decida en función de tipo elegido y lo mantienen hasta que el valor es leído por el vi, momento en el que retornan a su estado inicial.



En las figuras:

m → representa la acción del usuario sobre el ratón.

v → representa la respuesta el botón.

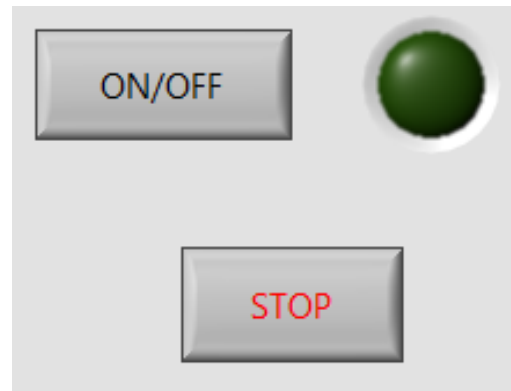
RD → Representa los momentos en los que el vi hace la lectura del control booleano.





### Panel Frontal: propiedades de los objetos.

**Ejercicio:** implementar el código del siguiente panel frontal con el que se pretende verificar los distintos comportamientos de un control booleano (botón ON/OFF) en función del tipo de modo de operación elegido.



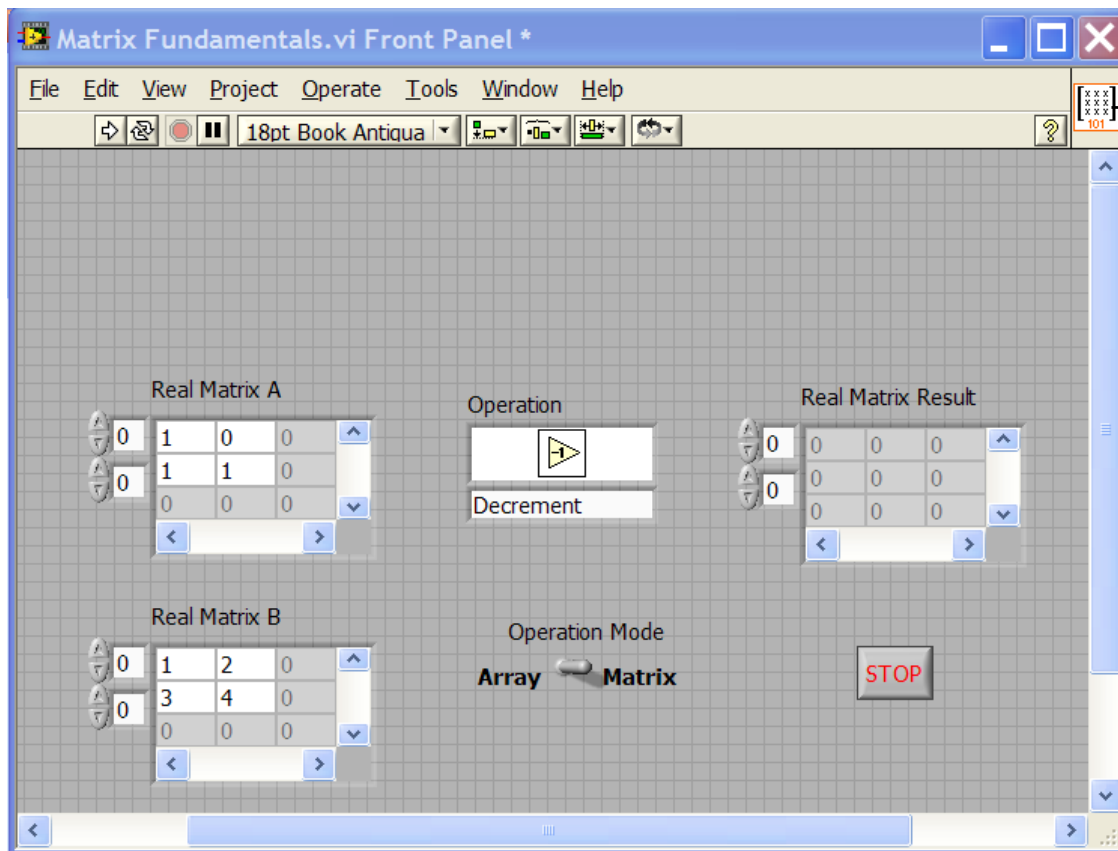
### Consideraciones:

- El botón ON/OFF enciende o apaga el led en función de su estado o valor.
- Cuando se seleccionen los modos tipo **Switch** no se debe temporizar el bucle principal del programa.
- El bucle de control del vi debe ejecutar iteraciones a intervalos de 500 ms cuando se seleccionen los modos de operación tipo **Latch**. De esta forma se puede verificar mejor como el led control e indicador led asociado cambian de estado cuando el vi lee el valor del control ON/OFF.
- El botón STOP finaliza la ejecución del vi.

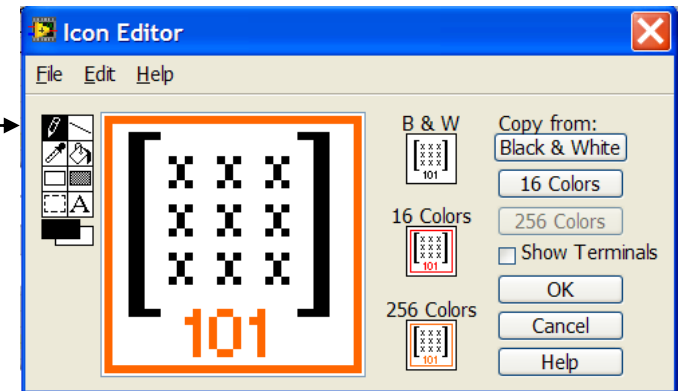


### Panel Frontal: Conector.

- Permite identificar los VI's.
- Puede contener texto e imágenes



Doble clic



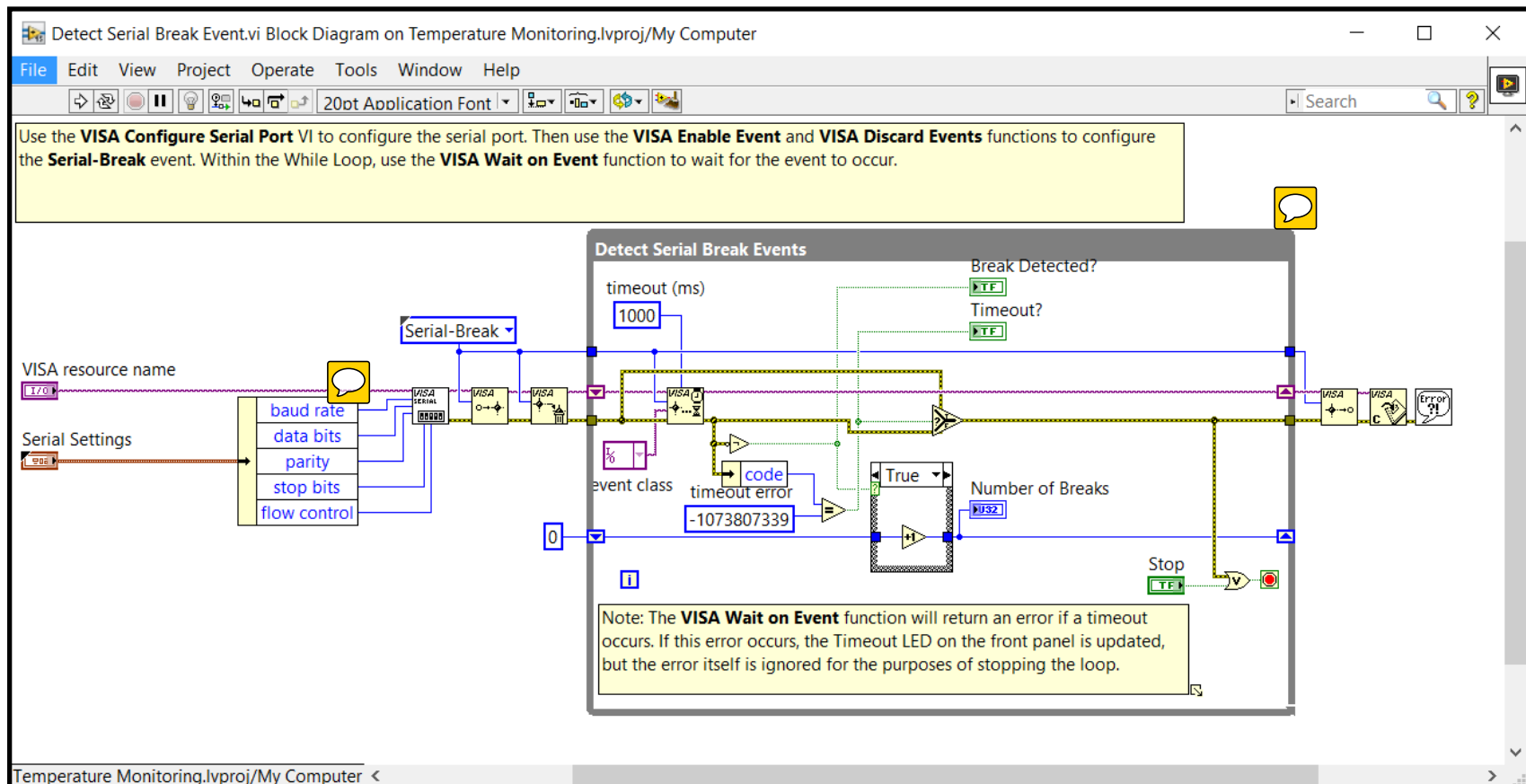
**Editor de iconos: solo accesible desde el panel frontal**





### Diagrama de bloques:

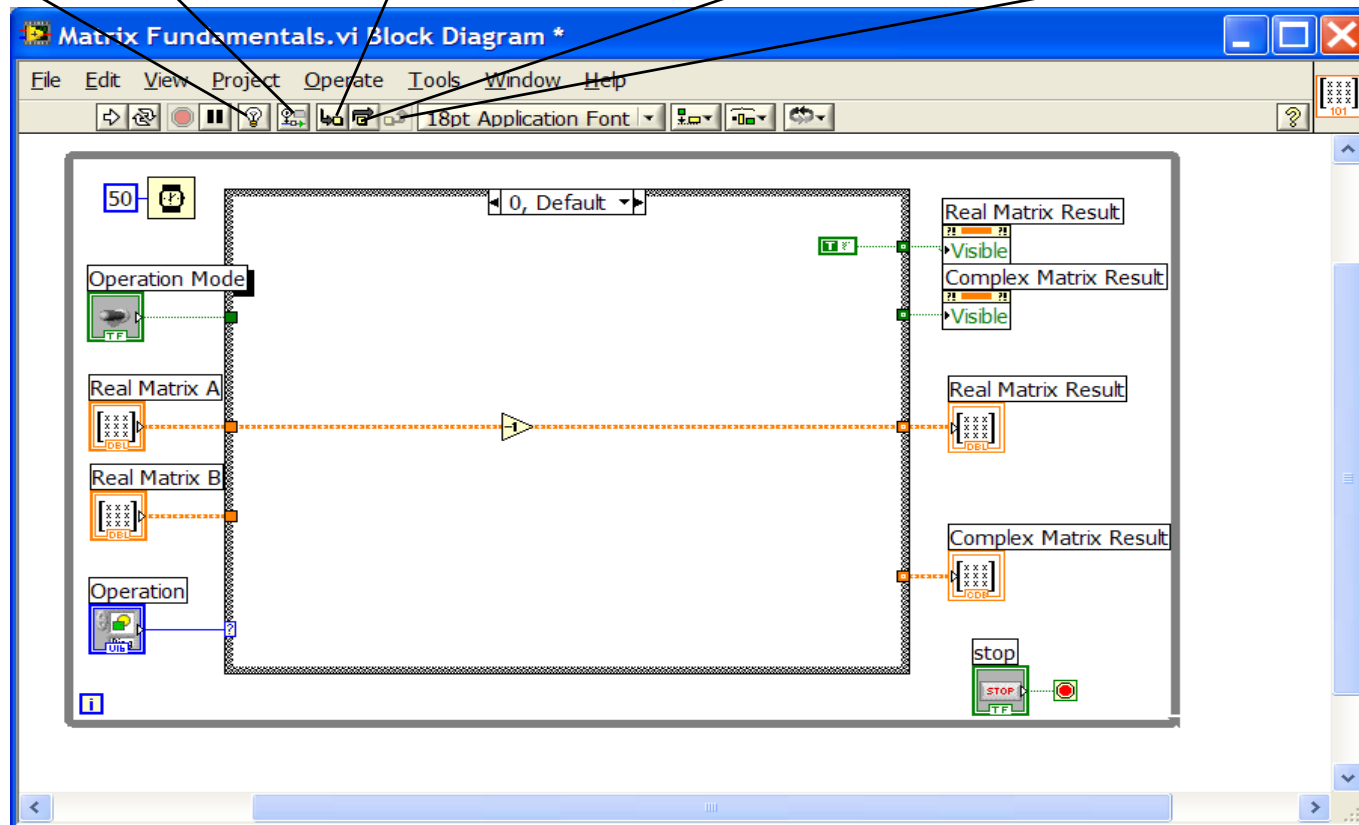
- Contiene el código fuente de la aplicación.
- Cada vez que insertamos un objeto en el panel frontal aparece representado en el diagrama de bloques.





### Menú del Diagrama de Bloques.

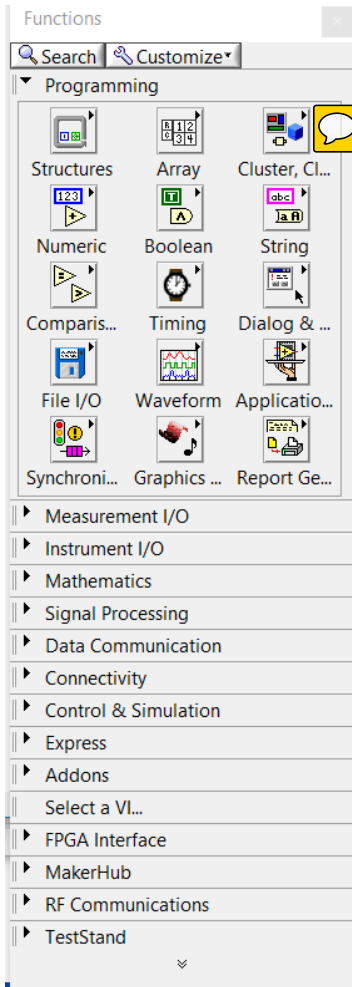
- Ejecución animada
- Mantiene valores en los cables
- Step Into:** Ejecución de los subVI's abriéndolos.
- Step Over:** Ejecución de los subVI's sin abrirlos.
- Step Out:** No ejecuta los subVI's.



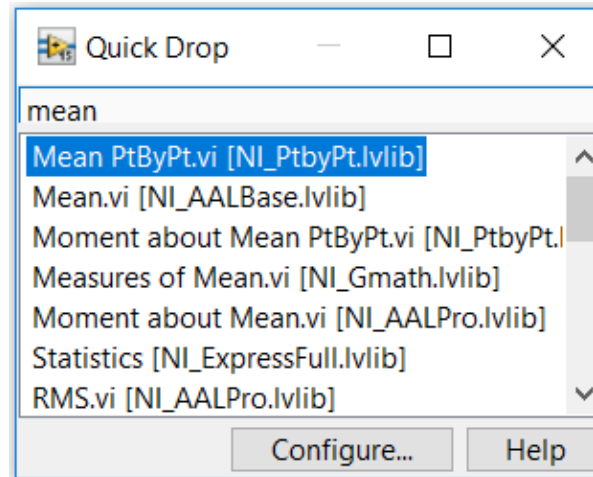


### Diagrama de bloques: paleta de funciones.

- A la paleta de objetos se accede haciendo *click* sobre el diagrama de bloques con el botón derecho del ratón.



- La paleta de funciones permite el acceso a elementos de programación y a las librerías de funciones y toolkits.
- Las estructuras y elementos de programación se encuentran en el ítem *Programming*.
- Las funciones se pueden buscar por su nombre a través de la ventana *Quick Drop* que aparece pulsando <Ctrl+barra espacio>.

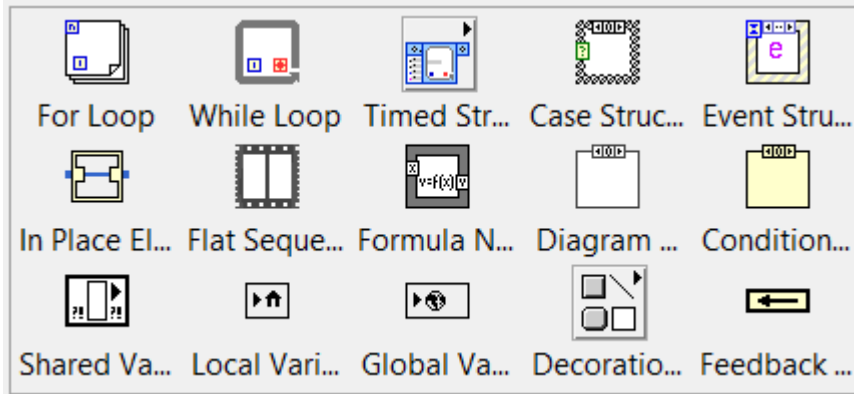




### Diagrama de bloques: paleta de funciones.

- Estructuras y elementos de programación:

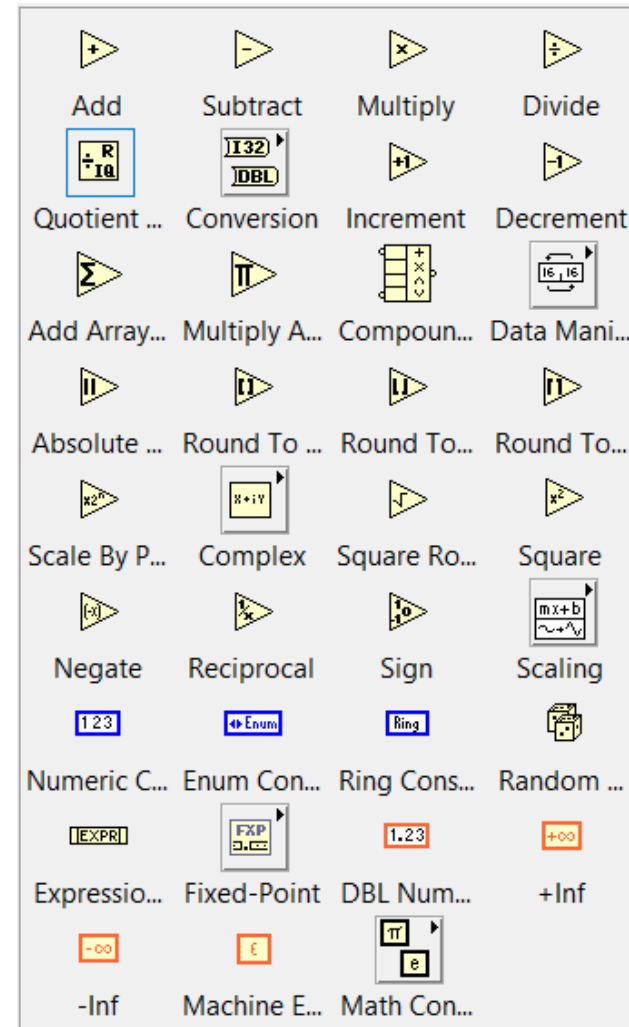
#### Estructuras



#### Boolean



#### Numeric





### Diagrama de bloques: paleta de funciones.

- Estructuras y elementos de programación:

#### String

String Len... Concaten... String Sub... Trim Whit... Normalize...

Replace S... Search an... Match Pat... Match Re... Path/Arra...

Scan From... Format Int... Format D... Build Text... Number/S...

Spreadshe... Array To S... To Upper ... To Lower ... Flatten/Un...

String Con... Empty Stri... Space Co... Tab Const... Additiona...

Carriage R... Line Feed ... End of Lin...

#### Comparación

Equal? Not Equal? Greater? Less? Greater Or...

Less Or Eq... Equal To 0? Not Equal... Greater Th... Less Than ...

Greater Or... Less Or Eq... Select Max & Min In Range ...

Not A Nu... Empty Arr... Empty Stri... Decimal ... Hex Digit?

Octal Digi... Printable? White Spa... Lexical Cl... Comparis...

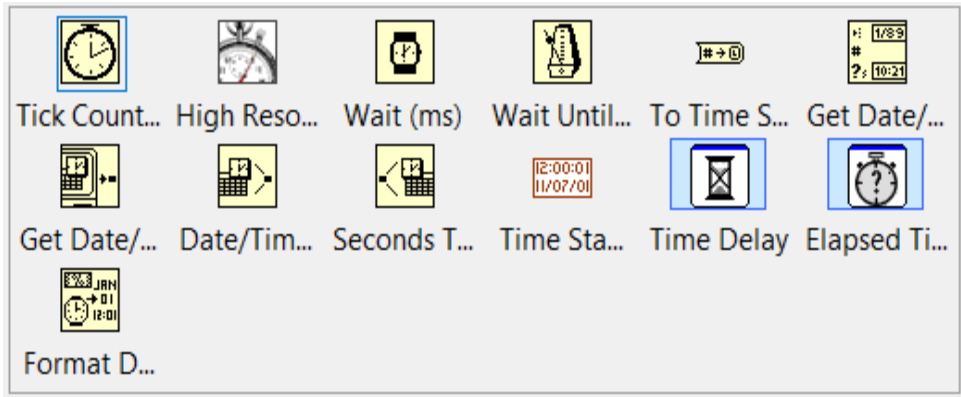
Is Path an... Fixed-Poi...



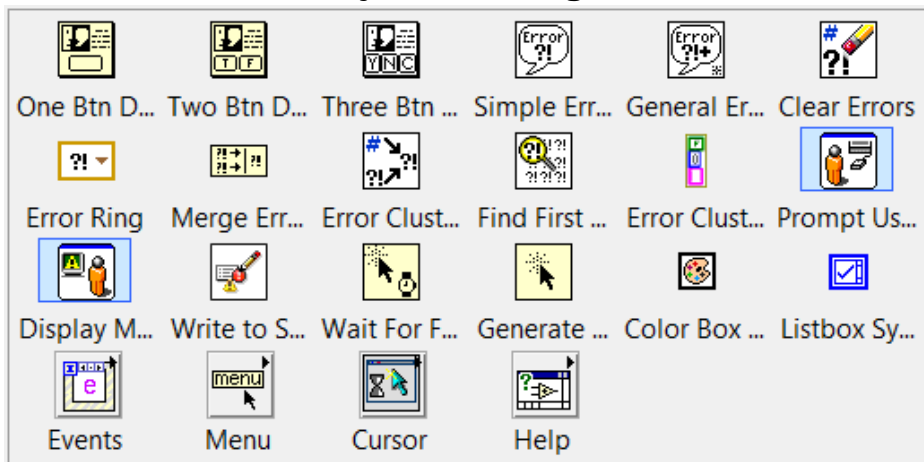
### Diagrama de bloques: paleta de funciones.

- Estructuras y elementos de programación:

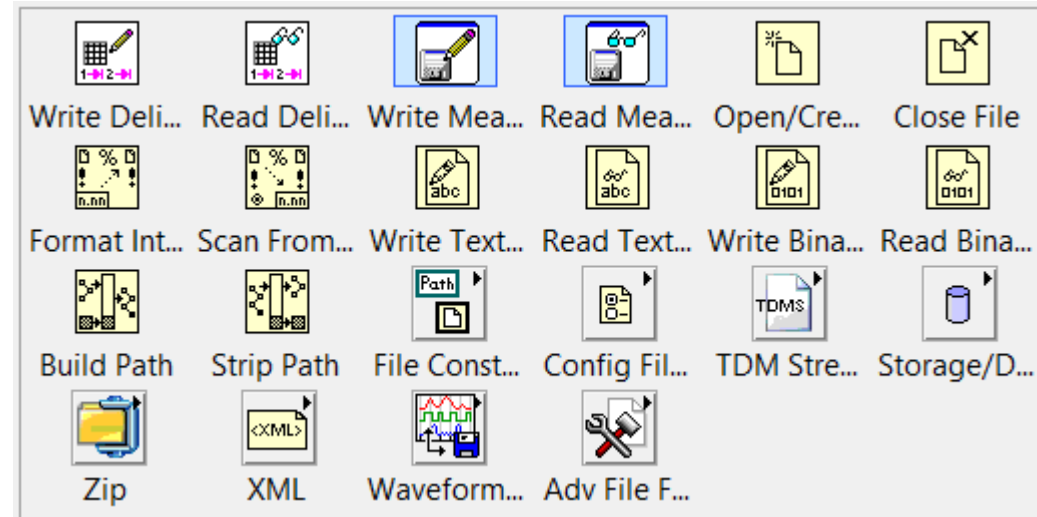
#### Temporización



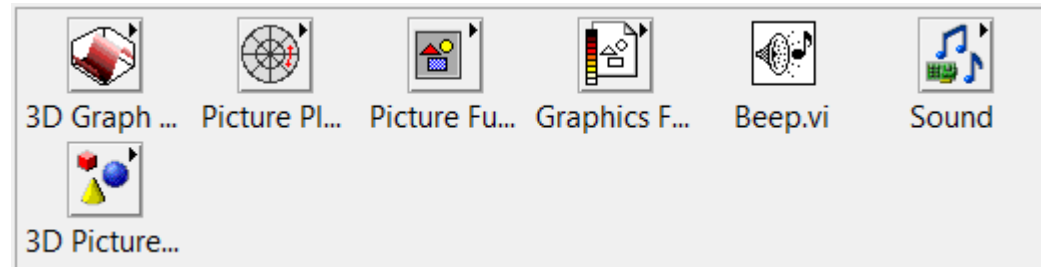
#### Cajas de diálogo



#### File I/O



#### Gráficos







### Diagrama de bloques: paleta de funciones.

- Librerías/Toolkits:

- ▶ Measurement I/O
- ▶ Instrument I/O
- ▶ Mathematics
- ▶ Signal Processing
- ▶ Data Communication
- ▶ Connectivity
- ▶ Control & Simulation
- ▶ Express
- ▶ Addons
- Select a VI...
- ▶ FPGA Interface
- ▶ MakerHub
- ▶ RF Communications
- ▶ TestStand

**Measurement I/O**

- NI-DAQmx
- TEDS
- System Co...
- Peer To P...

**Instrument I/O**

- DRIVERS
- Instr Drivers
- IVI Class D...
- Instr Asst
- VISA
- 488
- GPIB
- SERIAL
- Serial

**Mathematics**

- 123
- Numeric
- Elementary
- Linear Alg...
- Fitting
- Interp & E...
- Integ & Di...
- Prob & Stat
- Optimizat...
- Differenti...
- Geometry
- Polynomial
- Script & F...

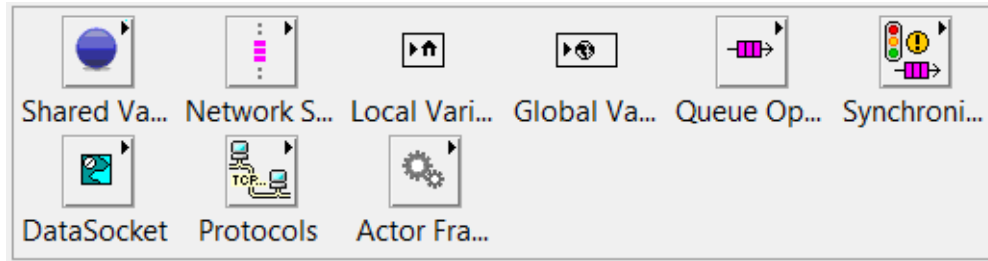
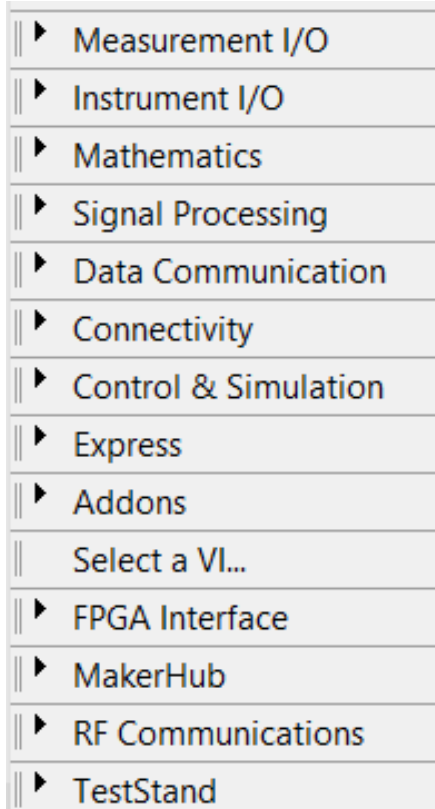
**Signal Processing**

- Wfm Gen...
- Wfm Con...
- Wfm Mea...
- Sig Gener...
- Sig Opera...
- Windows
- Filters
- Spectral
- Transforms
- Point By P...

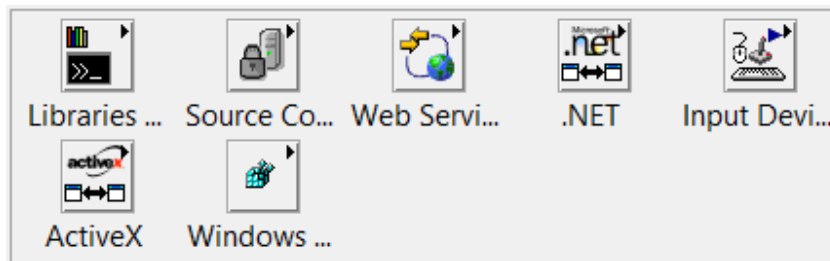


### Diagrama de bloques: paleta de funciones.

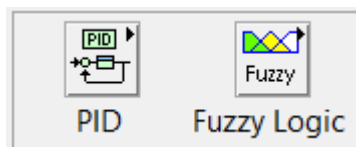
- Librerías/Toolkits:



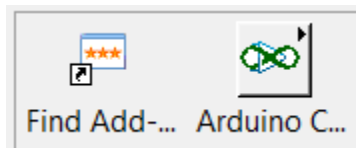
➔ Data Communication



➔ Connectivity



➔ Control & Simulation

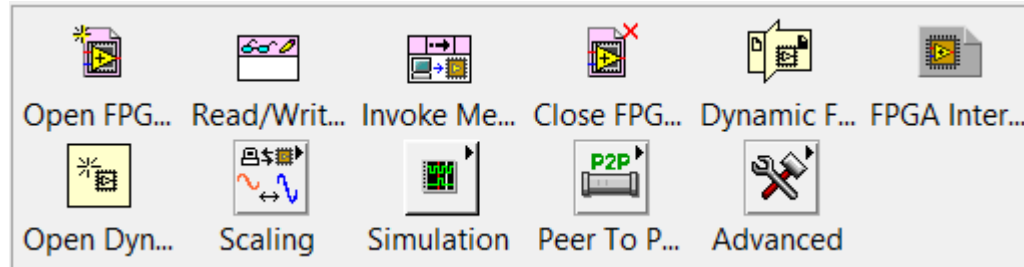


➔ Addons (p.e. Arduino)

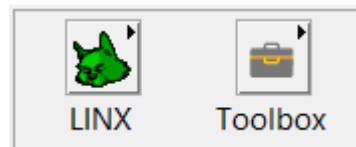


### Diagrama de bloques: paleta de funciones.

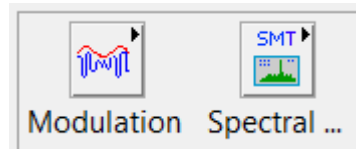
- Librerías/Toolkits:



→ **FPGA Interface**




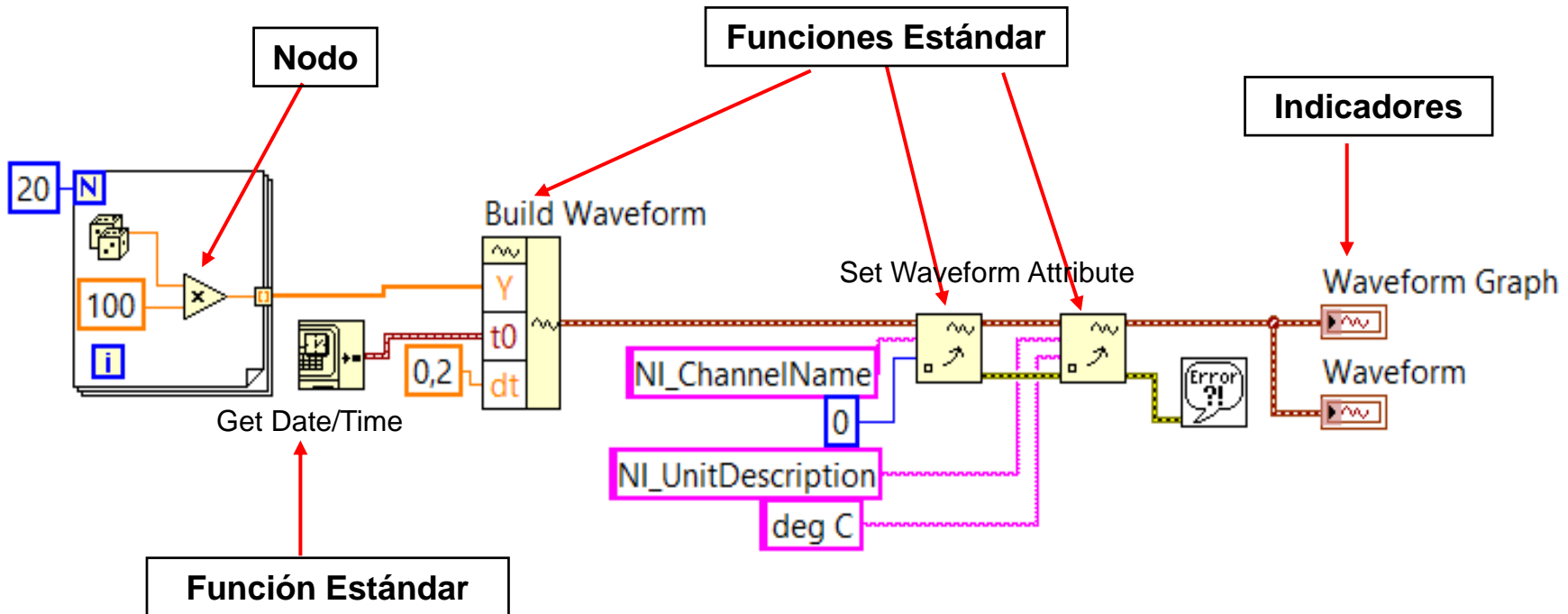
Librería externa *MakerHub* para programación de sistemas embebidos: Raspberri Pi, SparkFun, Arduino, BeagleBone Black, chipKIT™ Wi-FIRE™, etc.



**RF Communications**



**Diagrama de bloques:** Puede contener controles, indicadores, nodos, subví's, funciones estándar y funciones Express. 



- **Nodos:** son objetos que tienen entradas y/o salidas y realizan operaciones cuando se ejecuta el VI.
- **Funciones Estándar:** funciones de bajo nivel que en general proporcionan una salida en función de las entradas. Ejemplos: funciones para manipulación de array y string, funciones matemáticas, etc.

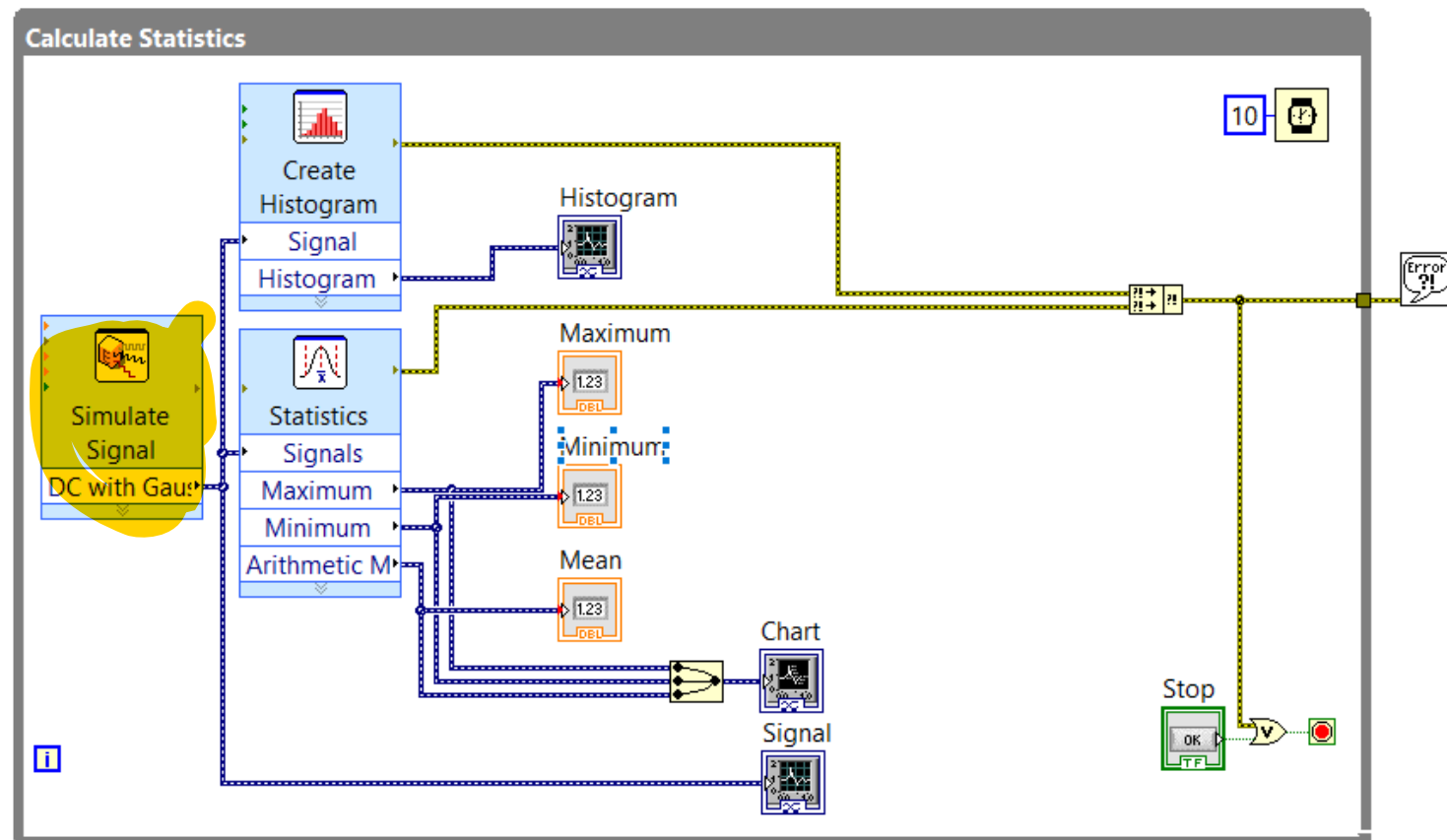


### Diagrama de bloques: Funciones vi Express.

- Las Funciones Express son funciones de alto nivel que permiten realizar tareas complejas de forma inmediata.
- Ejemplos: Generación de señal, cálculo de estadísticos, adquisición de datos, etc.

- Labview proporciona entorno a 40 funciones Express.

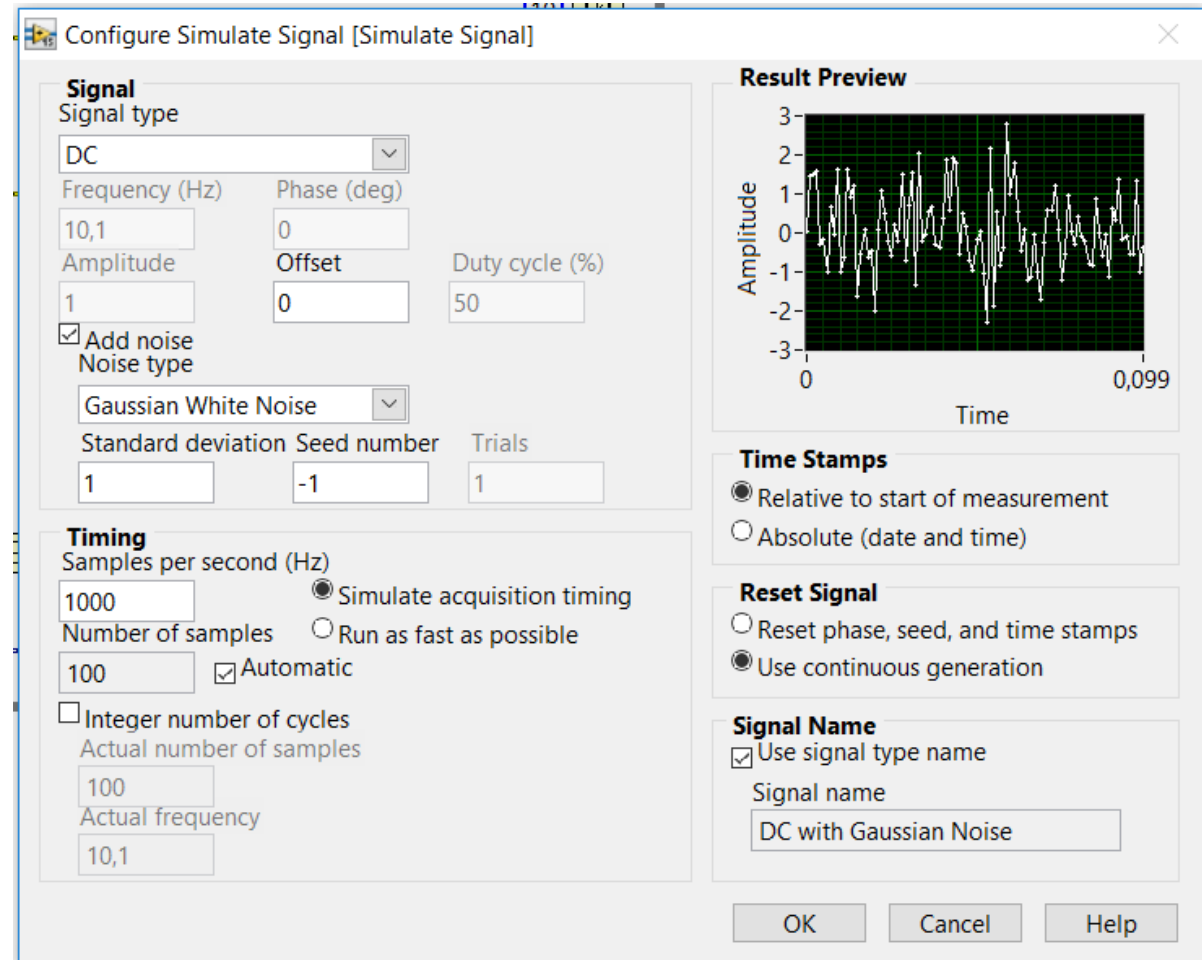
- Se caracterizan por su fondo de color azul.





### Diagrama de bloques: Funciones vi Express.

- Haciendo doble-click sobre la función o vi Express aparece una ventana que permite parametrizar la función.



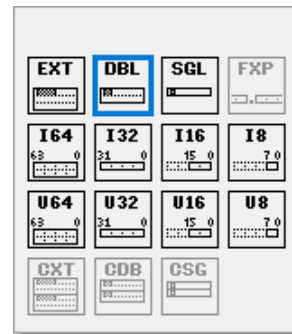


### Diagrama de bloques: tipos de datos.

- Cada control e indicador tiene un tipo de dato asociado:
- El color del cable indica el tipo de dato asociado:

- ✓ Numérico: Entero o real.
- ✓ Booleano: True o False.
- ✓ String: Secuencia de caracteres ASCII.

### Los datos numéricos soportan distintos formatos



	TIPO	COLOR	BITS	RANGO
EXP	REAL-Extendido	Naranja	128	±1.19e+4932
DBL	REAL-Doble	Naranja	64	±1.79e+308
SGL	REAL-Simple	Naranja	32	±3.4e+38
I32	Entero-LONG	Azul	32	±2147483647
I16	Entero-WORD	Azul	16	-32768..+32.67
I8	Entero-BYTE	Azul	8	-128..+127
U32	NATURAL	Azul	32	0..4294'967.295
U16	NATURAL	Azul	16	0..65.535
U8	NATURAL	Azul	8	0..255
CXT	COMPLEJO	Naranja	2*128	2*±1.19e+4932
CBD	COMPLEJO	Naranja	2*64	2*±1.79e+308
CSG	COMPLEJO	Naranja	2*32	2*±3.4e+38



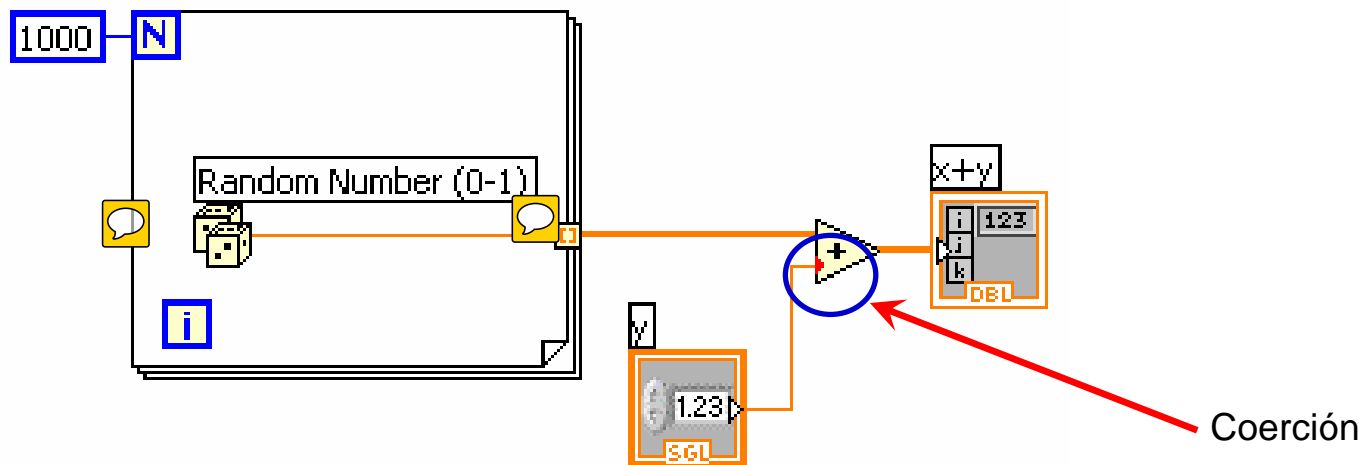


### Diagrama de bloques: tipos de datos.

- **Puntos de coerción (conversión de tipos):** los puntos de *coerción* indican que hay dos tipos de datos numéricos de diferente tipo unidos al mismo punto.

Las *coerciones* en Labview:

- Requieren copia de datos.
- En determinados casos se requieren mucha memoria, como en el siguiente caso en el que se deben almacenar todos los datos aleatorios que se van generando.

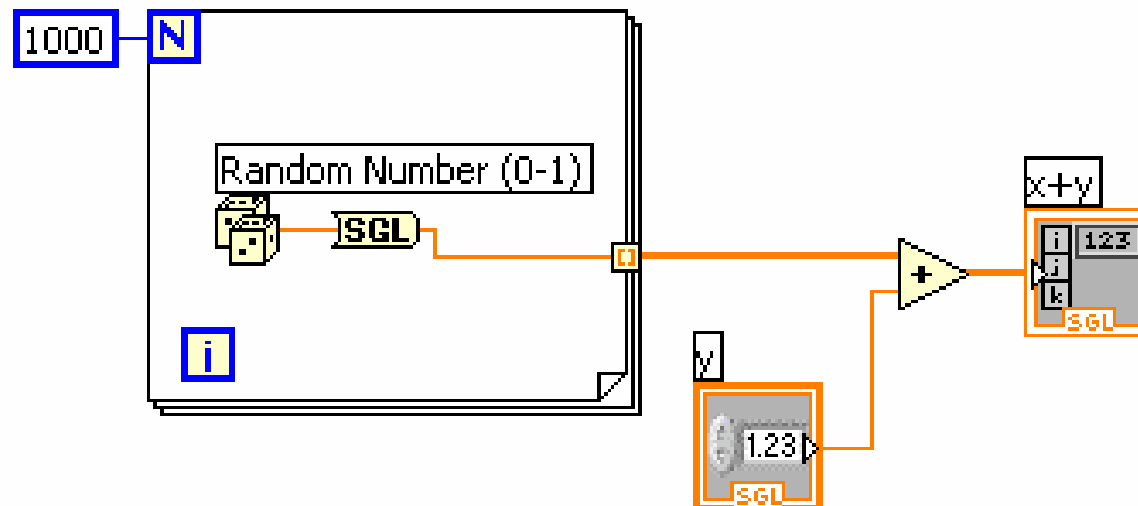






### Diagrama de bloques: tipos de datos.

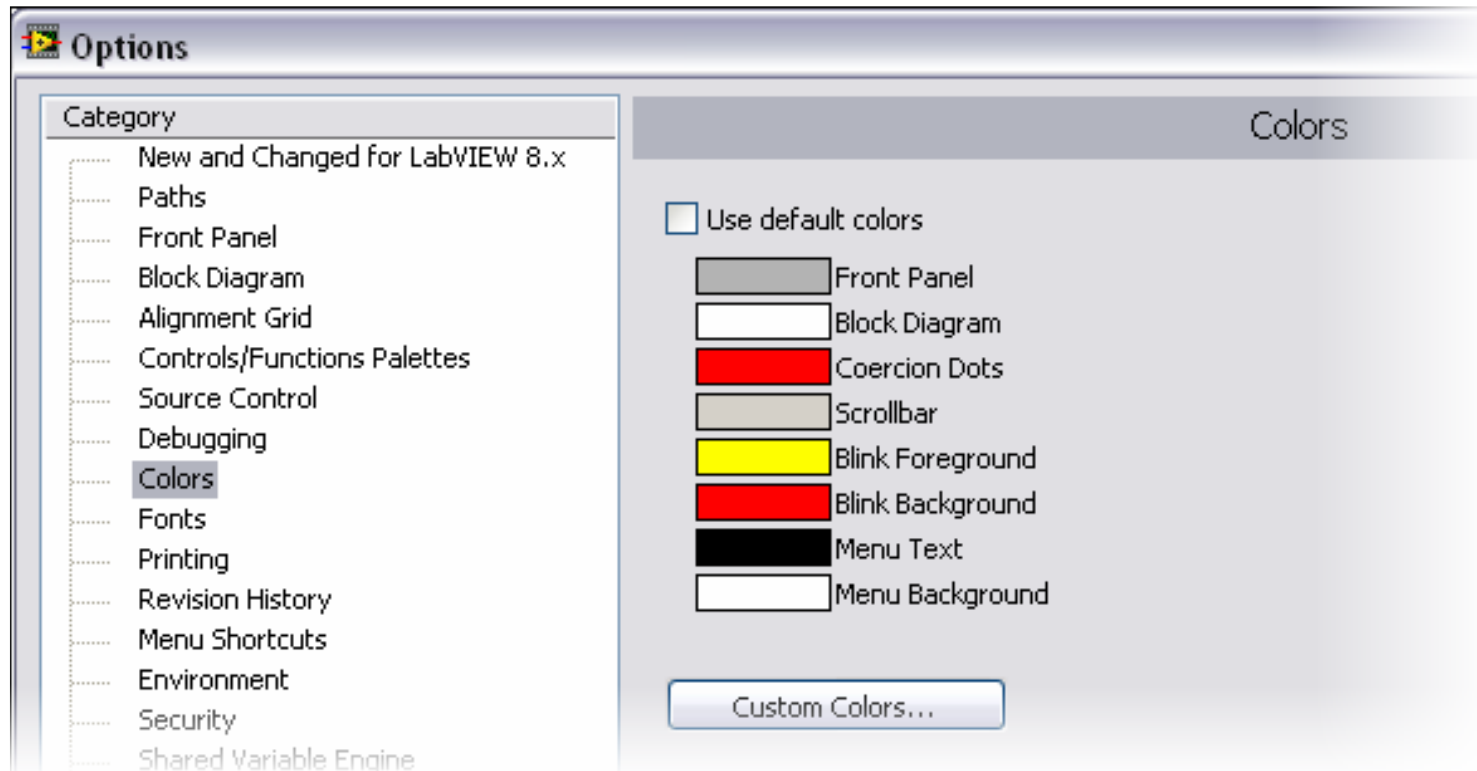
- Para evitar *coerciones* la mejor solución es realizar conversiones de tipos con los nodos disponibles para ello:
- En el ejemplo anterior se convertiría el número aleatorio a medida que se va creando, de esta manera se evita la conversión de un gran búfer de datos.





### Diagrama de bloques: tipos de datos.

- Configurando Puntos de Coerción → **Tools>>Options>>Colors to change coercion dot color**





### Paleta de herramientas (*Tools Palette*):



Tool	Icon	Description
Automatic Tool Selection		Automatically choose the appropriate tool
Operating Tool		Change the value of a control or select the text within a control
Positioning Tool		Position, resize, and select objects
Labeling Tool		Edit text and create free labels
Wiring Tool		Wire objects together on a block diagram

Scrolling Tool		Scroll the window without using the scroll bars
Breakpoint Tool (Used for debugging)		Set breakpoints on VIs, functions, wires, loops, sequences, and cases
Probe Tool (Used for debugging)		Create probes on wires and display intermediate values on a wire in a running VI
Get Color Tool		Copy colors for pasting with the Color Tool
Coloring Tool		Set the foreground and background colors



### Técnicas de Edición.

- En el panel frontal se puede “conmutar” de la herramienta de posición a la de operación pulsando la barra espaciadora.
- En el diagrama de bloques se puede conmutar de la herramienta de posición a la de cableado pulsando la barra espaciadora.
- Se puede conmutar entre elementos de la paleta de herramientas pulsando <Tab>.
- Para realizar copias de objetos tanto en el panel frontal como en el diagrama de bloques basta con pulsar <Ctrl> y arrastras el objeto hasta el sitio deseado.
- <Ctrl+E> → Permite conmutar del panel frontal al diagrama de bloques.




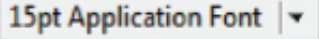






### Técnicas de Edición.

#### Keyboard Shortcuts

File					
Ctrl-N	Create new VI	Ctrl-Z	Undo last action	Right-Click	Display controls/ functions palette
Ctrl-S	Save VI	Ctrl-Shift-Z	Redo last action	Shift-Right-Click	Display tools palette
Ctrl-P	Print	<u>Operate</u>		Ctrl-T	Tile block diagram and front panel windows
Edit			Ctrl-R	Run VI	
Ctrl-V	Paste object	Ctrl-.	Abort VI		
Ctrl-U	Clean up diagram	<u>Window</u>		Ctrl-H	Display context help
Ctrl-Space	Activate quick drop	Ctrl-E	Display block diagram/ front panel		
<b>Ctrl-B</b>	<b>Remove broken wires</b>				
Ctrl-C	Copy an object				
Ctrl-X	Cut object				

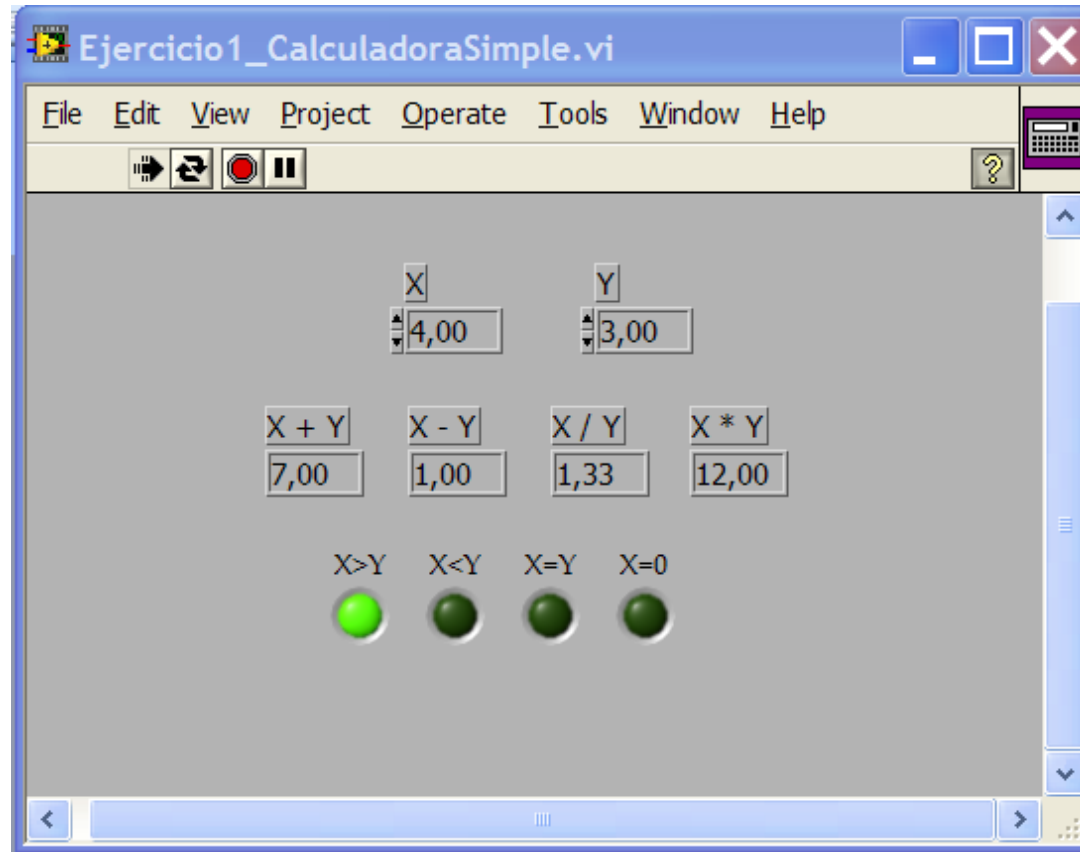


### Técnicas de Edición.

Editing Tools		
Tool	Icon	Description
Show Context Help		Display the context help window
Text Settings		Change the font setting for the VI, including size, style, and color
		
Align Objects		Align selected objects
Distribute Objects		Space objects evenly
Resize Objects		Resize multiple front panel objects to the same size
Reorder		Reorder the layers of the objects
Clean Up Diagram		Rearrange wires and objects on the block diagram
Enter		Appears when a new value is available to replace an old value



### Ejercicio 1: Calculadora

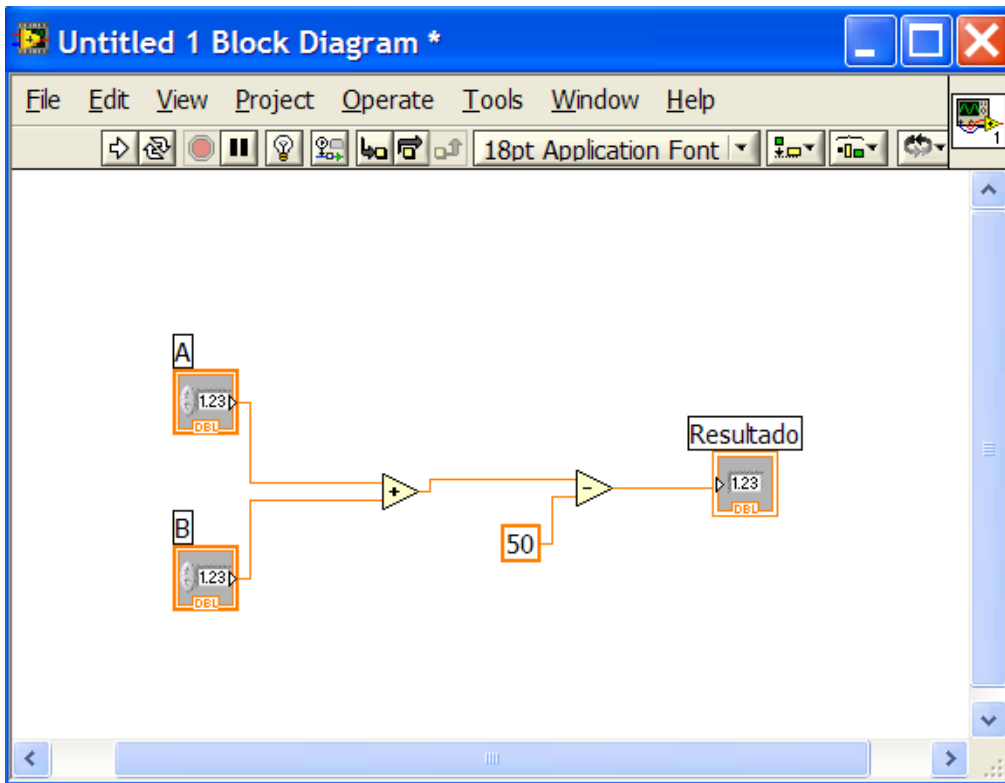




### Flujo de programa

- En Labview NO existe un orden establecido (Derecha a Izquierda, arriba-abajo, etc.) para la ejecución del código: **un nodo o función del diagrama de bloques se ejecuta cuando todas sus entradas están disponibles.**

### Ejemplo 1:



Este código se ejecuta de izquierda a derecha por el hecho de que hasta que no se disponga del resultado de la suma no se puede realizar la resta

### Trucos Cableado:

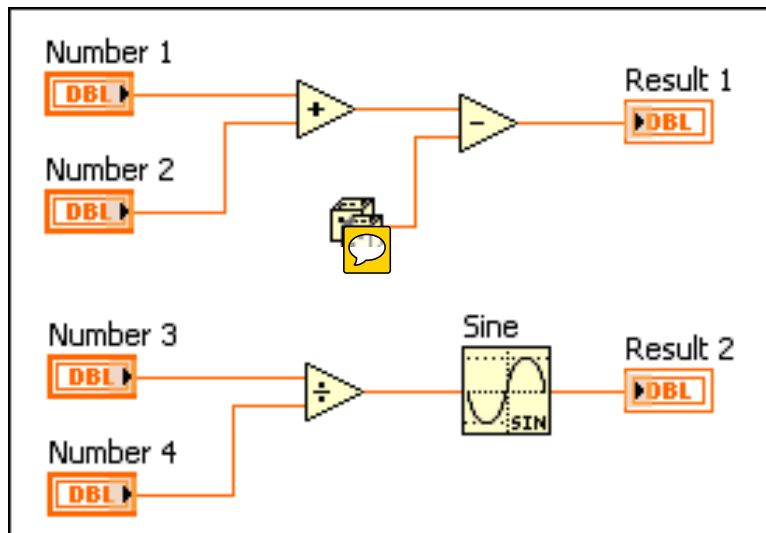
- Utilizar la barra espaciadora para cambiar la dirección del cableado.
- Para eliminar cables erróneos que aparecen marcados con línea discontinua: <Ctrl+B>.





### Flujo de programa (cont.)

#### Ejemplo 2:



¿Qué se ejecuta primero, la suma, la resta o la división?





### Depuración.

Debugging Tools		
Tool	Icon	Description
Run		Execute the VI
List Errors		List errors that prevent the VI from running
Run Continuously		Execute the VI continuously until abort or pause is pressed
Abort Execution		Stop VI execution immediately
Execution Highlighting		Animate data movement on the block diagram wires
Pause		Temporarily stop execution to debug a portion of the VI
Step Into		Single-step into a subVI or structure to debug it
Step Over		Execute a subVI or structure and pause at the next one
Step Out		Execute a subVI or structure and resume single-stepping



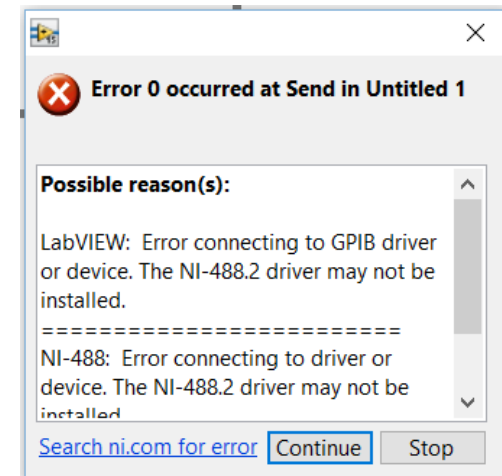
### Depuración: evitar *warnings* durante el desarrollo de una aplicación.

- Las herramientas de programación pueden funcionar en modo depuración (*debug*) o en modo ejecutable (*release*).
- Cuando estamos desarrollando código por defecto la herramienta se encuentra en modo depuración (*debug*) de manera que nos avisa de posibles anomalías (*warnings*) que nada tienen que ver con errores en el código.

#### Ejemplos:

1. Se intenta acceder a un dispositivo no conectado al ordenador. El código es correcto pero al no encontrarse conectado el dispositivo, la aplicación genera un *warning* o aviso para indicarnos esta circunstancia.
  2. Variables declaradas que no se utilizan → *warning: unused variable `...`*
  3. Se invoca a una función no definida en ningún sitio → *undefined reference to `...`*
- En modo ejecución la aplicación no avisa de estas posibles anomalías.

En modo depuración, cuando se ejecuta la aplicación y se produce un *warning* esta pausa su ejecución y nos avisa mediante un *pop-up* dando la opción de continuar o finalizar la aplicación.

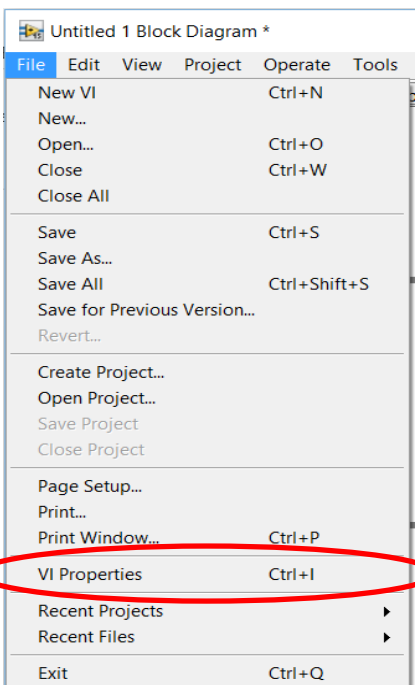




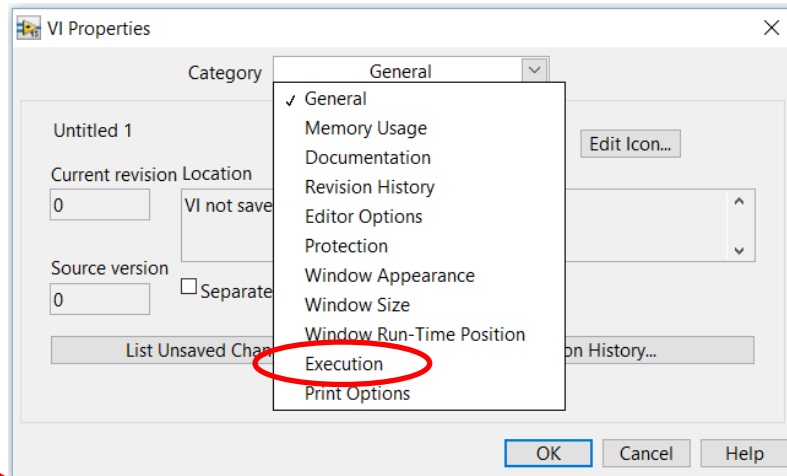
### Depuración: evitar *warnings* durante el desarrollo de una aplicación.

- Para cambiar el modo de ejecución (modo ejecutable a depuración o viceversa) en Labview:
  1. Seleccionar **VI Properties** del menú File del vi.
  2. Aparece una nueva ventana en la que se pueden modificar las propiedades del vi. Las propiedades del vi aparecen clasificadas por categorías. Seleccionar la categoría **Execution**.
  3. Para deshabilitar el aviso de *warnings* desmarcar la **checkbox *Enable automatic error handling***.

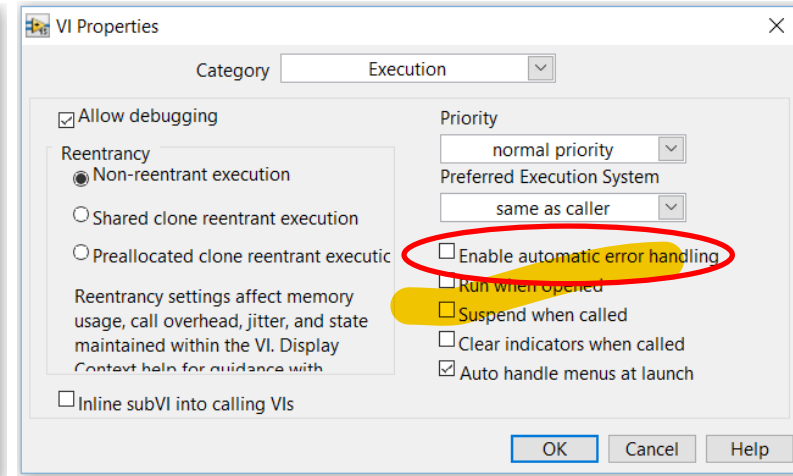
1



2



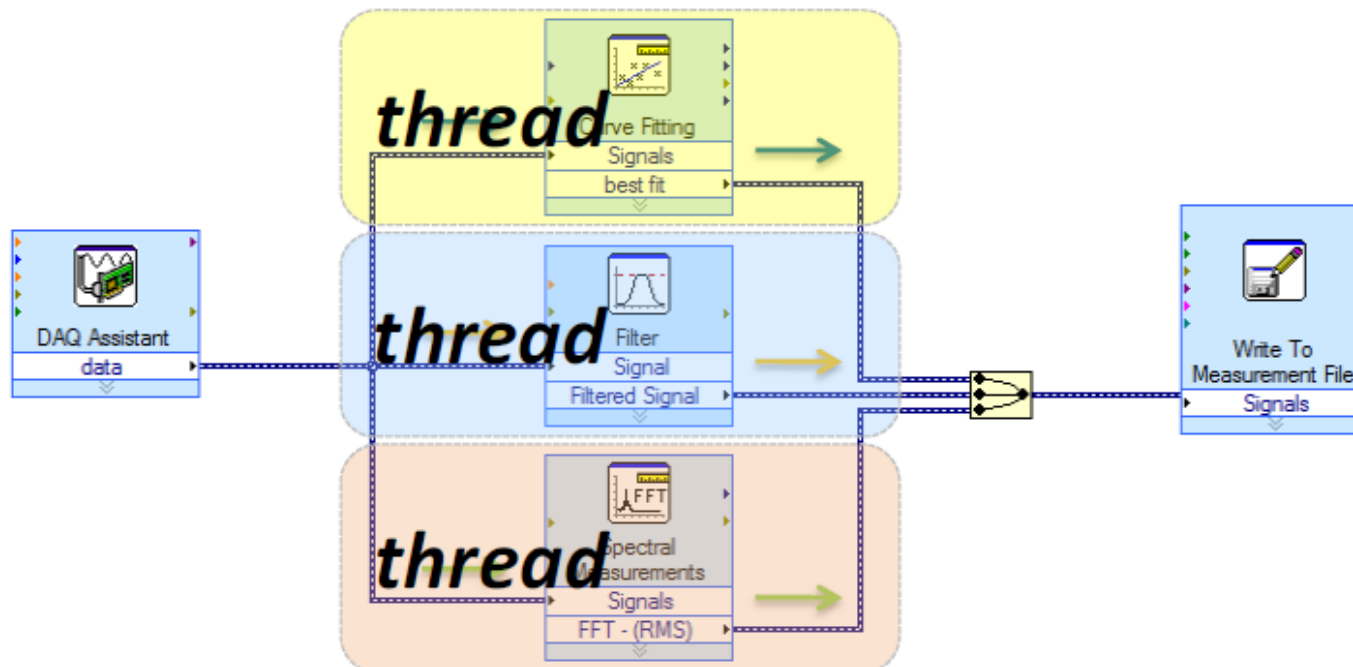
3





### Multihilo.

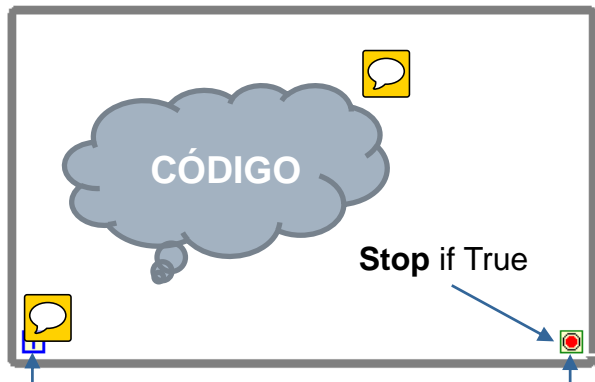
- Labview divide automáticamente el código de cada aplicación en múltiples hilos.
- Divide automáticamente cada programa en dos hilos: un hilo para la gestión de interface de usuario y otro hilo para la ejecución del código para que el Sistema operativo pueda ejecutar la aplicación de manera más efectiva.
- El usuario no debe poseer conocimientos sobre la compleja programación multihilo para beneficiarse de sus ventajas.



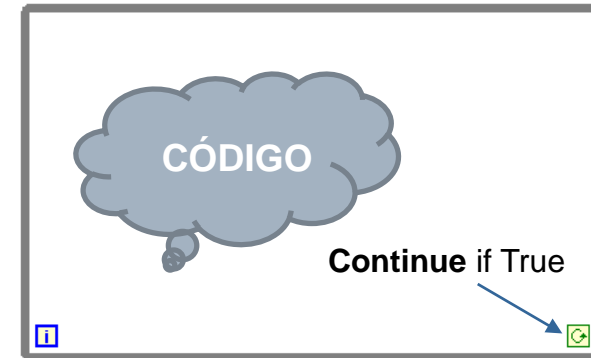
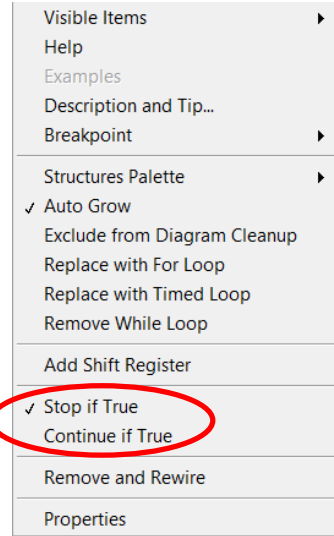


### Bucles: While.

- Ejecuta el subdiagrama contenido en el rectángulo hasta que la condición indicada en el terminal condicional se cumple.

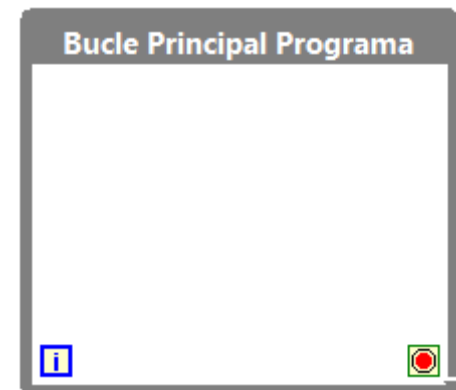
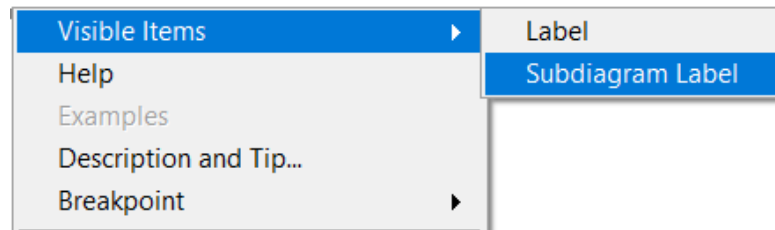


Click botón dcho. ratón



Se puede añadir una etiqueta al while seleccionando:

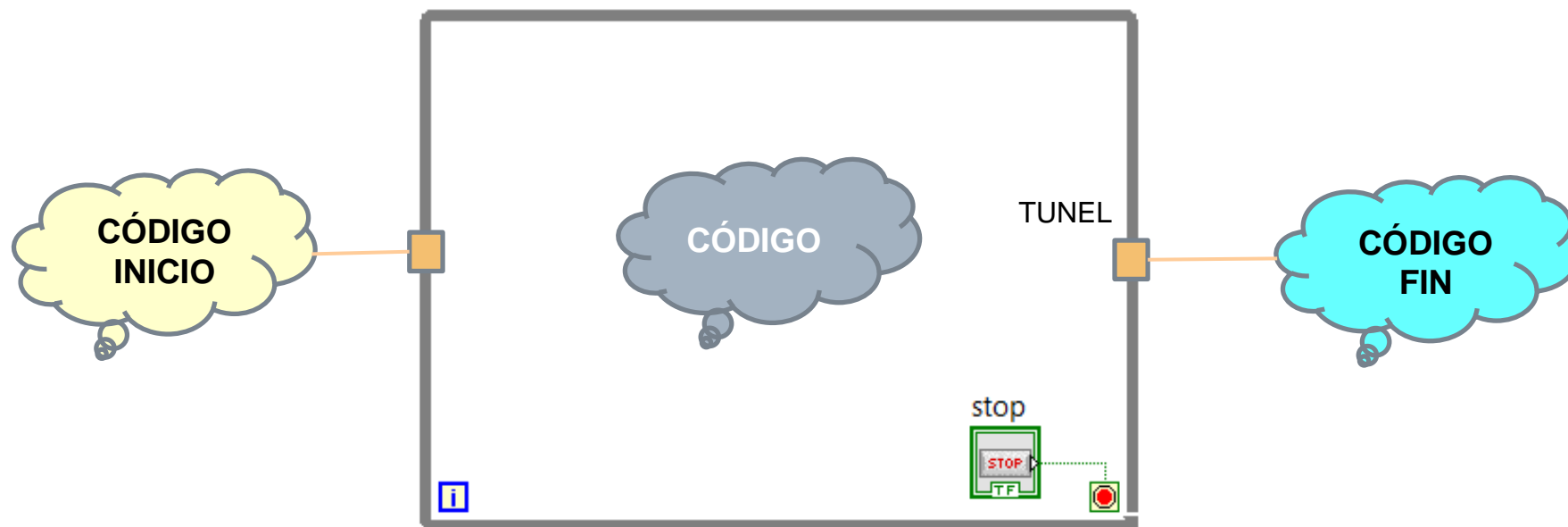
*Visible Items* → *Subdiagram Label*





### Bucles: While.

- Las aplicaciones Labview basan su ejecución en un código que se encuentra dentro de un bucle While gobernado con un botón de fin de aplicación (stop).

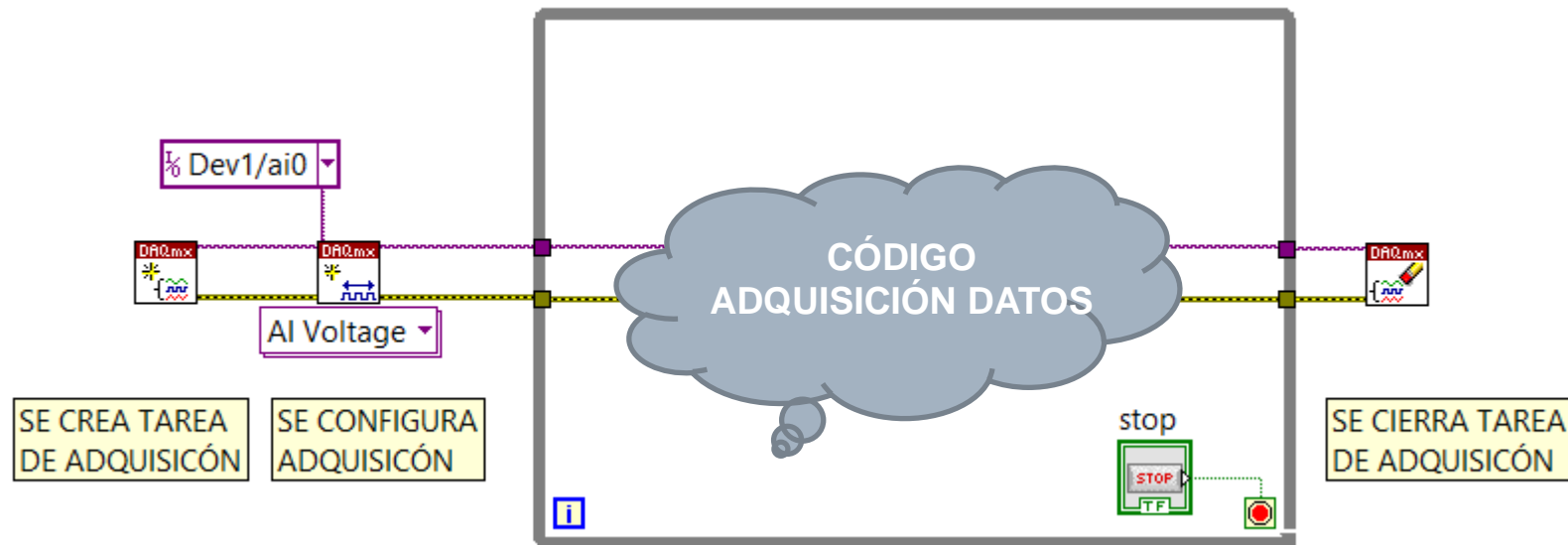


- Se puede secuenciar para que se ejecute un código inicial antes del bucle principal del programa y otro una vez que finaliza este.



### Bucles: While.

- El código inicial (CÓDIGO INICIO) suele contener inicializaciones de variables y configuraciones o inicialización de parámetros de los diferentes elementos hardware externos gobernados por el programa principal. Por ejemplo, apertura de puerto serie, inicialización de tarjetas de adquisición, inicialización de sistemas embebidos externos, etc.
- El código final (CÓDIGO FIN) suele contener el cierre de drivers de diferentes elementos hardware gobernados por el programa principal para liberar recursos: cierre de puertos serie, cierre de drive de tarjetas de adquisición, etc.
- **Ejemplo aplicación de adquisición de datos:**



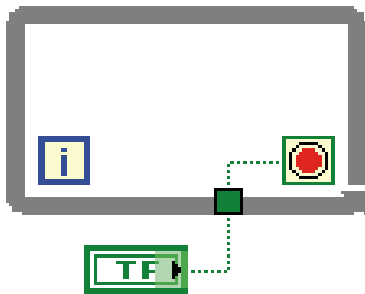




### Bucles: While.

- **Bucles infinitos:** error muy típico en programación que implica que un bucle no finalice nunca.
- **Ejemplos:**

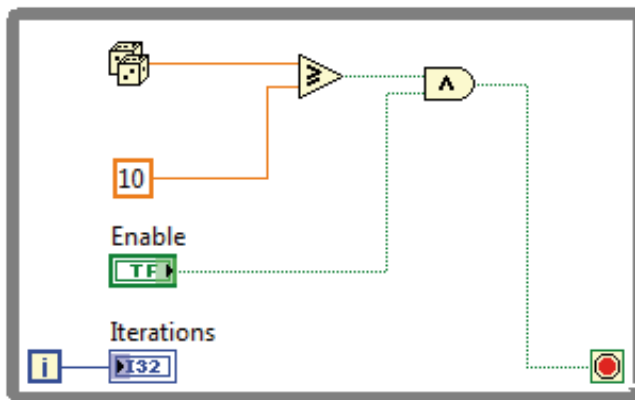
1



Si el control booleano que controla la finalización del bucle tiene valor FALSE cuando se inicia la aplicación se produce un bucle infinito.

Aunque el control booleano cambie de estado, ya no se leerá por estar ubicado fuera del bucle.

2



El bucle se ejecuta hasta que la salida del subVI *random* sea mayor o igual que 10 **y (and)** el control Enable sea True.

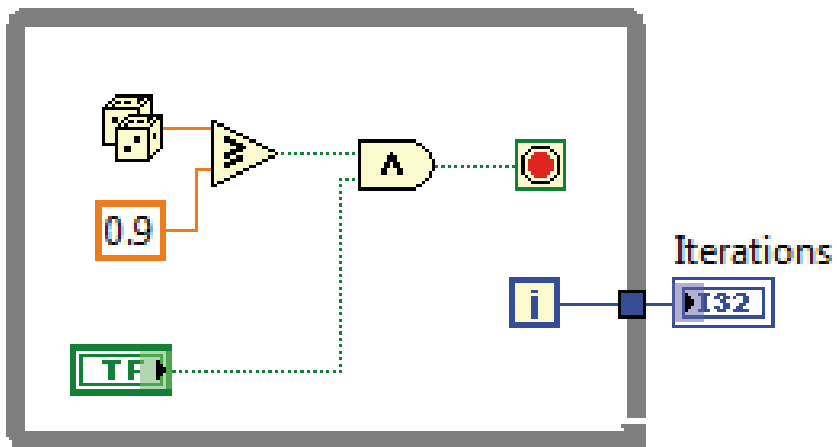
Como el subvi *random* solo genera números entre 0 y 1, el bucle no finalizará nunca.



### Bucles: While (Cont.)

#### Túneles:

- Permiten pasar datos desde el interior al exterior del bucle.
- Los datos que se pasan a través de los túneles no “salen al exterior” hasta que finaliza el bucle while.
- El túnel aparece de forma automática en el momento en que cualquier cable cruza el límite del rectángulo correspondiente al while.
- **Ejemplo:**

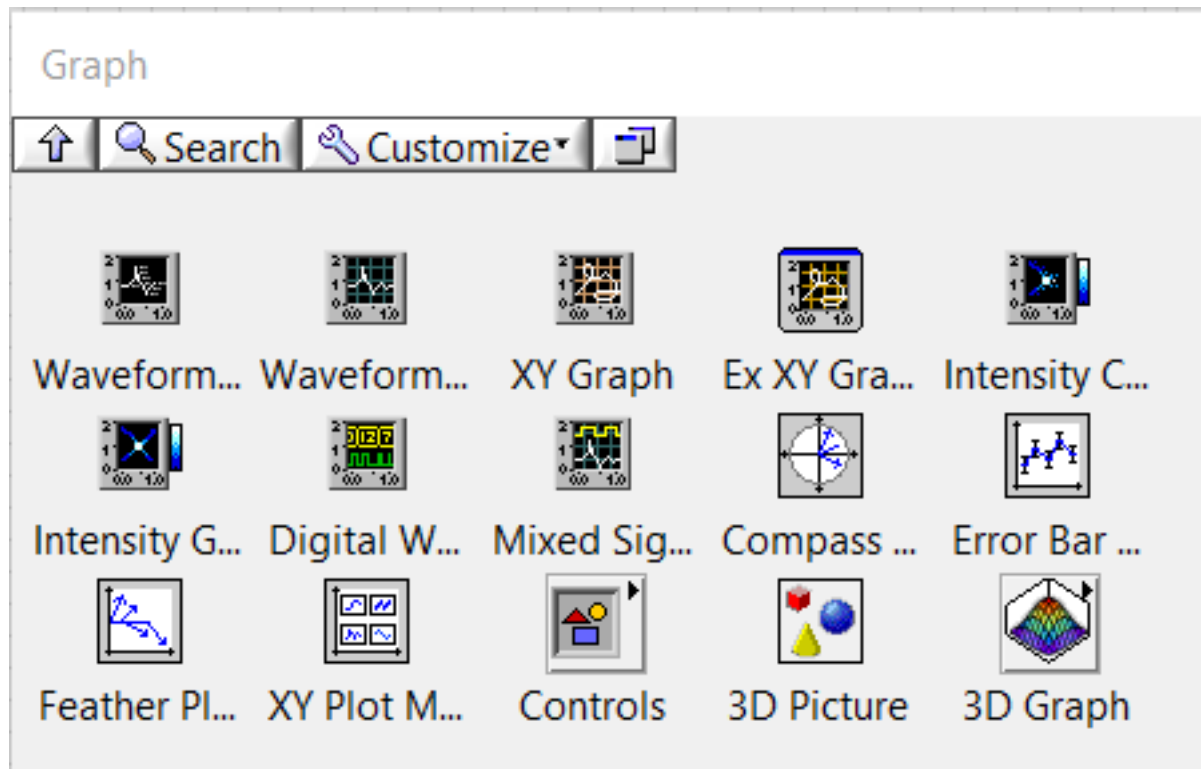


El número de iteraciones NO pasa al indicador **Iterations** hasta que finaliza el bucle.



### Gráficos.

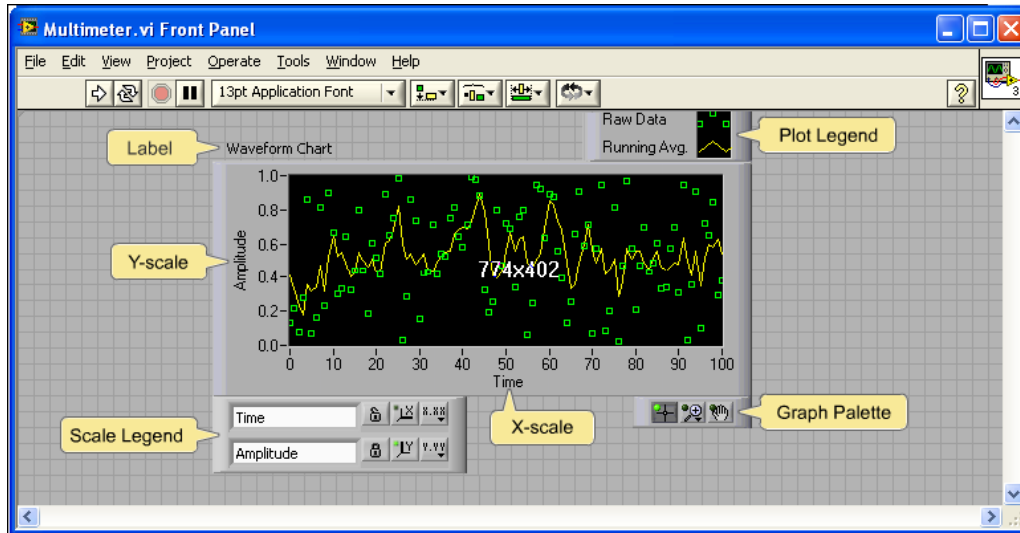
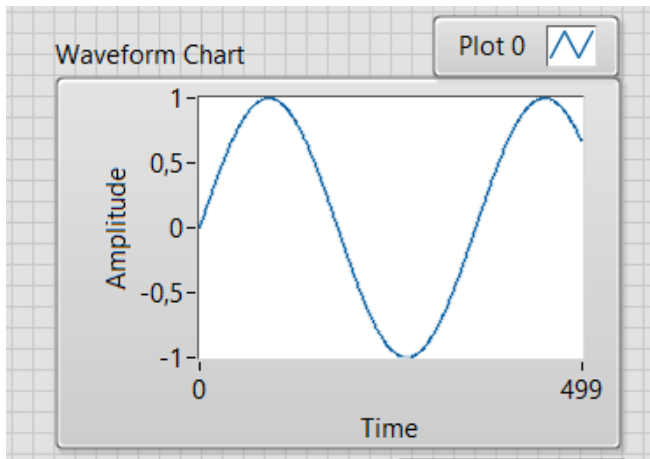
- Proporciona diferentes tipos de gráficos (2D, 3D, gráficos de intensidad, etc.).
- A los gráficos se accede desde el panel frontal:





### Gráficos: Gráfico Waveform Chart 2D.

- Útil para representar datos de manera continuada.



- Se pueden representar varias señales en un mismo gráfico utilizando el elemento **Bundle** desde *Funtions* → *Cluster & Variants*



accesible





### Gráficos: Gráfico Waveform Chart 2D.

• En un gráfico de tipo chart se puede representar los siguientes tipos de datos:

1. **Escalares:** se puede representar punto a punto.
2. **Arrays de 1D:** array que contiene varios puntos.
3. **Array 2D:** array que contiene puntos de varias señales. Para esta funcionalidad es necesario “mezclar” estas señales mediante la función *Bundle*.
4. **Waveform Data (WDT):** se trata de un tipo de datos específico utilizado por Labview y software de National Instruments (SignalExpress, TDMS, DAQmx, etc.) para almacenar información de señales periódicas.

Consiste en un cluster que contiene la siguiente información:

Field	Value
t0	00:00:00 DD/MM/YY
dt	1,00m
Y	0 0,062791 0,125333 0,187381 0,24869

**t0** → Se trata de una marca de tiempo (*timestamp*) que indica el momento en el que comienza la señal.

**dt** → se trata de un dato de tipo numérico que indica el intervalo o diferencia de tiempo entre dos muestras consecutivas de la señal almacenada en el WDT.

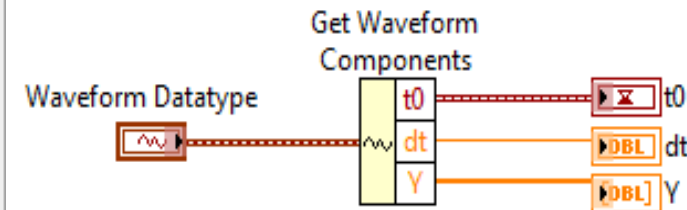
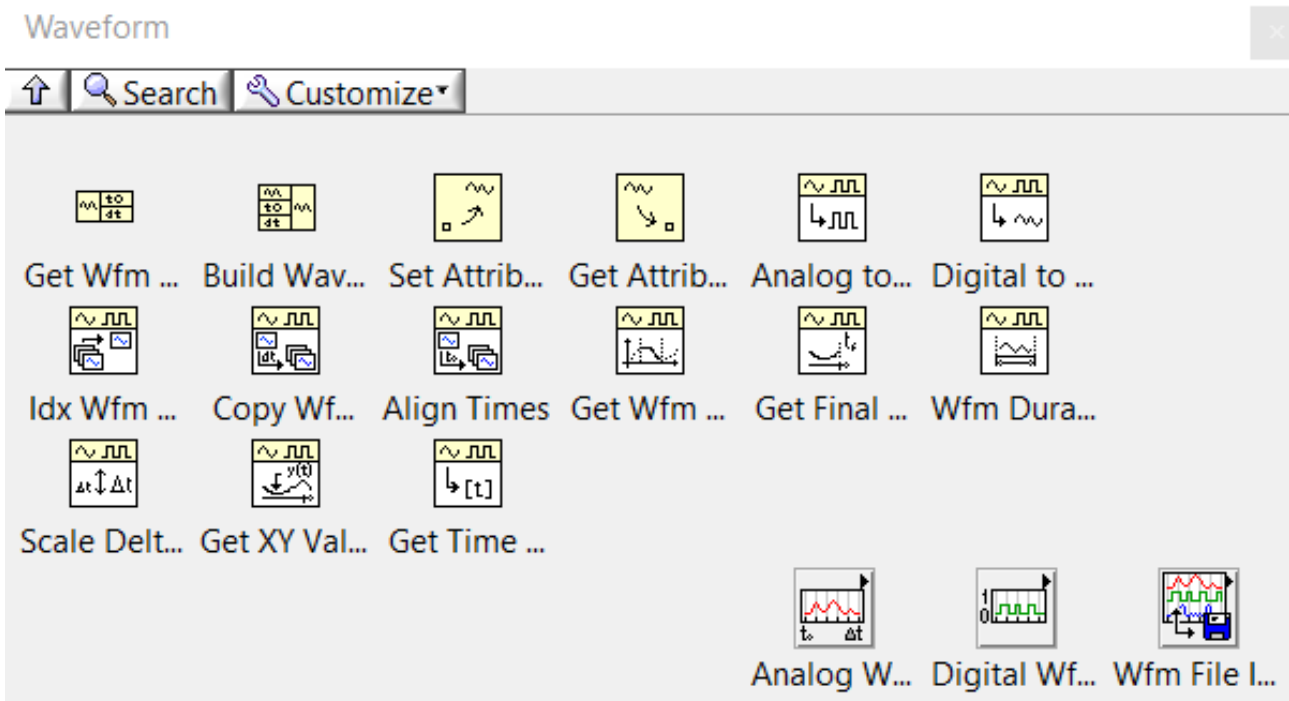
**Y** → array 1D que contiene los valores de las muestras de la señal almacenada.



### Gráficos: Gráfico Waveform Chart 2D.

#### 4. Waveform Data (WDT) (Cont.):

- Los cluster WFD tienen su propia paleta de funciones para su gestión. Se trata de la paleta Waveform accesible desde el diagrama de bloques:



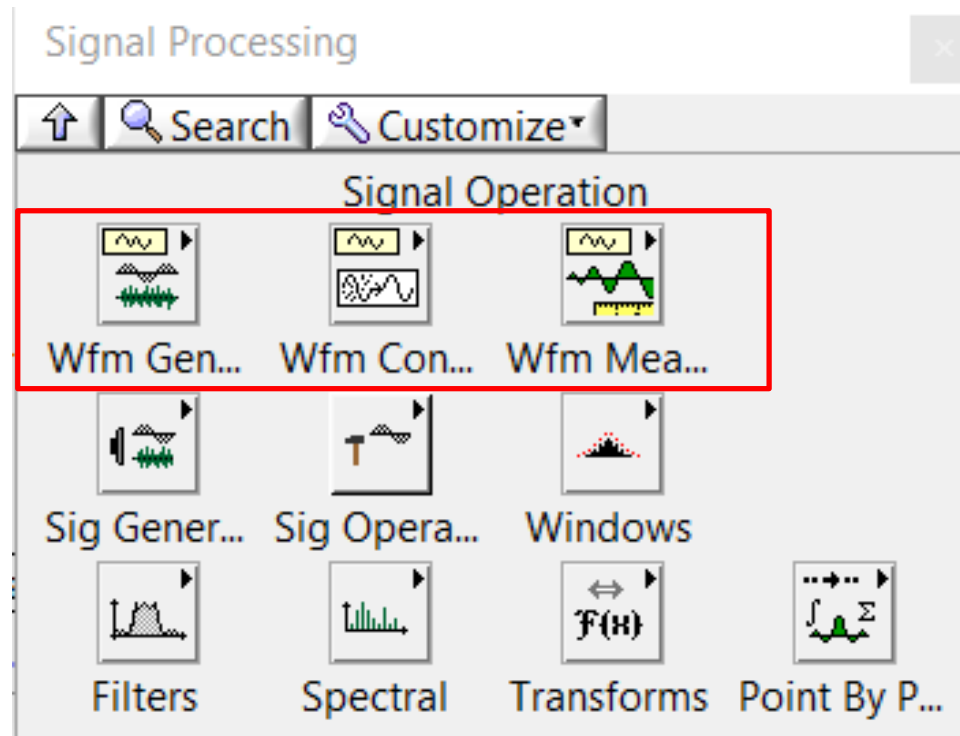


### Gráficos: Gráfico Waveform Chart 2D.

#### 4. Waveform Data (WDT) (Cont.):

- Existe un conjunto de vi's para generación, acondicionamiento y medida de señales en formato WDT accesibles desde la librería **Signal Processing** o desde la paleta **programming** → **waveform** del diagrama de bloques:

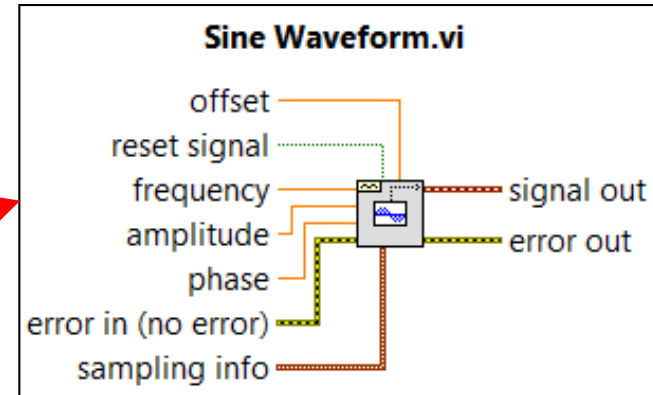
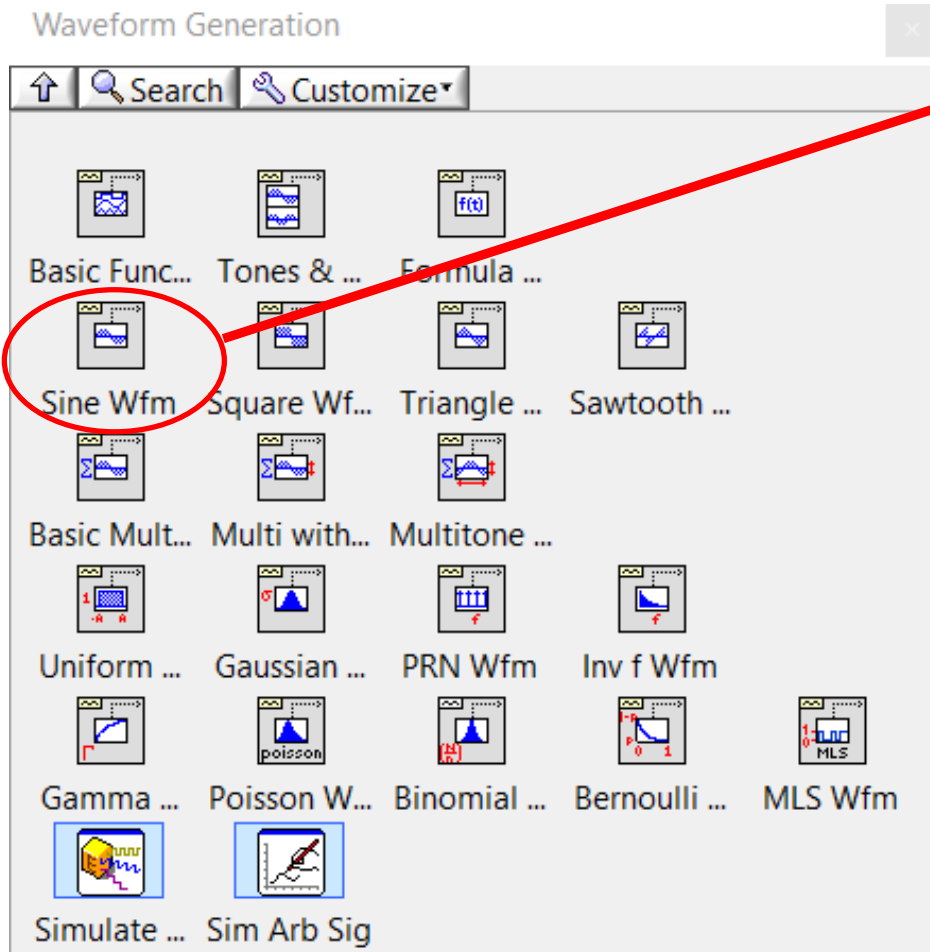
1. Waveform Generation.
2. Waveform Conditioning.
3. Waveform Measurement.





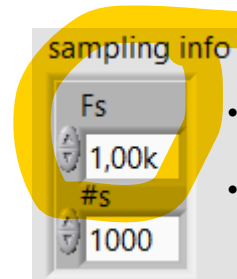
### Gráficos: Gráfico Waveform Chart 2D.

#### 4. Waveform Data (WDT) (Cont.):



Siendo:

- **frequency**: frecuencia de la señal generada,  $F_{\text{señal}}$ .
- **amplitude**: amplitud de la señal generada
- **phase**: fase de la señal generada.
- **offset**: componente continua (DC) de la señal generada.
- **reset signal**: si es *true* resetea la fase de la señal generada y la marca de tiempos (*timestamp*). Su valor por defecto es *false*.
- **sampling info**: se trata de un cluster que contiene información del muestreo de la señal generada:



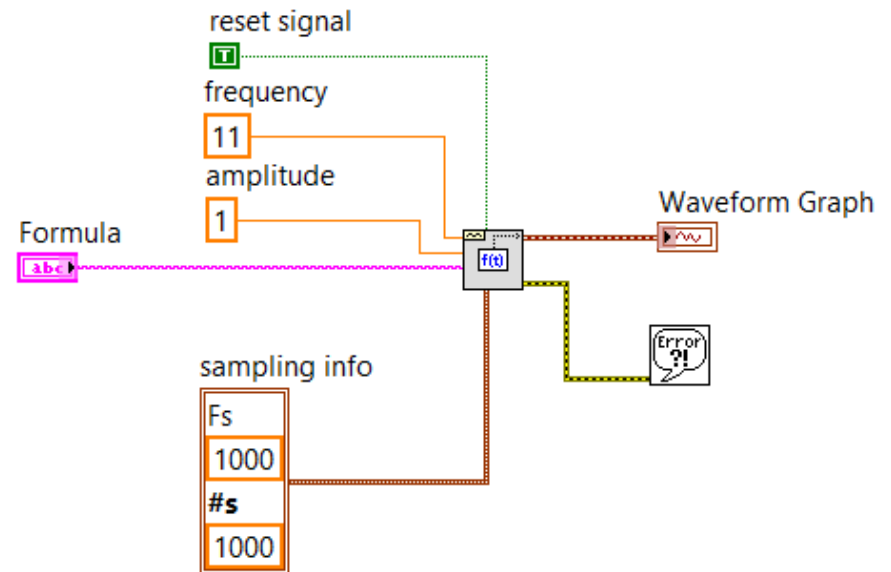
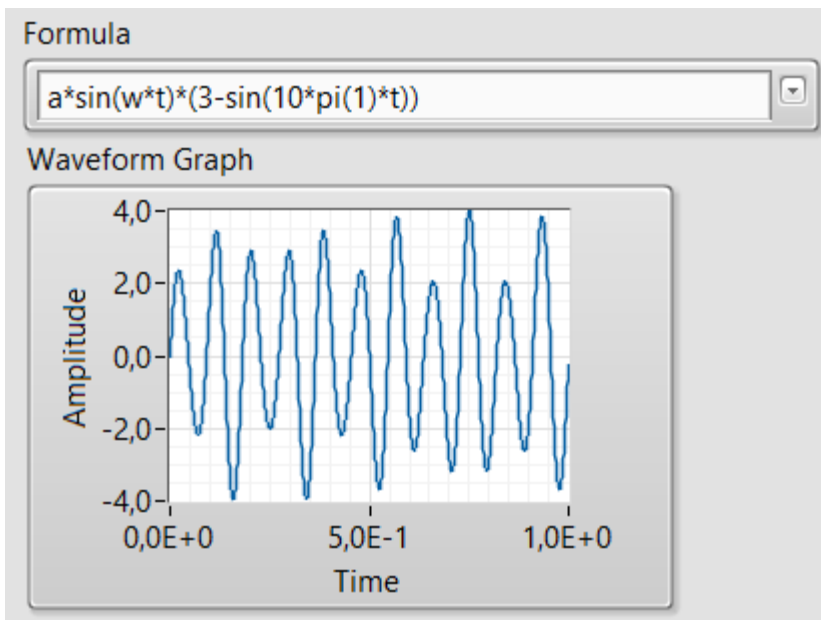
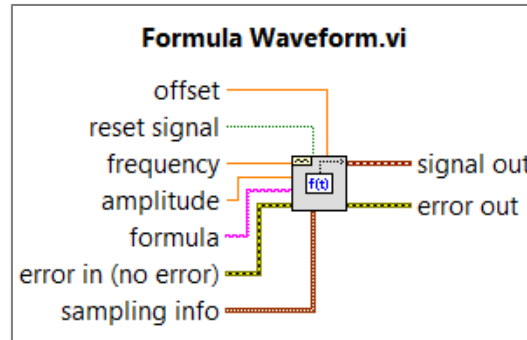
- **Fs**: frecuencia de muestreo en muestras/segundo. Valor por defecto 1000.
- **#s**: Número de muestras a generar. Valor por defecto 1000.





### Gráficos: Gráfico Waveform Chart 2D.

#### 4. Waveform Data (WDT) (Cont.):

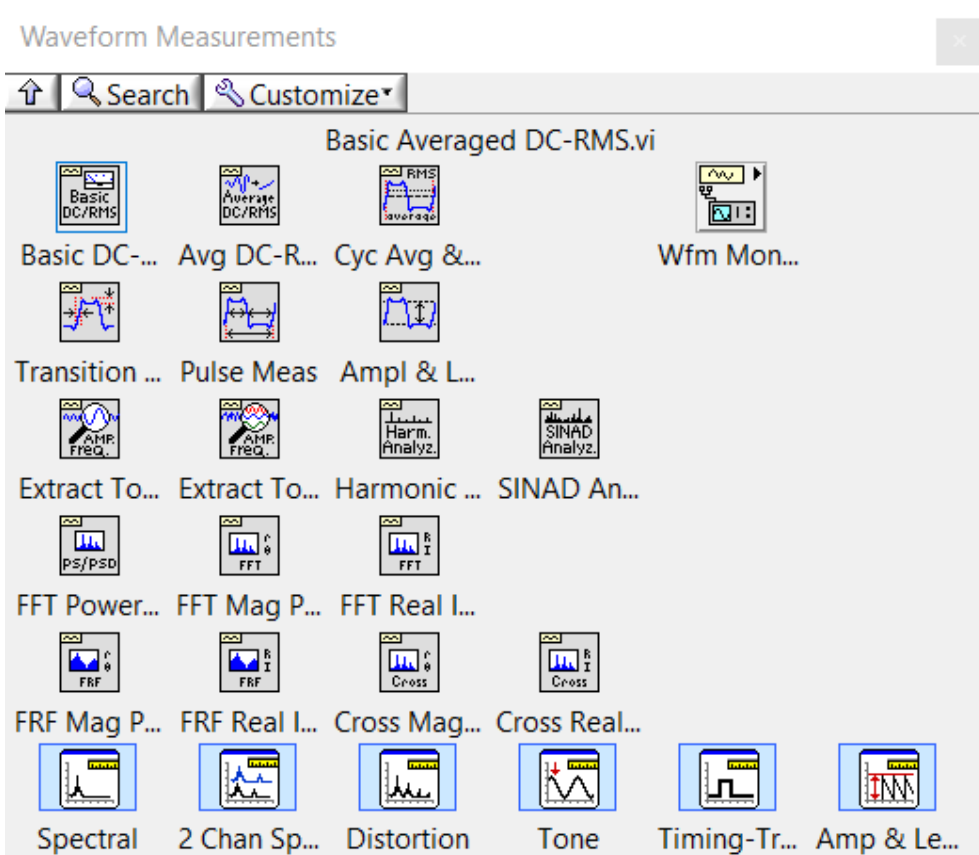




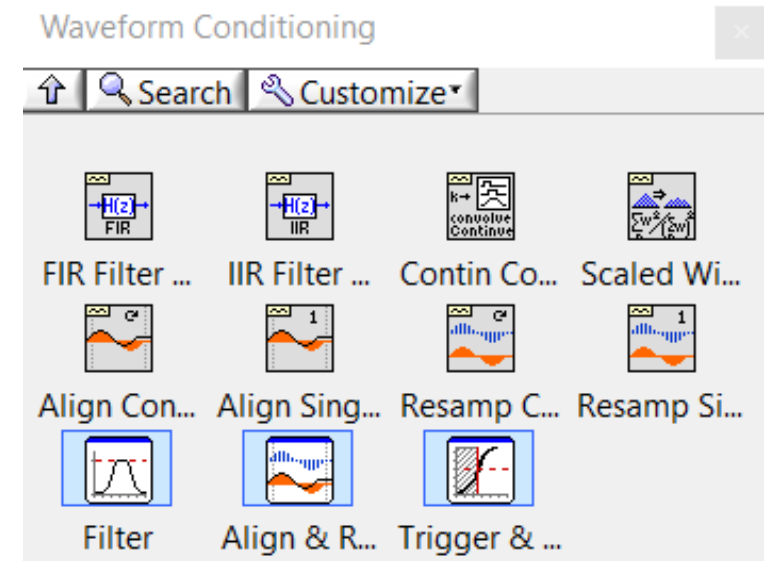
### Gráficos: Gráfico Waveform Chart 2D.

#### 4. Waveform Data (WDT) (Cont.):

Funciones para obtener medidas de las señales Waveform Data: **Waveform Measurements.**



Funciones para acondicionamiento de las señales Waveform Data: **Waveform Conditioning.**





### Gráficos: Gráfico Waveform Chart 2D.

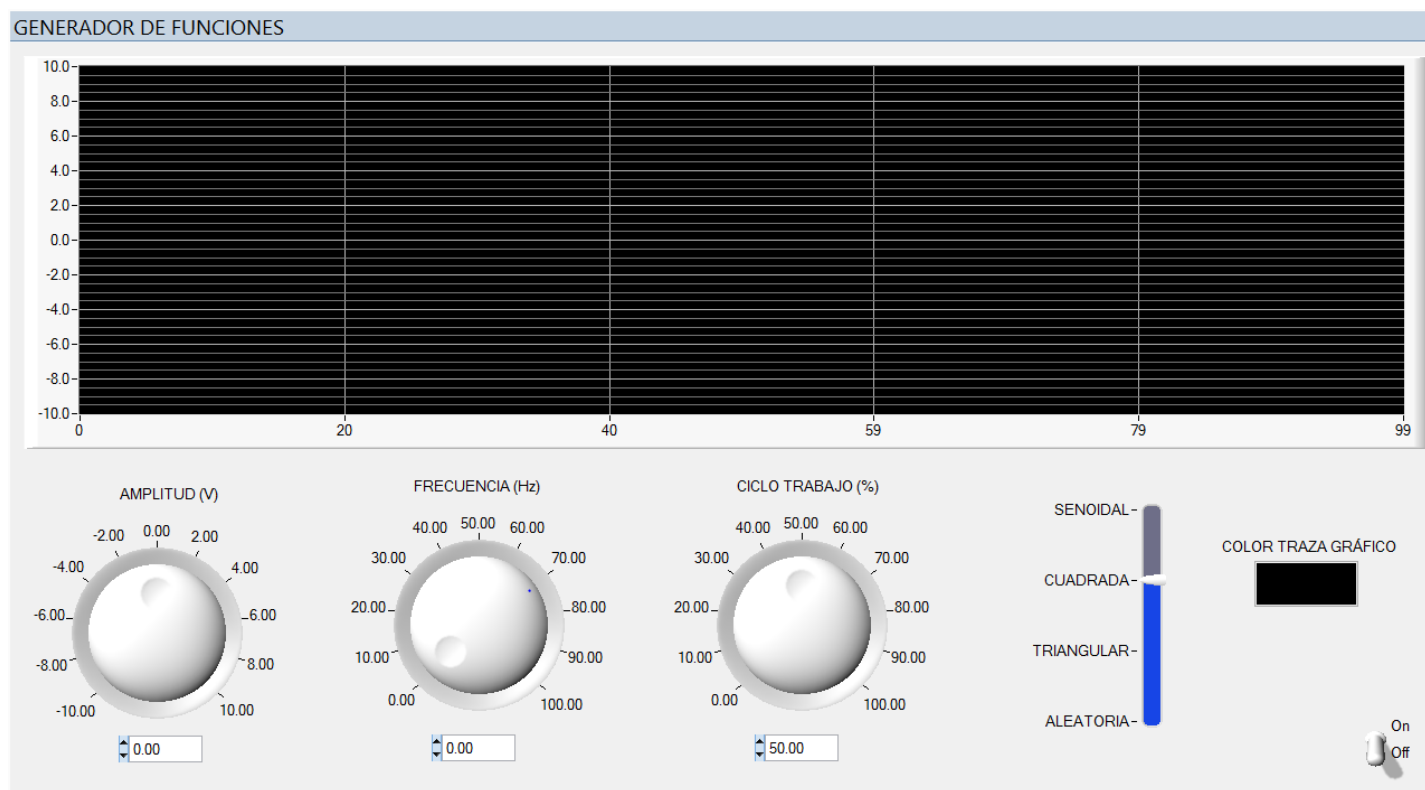
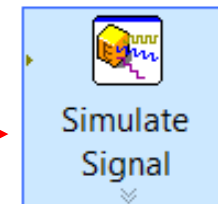
**Ejercicio:** Generar dos ciclos de una señal senoidal mediante el vi **sine** y de una cosenoidal (vi **cosine**) disponible en la librería *Mathematics* (*Mathematics* → *Elementary* → *Trigonometric*).

1. Representarla en un gráfico de tipo Chart los valores obtenidos punto a punto a intervalos de 20 ms ( $T_s=20\text{ms}$ ).
2. Almacenar los valores obtenidos en dos arrays 1D y representarlos posteriormente en un gráfico de tipo Chart.
3. Almacenar los valores obtenidos en un array 2D y representarlo posteriormente en un gráfico de tipo Chart.
4. Añadir una barra de desplazamiento horizontal al gráfico que permita desplazarse a través de la señal.



**Ejercicio:** Realizar el vi correspondiente al generador de funciones de la figura:

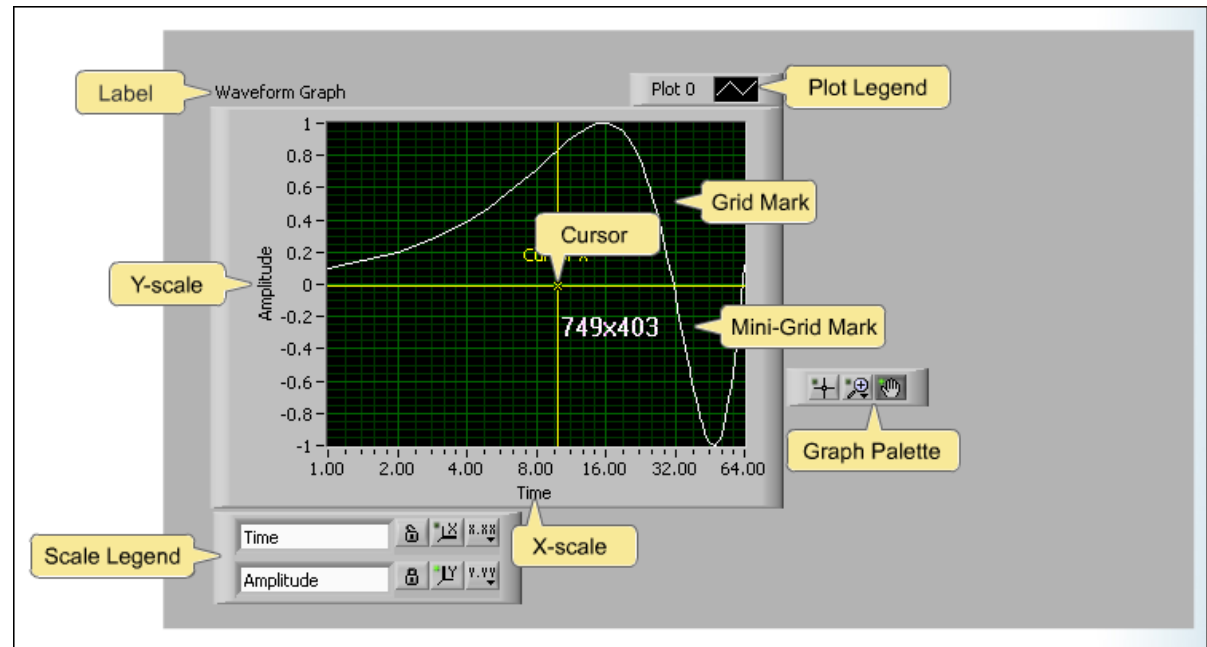
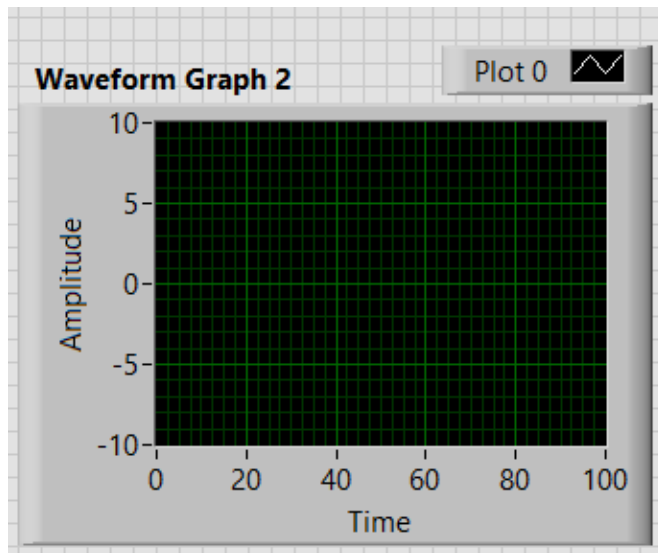
1. Utilizando las funciones para generación de señal en formato WDT (*Waveform Data*).
2. Mediante el vi express par generación de señal ***Simulate Signal*** →





### Gráficos: Gráfico Waveform Graph 2D.

- Útil para representar un número limitado de datos.
- Los datos a representar se deben encontrar previamente almacenados en un array. NO permite la representación de escalares
- Se representan los valores de la función contra su índice.

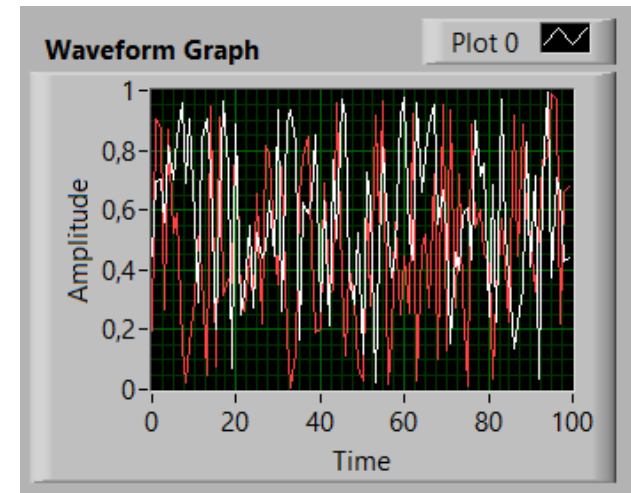
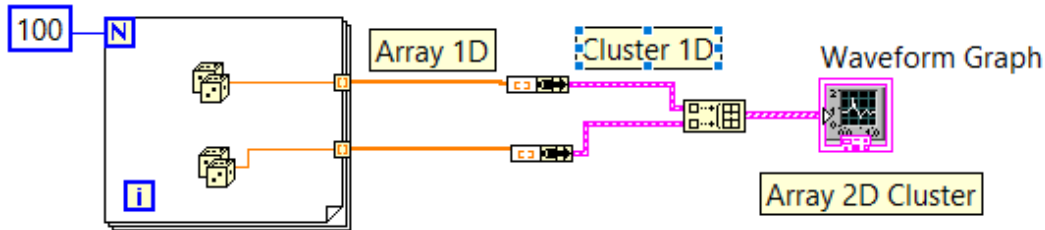




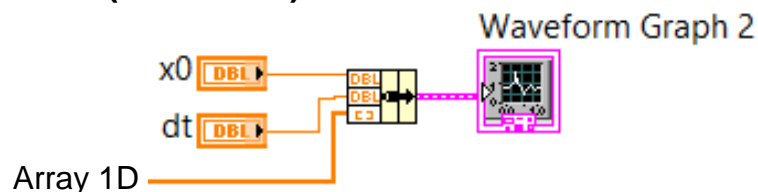
### Gráficos: Gráfico Waveform Graph 2D.

• En un gráfico de tipo Graph se pueden representar los siguientes tipos de datos:

1. **Arrays de 1D:** array que contiene varios puntos/muestras.
2. **Array 2D:** array que contiene puntos/muestras de varias señales. Para esta funcionalidad es necesario “mezclar” estas señales mediante la función *Bundle*.
3. **Waveform Data (WDT).**
4. **Array 2D de Cluster:**



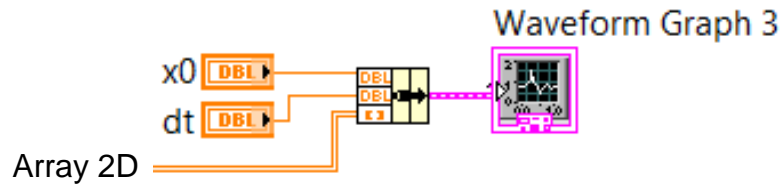
5. **Cluster en formato WDT (una señal):**



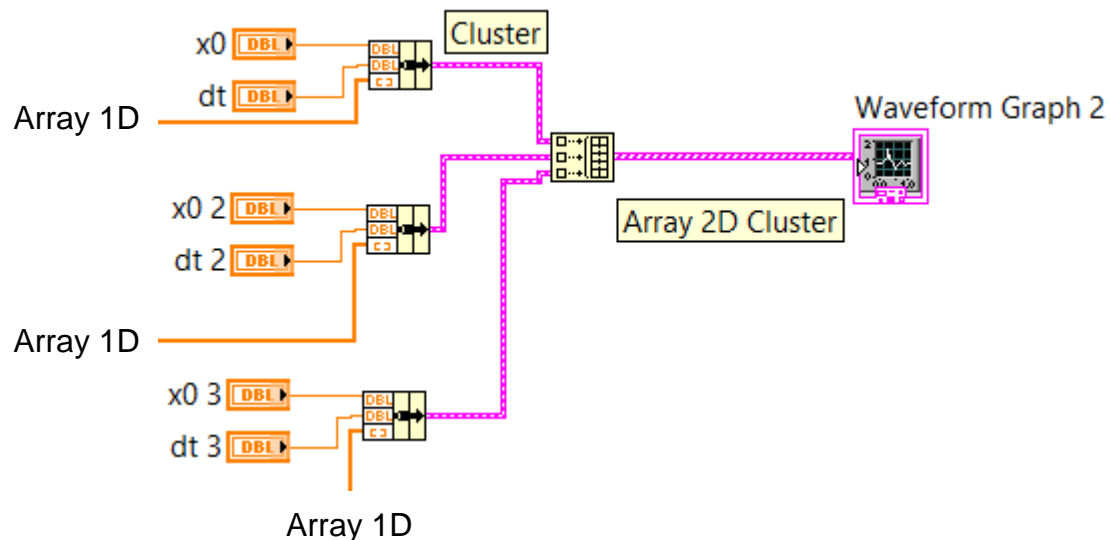


### Gráficos: Gráfico Waveform Graph 2D.

6. Cluster en formato WDT (varias señales que comparten el mismo  $x0$  y  $dt$ ):



7. Array 2D de cluster: Cada gráfica tiene sus valores de  $x0$  y  $dt$ .



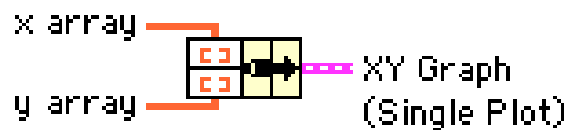


### Gráficos: Gráfico Waveform XY

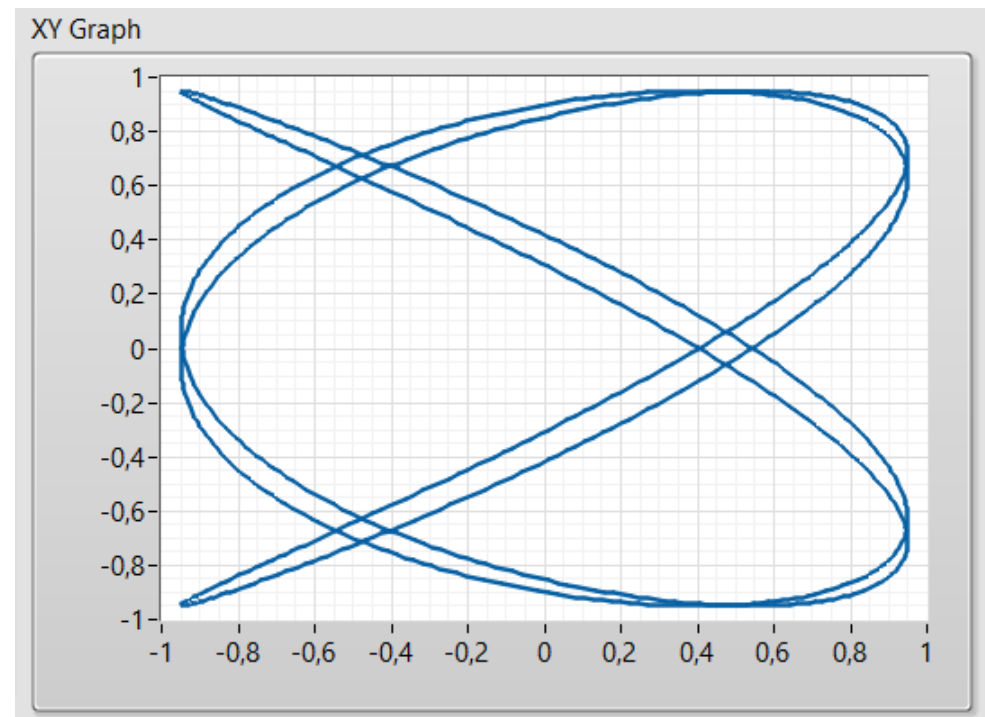
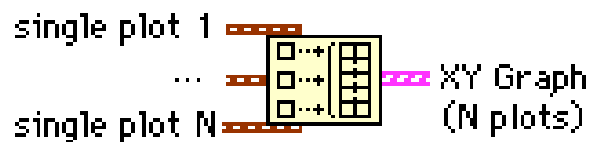
- Permite representar una array de datos que contiene los valores del eje **x** contra otro que contiene los valores del eje **y**.
- Los valores de los ejes **x** e **y** se encuentran en dos vectores 1D.

#### XY Graphs:

Single Plot XY Graph :



Multiplot XY Graph :

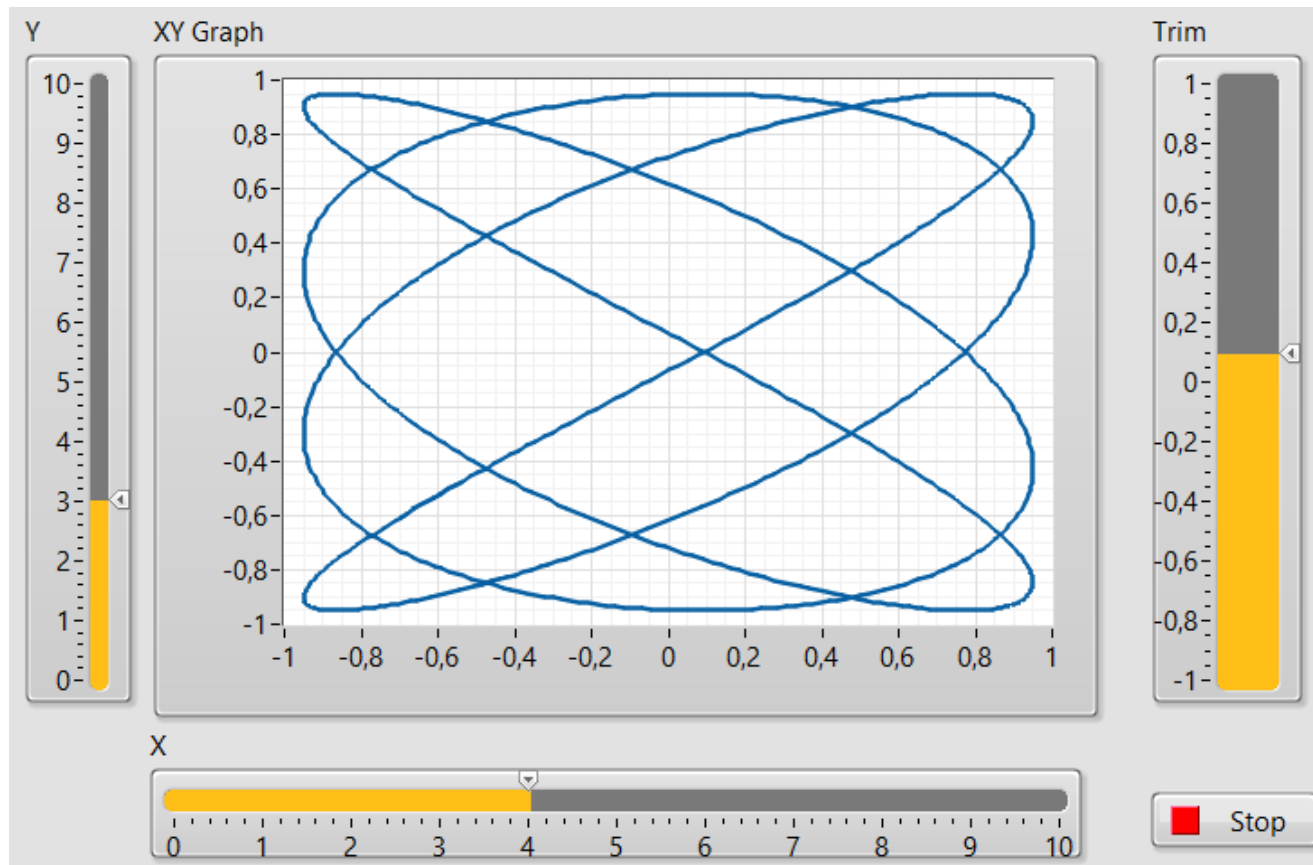






### Gráficos: Gráfico Waveform XY

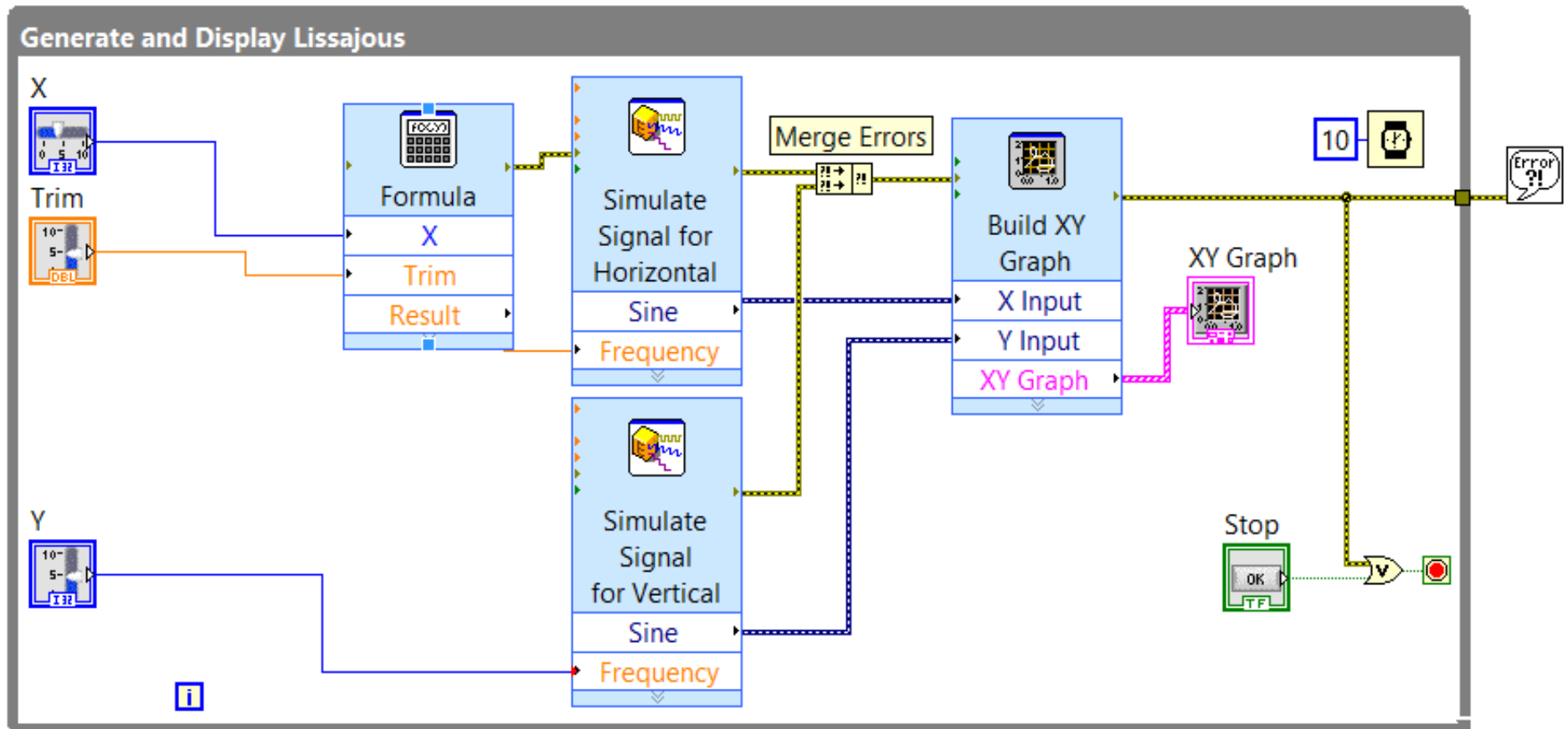
- **Ejemplo:** generación figuras de Lissajous.





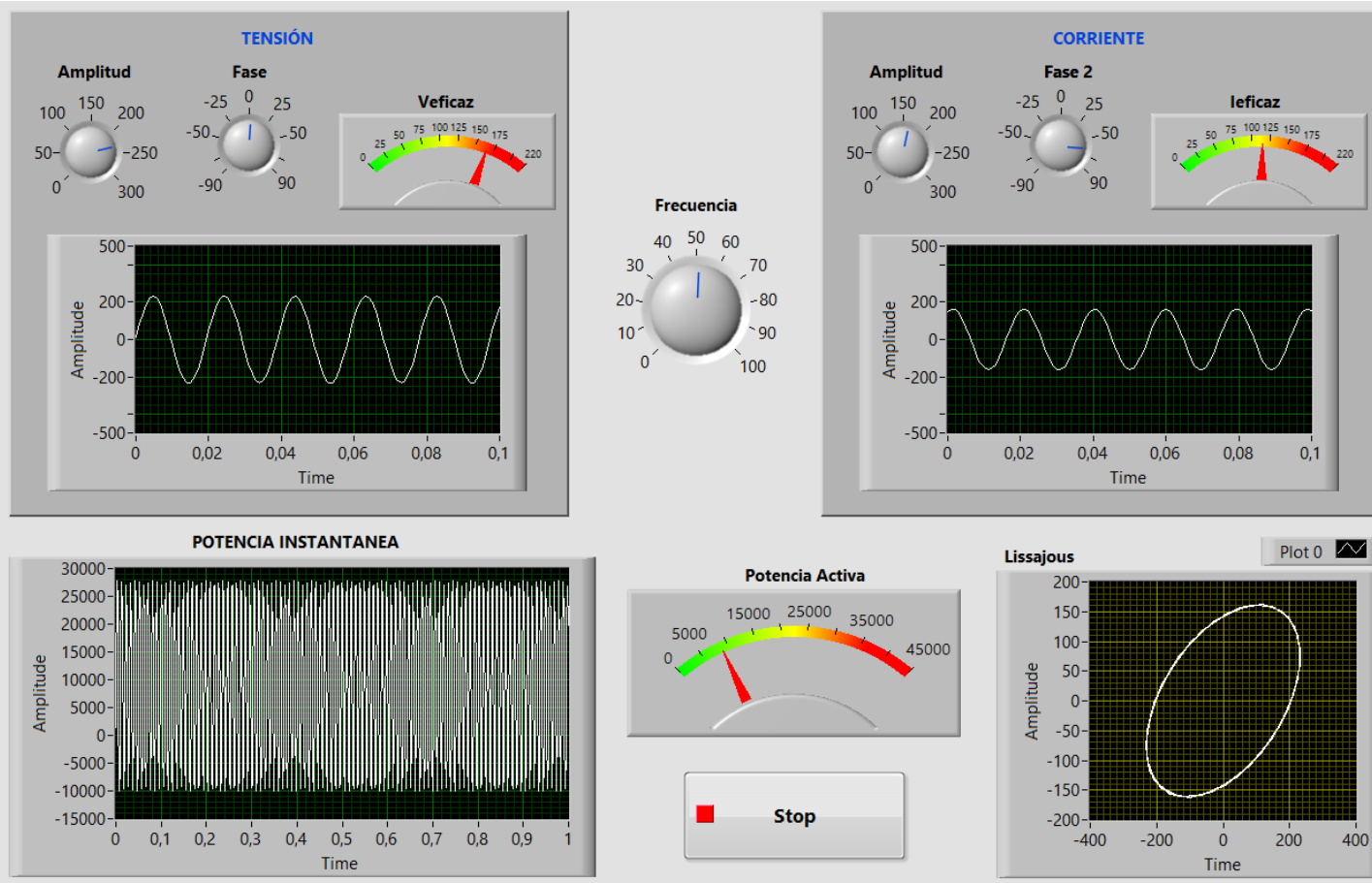
### Gráficos: Gráfico Waveform XY.

- **Ejemplo:** generación figuras de Lissajous.





**Ejercicio:** Realizar el código que genere dos señales senoidales de la misma frecuencia que representan la tensión y corriente sobre una carga utilizando para ello la función *Sinewaveform*.



- Obtener a partir de estas señales: sus valores eficaces, potencia instantánea, potencia activa y representación de Lisajous de ambas señales.

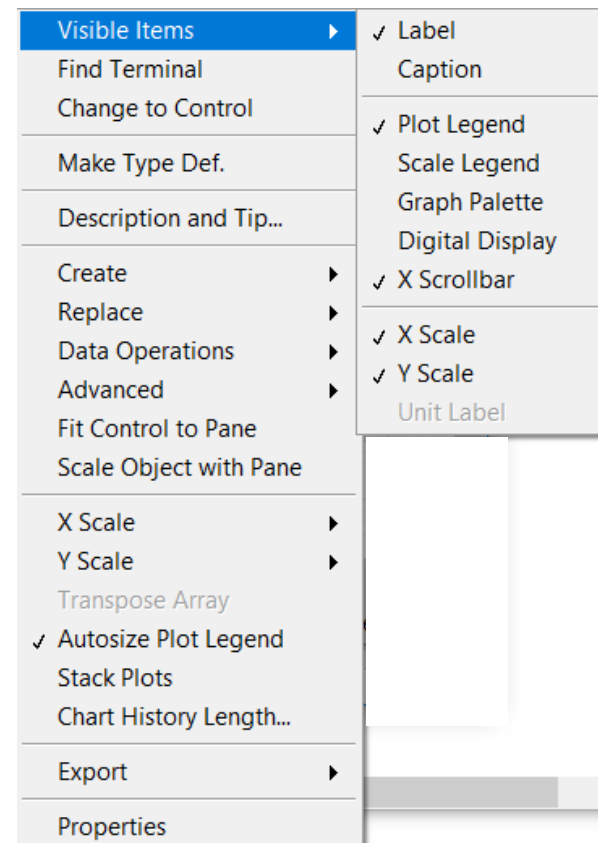
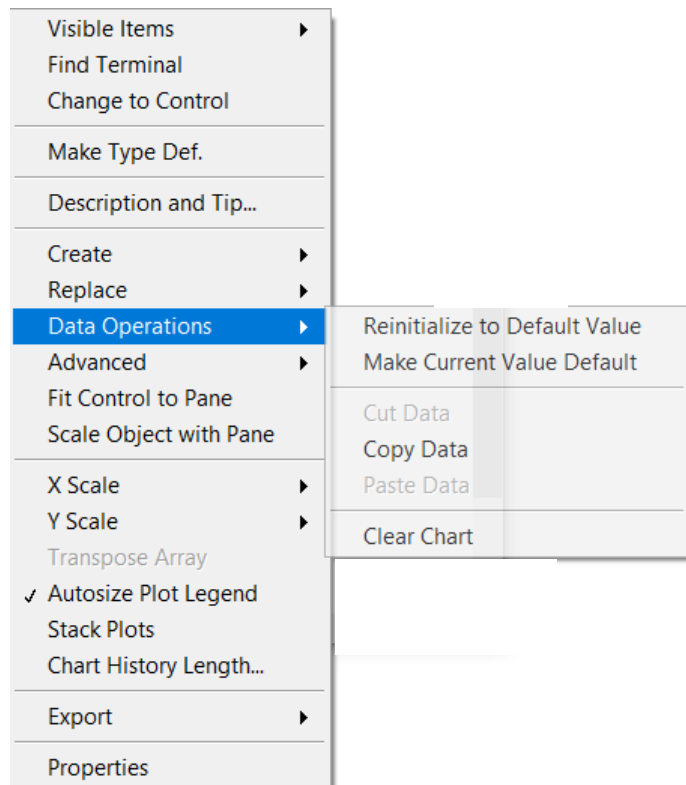
- Representar la información tal y como se muestra en la figura:

**NOTA:** Potencia activa es el valor medio de la potencia instantánea.



### Gráficos: menús.



- Se pueden realizar operaciones sobre los gráficos a través del menú flotante que aparece al pulsar el botón derecho del ratón sobre estos.
- Operaciones: reset del gráfico, apariencia, barras de desplazamiento, escalas, etc. .





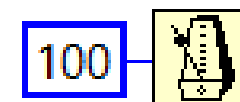
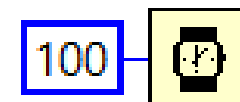
### Temporización

- En un bucle, cuando finaliza una iteración comienza a ejecutarse la siguiente de forma inmediata, a no ser que se alcance la condición de STOP.
- En muchas ocasiones se hace necesario controlar el tiempo entre iteraciones. Por ejemplo si se quiere adquirir una muestra cada 10 segundos, se necesita que la iteración se ejecute cada 10 segundos.
- Durante el tiempo de espera, el procesador puede ejecutar otras tareas.
- Para temporizar los bucles y sincronizar tareas se pueden utilizar los vi's **Wait** y **Wait Until Next ms Multiple**.

Wait Functions	Features
<b>Wait Until Next ms Multiple</b> 	Monitors a millisecond counter and waits until the millisecond counter reaches a multiple of the amount specified. This function is primarily used to synchronize activities. For instance, you can place this function in a loop to control the loop execution rate. For this function to be effective, your code execution time must be less than the time specified for this function. The execution rate for the first iteration of the loop is indeterminate.
<b>Wait (ms)</b> 	Waits until the millisecond counter counts to an amount equal to the input specified. This function ensures that the loop execution rate is at least the amount of the input that has been specified.



La duración de la primera iteración es indeterminada.

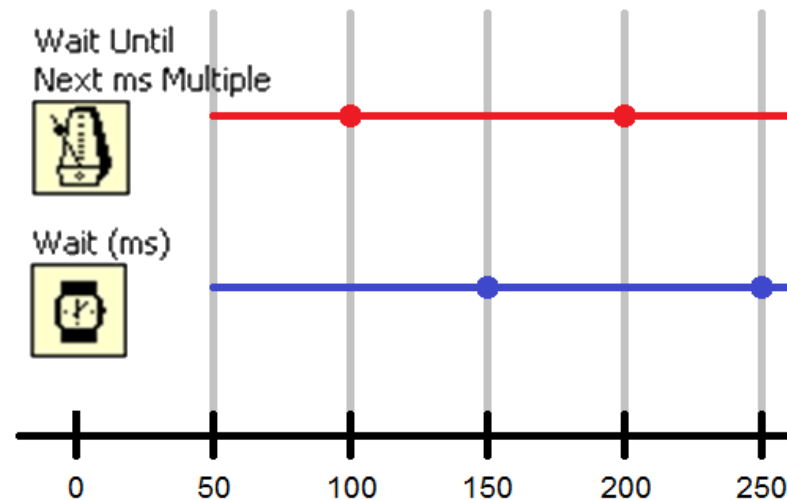




### Temporización

- Ambas funciones se utilizan para bloquear/parar la ejecución de un bloque de código durante un periodo de tiempo especificado.
- Normalmente se utilizan para controlar la velocidad a la que se ejecuta cada iteración de un bucle.
- La diferencia entre las funciones **Wait** y **Wait Until Next ms Multiple** son:
  - **Wait**: bloquea el código hasta que finaliza el tiempo especificado en su terminal **milliseconds to wait**.
  - **Wait Until Next ms Multiple**: bloquea el código hasta que el valor del reloj del sistema en milisegundos es divisible( múltiplo) por el valor especificado en su terminal **millisecond multiple**.
- En la siguiente figura se muestra el momento en el que ambas funciones bloquean el código supuesto que en sus respectivas entradas se ha cableado un valor de 100 ms. y que el código empieza a ejecutarse en t=50ms.

**Importante:** Cuando se insertan en un bucle, las dos funciones se ejecutan en paralelo con el código del bucle.

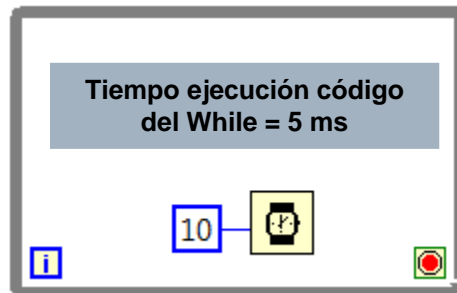




### Temporización (cont.)

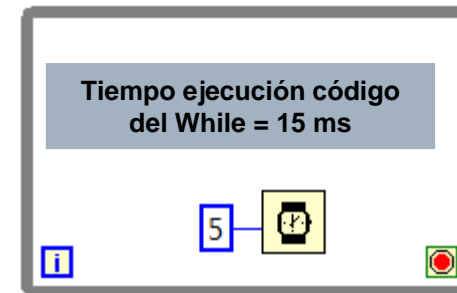
**Wait** 

- Espera la cantidad de tiempo especificada, equivale a un *Delay*.
- Se utiliza típicamente para establecer retardos entre segmentos de código (p.e. en estructuras *sequence*).
- Cuando se ubica en un bucle se ejecuta en paralelo con el resto de código.



El bucle finaliza transcurridos 10 ms.

Desde que finaliza el código del while (5ms) hasta los 10 ms que faltan para que finalice el Wait la CPU puede ejecutar otro código, es decir **el while no consume todo el tiempo de procesador.**



El bucle se ejecuta cada 15 ms. El Wait no sirve de nada en este caso por ser su duración inferior al tiempo que tarda en ejecutarse el código del While.

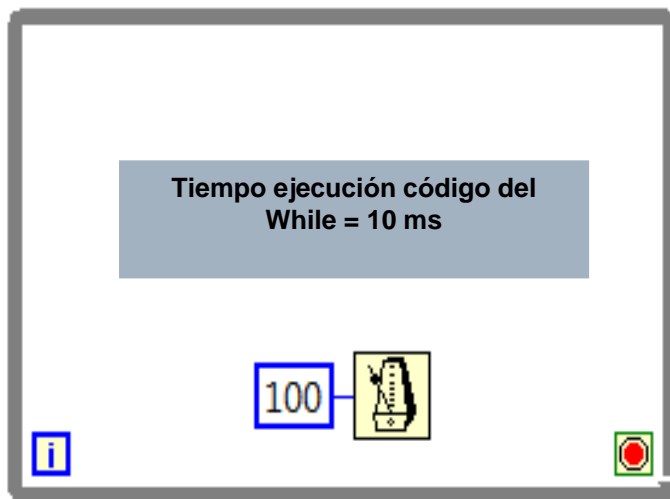


### Temporización (cont.)

#### Wait until next ms Multiple :

- El tiempo de la primera iteración es aleatorio entre 0 y el tiempo indicado al Wait until next ms. Esto debe a que este vi obtiene una marca de tiempos (*timestamp*) del reloj interno del PC y espera hasta que el valor de este *timestamp* es divisible por el valor que se le ha cableado.

Por este motivo, la primera vez que se ejecuta, el tiempo depende del valor absoluto del tiempo en el que el bucle empieza a ejecutarse. Las ejecuciones siguientes se encontrarán alineadas con este tiempo y esperaran el tiempo indicado.



- El bucle se ejecuta en los tiempos múltiplos de 100 ms.
- Transcurridos los 10 ms que tarda en ejecutarse el código contenido en el while y hasta los 100 ms indicados en el *wait until next ms* la CPU puede ejecutar otro código, es decir **el while no consume todo el tiempo de procesador.**
- Al igual que en el caso del **Wait**, si el código tiene una duración superior al del valor cableado al **Wait Until Next ms Multiple**, este último no tendrá efecto alguno.





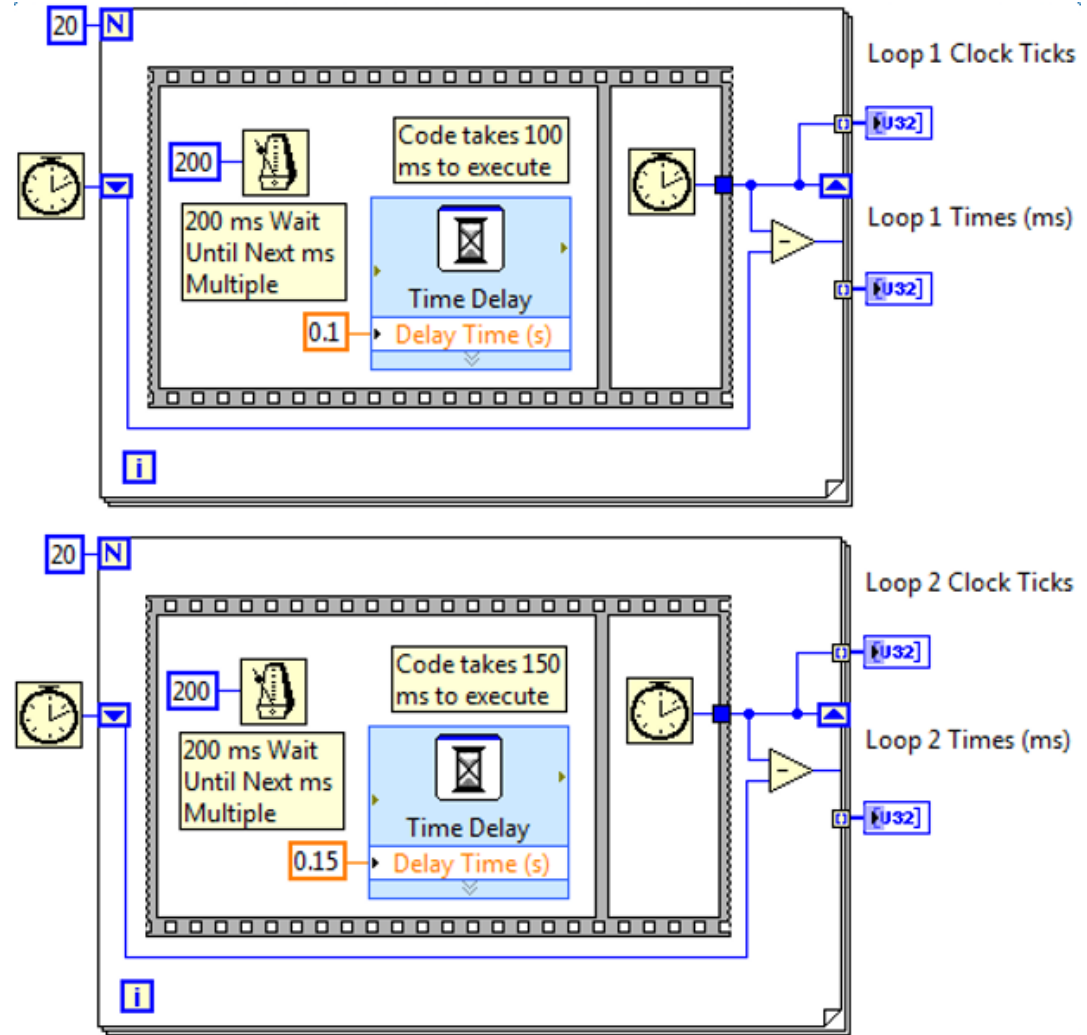
### Temporización (cont.)

#### Wait until next ms Multiple

• **Wait Until Next ms Multiple** se utiliza para sincronizar con el reloj del sistema bucles que se ejecutan en paralelo.

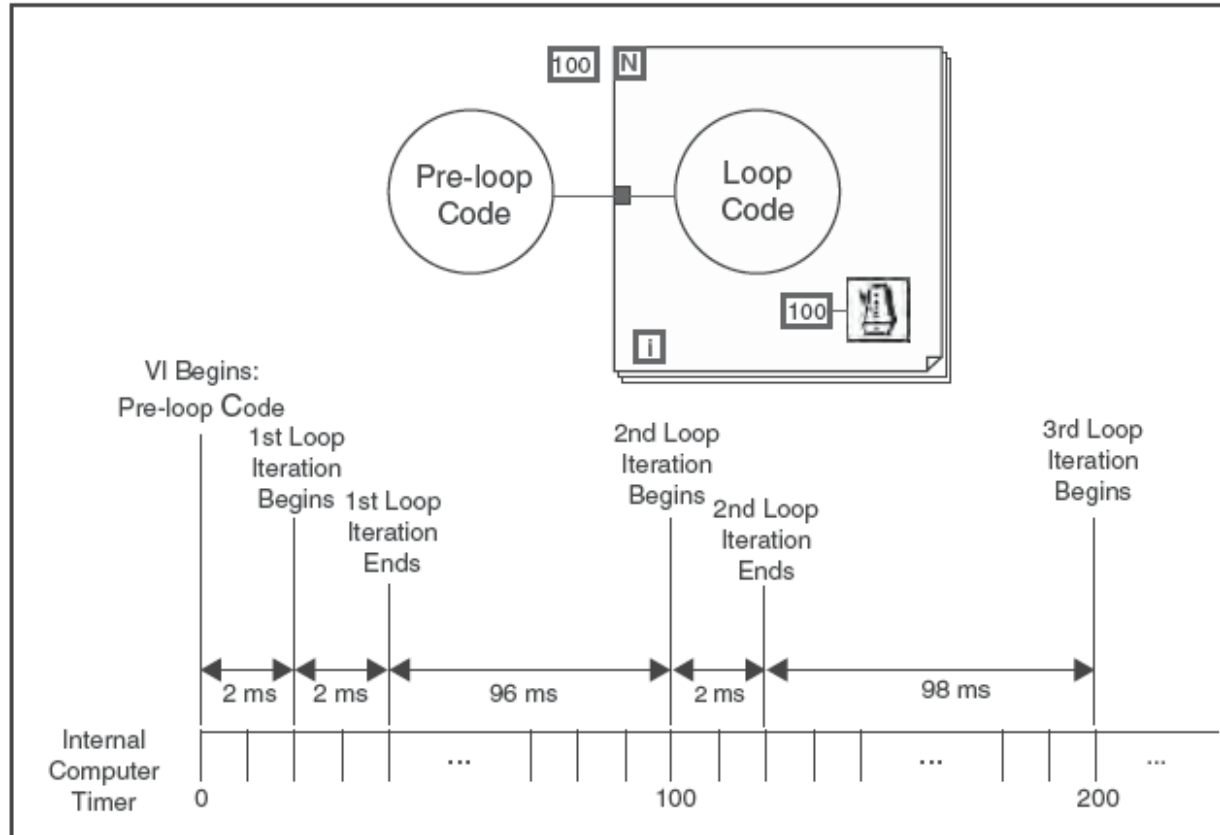
#### En el ejemplo:

- Los dos bucles se ejecutan en tiempos múltiples de 200 ms.
- El código del primer bucle tarda en ejecutarse 100 ms y el del segundo bucle 150 ms.
- Los dos bucles están sincronizados para ejecutar la próxima iteración cada tiempo del reloj del sistema múltiple de 200 ms.
- Mediante este método se asegura que cada bucle empieza la siguiente iteración en el mismo instante de tiempo.





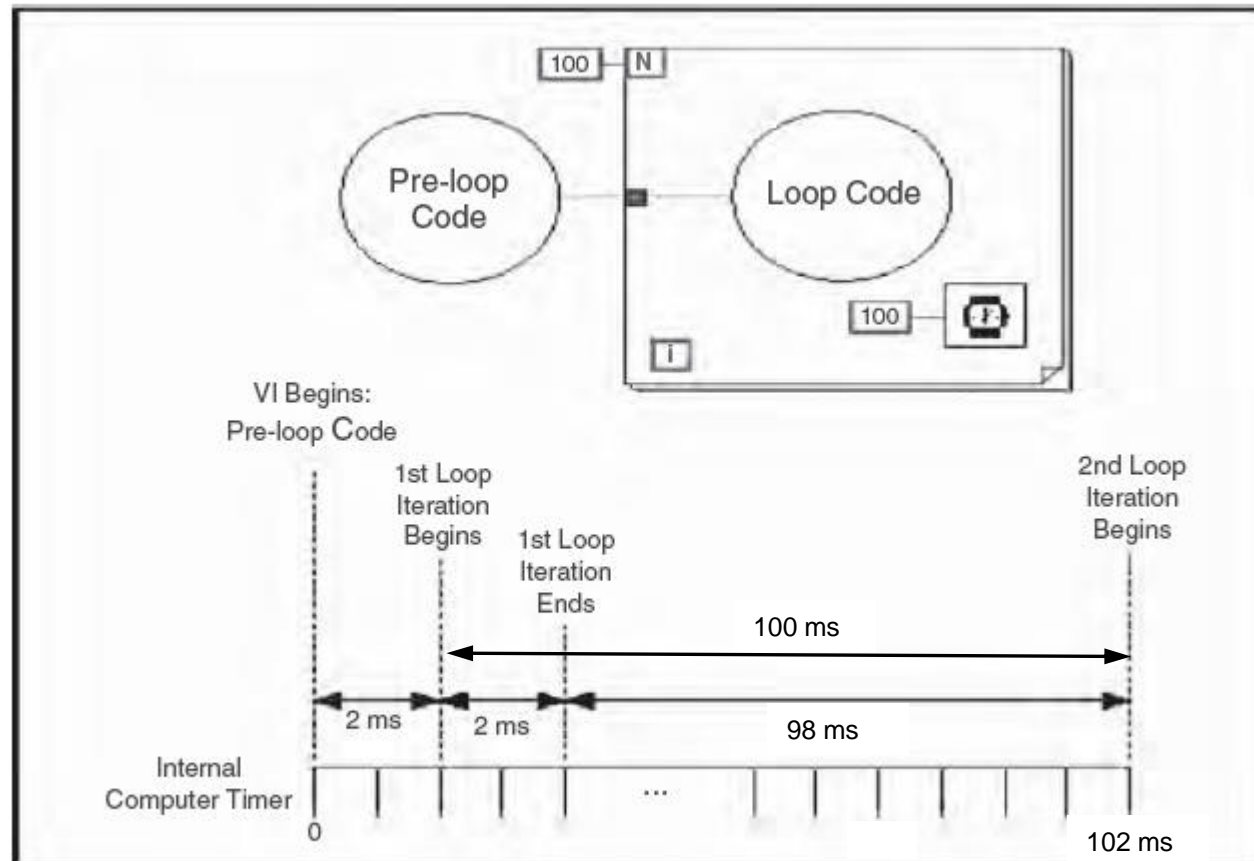
### Temporización:



Timing diagram for Wait Until Next ms Multiple function



### Temporización:

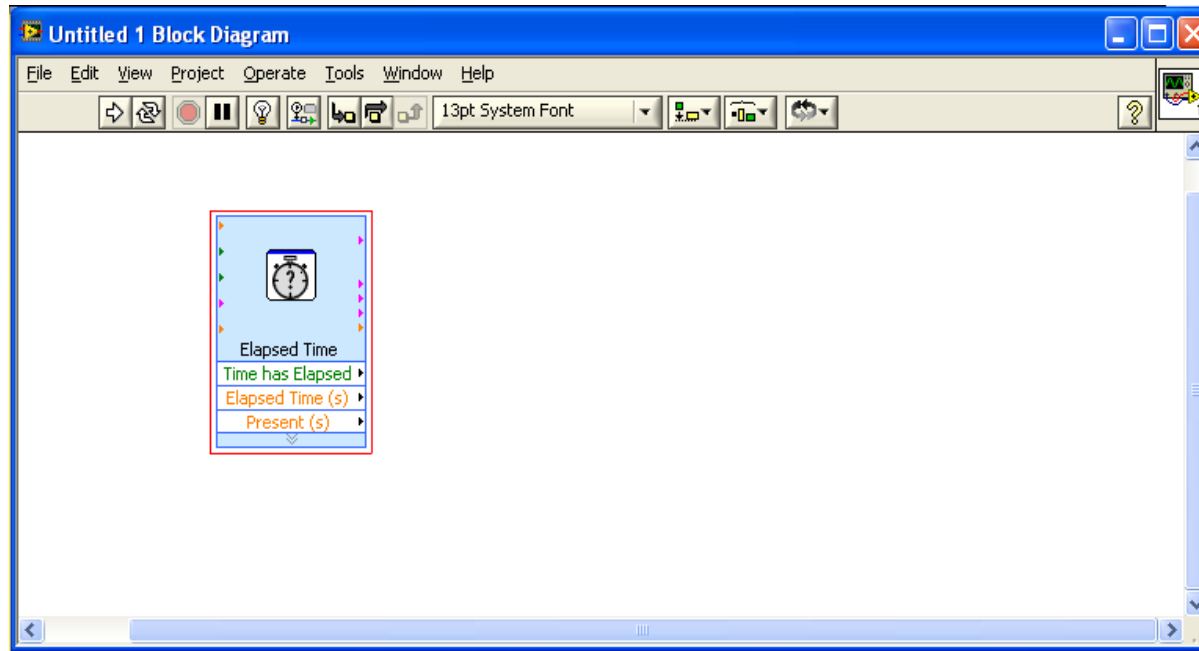


Timing diagram for Wait ms function



### Temporización: Elapsed Time Express

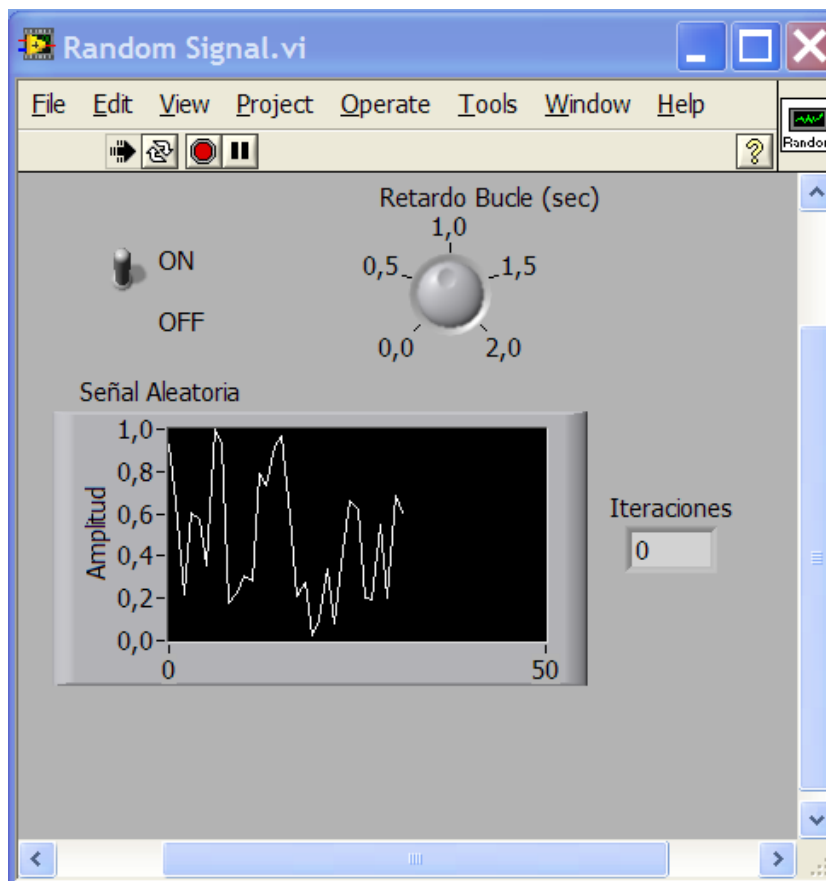
- El **Elapsed Time Express VI** indica la cantidad de tiempo que ha transcurrido desde el periodo de tiempo especificado en el parámetro de entrada *SetStartTime*.
- Esta función no proporciona tiempo al procesador para completar otras tareas.





### Ejercicio 3: bucle while.

- Generar números aleatorios y representarlos en un gráfico.



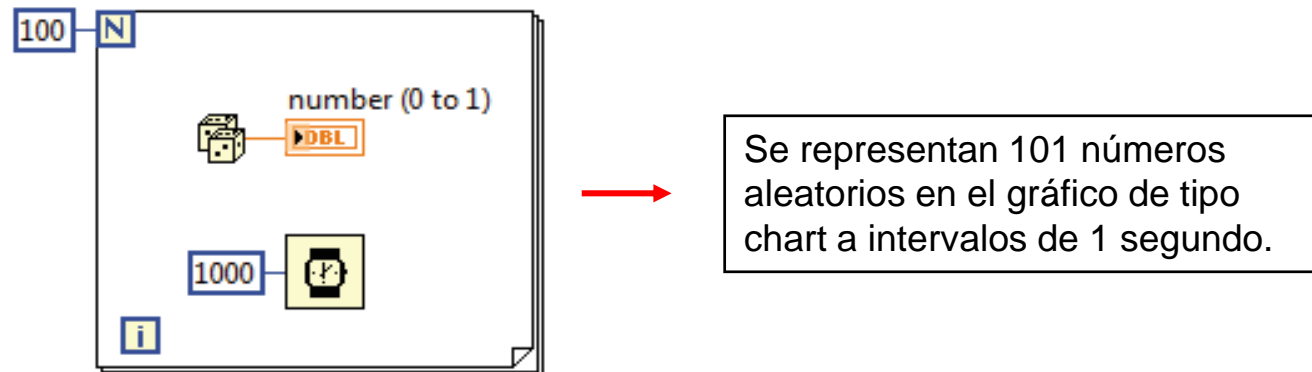


### Bucles: For

- Ejecuta el subdiagrama contenido en el rectángulo el número de veces indicado en el terminal de cuenta.



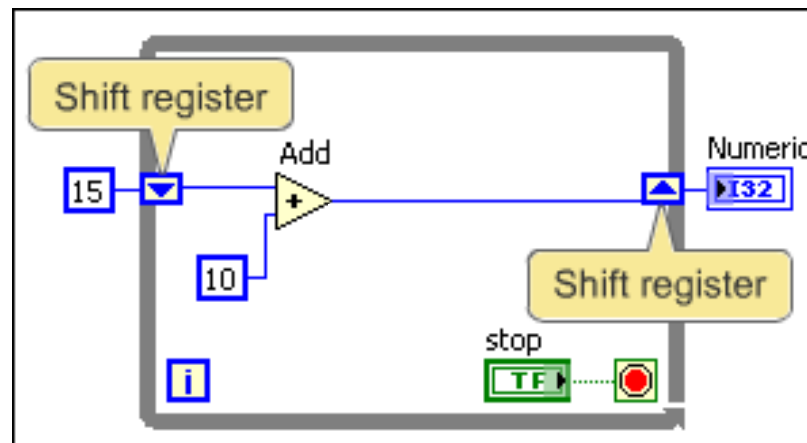
### Ejemplo:





### Registros de desplazamiento

- Los registros de desplazamiento transfieren valores de una iteración de los bucles While y For a la siguiente.
- Permiten acceder a información de iteraciones previas.
- Aparecen como un par de terminales enfrentados y situados en los lados verticales del borde que delimita la estructura de los bucles While y For.

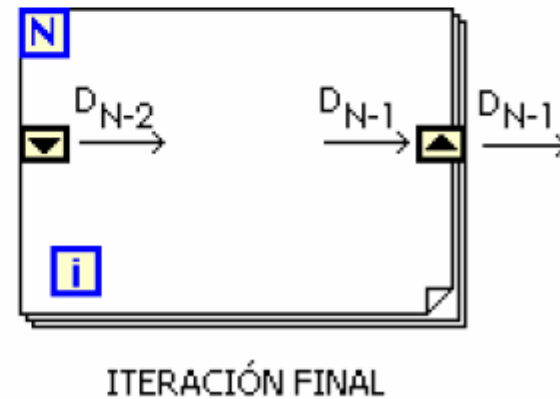
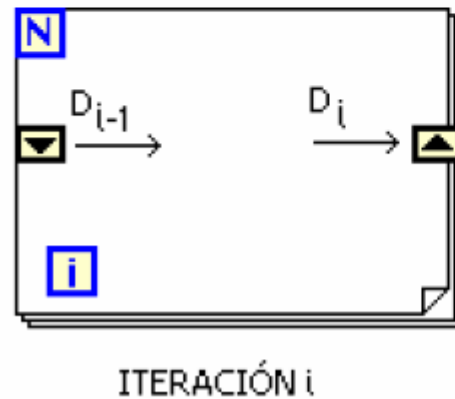
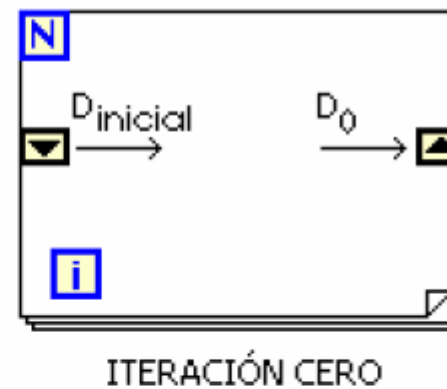


- El terminal de la derecha contiene una flecha hacia arriba y almacena los datos de la iteración que acaba de finalizar/ejecutarse.
- LabView transfiere los datos conectados a los terminales de la derecha al finalizar cada iteración.



### Registros de desplazamiento

- Funcionamiento de los registros de desplazamiento:

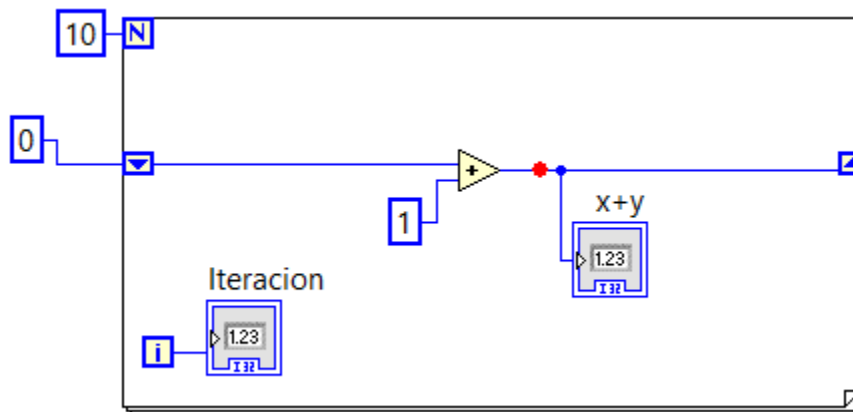






### Registros de desplazamiento

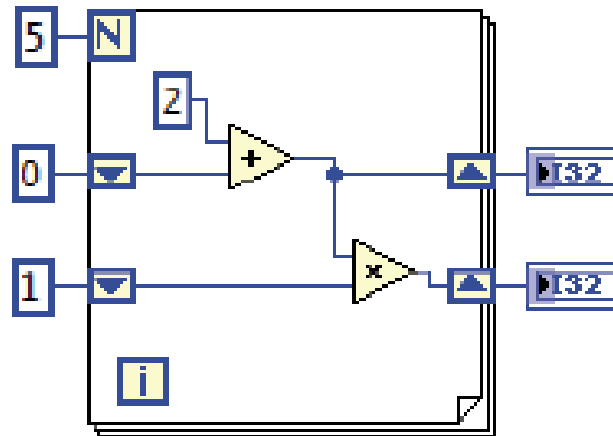
- **Ejercicio:** Verificar la secuencia de transmisión de valores de los registros de desplazamiento con la siguiente aplicación que simplemente suma una unidad en cada iteración del bucle del programa.





### Registros de desplazamiento

- Permite transferir cualquier tipo de dato.
- Los datos cableados a los terminales de un registro de desplazamiento deben ser del mismo tipo.
- Para añadir un registro de desplazamiento basta con hacer clic con el botón derecho del ratón sobre el borde que delimita las estructuras While y For y seleccionar **Add Shift Register** del menú flotante que aparece.
- Se pueden añadir tantos registros de desplazamiento como sean necesarios:

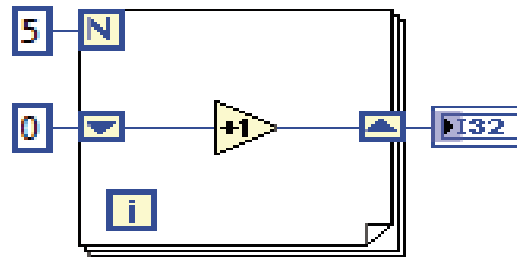


- Para eliminar un registro de desplazamiento basta con hacer clic con el botón derecho del ratón sobre el borde izquierdo o derecho de la estructura While o For y seleccionar **Remove All** del menú flotante que aparece.

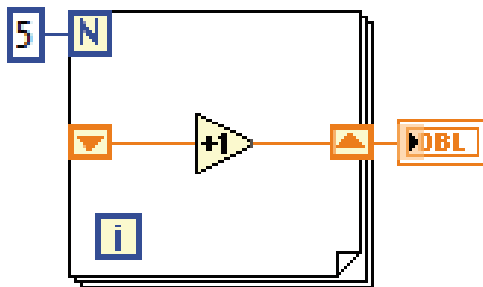


### Registros de desplazamiento

- Para evitar posibles comportamientos erráticos es conveniente inicializar los registros de desplazamiento.
- Para inicializarlos basta con cablear un control o constante al registro de desplazamiento. Cuando se ejecuta el vi el valor de inicialización pasa al registro de desplazamiento.



- Ejemplo de Registro de Desplazamiento sin inicializar

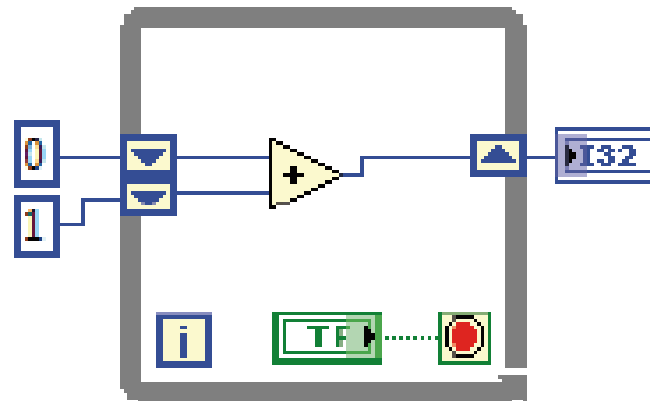


El VI finaliza cuando el registro de desplazamiento pasa el valor 5 al indicador. Cuando se vuelve a ejecutar el VI, el registro de desplazamiento empieza con un valor inicial de 5, que se corresponde con el último valor de la ejecución previa, y así sucesivamente.



### Registros de desplazamiento apilados (*Stacked Shift Registers*)

- Los registros de desplazamiento apilados permiten acceder a valores de varias iteraciones previas, de forma que puedan utilizarse en la iteración actual.
- Para añadir un registro de desplazamiento apilado hacer clic con el botón derecho del ratón sobre el lado izquierdo del borde que delimita la estructura While o For y seleccionar **Add Element** del menú flotante que aparece.

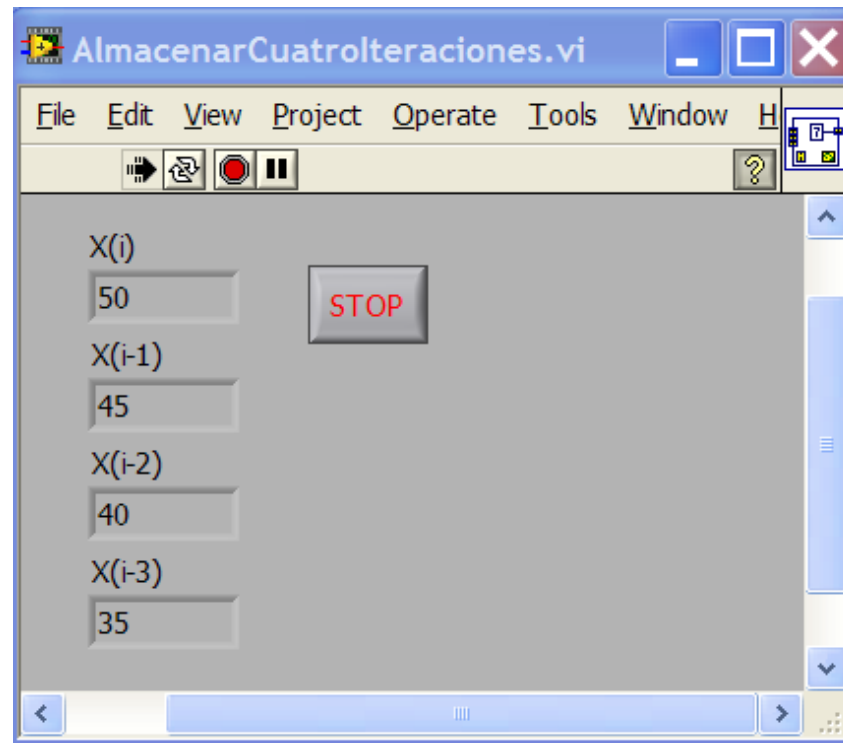


- En el ejemplo anterior el primer terminal contiene el valor de la última iteración y el segundo terminal contiene el de la iteración anterior. Si hubiera un tercer terminal, este almacenaría el valor de dos iteraciones anteriores y así sucesivamente.



### Ejercicio 4: registros de desplazamiento.

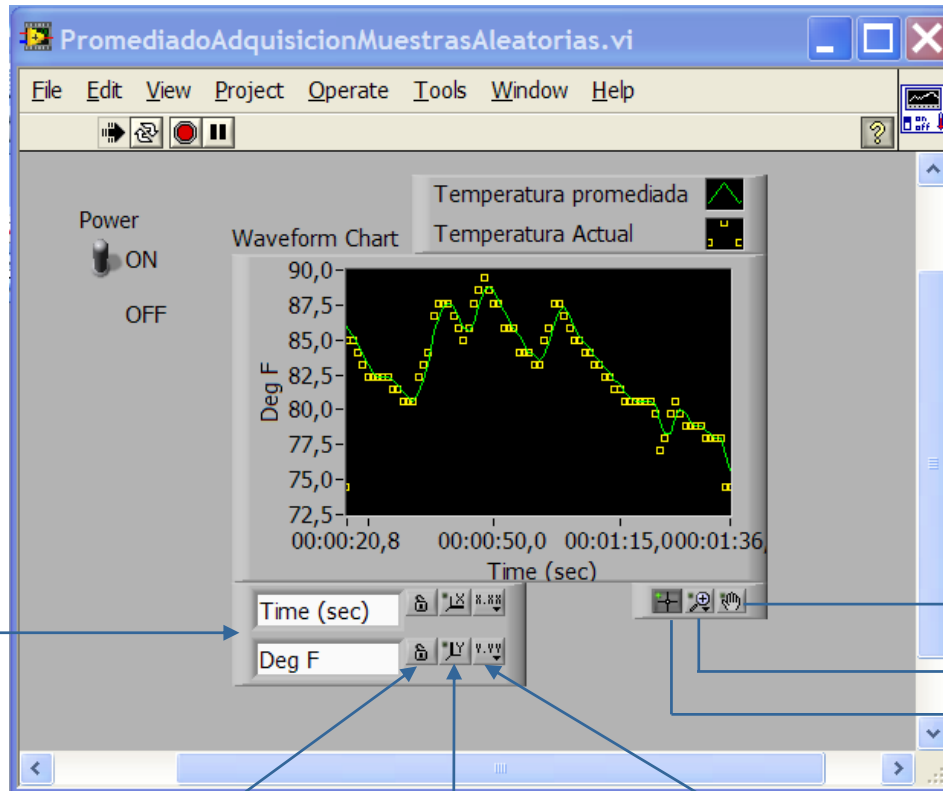
Representar en cuatro indicadores los cuatro últimos valores de las cuatro últimas iteraciones de un bucle While. En cada iteración, que debe ejecutarse cada segundo, el valor del registro de desplazamiento debe incrementarse en cinco unidades.





### Ejercicio 5: registros de desplazamiento.

Adquirir muestras aleatorias cada segundo y calcular la media de las últimas tres muestras adquiridas. En el gráfico deben representarse dos trazas: la de la muestra actual y la de la muestra promediada.



Etiquetas de los ejes

Zoom

Desplazamiento

Cursores

Bloqueo de escalas

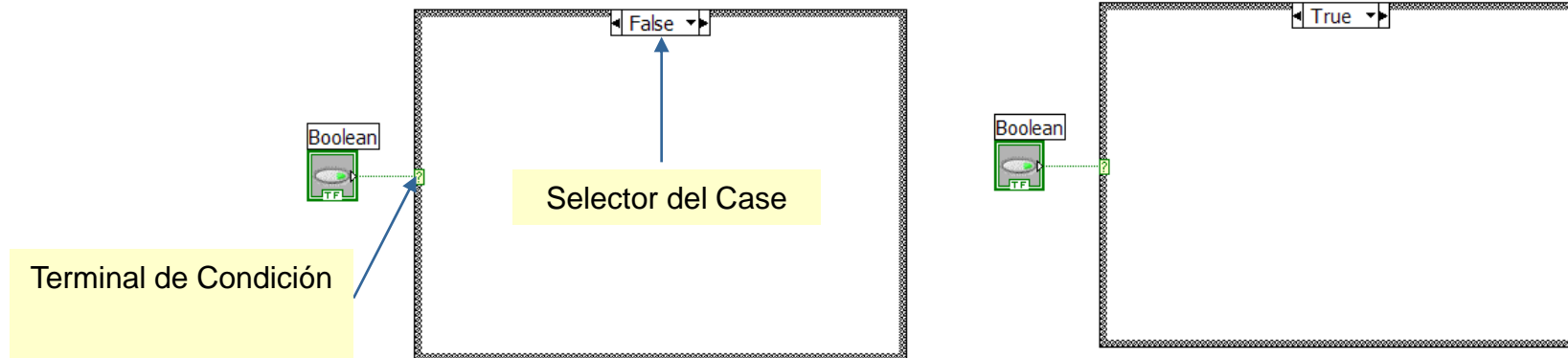
AutoScale

Formato de la escala



### Toma de decisiones: Estructura Case

- Una estructura Case contiene dos o más subdiagramas.
- Se ejecuta el subdiagrama correspondiente al valor que tiene cableado el terminal de condición.

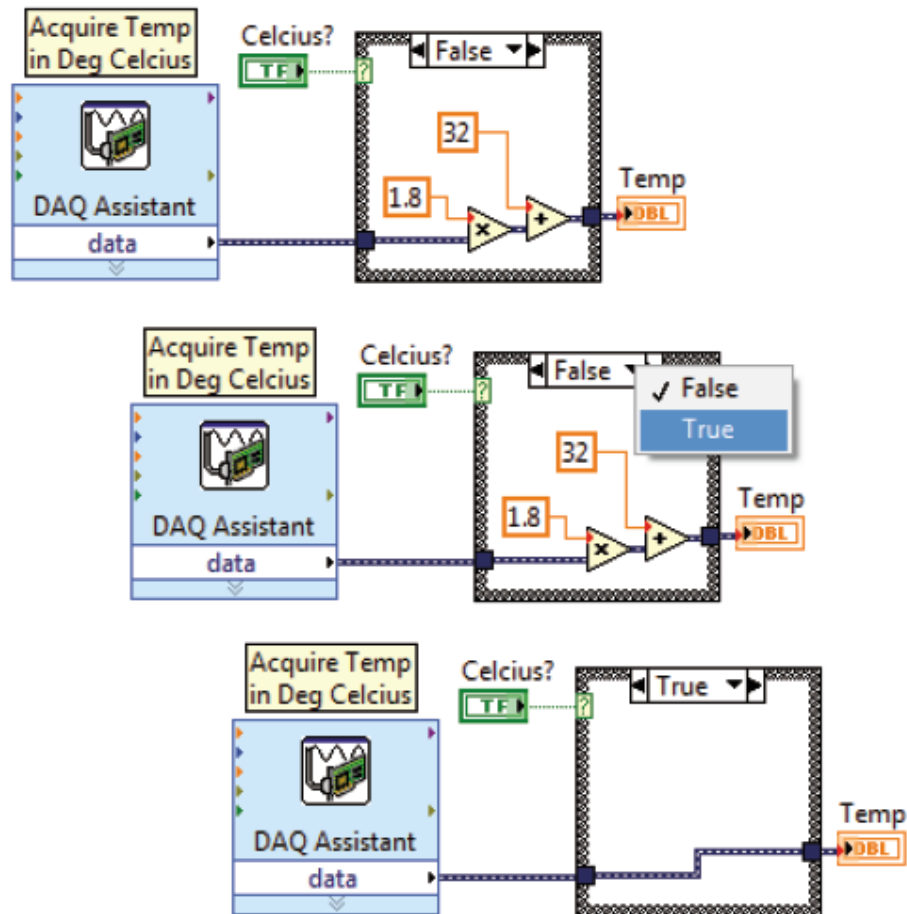


- Se puede especificar un subdiagrama para que se ejecute código en el caso de que el valor conectado al terminal de condición no coincida con ningún valor de los indicados en el selector. A este subdiagrama se le denomina “*Case por defecto*”.
- Para especificar Case por defecto: hacer clic con el botón derecho del ratón sobre el borde que limita la estructura Case y seleccionar la opción ***Make This The Default Case*** del menú flotante .



### Toma de decisiones: Estructura Case

- **Ejemplo:** Conversión de temperatura a grados Celsius.

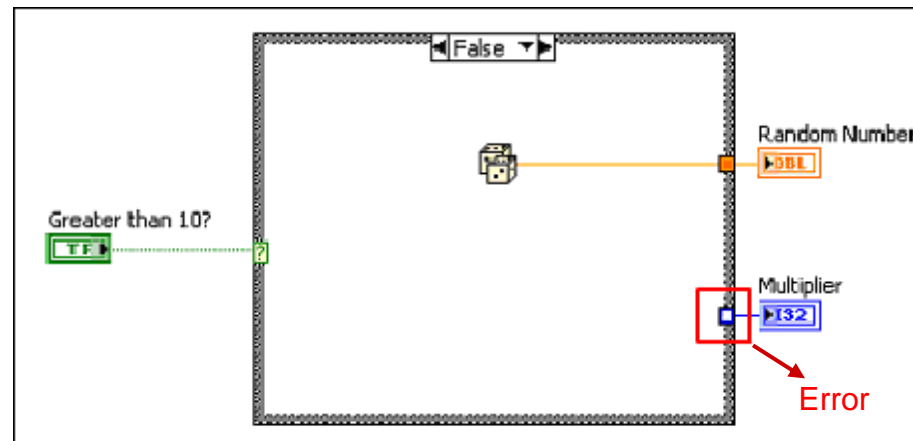






### Estructura Case: Entradas y túneles de salida







- Se pueden crear múltiples entrada y túneles de salida para una estructura Case:
  - Las entradas están disponibles para todos los Cases, aunque no todos los Cases las necesiten.
  - Los túneles de salida se definen para cada Case. Cuando un Case no tiene un valor de salida cableado al túnel se produce un error. LabView indica dicho error rellenando de color blanco el centro del rectángulo que identifica al túnel.





### Estructura Case: Entradas y túneles de salida

- Para corregir este error situarse sobre el case que contiene el túnel sin cablear y seleccionar del menú flotante la opción **Use Default If Unwired** para asignar el valor por defecto al túnel sin cablear.
- Los valores por defecto en función del tipo de dato son los siguientes:

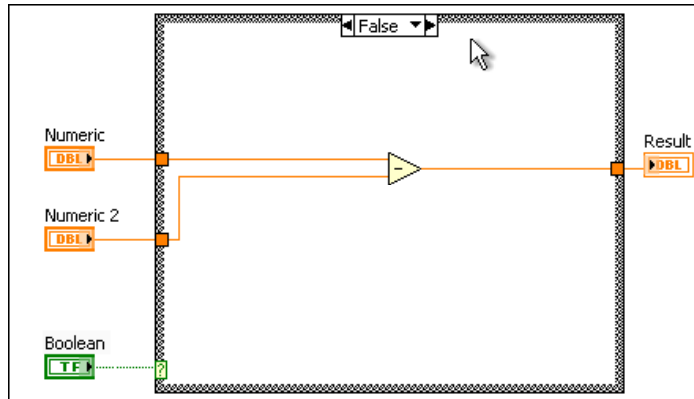
Data Type	Default Values
Numeric 	0 
Boolean 	FALSE 
String 	empty ("") 

- Es conveniente evitar la opción **Use Default If Unwired** por los siguientes motivos:
  - El diagrama de bloques no queda debidamente documentado lo que puede ser confuso para otros programadores que utilicen ese código.
  - La depuración del código es más compleja.

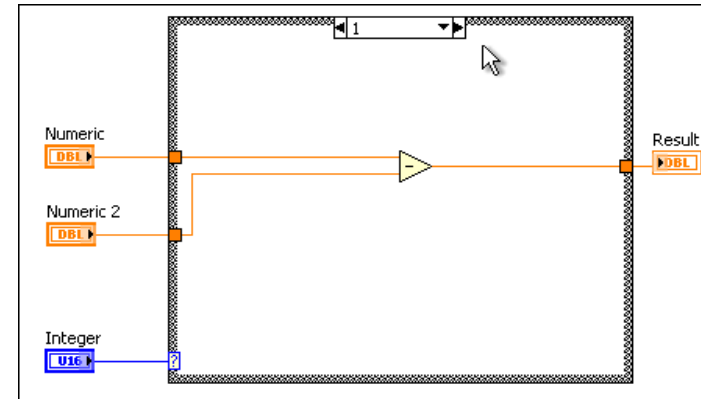


### Estructura Case: Tipos de Case

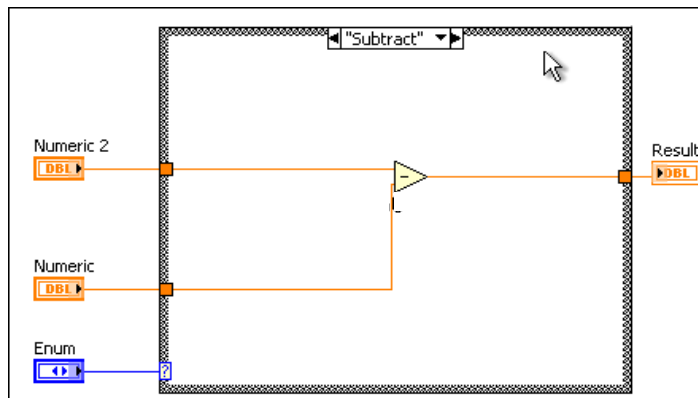
- Depende del tipo de dato que se conecte al terminal de condición o selector, así se tiene:



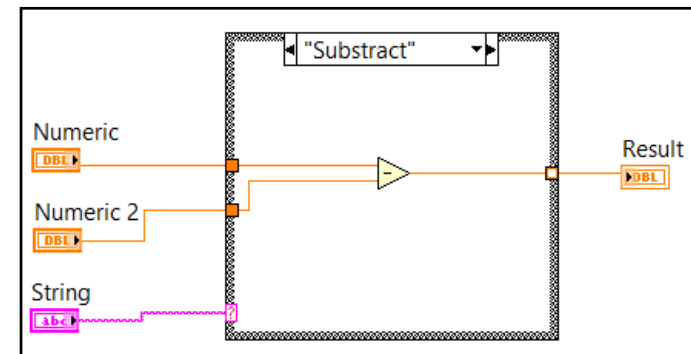
Case Boolean



Case Integer



Case Enumerado



Case String



### Estructura Case: Selector

- En el selector de un Case se puede especificar un único valor o un rango de valores:

#### Ejemplos:

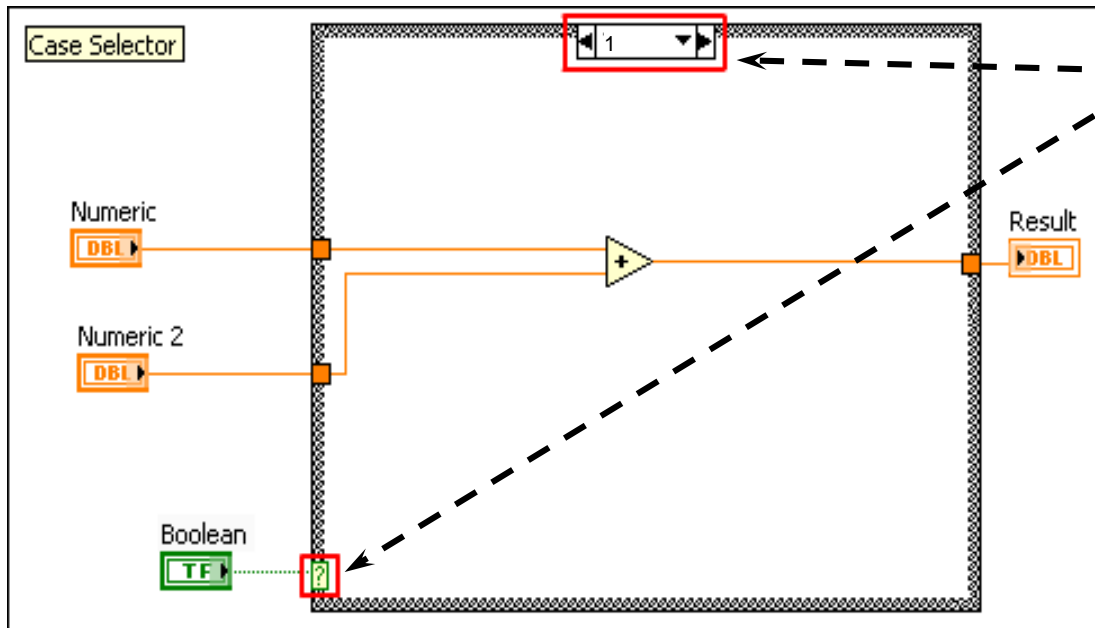
- ① 10..20 → Especifica que incluye todos los números comprendidos entre 10 y 20 incluidos estos dos valores.
- ② ..50 → Representa todos los números menores o iguales a 50.
- ③ ..10, 12..14 → Comprende números menores o iguales a 10 y comprendidos entre 12 y 14 (incluidos)

- Si se cablea una variable en coma flotante al terminal de condición, LabView redondea su valor al entero más próximo. **El selector del Case no permite variables en coma flotante.**



### Estructura Case: Selector

- Si el selector contiene un dato de tipo diferente al del terminal de condición, el valor aparecerá marcado en rojo para indicar esta circunstancia.

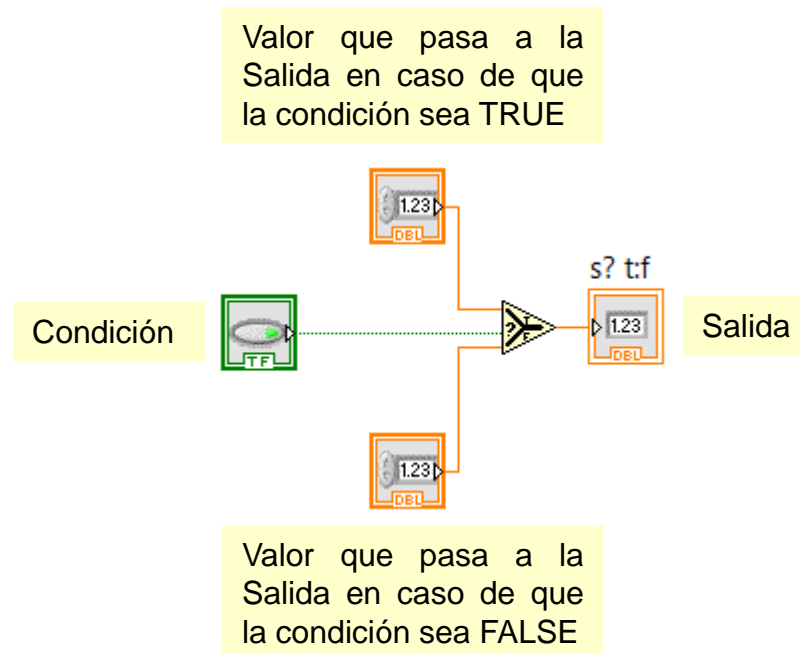


No coinciden los tipos del Selector del Case y del terminal de condición.



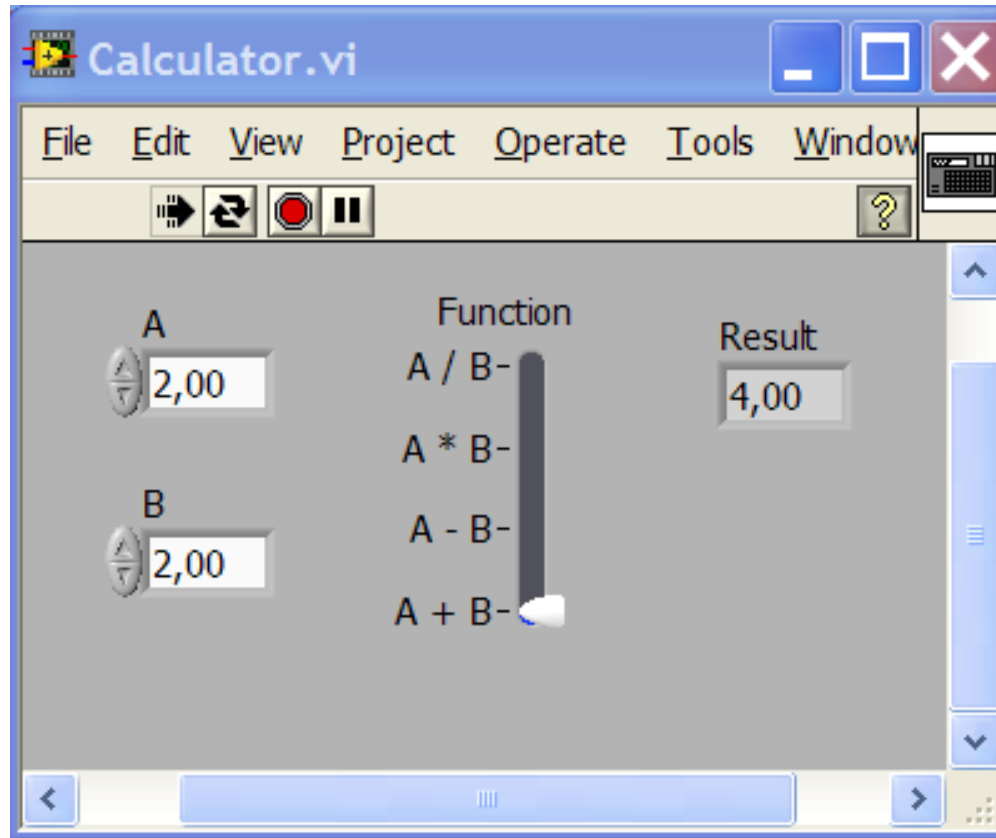
### Estructura Case: Selector

- **Función *Select*** : permite determinar que dato se selecciona en función de que la condición sea verdadera o falsa.
  - Se encuentra en la subpaleta **Comparison** del diagrama de bloques.



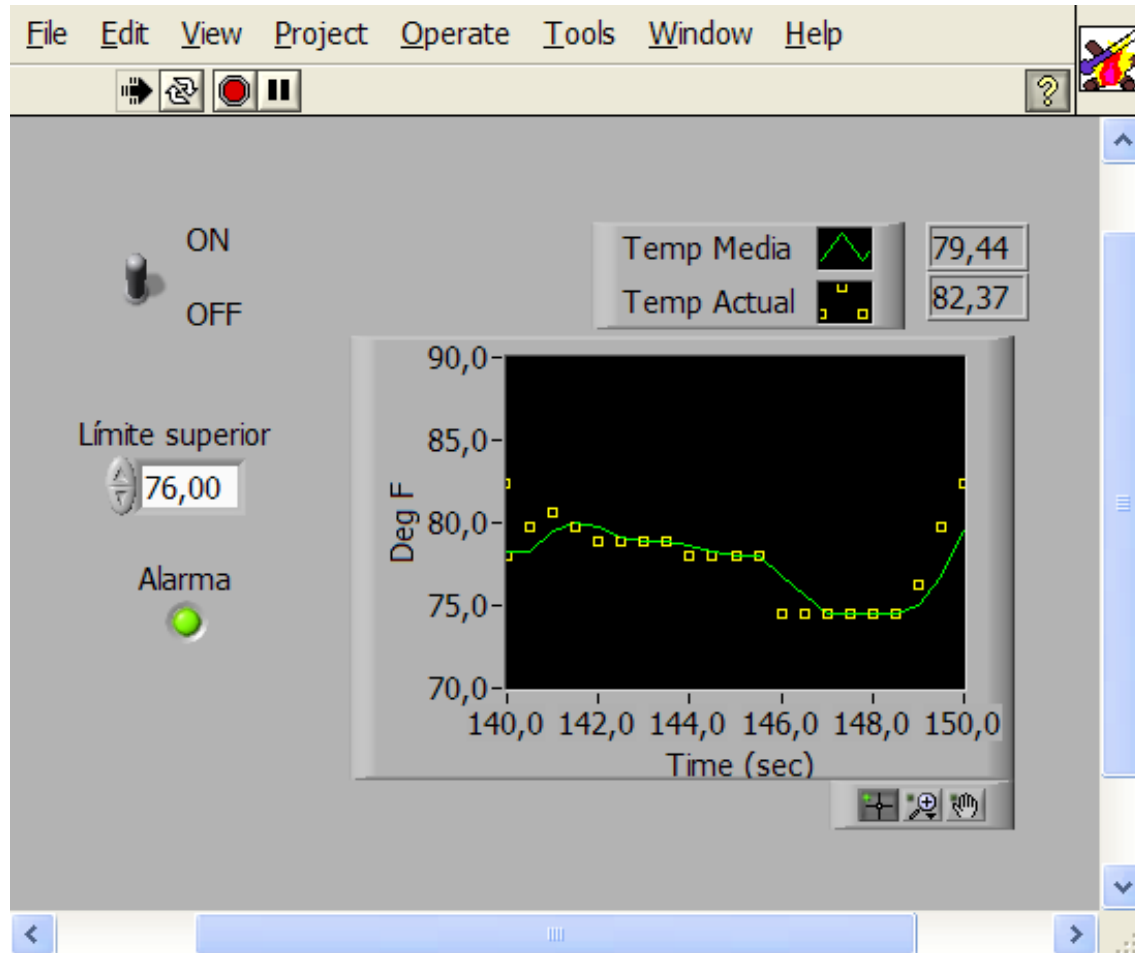


### Ejercicio 6: calculadora con estructura Case.








### Ejercicio 7: control de temperatura.







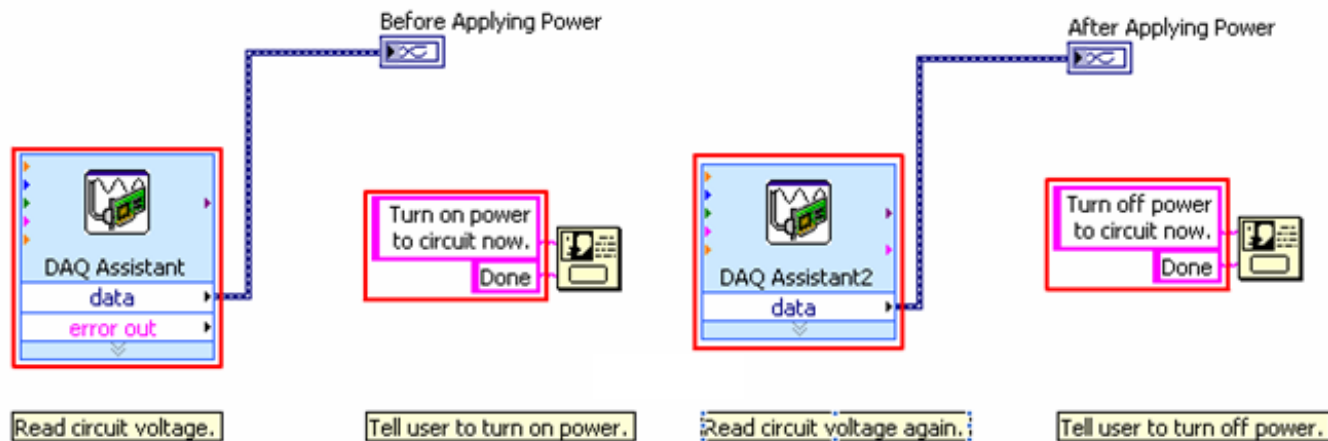
### Ejercicio 7: control de temperatura (Continuación).

- 1.- En el gráfico de tipo **Waveform Chart** se representan dos trazas, una que contiene la temperatura actual (traza amarilla) y otra que contiene el promedio de las tres últimas temperaturas (traza verde).
- 2.- Para simular la temperatura actual utilizar una función *random* que genere temperaturas entre 70 y 90°C.
- 3.- Cuando la temperatura actual supere el límite prefijado en el control *Límite Superior* deberá encenderse el led *Alarma* y emitirse un sonido Beep  (*Paleta de funciones* → *Programming* → *Graphics & Sounds*).
- 4.- Las muestras deben adquirirse cada 500 ms.
- 5.- Para almacenar los valores de las últimas temperaturas y poder promediarlas deben utilizarse registros de desplazamiento.
- 6.- Para hacer el promediado utilizar el nodo **Compound Arithmetic**  (*Paleta de funciones* → *Arithmetics* → *Numeric*)
- 7.- Para representar las dos trazas en el gráfico utilizar el nodo **Bundle**  (*Paleta de funciones* → *Programming* → *Cluster & Variant*).



### Estructura Sequence

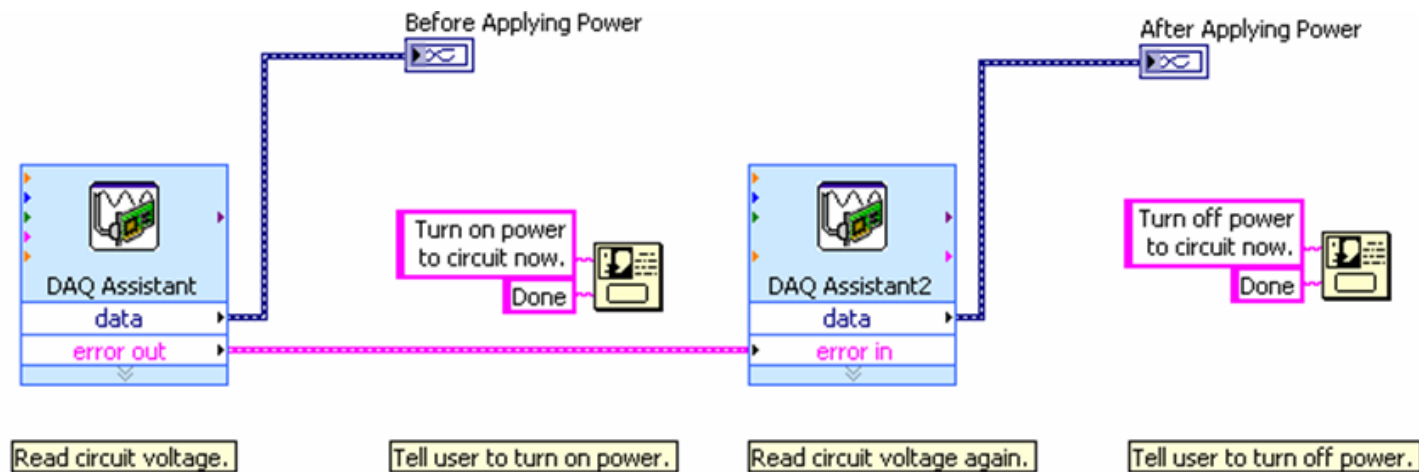
- La mayoría de VI's programados con LabView realizan tareas secuenciales que pueden programarse de diferentes maneras.
- Considerar por ejemplo el siguiente VI:
  - Se adquiere una tensión.
  - Se muestra una caja de diálogo para indicar al usuario que conecte la alimentación del circuito.
  - Se adquiere otra tensión.
  - Se muestra otra caja de diálogo indicándole al usuario que desconecte la alimentación del circuito





### Estructura Sequence

- En el VI anterior no existe ningún elemento que indique el orden de ejecución, por tanto no se puede saber cual se ejecutará en primer lugar.
- Para solucionar este problema se pueden cablear los subVI's a través del *error cluster* tal y como se muestra en la siguiente figura:

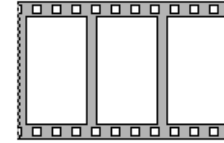


- De esta manera se fuerza el orden de ejecución de los DAQ Assistants, pero no el de las cajas de diálogo ya que están no contienen *error cluster*.

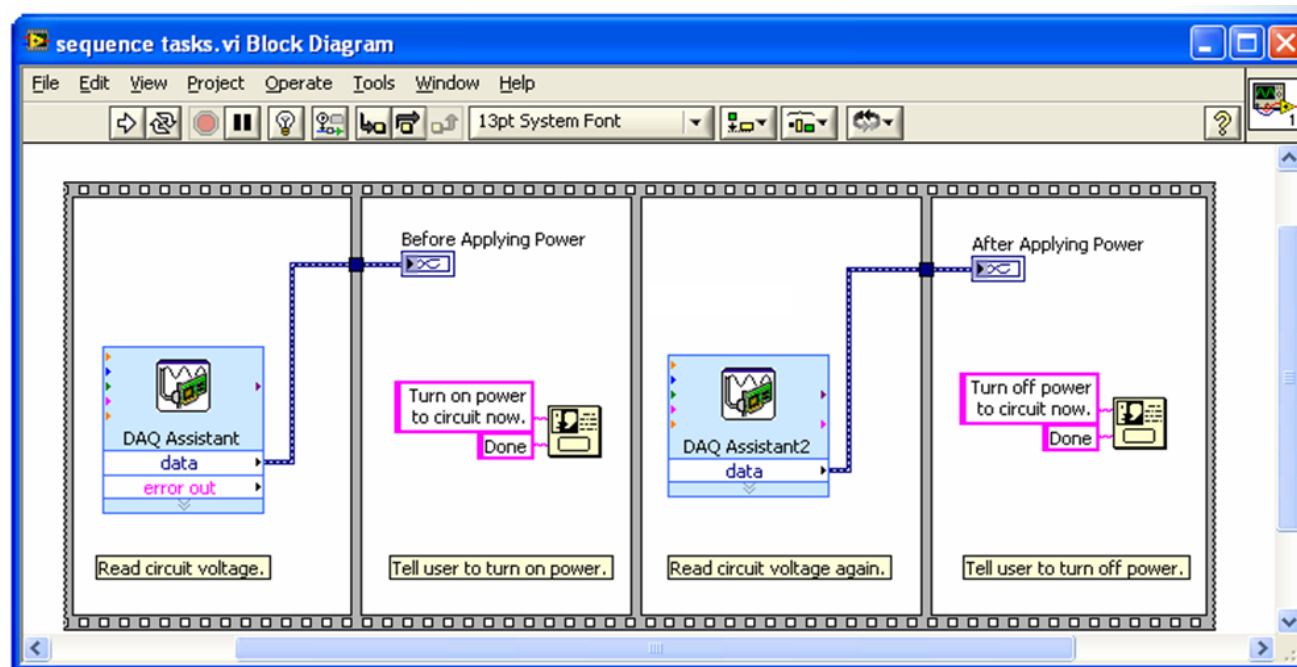


### Estructura Sequence

- Para solucionar este problema se utiliza la estructura SEQUENCE.



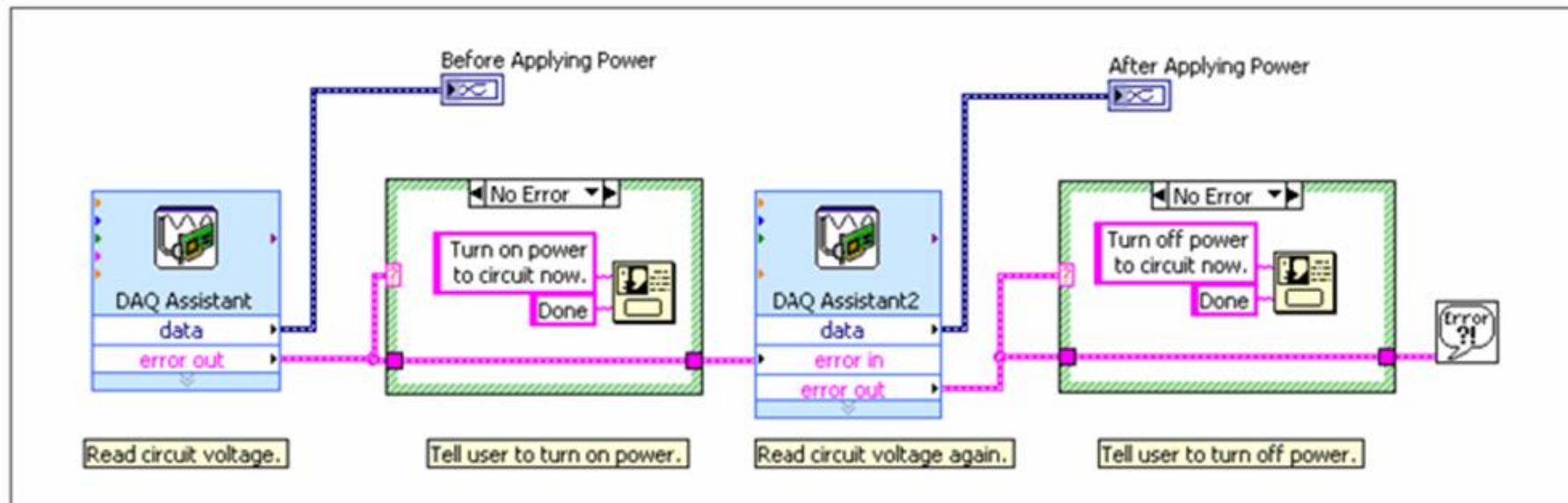
- La estructura SEQUENCE fuerza el orden en que se ejecuta el código. Está formada por varios fotogramas, de manera que se ejecuta el contenido de cada fotograma de manera secuencial. Hasta que no se completa la ejecución de un fotograma no se ejecuta el siguiente.
- El ejemplo anterior quedaría tal y como se muestra en la siguiente figura:





### Estructura Sequence

- Aunque la estructura SEQUENCE garantiza el orden de ejecución, por tanto **prohíbe la ejecución de operaciones en paralelo**, su utilización debería por tanto evitarse siempre que sea posible con el fin de aprovechar al máximo las ventajas del paralelismo que intrínsecamente ofrece Labview.
- Otro aspecto negativo es que la estructura SEQUENCE NO permite parar la ejecución a través de la secuencia.
- La mejor manera para establecer la secuencialidad de los VI's es utilizar estructuras Case y cablear el error cluster al terminal de operación del Case, permitiendo de esta manera que el código contenido en el Case (las cajas de diálogo en el ejemplo anterior), se ejecute únicamente si no hay errores:



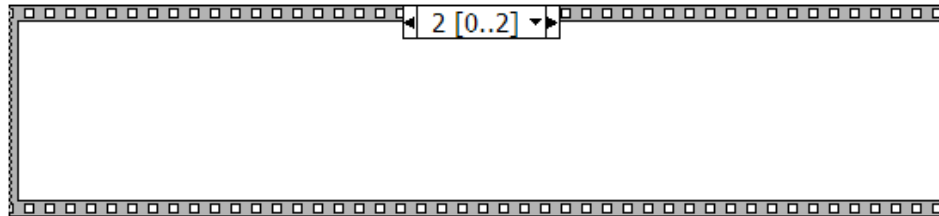


### Estructura Sequence

- Hay dos tipos de estructura SEQUENCE: STACKED SEQUENCE y FLAT SEQUENCE.

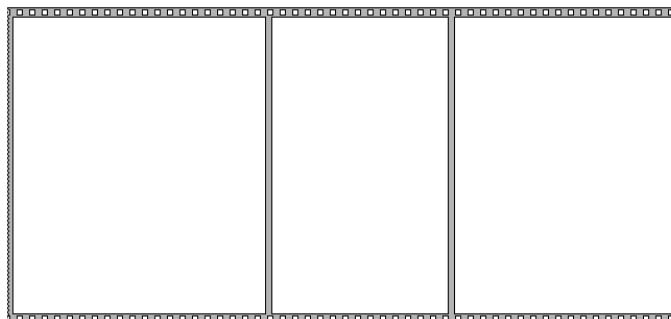
- **STACKED SEQUENCE.**

Tiene un menú en la parte superior en el que se indica la numeración del fotograma que se muestra y el número total de estos que contiene.



- **FLAT SEQUENCE.**

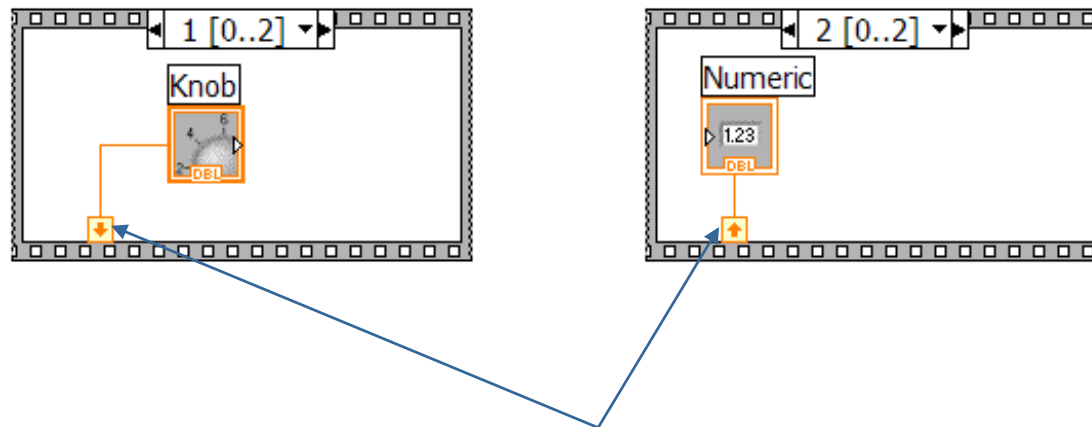
Es más visual, en este caso los fotogramas se visualizan uno a continuación de otro.





### Estructura Sequence

- Para pasar datos de un fotograma a otro de una estructura STAKED SEQUENCE se utiliza un **terminal de secuencia local** (*sequence local terminal*).



**Terminales de secuencia local** para pasar datos de un fotograma a otro en una estructura STAKED SEQUENCE

- La estructura STAKED SEQUENCE puede tener túneles y se tratan de la misma manera que en el CASE: solo pueden tener una salida y puede haber distintas entradas en cada fotograma.



### Estructura Event: programación basada en eventos.

- **Programación basada en eventos:** método de programación en el que los programas esperan que se produzca un evento para ejecutar determinadas acciones:

#### Evento:

- Acción de usuario sobre el panel frontal.
- Generado por dispositivo de I/O (RS232, SAD, etc.).



Ejecución de código asociado al evento en el diagrama de bloques.

#### • Programación basada en Polling:

- Método de programación en el que mediante un bucle se chequea de forma continua si se ha producido algún cambio en el panel frontal.
- Este método “consume” mucho tiempo de CPU.
- Puede fallar a la hora de detectar determinados cambios si estos se producen en instantes de tiempo similares.

#### • Programación mediante estructura **Event**:

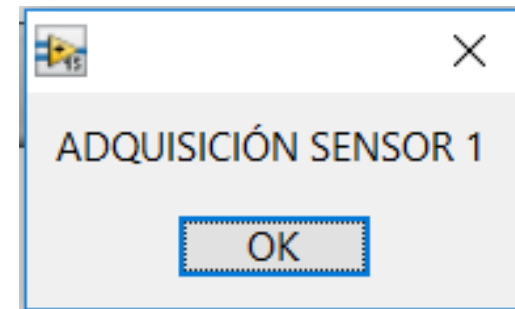
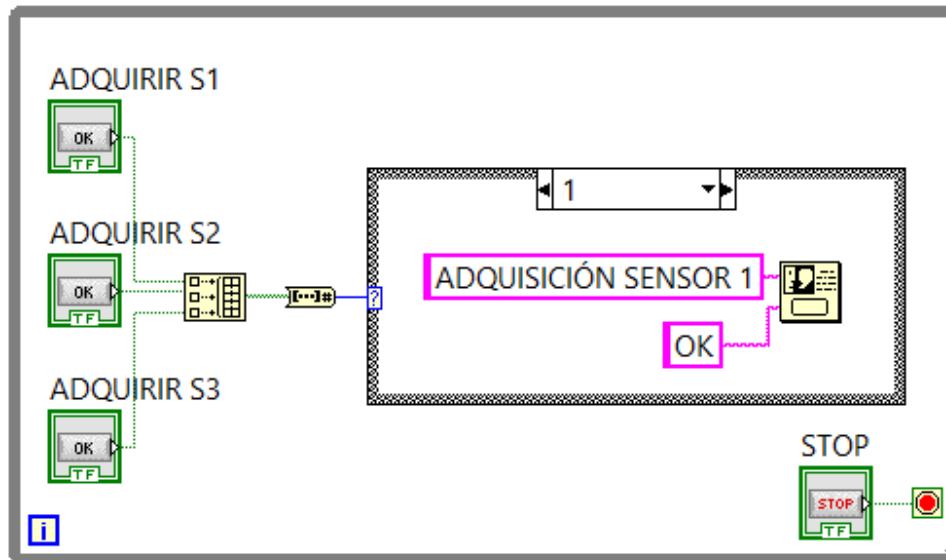
- Método de programación basado en eventos que elimina la necesidad de sondeo continuo.
- Se reducen notablemente los requerimientos de CPU.
- Simplifica el código del diagrama de bloques.
- No hay posibilidad de pérdida de eventos → los eventos recibidos se almacenan en una cola de eventos.
- Maneja/gestiona los eventos generados en el interface de usuario (panel frontal).





### Estructura Event: programación basada en eventos.

- Ejemplo de programación basada en eventos por **Polling** → Bucle continuo → Poco eficiente desde el punto de vista de CPU.





### Estructura Event: programación basada en eventos.

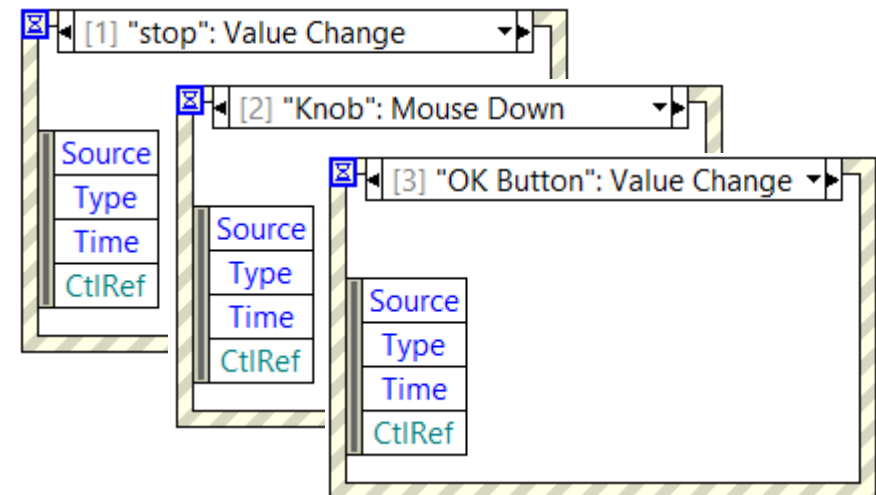
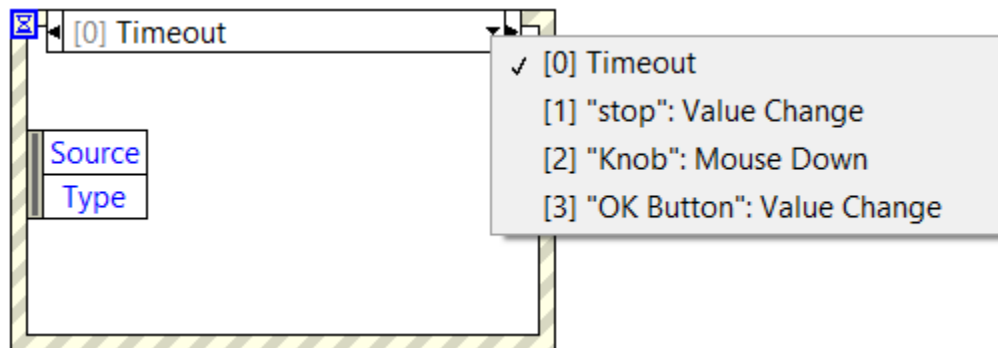
- Conceptos **Programación Basada en Eventos:**

1. **Definición de evento:** es una notificación asíncrona de que algo ha ocurrido, por ejemplo cualquier acción generada por el usuario sobre el interface de usuario de una aplicación: pulsar un botón determinado del interface de usuario con el botón izquierdo del ratón o con el derecho, pulsar una determinada tecla, mover el ratón, mover un potenciómetro para que cambie de valor, etc.  
  
**NOTA:** aunque la estructura Event maneja los eventos generados en el interface de usuario, los eventos pueden tener otros orígenes como dispositivos de entrada/salida (puerto serie, tarjetas de adquisición , etc) u otras partes de código.
2. **Cuando se produce un evento el sistema operativo (SO) es el responsable de identificar y responder al evento.** El SO indica/informa a la aplicación los eventos que el usuario genera sobre los distintos objetos u elementos que forman el interface de usuario. Para ello se debe indicar al sistema operativo (SO) cuales son los objetos del interface de usuario y los eventos asociados a estos que se desean detectar para ejecutar algún tipo de acción → **registro de eventos en el SO.**
3. Registrados los eventos en el SO, cada vez que se produzca uno de ellos, el SO “avisa” a la aplicación y está ejecuta el código asociado al evento generado por el usuario.
4. Cuando se produce un evento, la aplicación ejecuta el código asociado y queda a la espera del siguiente evento.
5. Los eventos se ejecutan en el orden en los que se han generado o producido.



### Estructura Event: programación basada en eventos.

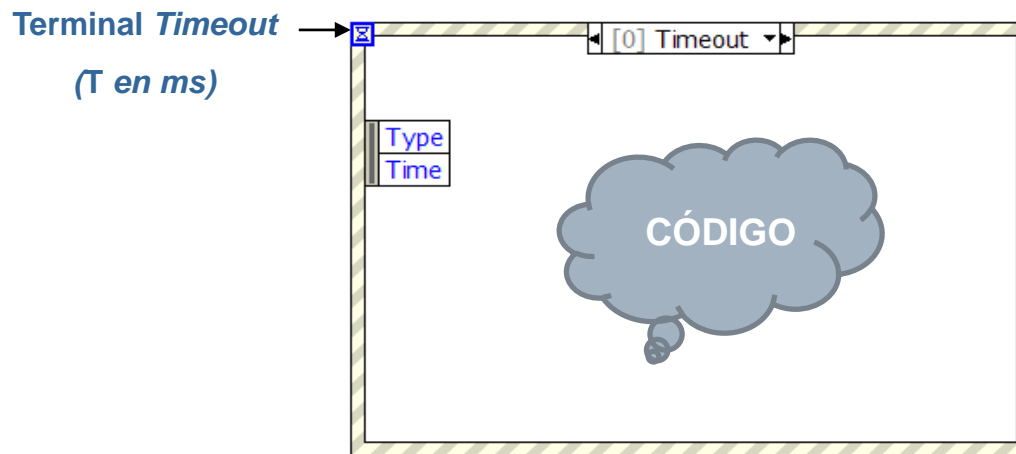
6. Es la estructura **más eficiente** para capturar la actividad (eventos) en el interface de usuario de una aplicación ya que **paraliza/duerme** la **ejecución de los bucles en los que se encuentra hasta que recibe del SO uno de los eventos** previamente registrados por la estructura.
7. Mientras la estructura está a la espera de eventos enviados desde el sistema operativo **NO se consume tiempo de CPU**, de ahí su mayor eficiencia. **La estructura EVENT bloquea el programa (no se leen el resto de controles ubicados fuera de la estructura event)** hasta que se genera un evento: **TIMEOUT, pulsar un botón, etc.**
8. Contiene varios subdiagramas en los que se definen los distintos eventos registrados para el sistema operativo. Para añadir subdiagramas: opción *Add Event Case* del menú flotante.





### Estructura Event: programación basada en eventos.

9. Evento por defecto: evento **TIMEOUT**. El código del subdiagrama para el evento TIMEOUT se ejecuta cuando pase el número de milisegundos indicados en el terminal *Event Timeout* **sin que se haya producido alguno de los eventos registrados en el resto de subdiagramas**.



**Ejemplo** → Si se cablea al terminal TIMEOUT el valor 1000 (1 segundo), el sistema operativo generará el evento TIMEOUT si transcurrido 1 segundo no se ha generado/producido alguno de los eventos definidos en el resto de subdiagramas.

En definitiva, el timer que controla la generación del evento TIMEOUT se resetea cada vez que se genera/produce alguno de los eventos registrados en la estructura Event.



### Estructura Event: Funcionamiento.

- Por defecto el valor del terminal TIMEOUT es -1, lo que indica que al sistema operativo que **NO** debe generar nunca el evento TIMEOUT.
- En este caso, la estructura EVENT solo responde a los eventos generados por el usuario. El subdiagrama del evento TIMEOUT es como si no existiera. Por tanto, solo cuando se genere uno de los eventos registrados es cuando se leerá y se ejecutará el código ubicado fuera de la estructura EVENT.
- Por este motivo se debe añadir un subdiagrama para el botón STOP, si no se registra evento para el botón STOP su estado no se lee y por tanto el programa no finalizaría nunca.

### NOTAS:

1. Cuando se utiliza la estructura EVENT para la gestión del interface de usuario es conveniente que se incluyan en este todos los controles gestionados por sus correspondientes eventos.
2. No deben incluirse dos o más estructura EVENT en el mismo programa (vi), en caso contrario obtendremos funcionamientos anómalos.

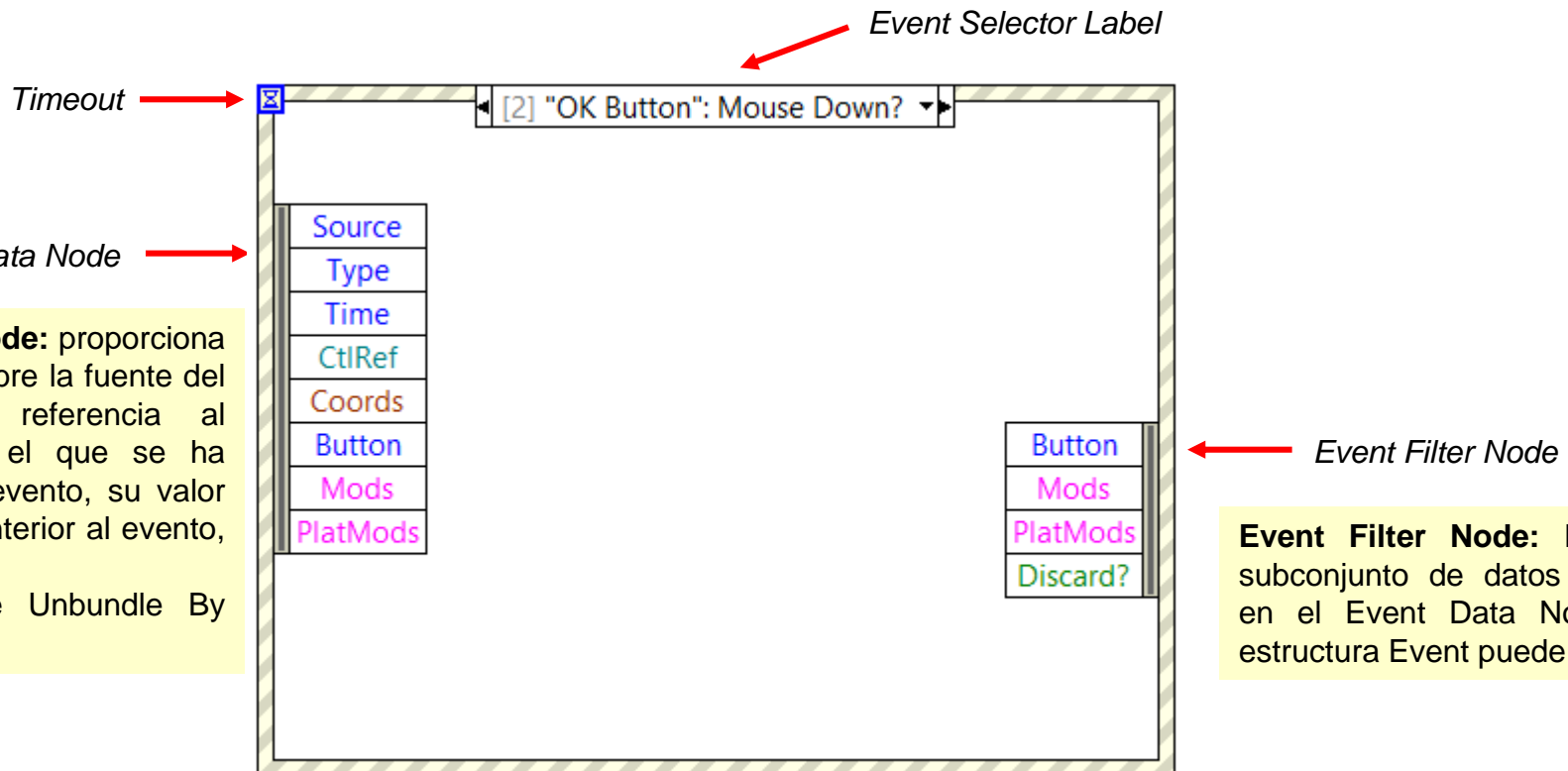
(véase <http://zone.ni.com/reference/en-XX/help/371361M-01/lvhowto/caveatsrecmndtnsevnts/>).

- Eliminar el subdiagrama del evento TIMEOUT equivale a cablearle -1 al terminal de TIMEOUT. De hecho, si no se va a utilizar el evento TIMEOUT se puede eliminar este subdiagrama.



### Estructura Event: elementos estructura Event.

- La estructura Event está formada por los siguientes elementos: *Event Data Node*, *Event Filter Node*, *Event Selector Label*, terminal para *Timeout*. que aporta información sobre la fuente del evento: una referencia al control sobre el que se ha provocado el evento, su valor actual, valor anterior al evento, etc.



**Event Data Node:** proporciona información sobre la fuente del evento: una referencia al control sobre el que se ha provocado el evento, su valor actual, valor anterior al evento, etc.

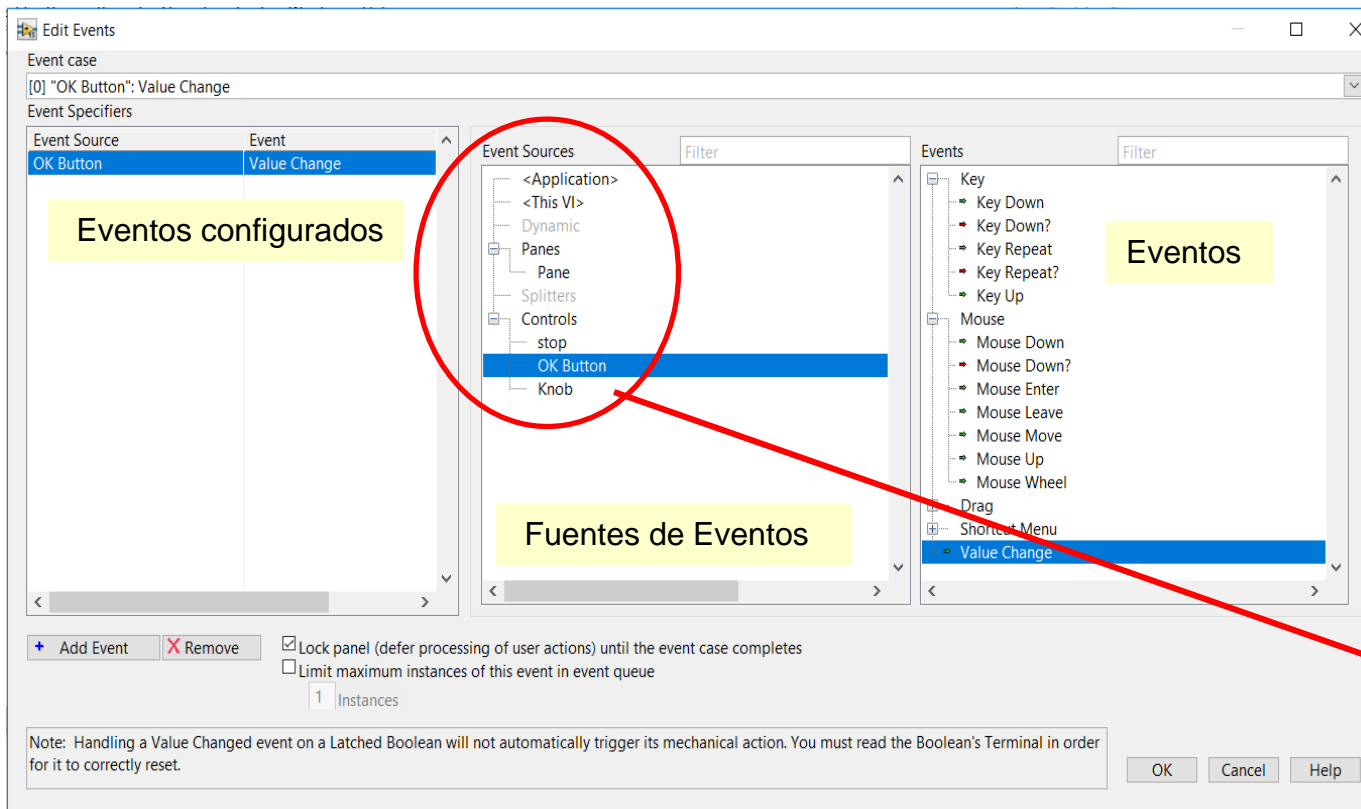
Similar to the Unbundle By Name function.

**Event Filter Node:** Identifica el subconjunto de datos disponibles en el Event Data Node que la estructura Event puede modificar.



### Estructura Event: programación basada en eventos.

- Los eventos que controlan la ejecución de cada subdiagrama (eventos registrados en el sistema operativo) se asocian a través del panel *Edit Events*.
- Los subdiagramas se añaden de la misma manera que en las estructuras CASE, a través del menú contextual que aparece al hacer clic con el botón derecho del ratón sobre el borde de la estructura.



- Algunas características de la estructura *Event*.

– No hace polling, por lo que no se sobrecarga al procesador.

– Se garantiza la captación de todos los eventos.

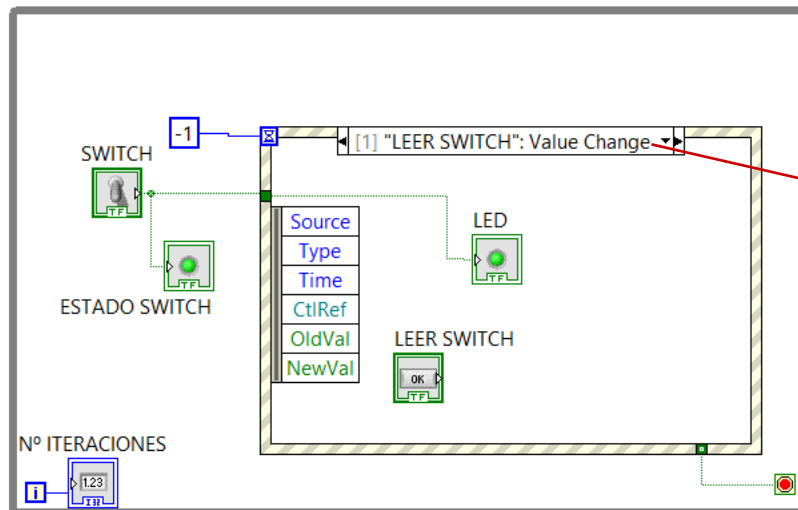
– Los subdiagramas (cases) de eventos se ejecutan en el orden en que se han generado o producido.

Se pueden registrar eventos asociados al VI (<This VI>), al panel (*Pane*) o a cualquier control presente en el interface de usuario (*Controls*).



### Ejemplo para verificar funcionamiento de la Estructura Event

- Para verificar el funcionamiento de la estructura EVENT se diseña el siguiente programa:



En el código solo se registran los eventos asociados a los botones LEER ESTADO SWITCH Y STOP.

- Según se puede deducir del código fuente se pretende el siguiente funcionamiento:
  - El led ESTADO SWITCH indica el estado del control SWITCH.
  - Al pulsar el botón LEER ESTADO SWITCH el indicador LED muestra también el estado del control SWITCH.
  - El indicador Nº ITERACIONES muestra el número de iteraciones del bucle que se han ejecutado





### Ejemplo para verificar funcionamiento de la Estructura Event (Cont.)

- Modificar el código cableando los siguientes tiempos al terminal de TIMEOUT:
  - TIMEOUT = -1
  - TIMEOUT = 2000 (2 segundos)
  - TIMEOUT = 0
- Observe y **justifique** el comportamiento del código en cada caso.

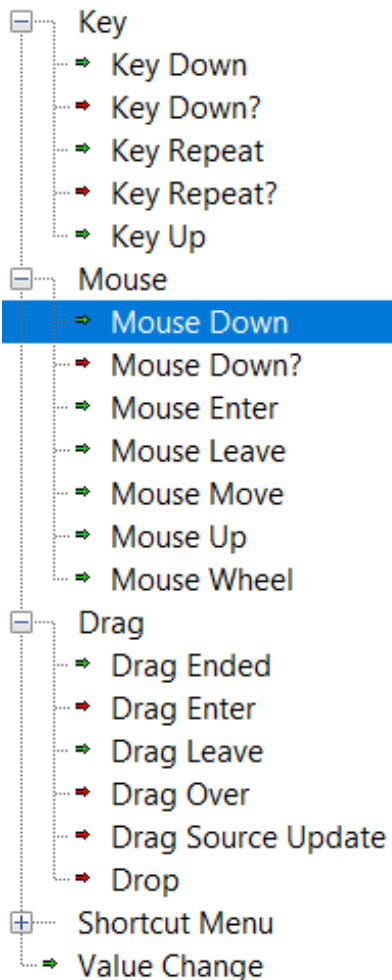


### Estructura Event: Tipos de eventos.


- Los eventos se dividen en dos categorías: **Filtrado (*Filter Events*)** y **Notificación (*Notify Events*)**.
  1. **Eventos de Notificación:** son los que informan de los eventos generados sobre el panel frontal.
    - **Ejemplos eventos asociados a operaciones sobre el panel frontal:** se ha pulsado botón del ratón, se ha liberado botón del ratón, se ha movido el ratón, se ha arrastrado un objeto, se ha pulsado una tecla, etc.
  2. **Eventos de Filtrado:** son eventos capturados por la estructura *Event* antes de que Labview los procese.
    - El código asociado al case del filtro de eventos se utiliza para decidir si el evento debería ser o no procesado.
    - **Ejemplo:** Filtro de evento "*Panel Close?*" utilizado para presentar en pantalla un pop-up que indique al usuario si realmente desea finalizar la aplicación.



### Estructura Event: Tipos de eventos.



Los eventos de Notificación se identifican mediante una flecha de color verde:

 **Evento de Notificación**

Los eventos de Filtrado se identifican mediante una flecha de color rojo:

 **Evento de Filtrado**

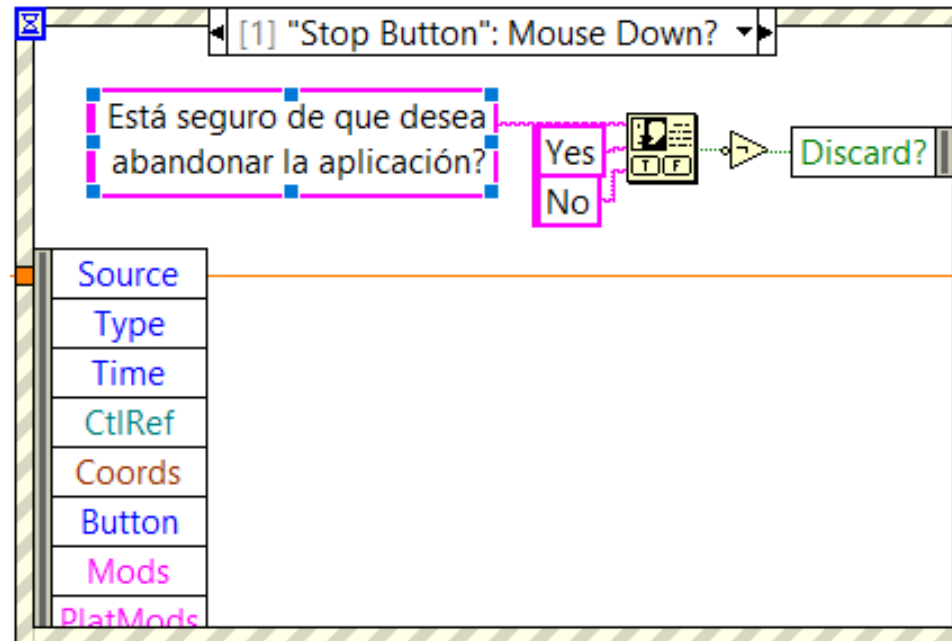
Detalle y descripción de eventos en:

[http://zone.ni.com/reference/en-XX/help/371361B-01/lvprop/control\\_e/](http://zone.ni.com/reference/en-XX/help/371361B-01/lvprop/control_e/)



### Estructura Event: Tipos de eventos.

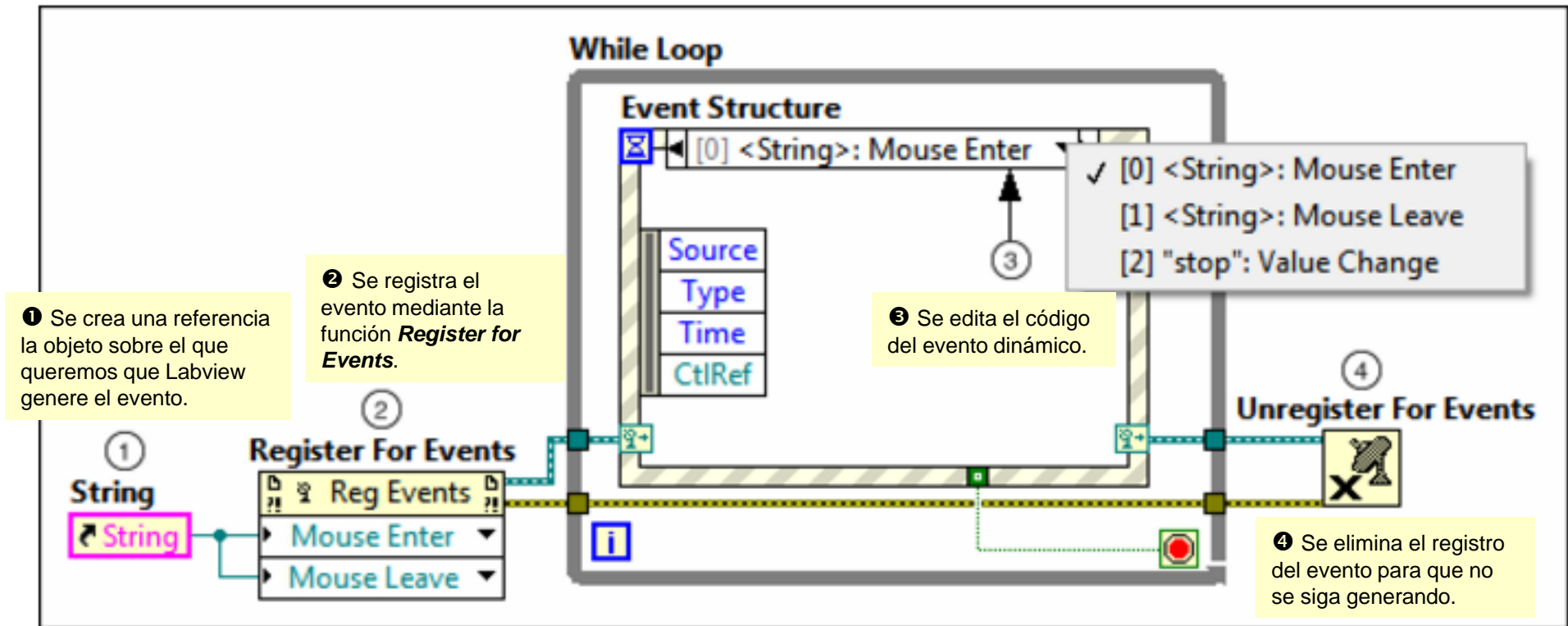
**EJEMPLO:** Filtro de eventos para consultar si realmente se desea abandonar la aplicación.





### Estructura Event: eventos dinámicos.

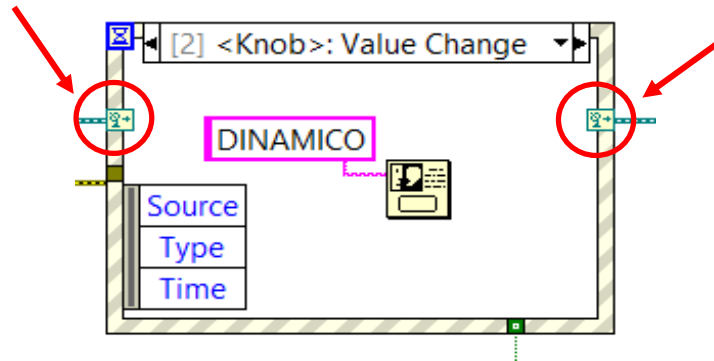
- **EVENTOS DINÁMICOS:** utilizados para registrar eventos de la aplicación, vi o controles del panel frontal en tiempo de ejecución.
- Paleta **Dialog & User Interface** → **Events** → **Register for Events**.
- El proceso se realiza en cuatro pasos (ver [http://zone.ni.com/reference/en-XX/help/371361K-01/lvhowto/dynamic\\_register\\_event/](http://zone.ni.com/reference/en-XX/help/371361K-01/lvhowto/dynamic_register_event/)):



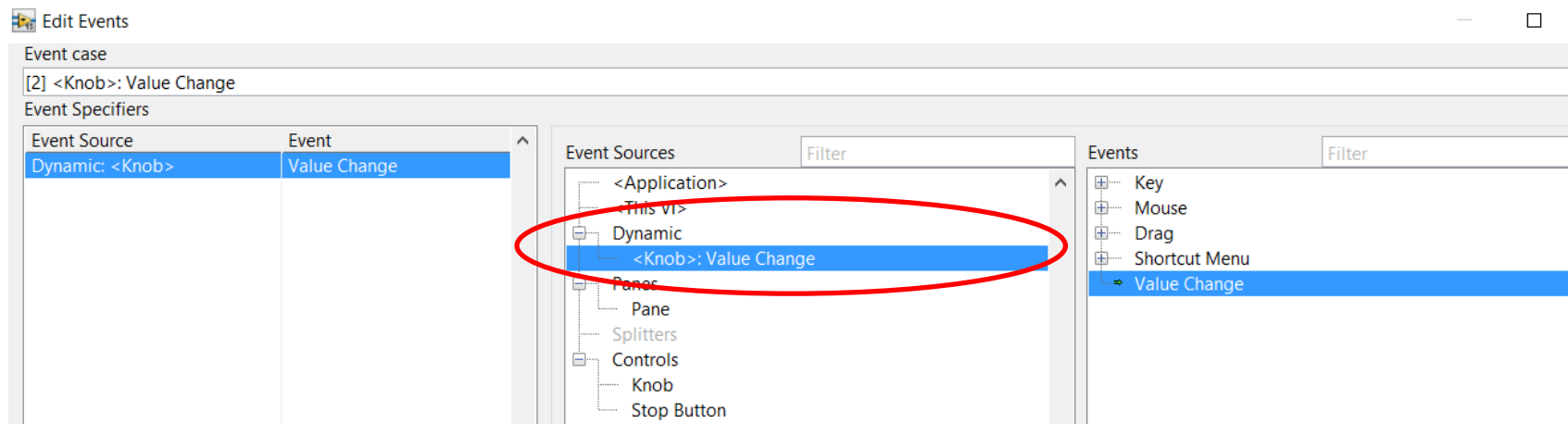


### Estructura Event: Eventos Dinámicos.

- Para que aparezcan los icono que indican que se trata de un evento dinámico seleccionar *Show Dynamic Event Terminals* del menú flotante de la estructura Event.



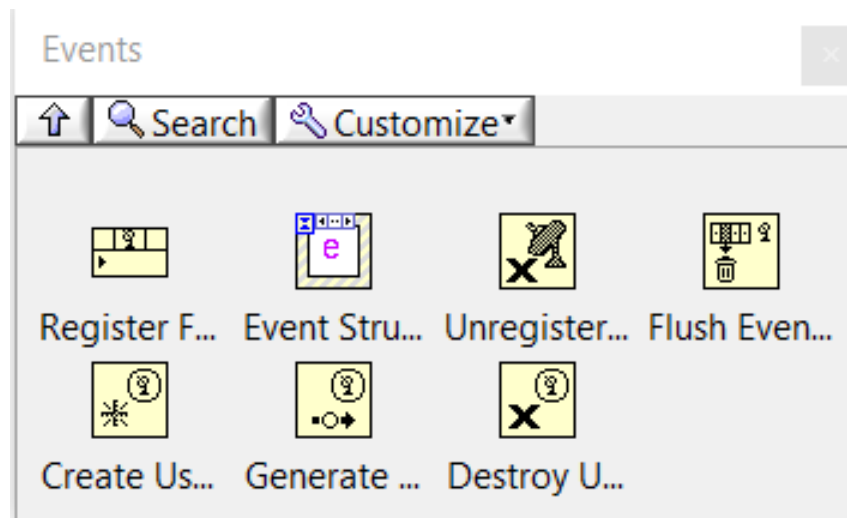
- Para editar el subdiagrama correspondiente al evento dinámico seleccionarlo en la ventana de edición de eventos en la sección **Dynamic**:





### Estructura Event

- **EVENTOS DINÁMICOS:** en la paleta *Dialog & User Interface* → *Events* → *Register for Events* se encuentran las funciones o vi's para la gestión de eventos dinámicos.

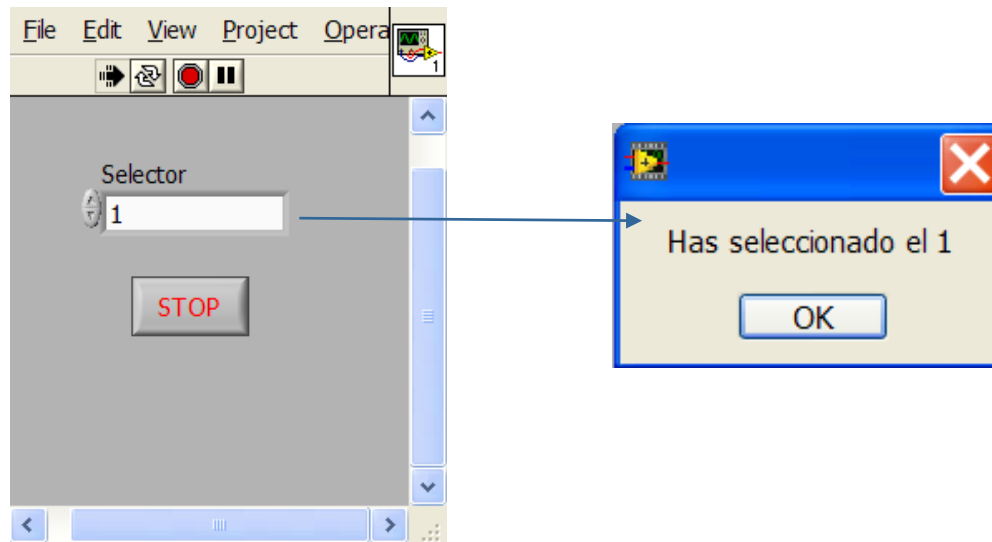




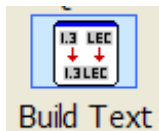
### Ejercicio 8: estructura Event.

Cada vez que se seleccione con el control *Selector* un número comprendido entre 0 y 3, deberá aparecer un mensaje pop-up indicando el número que se acaba de seleccionar.

Si transcurren más de 5 segundos sin modificar el valor del selector, se deberá mostrar un mensaje indicando esta circunstancia.



Para construir el mensaje a presentar en el pop-up utilizar el VI express *BUILT TEXT* (*Functions* → *Express* → *Out* → *Built Text*)



Crea un string de salida como combinación de un texto y de uno o varios parámetros de entrada





### Ejercicio 9: estructura Event.

Programar vi para que en la caja de texto aparezca un mensaje indicando el botón que se ha pulsado, tal y como se muestra en la figura.

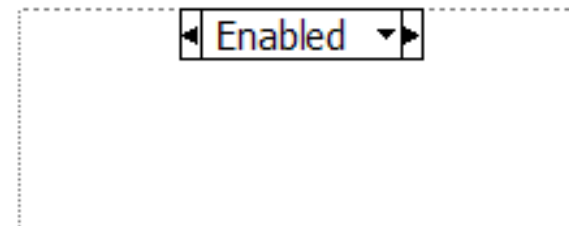
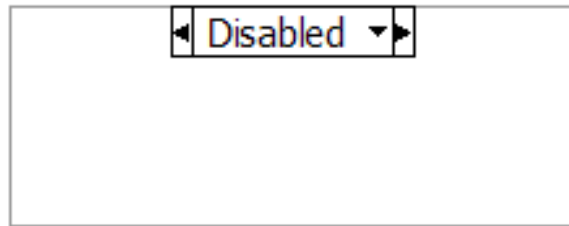
1. Programarlo utilizando técnicas de polling tradicionales, esto es, sin utilizar la estructura *Event*.
2. Programarlo utilizando la estructura *Event*.





### Estructura Diagram Disable

- Esta estructura se utiliza para comentar el código, por lo que son especialmente útiles en los procesos de depuración.

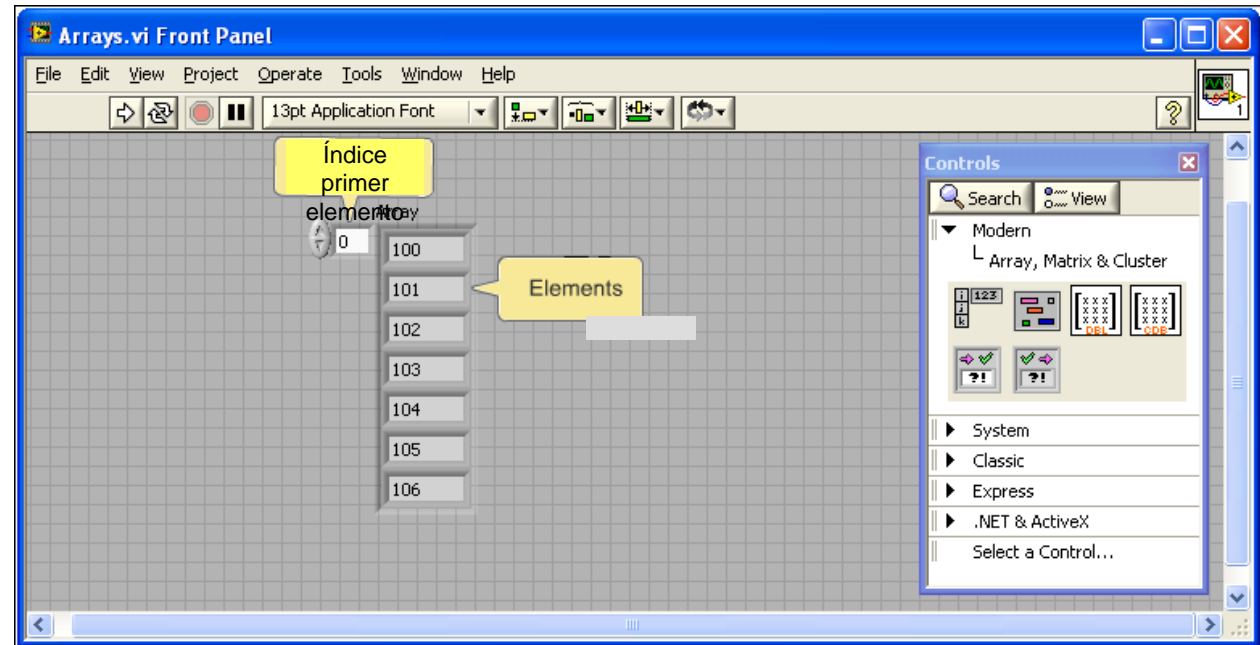
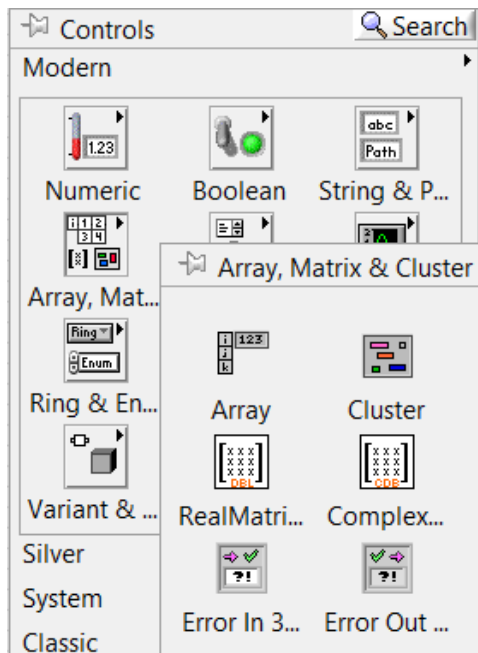


- Funciona como una CASE con dos subdiagramas: *Disabled* y *Enabled*. Se ejecuta únicamente el código contenido dentro del subdiagrama *Enabled*.



### Arrays

- Los arrays se crean en el panel frontal → Paleta Array, Matrix & Cluster → Array.
- Una vez creado, en el panel frontal se visualiza el índice del primer elemento mostrado y elementos del array. Si se desea se pueden añadir barras de desplazamiento.



- El índice del primer elemento del array es cero.

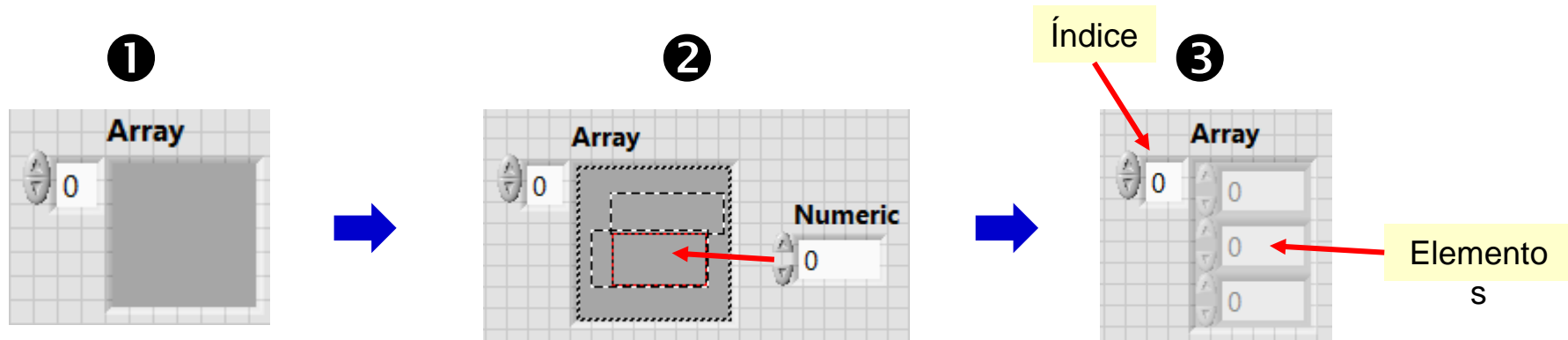


### Arrays: crear arrays.

1. Se crea el array “vacío” **Paleta Array, Matrix & Cluster** → **Array** del panel frontal.
2. Se define el tipo de datos del array. Para ello se crea un control o indicador del tipo deseado y se arrastra al interior del array. Puede definirse un array de cualquiera de los tipos contemplados por Labview: numéricos enteros, reales, booleanos o string.

Si se desea crear un array constante, se debe arrastrar a su interior una constante.

3. Se visualiza el array creado: índice y elementos.



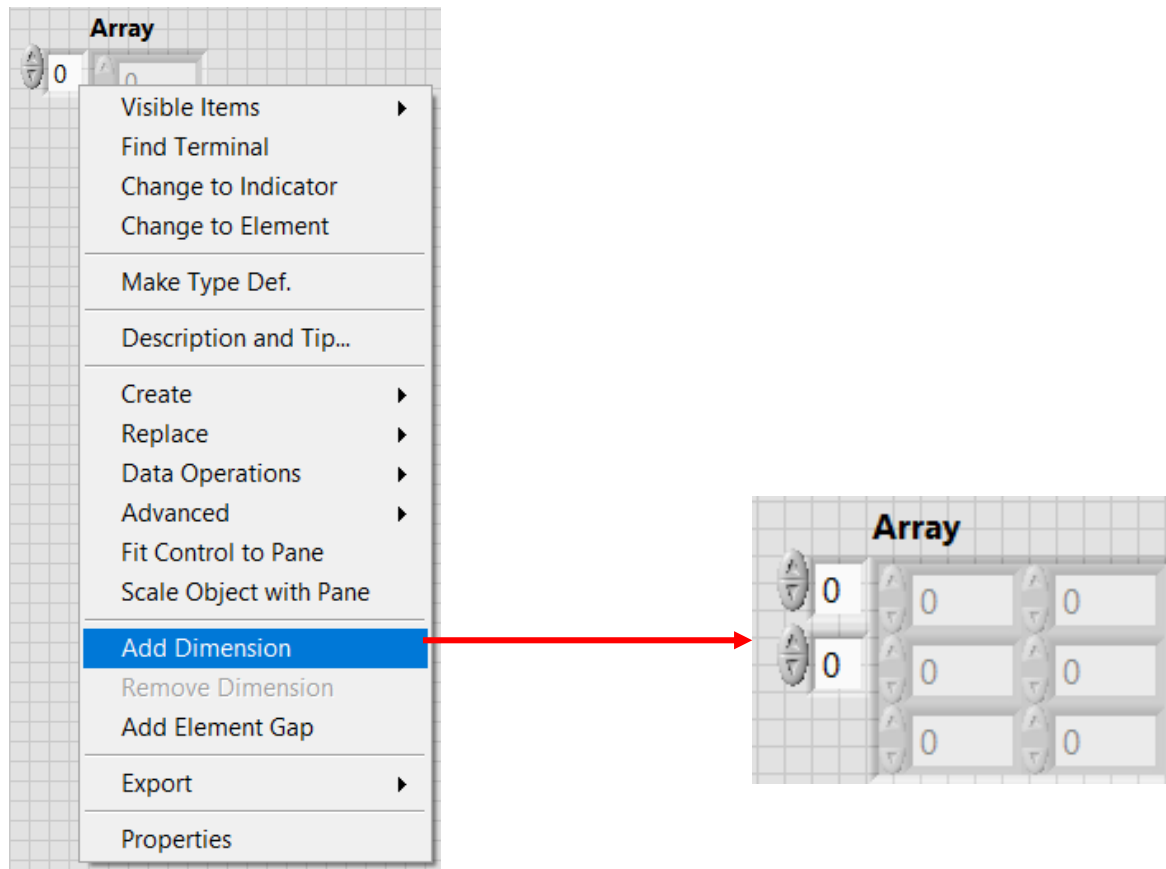
- Se pueden crear arrays constantes desde la paleta de funciones del diagrama de bloques:

*Functions* → *Programming* → *Arrays* → *Array Constant*



### Arrays: crear Arrays de varias dimensiones.

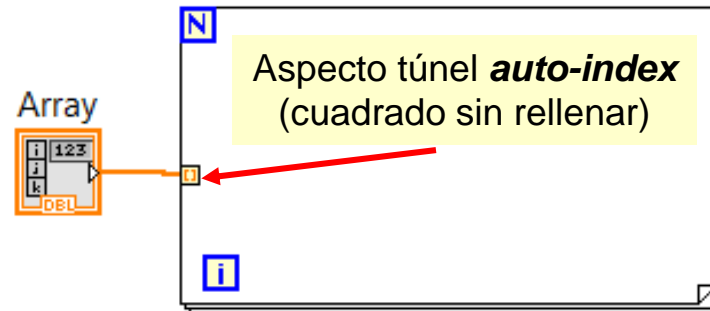
- Para definir arrays de más de una dimensión basta con situarse sobre el índice del array (en el **panel frontal**), hacer clic con el botón derecho del ratón y seleccionar la opción **Add dimension** del menú flotante que aparece en pantalla:



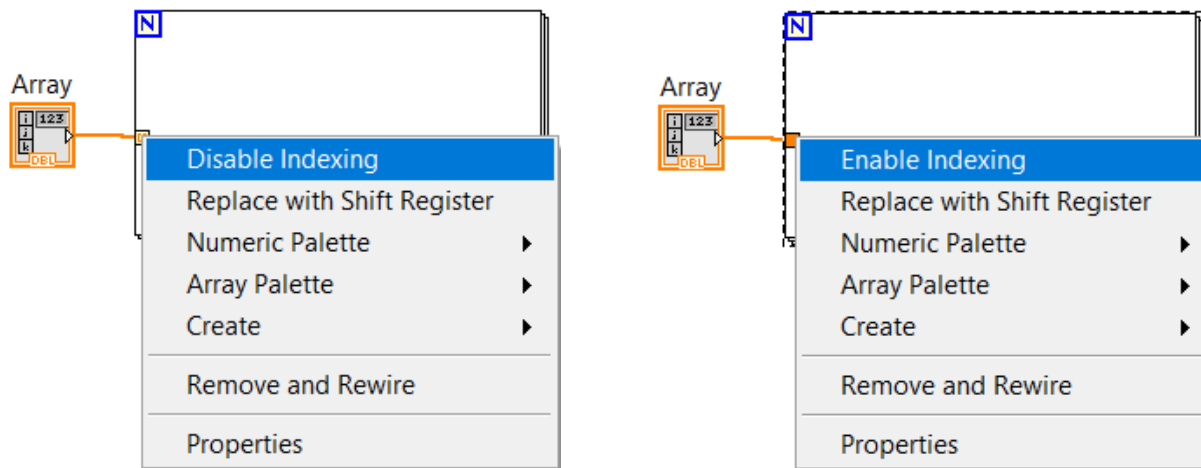


### Arrays: *Auto-Indexing*.

- Si se cablea un array mediante un túnel de entrada a una estructura For o While se crea por defecto un túnel de tipo **auto-index**. Este tipo de túnel permite leer/escribir un elemento del array en cada iteración del bucle.



- El túnel **auto-index** se puede activar o desactivar mediante las opciones **Enable Indexing** y **Disable Indexing** que aparecen en el menú flotante que aparece al hacer clic con el botón derecho del ratón sobre el túnel.



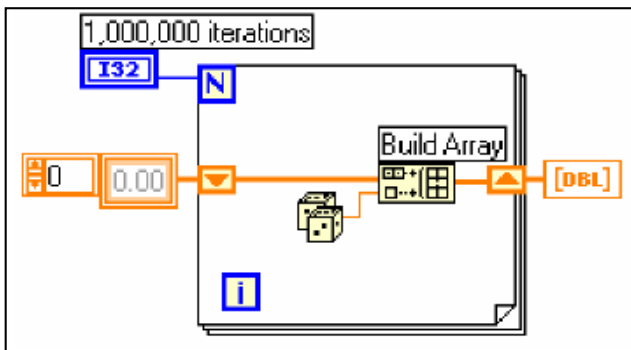


### Arrays: Algunas consideraciones.

- Reubicar memoria tiene un alto coste computacional cuando se realiza de manera continua/iterativa.
- Debe limitarse el uso de las funciones que tienden a causar un “reacomodo” de memoria:
  - En el caso de los Arrays: *Build Array*.
  - En el caso de cadenas de caracteres: *Concatenate Strings*.

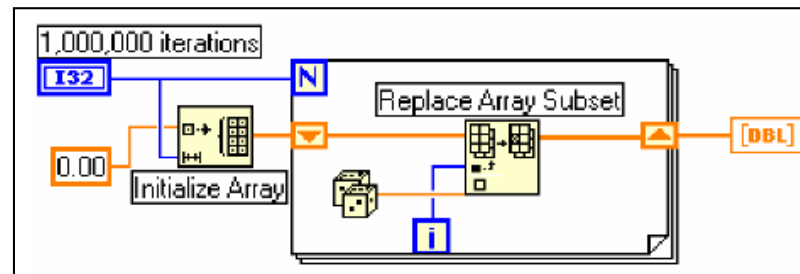
### Ejemplo: generación de un array para almacenar array: 1.000.000 de muestras.

1



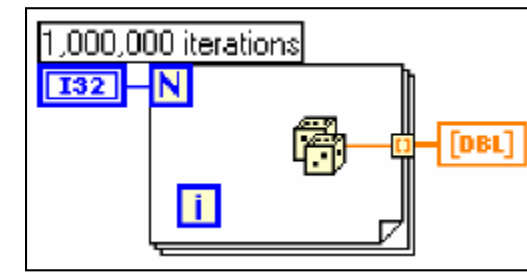
Muy lento debido a que cada iteración implica reubicar memoria.

2



Mucho más rápido ya que solo ubica memoria una vez.

3

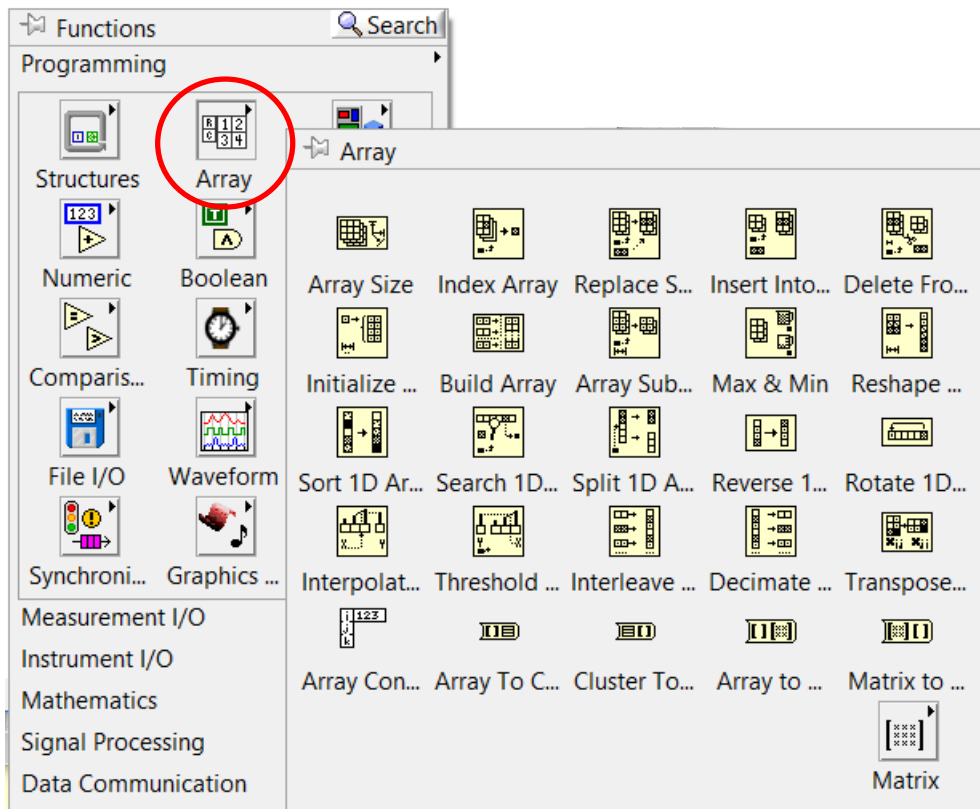


El método más rápido y claro → **auto-index**.



### Arrays: Funciones para gestión de Arrays.

- Disponibles en la paleta de funciones del **Diagrama de bloques**: *Functions* → *Programming* → *Array*



Dentro de las funciones cabe destacar las siguientes:

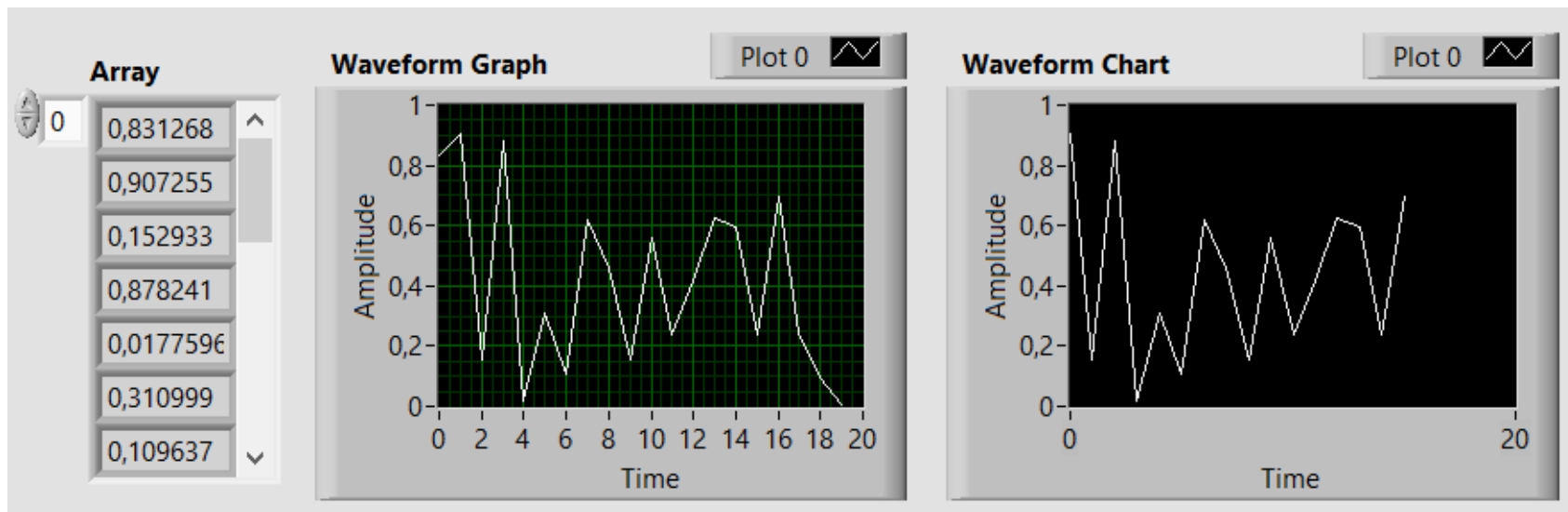
- 1.- **Array Size**: Devuelve el número de elementos de un array.
- 2.- **Initialize Array**: Crea un array n-dimensional e inicializa sus elementos a un valor determinado.
- 3.- **Build Array**: Concatena varios arrays o añade elementos nuevos a arrays n-dimensionales.
- 4.- **Array Subset**: Permite obtener un subarray a partir de un array.
- 5.- **Index Array**: Permite obtener determinados elementos de un array. En el caso de arrays 2D permite obtener filas o columnas completas. **Para extraer filas o columnas completas no se puede utilizar la función *subset array* hay que utilizar la función *index array*.**
- 6.- **Max & Min**: Permite obtener los elementos mayor y menor del array.
- 7.- **Search**: Permite buscar un elemento en un array.
- 8.- **Transposes**: Permite calcular la transpuesta.





### Ejercicio 9: Arrays.

1. Crear un array de 20 números aleatorios con la función *random* para almacenarlos en un array y representarlos en un gráfico de tipo Graph.
2. Una vez generados y representados los 20 números y solamente en el caso de que se tenga garantías de que estas dos operaciones han finalizado, leer los 20 números almacenados en el array y representarlos uno a uno en un gráfico de tipo Chart a intervalos de 500 milisegundos.
3. Cada vez que se ejecute el programa el array y los dos gráficos deberán resetearse.



#### NOTAS:

- Utilizar *Property Nodes* para inicializar los objetos.
- No puede utilizarse la estructura **Sequence** para secuenciar la lectura y escritura del array



### Ejercicio 10: Arrays.

• Generar el siguiente interface de usuario de forma que a partir de dos Arrays originales de 1D y 2D se realicen las operaciones indicadas teniendo en cuenta las siguientes consideraciones:

1. Los Arrays originales 1D y 2D deben generarse mediante código utilizando bucles.
2. Obtener la siguiente información de los Arrays originales 1D y 2D: tamaño, valor máximo y mínimo y las posiciones (índices) en los que se encuentran.
3. Ordenar el array 1D original en orden ascendente y descendente.
4. Reemplazar un elemento en el array 1D y 2D original.
5. Reemplazar la fila indicada en el array 2D por el array 1D.
6. Reemplazar la columna indicada en el array 2D por el array 1D.

**Array 1D Original**

100
101
102
103

**INFORMACIÓN ARRAY 1D**

Tamaño: 4

Máximo: 103 Índice Máximo: 3

Mínimo: 100 Índice Mínimo: 0

**ORDENAR (Ascendente)**

Array Ascendente

100
101
102
103

**ORDENAR (Descendente)**

Array Descendente

103
102
101
100

**REEMPLAZAR**

Índice Elemento: 1

Array Reemplazado

100
55
102
103

Valor Nuevo: 55

---

**Original 2D Array**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

**INFORMACIÓN ARRAY 2D**

Tamaño 2D: 4 4

Máximo 2D: 15 Posición Máximo: 3 3

Mínimo 2D: 0 Posición Mínimo: 0 0

**REEMPLAZAR (Elemento)**

Índice Fila: 1

Índice Columna: 3

Valor Nuevo: 55

Nuevo Array

0	1	2	3
4	5	55	7
8	9	10	11
12	13	14	15

**REEMPLAZAR COLUMNA**

Índice Columna: 3

Array Columna Reemplazada

0	1	2	100
4	5	55	101
8	9	10	102
12	13	14	103

**REEMPLAZAR FILA**

Índice Fila: 2

Array Fila Reemplazada

0	1	2	3
4	5	55	7
100	101	102	103
12	13	14	15



### Ejercicio 11: Arrays.

- Generar el seno y coseno de los ángulos comprendidos entre 1 y 1000 grados para posteriormente representarlos en un gráfico de tipo GRAPH. Mediante el Selector **Selección señal a representar** se elige la señal a representar en el gráfico (seno o coseno).
- Representar los valores seno y coseno obtenidos en los Arrays FILA y COLUMNA tal y como aparecen en el interface de usuario de la figura. En el array *Array a Representar* se deben mostrar los valores de la señal elegida para representar.

**Array Seno-Coseno (FILA) (2x1000)**

0	0,017452	0,034899	0,052336	0,069756	0,087155	0,104528	0,121869	0,139173	0,156434	0,173648	0,190809	0,207912	0,224951	0,241922	Seno
1	0,999848	0,999391	0,99863	0,997564	0,996195	0,994522	0,992546	0,990268	0,987688	0,984808	0,981627	0,978148	0,97437	0,970296	Coseno

Size array FILA: 0 2 1000

**Selección señal a representar**

Seno

**Array Seno-Coseno (COLUMNA) (1000x2)**

	Seno	Coseno
4	0,0697565	0,997564
0	0,0871557	0,996195
	0,104528	0,994522
	0,121869	0,992546
	0,139173	0,990268
	0,156434	0,987688
	0,173648	0,984808
	0,190809	0,981627
	0,207912	0,978148
	0,224951	0,97437
	0,241922	0,970296
	0,258819	0,965926
	0,275637	0,961262
	0,292372	0,956305
	0,309017	0,951057

Size array COLUMNA: 0 1000 2

**Array a Representar (1000x1)**

(Seno o Coseno)
0
0,0174524
0,0348995
0,052336
0,0697565
0,0871557
0,104528
0,121869
0,139173
0,156434
0,173648
0,190809
0,207912
0,224951
0,241922

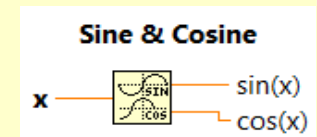
**Waveform Graph**

**Selección señal a representar**

✓ Seno  
Coseno

### NOTA:

- vi para calcular seno y coseno:

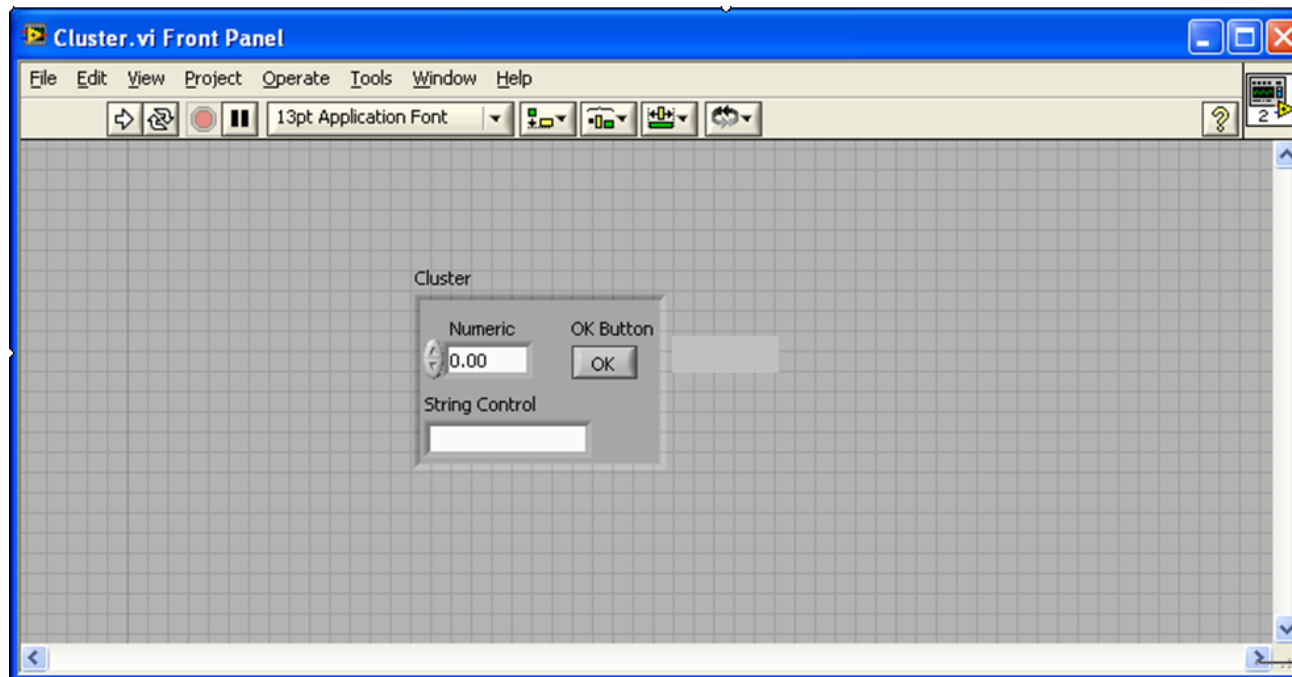


La entrada  $x$  debe estar en radianes.



### Cluster

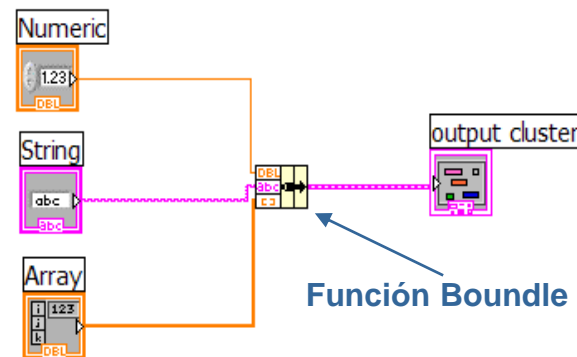
- Un cluster en LabView es el equivalente a las estructuras de los lenguajes basados en texto: una colección de variables de diferentes tipos.
- Se crean desde el panel frontal → **Controls Modern/Classic** → **Array, Matrix & Cluster**
- Para crear un cluster se coloca uno vacío y se arrastran a su interior los tipos de datos que se desea contenga el cluster.
- Se pueden crear también arrays de cluster.



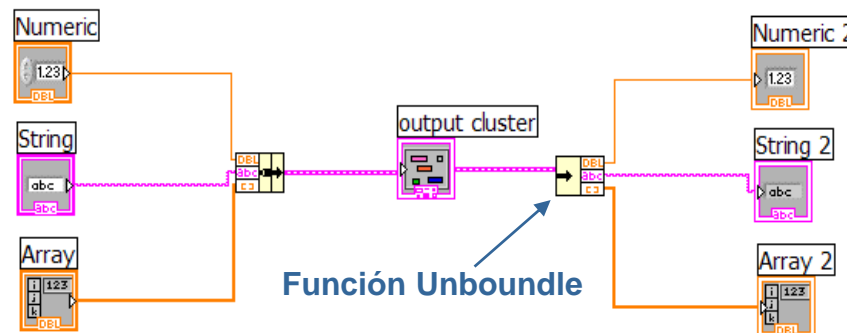


### Cluster

- Se pueden crear cluster en tiempo de programación cableando varios controles o indicadores a través de la función **Bundle** que se encuentra en la paleta de funciones del diagrama de bloques (**Functions**→**Programming**→**Cluster & Variant**→**Bundle**):



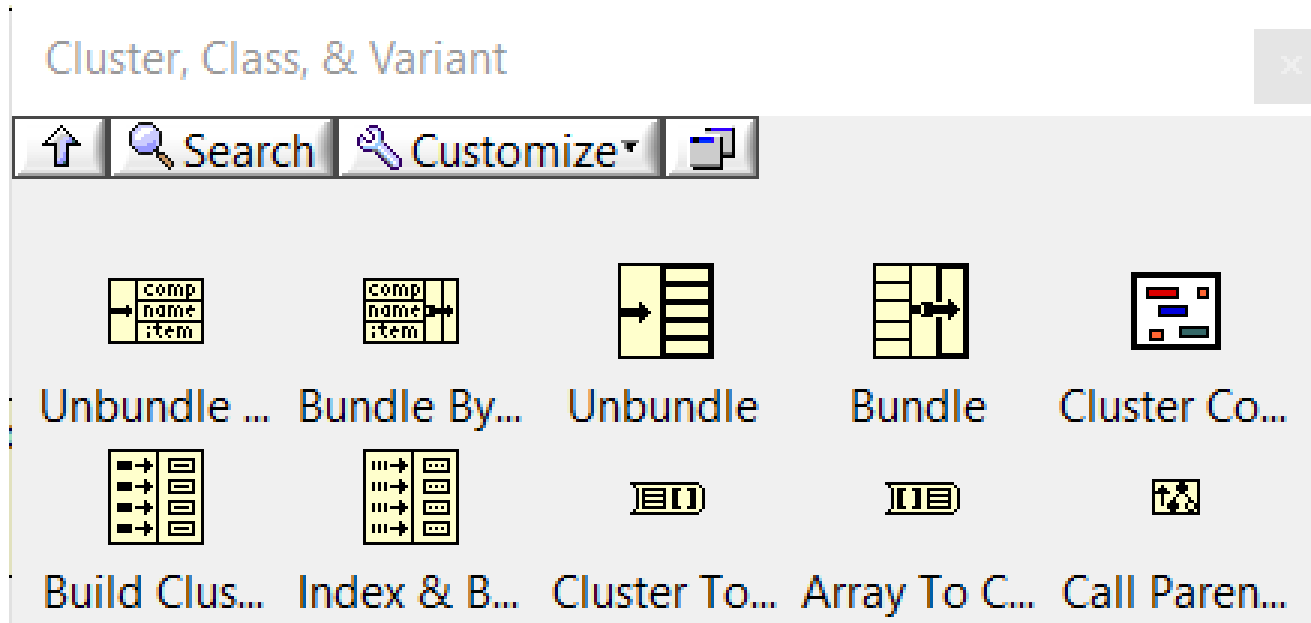
- Para acceder a los elementos de un cluster de manera individual se puede utilizar la función **Unbundle** (**Functions**→**Programming**→**Cluster & Variant**→**Bundle**):





### Cluster

- Funciones para gestión de Cluster:

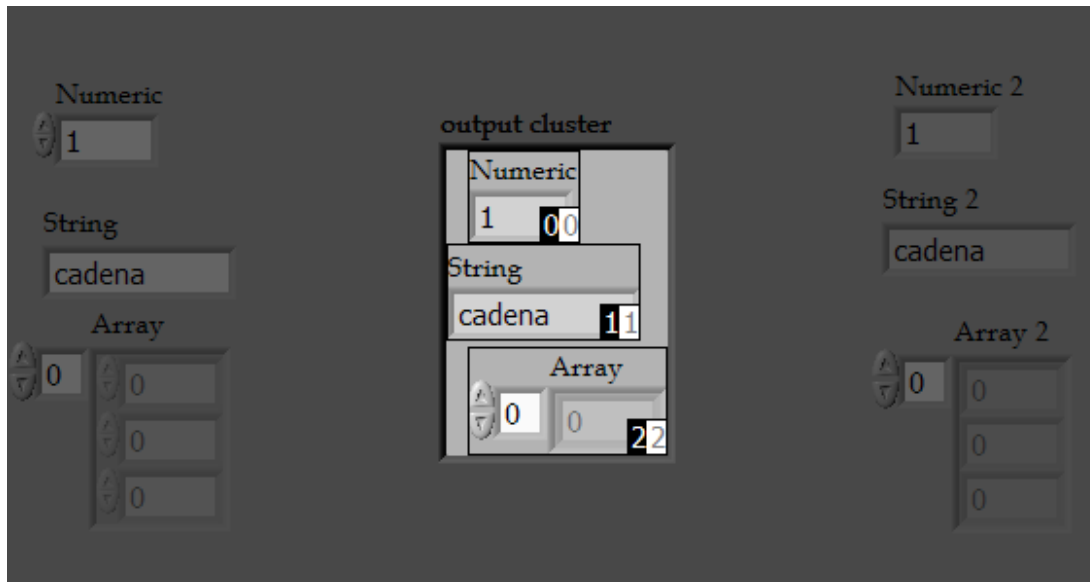


- Un Cluster se puede convertir en array mediante la función ***Kluster to Array*** siempre que los datos del cluster seán del mismo tipo.
- Un array se puede convertir en cluster mediante el vi ***Array to Cluster***,



### Cluster

- Los elementos del cluster tienen un orden lógico que en ningún caso está relacionado con su posición. El primer elemento del cluster es el 0.
- Cuando se elimina un elemento el orden se ajusta automáticamente.
- El orden de los elementos determina el orden en el que los elementos aparecerán en los terminales de las funciones **Bundle** y **Unbundle**.
- El orden de los elementos se puede modificar haciendo clic con el botón derecho del ratón sobre el borde del cluster y seleccionando la opción **Reorder Controls In Cluster**.

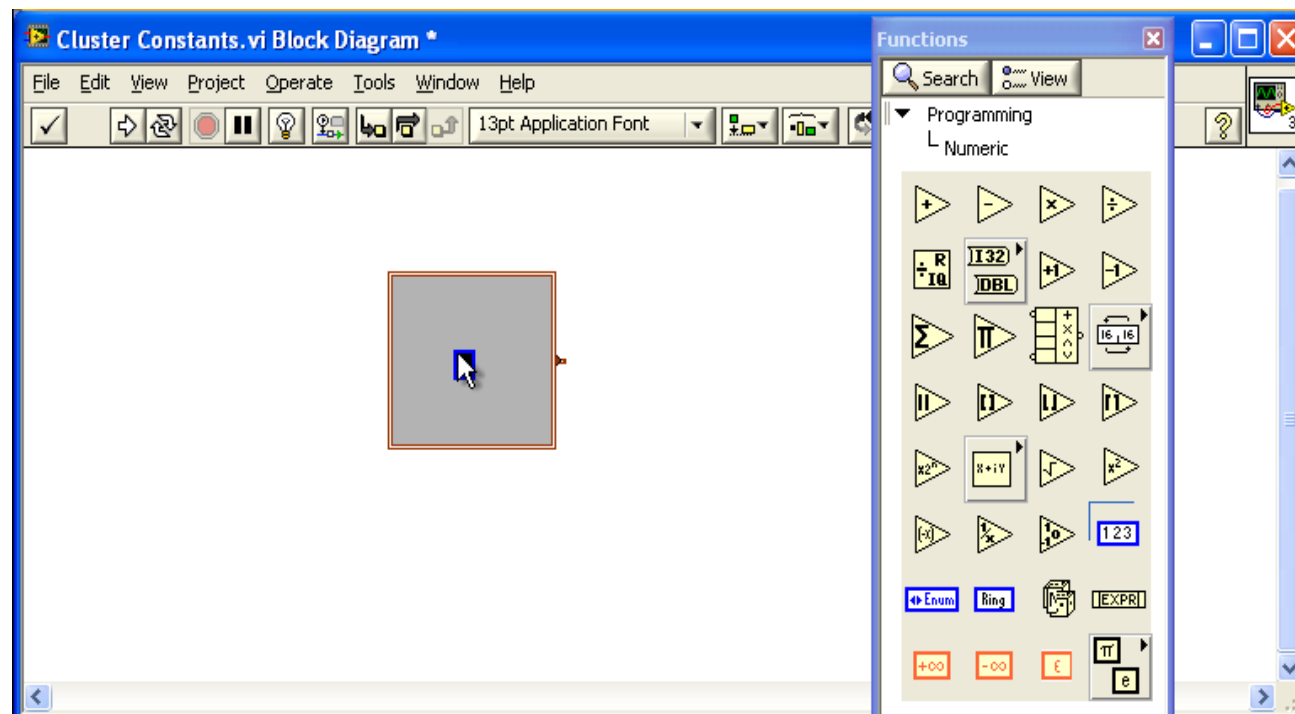


Para conectar dos cluster, además de coincidir el número y tipo de todos los elementos, debe coincidir también su número de orden.



### Cluster

- Como en el caso de los arrays se pueden crear cluster constantes. Para ello en la paleta de funciones del diagrama de bloques seleccionar: **Functions** → **Programming** → **Cluster & Variant** → **Cluster Constant**.
- Creado el cluster constante se deberá arrastrar a su interior los elementos constantes que se desea lo formen.

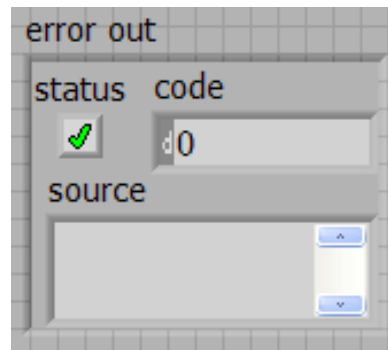






### Cluster de error

- LabView contiene un cluster *custom* denominado cluster de error que puede utilizarse en la fase de depuración para obtener información de los errores.

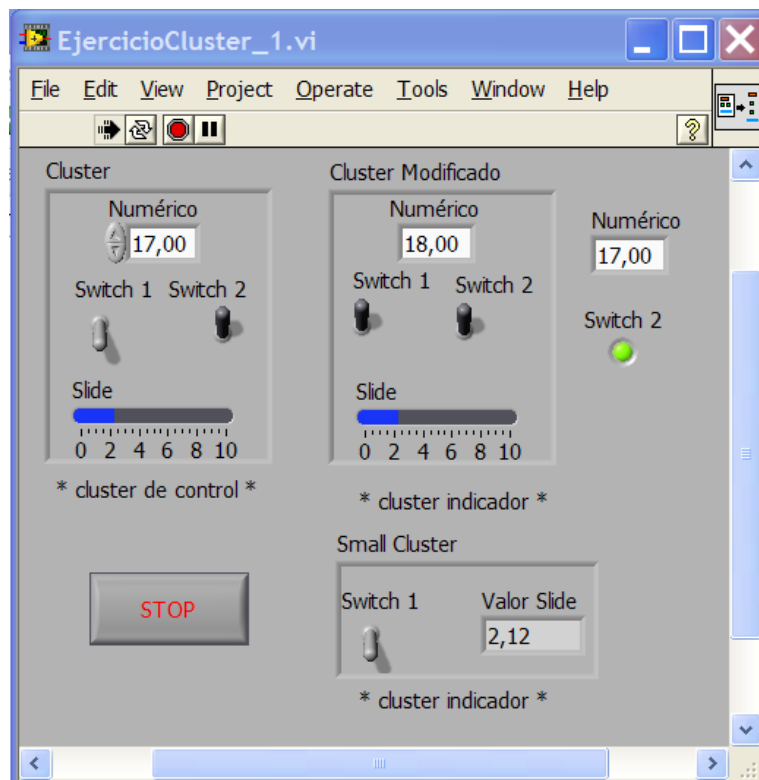


- Como se puede observar contiene tres elementos:
  - 1.- **Status**: indica si la ejecución ha sido o no correcta.
  - 2.- **Code**: indica el código del error.
  - 3.- **Source**: describe el origen del error.



### Ejercicio 11: Cluster.

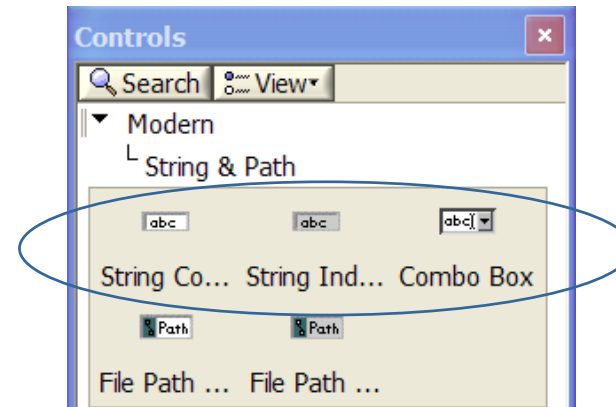
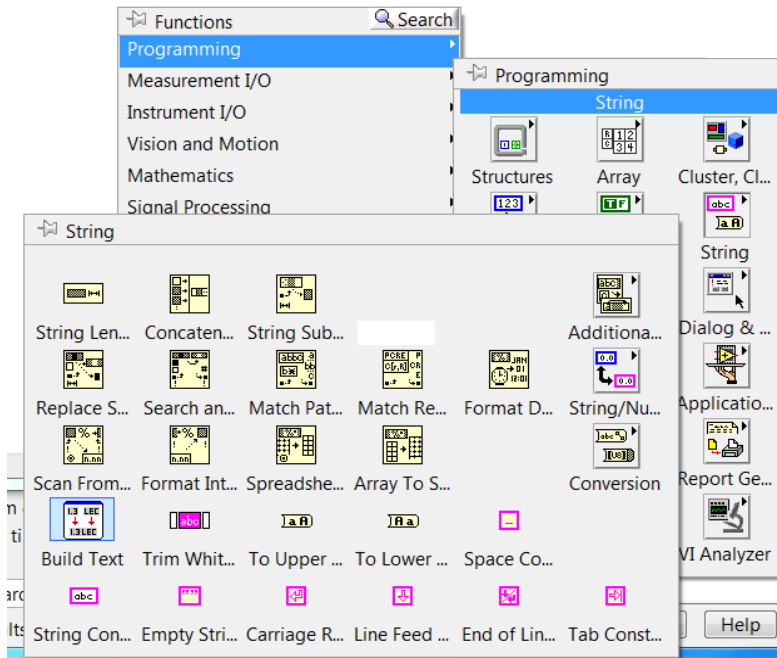
Crear un cluster que contenga un control numérico, dos switches, y una barra de desplazamiento (*Slide*). A partir de este cluster crear otro que contenga los mismos elementos con algunas modificaciones: el indicador numérico muestra una unidad más, el switch 1 está complementado. Crear también dos elementos individuales uno numérico que muestre el mismo valor que el control numérico del cluster y un led que indique la posición del switch 2. Además se deberá crear otro pequeño cluster que contenga otro switch con el mismo valor que el switch 1 y un indicador numérico con el valor señalado por la barra de desplazamiento.





### String

- Para insertar un control o indicador String seleccionar en la paleta de controles del **Diagrama de Bloques Programming** → **String** o en el **Panel Frontal Strings & Paths**:



- Las cadenas de texto pueden contener caracteres no imprimibles como: *fin de línea* ( $\backslash n$ ), *retorno de carro* ( $\backslash r$ ), *espacio* ( $\backslash s$ ), *tabulador* ( $\backslash t$ ), *fin de documento* ( $\backslash f$ ), etc.
- Se pueden crear cadenas constantes desde el panel de funciones del diagrama de bloques:

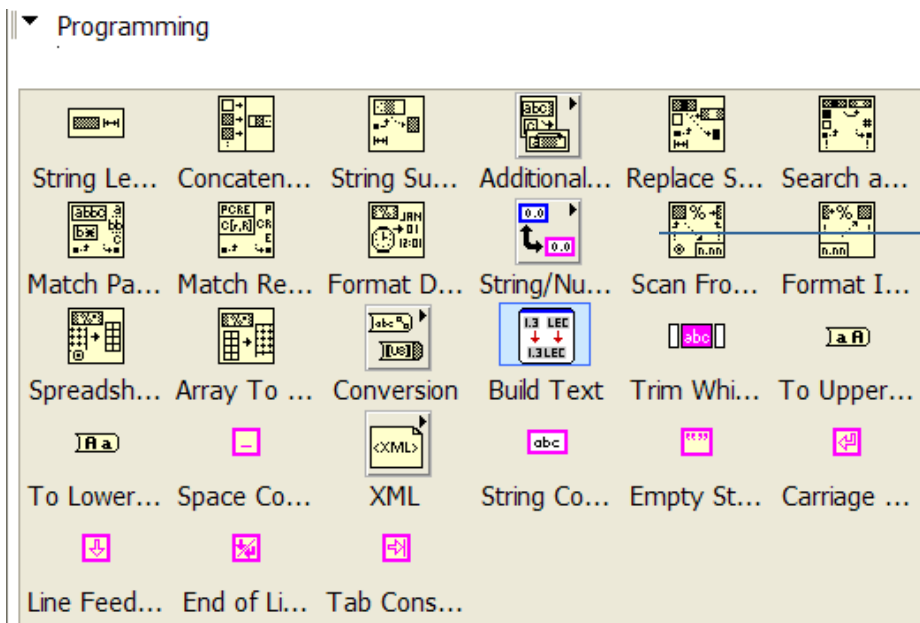
**Function → Programming → String → String Constant**



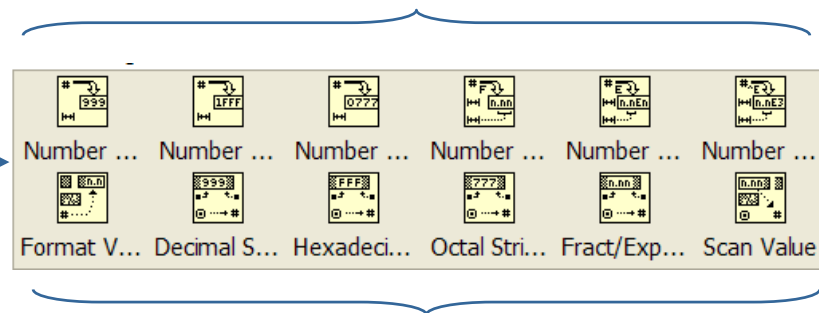
### String

- En la paleta de funciones se encuentran todas las funciones para manipular strings:

**Function → Programming → String**



Conversión de datos de tipo numérico a string en distintos formatos (decimal, octal, hexadecimal, exponencial, etc.)

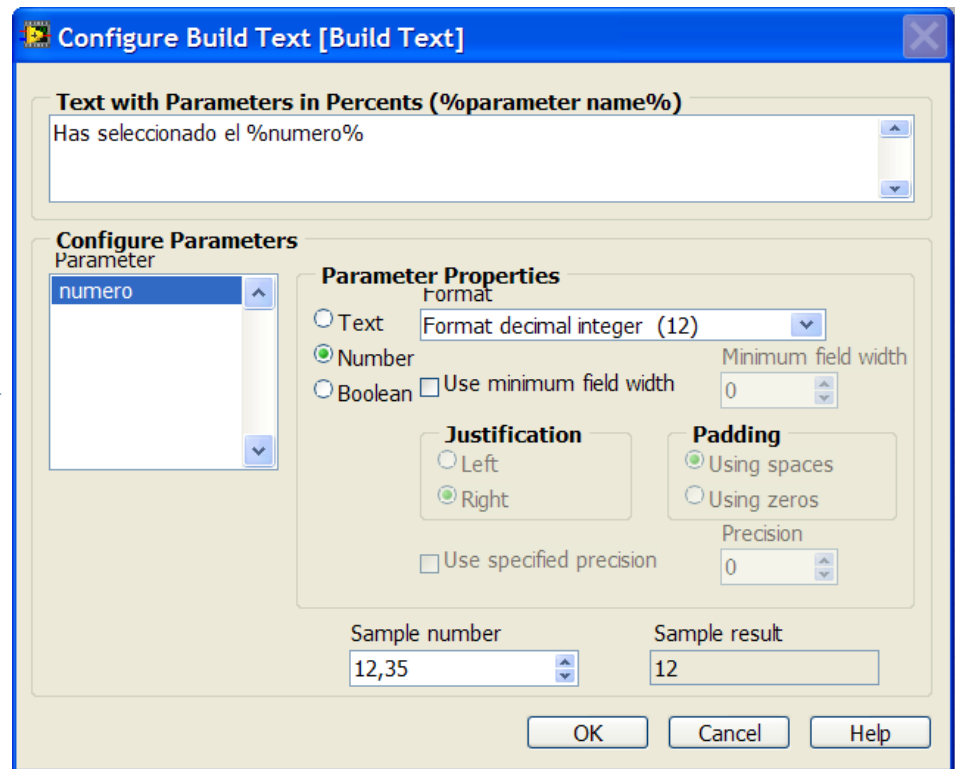
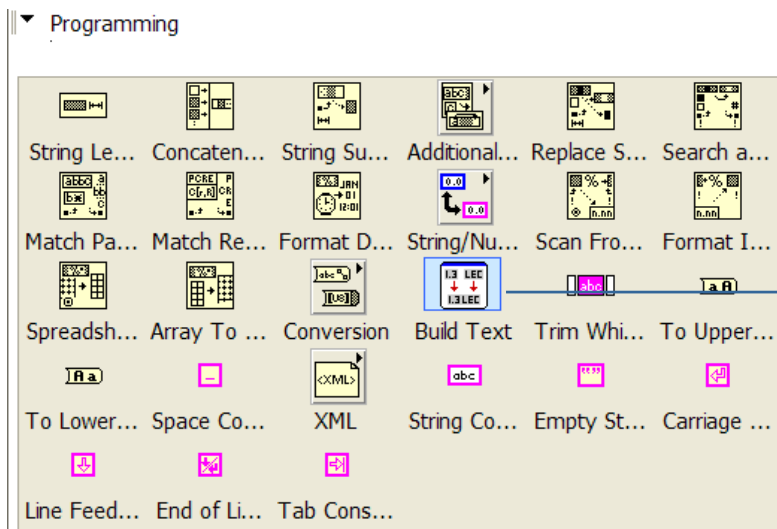


Conversión de string a datos de tipo numérico en distintos formatos (decimal, octal, hexadecimal, exponencial, etc.)



### String: Build text Express

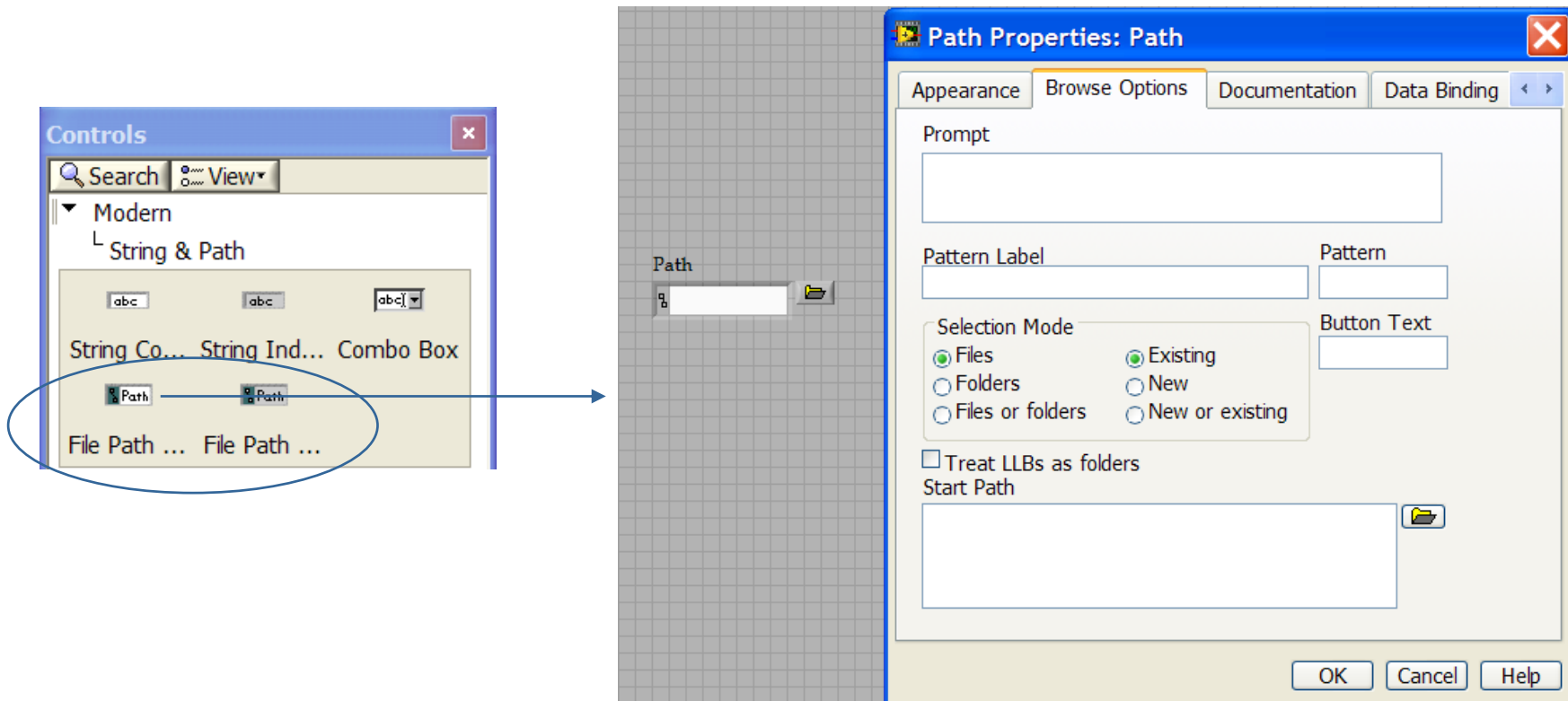
- Permite crear un *string* a partir de un texto y una serie de entradas parametrizadas.





### Path

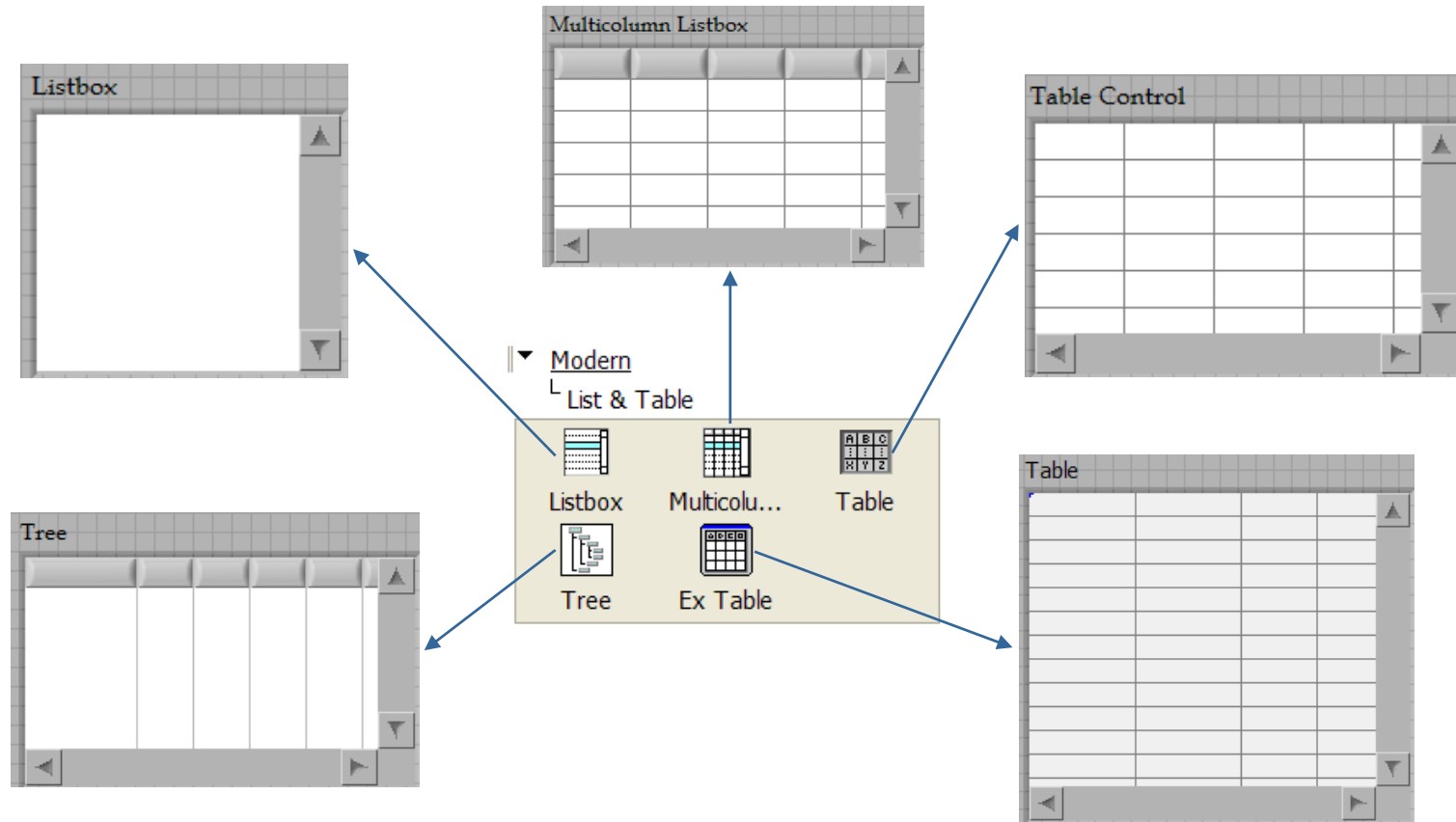
- Sirven para indicar u obtener rutas absolutas o relativas a directorios o ficheros locales o de máquinas pertenecientes a otra red.
- Para insertar un control o indicador Path seleccionar en la paleta de controles del panel frontal **Strings & Paths**:





### Listas y Tablas

- Para insertar una lista o tabla seleccionar en la paleta de controles del Panel Frontal **List & Table**:



- **NOTA:** Los datos de entrada a una tabla son de tipo *String*.



### Ejercicio 12: Tablas y Arrays.

- Realizar programa de forma que para representar en una tabla 10 números aleatorios, su cuadrado y su raíz cuadrada con cuatro decimales tal y como se muestra en la siguiente figura:

	x	x <sup>2</sup>	sqrt(x)
0	0,0000	0,0000	0,0000
0	0,7826	0,6125	0,8847
	0,2856	0,0816	0,5344
	2,2092	4,8805	1,4863
	2,2424	5,0283	1,4975
	2,6902	7,2371	1,6402
	0,1498	0,0224	0,3871
	3,8632	14,9240	1,9655
	0,6932	0,4805	0,8326
	0,8460	0,7157	0,9198

#### • Elementos necesarios:

- Build Array
- Transpose 2D Array
- Number To Fractional String





### Otras estructuras: *Formula Node*.

- Evalúa una expresión matemática escrita como texto en la que pueden utilizarse los siguientes operadores:

\*\* → Exponenciación.

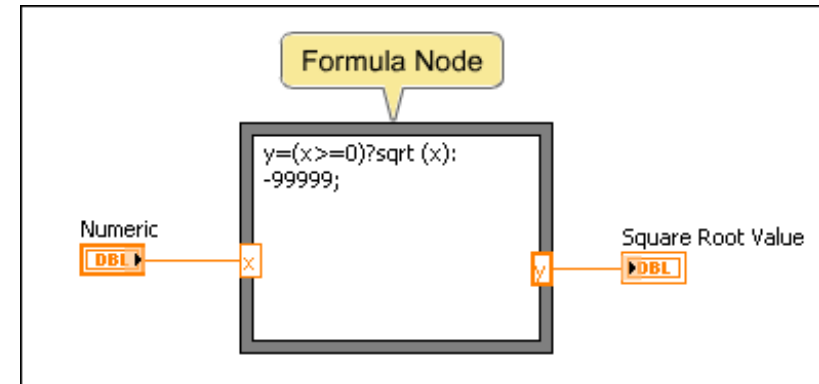
+, -, \*, /, % → Operaciones matemáticas básicas.

<<, >> → Desplazamientos lógicos.

&, |, &&, ||, ! → Operadores lógicos.

...?..... → Evaluación condicional.

=: → Asignación.



- Como se puede observar utiliza una sintaxis muy parecida a la de lenguaje C.
- Permite utilizar funciones matemáticas como: *abs*, *cos*, *sin*, *acosh*, *asin*, *atan*, *tan*, *floor*, *ceil*, *cot*, *pow(x,y)*, *sqrt*, *log*, *log2*, *max*, *mod*, etc.

Se pueden utilizar prácticamente todas las funciones de la paleta de funciones *Mathematics*.



### Otras estructuras: *Formula Node*.

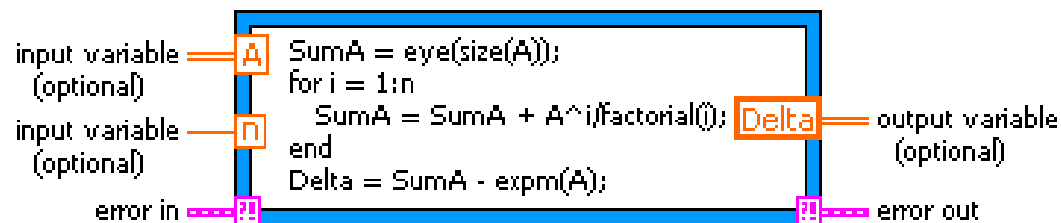
- Permite también el uso de las siguientes estructuras:
  - *if* (condición) {Sentencias\_if} *else* {Sentencias\_else}
  - *do* {Sentencias} *while* (condición)
  - *while* (condición) {sentencias}
  - *for* (asignación;condición;operación) {sentencias}
  - *switch* (condición) cases
- Las variables de entrada y salida se crean haciendo clic sobre el borde de la estructura y seleccionando *Add input* o *Add output* respectivamente del menú flotante que aparece en pantalla.





### Otras estructuras: *MathScript Node*.

- Este nodo permite la inserción de código muy semejante al de MATLAB para su ejecución.

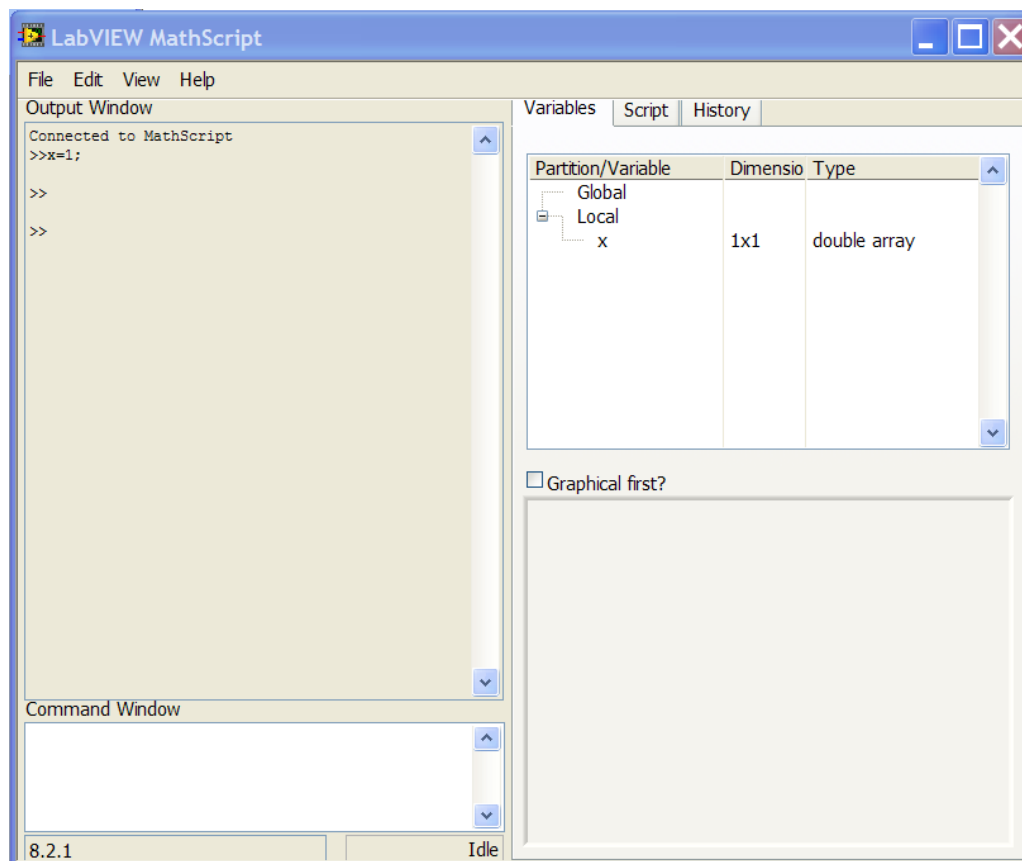


- A diferencia de versiones anteriores, el MathScript Node no realiza llamadas a otros programas y su código se compila junto con el resto de elementos del VI.
- Este elemento dispone de cientos de funciones que pueden consultarse en la ayuda.
- Como en el caso del *Formula Node*, se deben crear variables de entrada y salida. En este caso, a las variables de salida se les debe asignar un tipo de datos (Real, complejo, array 1D real, array 1D complejo, array 2D real, etc).



### Otras estructuras: *MathScript Node*.

- Desde el diagrama de bloques, seleccionando en el menú *Tools* la opción *MathScript Window* (*Tools* → *MathScript Window*) se puede acceder a una ventana que permite editar y depurar el código del Script:

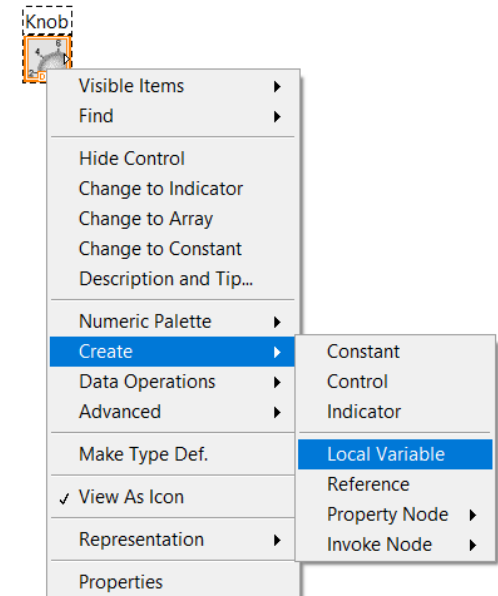




### Variables locales.

- Las variables locales son copias del terminal de un control o indicador del panel frontal que se pueden utilizar en cualquier lugar del diagrama de bloques para leer o escribir datos al control o indicador al que están asociadas.
- **Se utilizan cuando se quiere realizar el acceso, bien de lectura, bien de escritura, a un control o indicador desde subdiagramas en los que este no está visible.**
- Su alcance se limita al **vi** en el que están presentes.
- Solo es posible crear variables locales de controles e indicadores que tienen etiqueta (*label*).

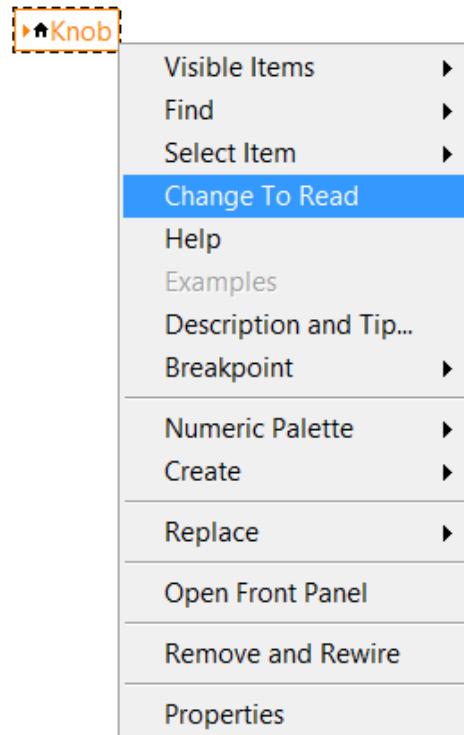
Para crear una variable local basta con situarse sobre el control o indicador al que va ir asociada, hacer clic con el botón derecho del ratón y seleccionar la opción **Create>>Local Variable** del menú flotante:





### Variables locales.

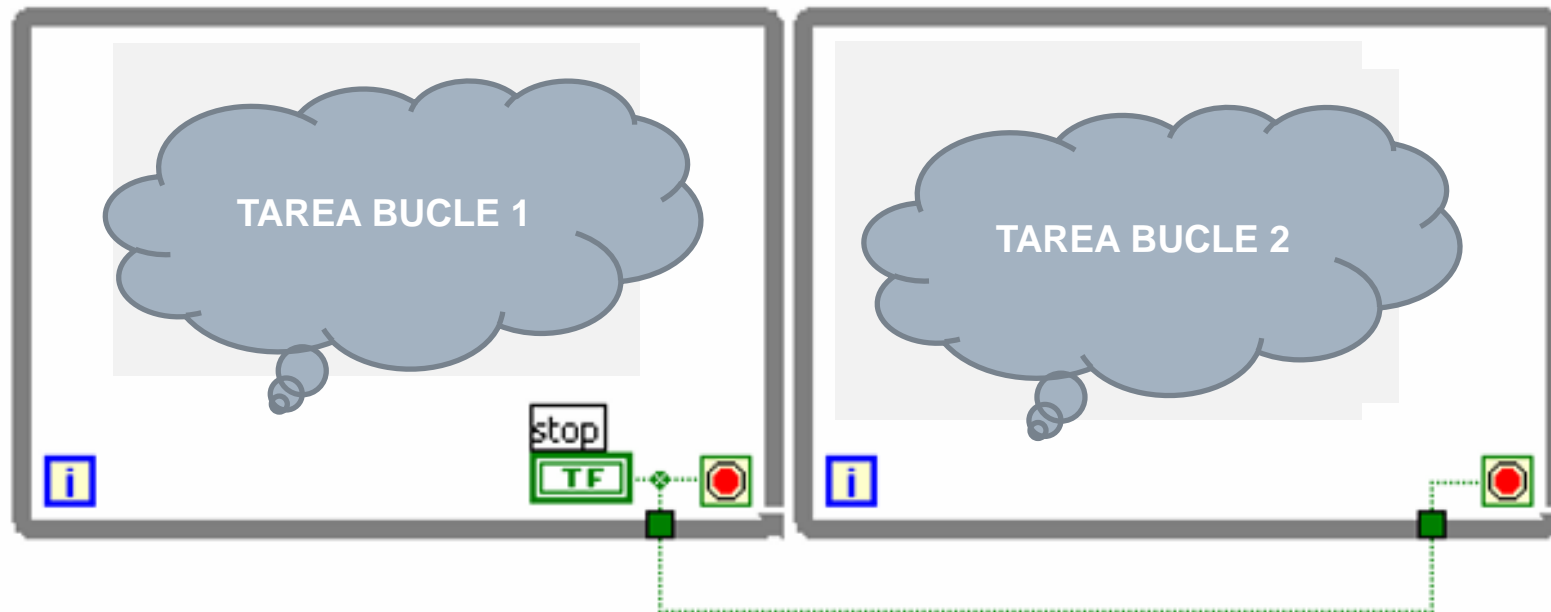
- El aspecto de la variable local es similar al de un terminal, toma el color representativo del tipo de la variable y muestra en su interior la etiqueta del control o indicador asociado.
- Por defecto las variables locales se crean en modo escritura (operan como si fueran un indicador). Se pueden cambiar a modo lectura seleccionando la opción **Change To Read** del menú flotante:





### Variables locales.

- **Ejemplo utilización variables locales:** Ejecución y finalización de dos bucles simultáneos.
  - Se requiere que las dos tareas se realicen simultáneamente.

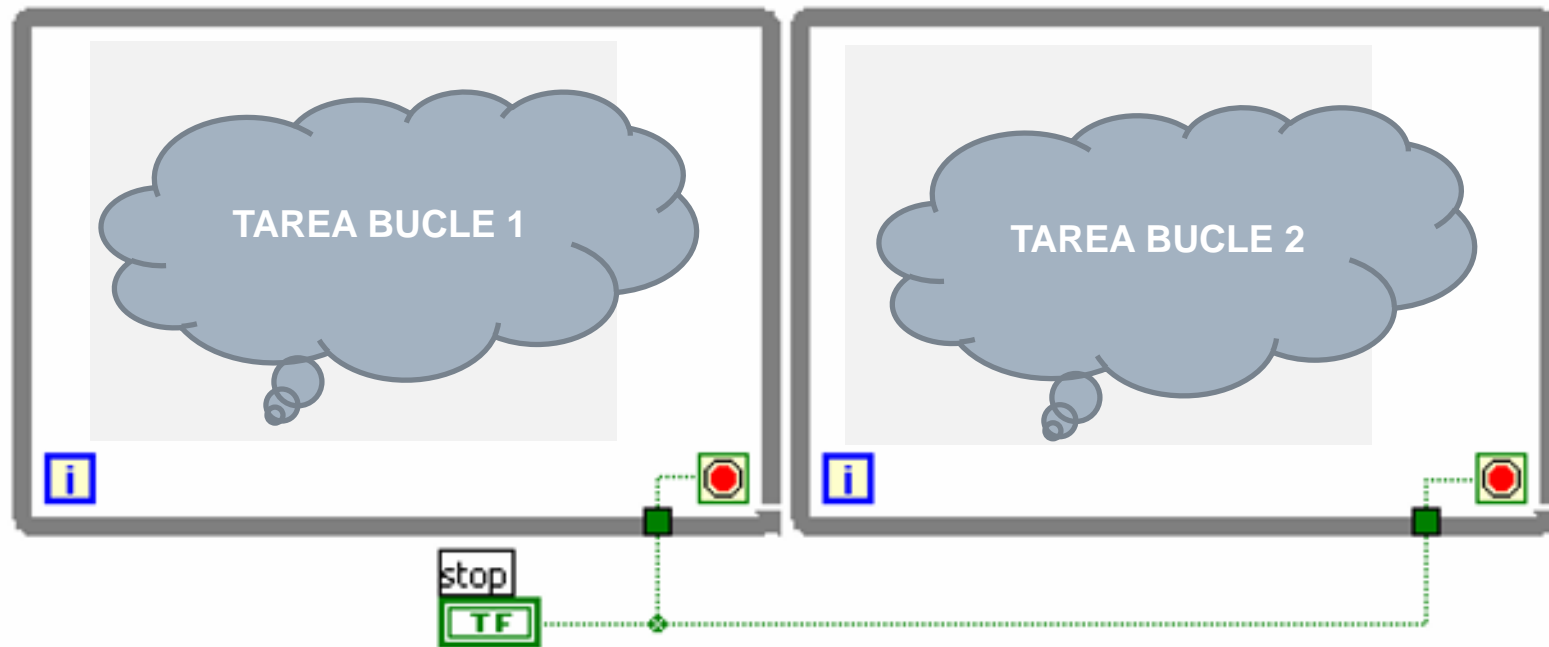


**OPCIÓN INCORRECTA:** La tarea del bucle 2 comienza su ejecución cuando el usuario pulse el botón stop y solo se ejecutará una vez.



### Variables locales.

- **Ejemplo utilización variables locales:** Ejecución y finalización de dos bucles simultáneos.



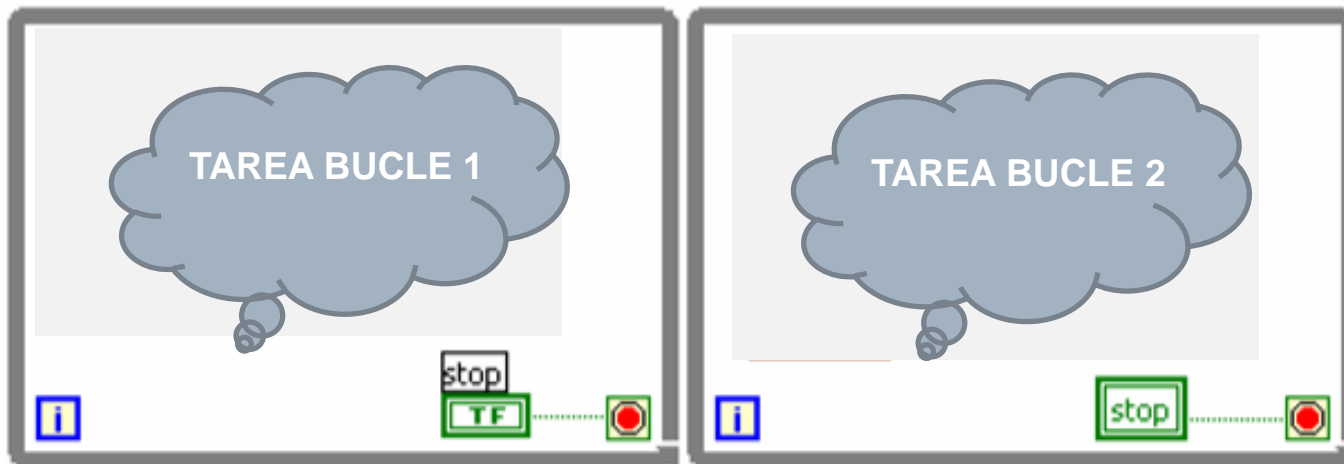
**OPCIÓN INCORRECTA:** Los dos bucles se ejecutan en paralelo pero el botón **stop** se lee al comenzar la ejecución del programa por lo que esta entra en un bucle infinito.





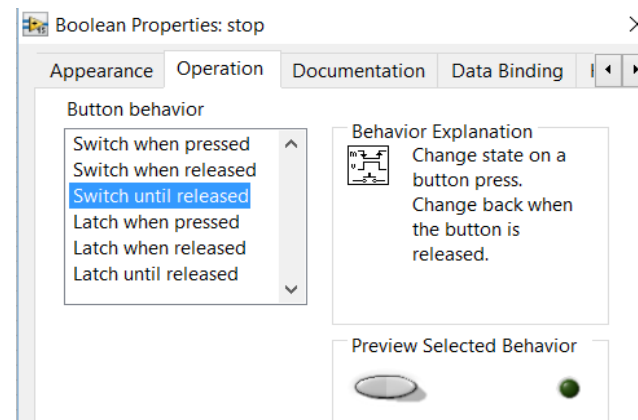
### Variables locales.

- **Ejemplo utilización variables locales:** Ejecución y finalización de dos bucles simultáneos.



**OPCIÓN CORRECTA:** Creando una variable local del botón **stop**.

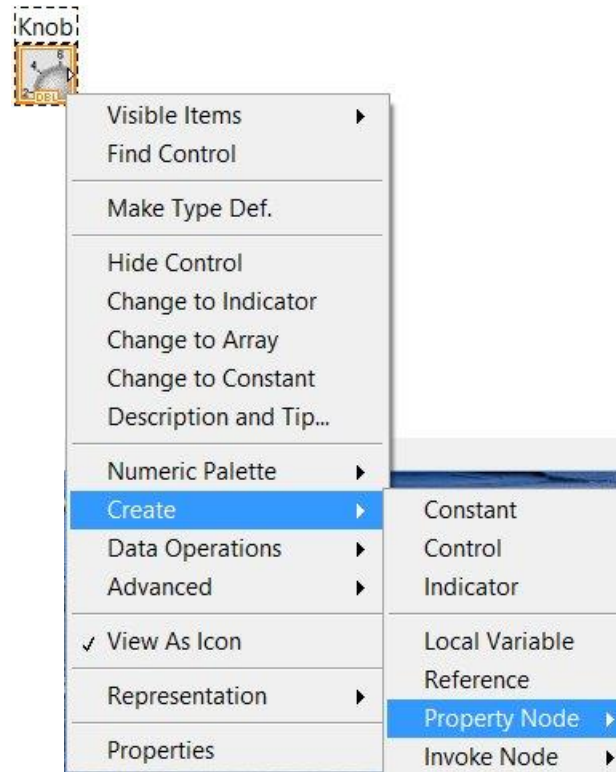
**NOTA:** Los controles de tipo BOOLEANO con acción mecánica tipo *Latch* no permiten crear variables locales. Para ello hay que cambiarles el comportamiento (acción mecánica) a cualquiera de las tres opciones de tipo *Switch* (p.e *Switch until released*).





### Nodos de propiedades (*Properties Nodes*).

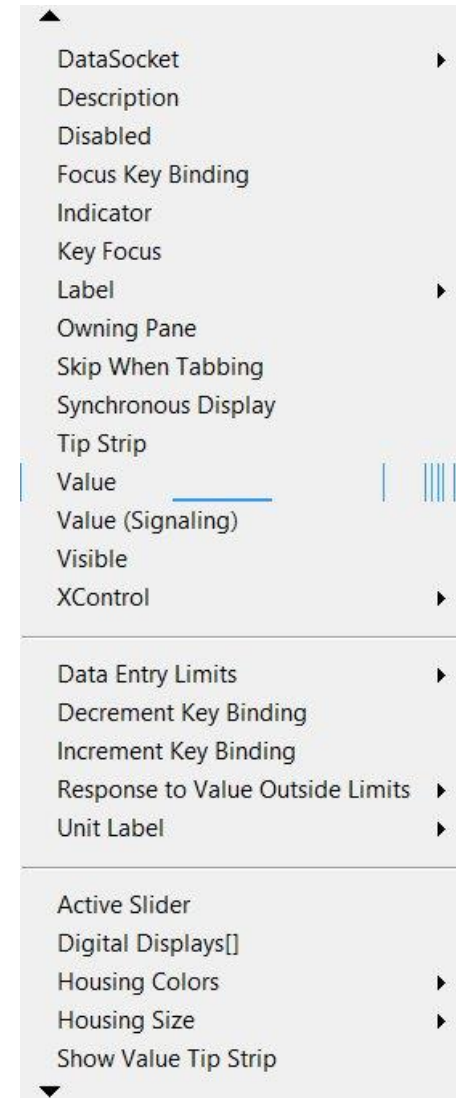
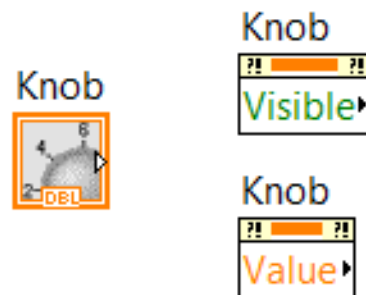
- Permiten modificar las propiedades de los controles e indicadores en **tiempo de programación**. En definitiva, los Property Nodes representan una propiedad o estado de los objetos que puede ser escrito o leído.
- Los nodos de propiedades se crean seleccionando la opción **Create→Property Node** del menú flotante del control o indicador.





### Nodos de propiedades (*Properties Nodes*).

- Algunas de las propiedades que se pueden modificar: valores por defecto al ejecutarse la aplicación, rangos, visibilidad, colores, etc.
- Cuando se crea el nodo de propiedades presenta el aspecto de un terminal.

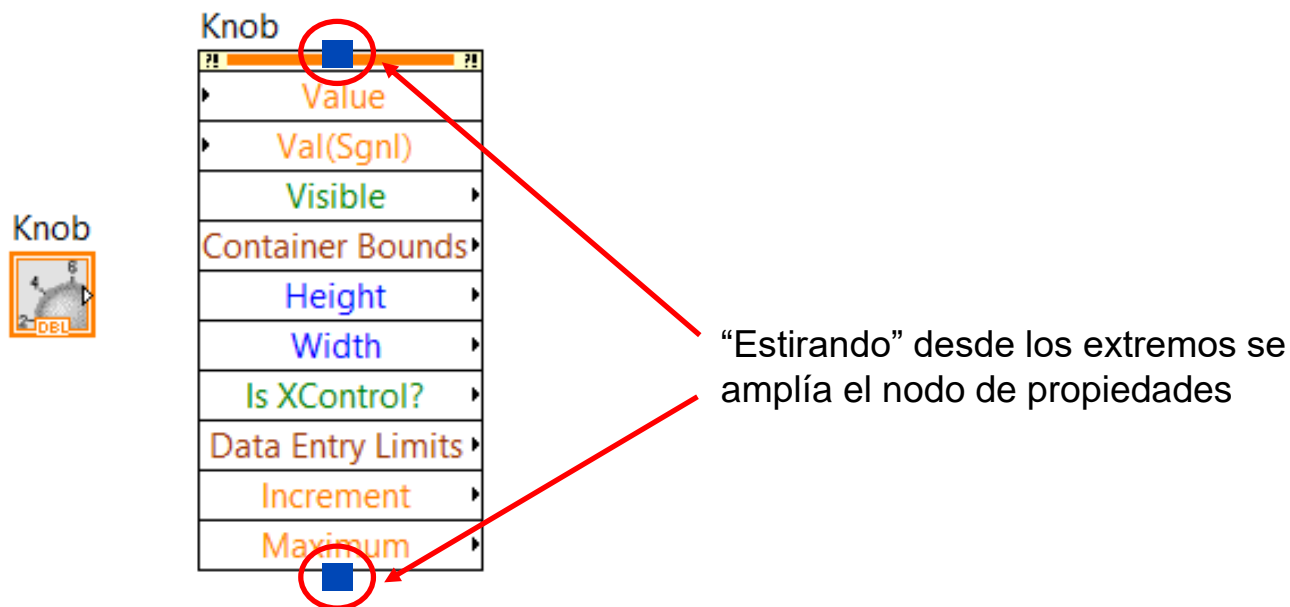


- Dependiendo del tipo de propiedad el nodo creado puede ser de lectura o escritura. En cualquier caso se puede cambiar de un modo a otro a través del menú flotante que aparece al hacer clic con el botón derecho del ratón sobre el nodo.



### Nodos de propiedades (*Properties Nodes*).

- Creado un nodo de propiedades se puede utilizar para leer/escribir varias propiedades simultáneamente, basta con ampliar el tamaño del nodo creado.

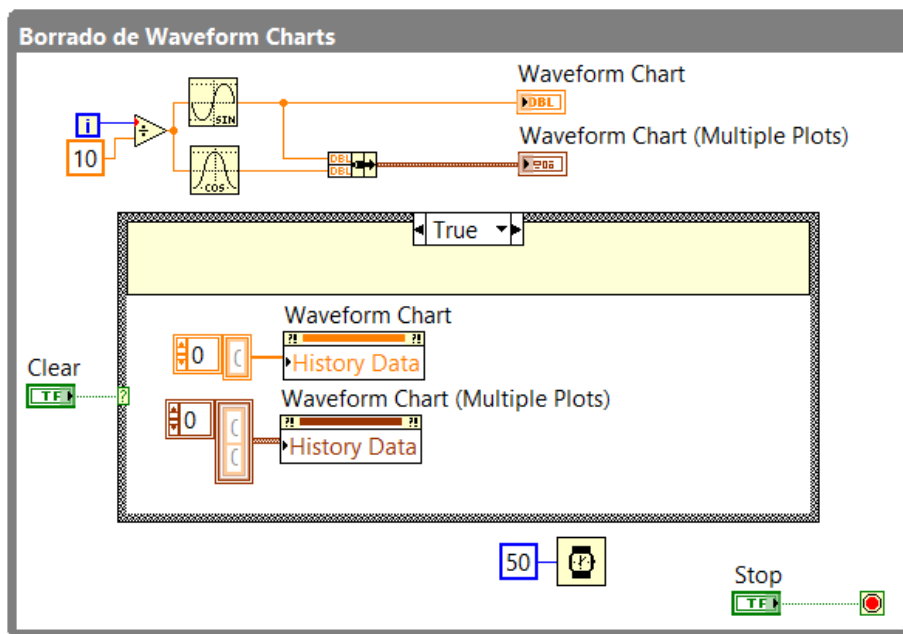




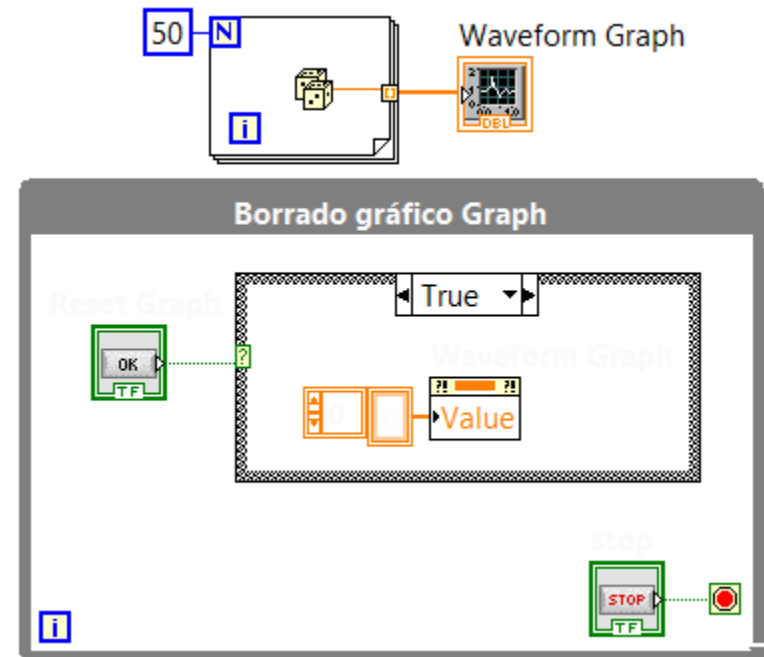
### Nodos de propiedades (*Properties Nodes*).

**Ejemplo:** borrar gráficos *Waveform* y *Graph* en tiempo de ejecución.

- Los gráficos *Waveform* se borran escribiendo un array vacío en la *Property Node History Data*.
- Los gráficos *Graph* se borran escribiendo un array vacío en la *Property Node Value*.



**Borrado gráfico *Waveform Chart***



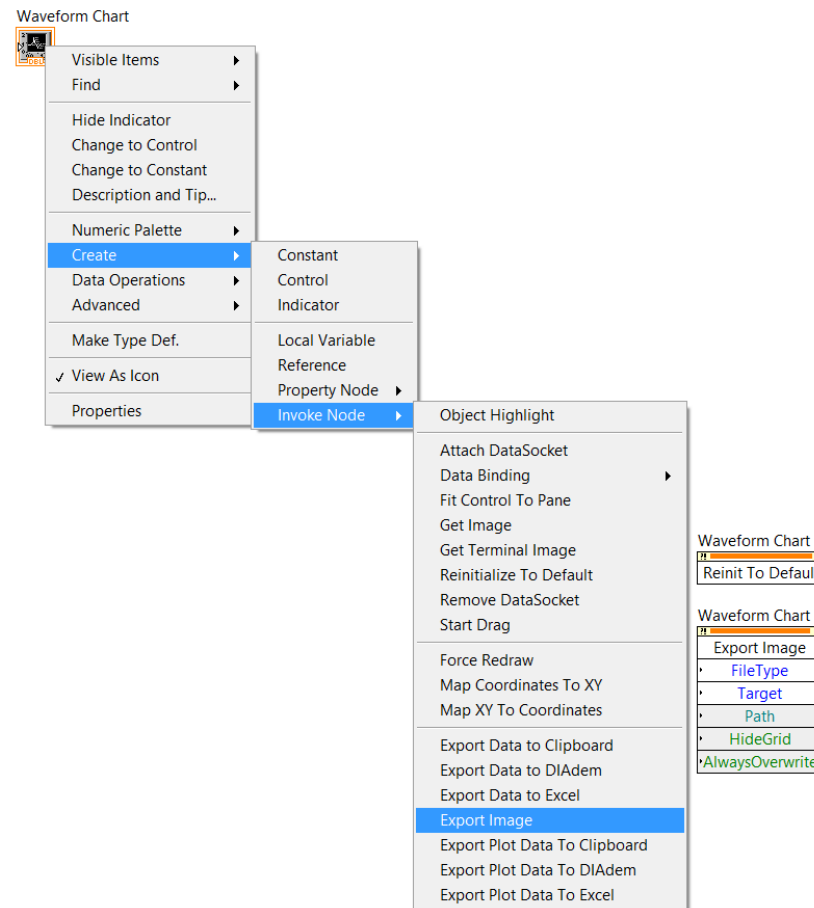
**Borrado gráfico *Graph***



### Invoke Node (Métodos).

- Los **Invoke Nodes** permiten ejecutar **métodos** (acciones) sobre los objetos de un panel frontal.
- Dependiendo del *Invoke Node* puede requerir parámetros o argumentos de entrada y/o salida.
- Se crean de la misma manera que las variables locales y *property nodes*:

Botón derecho del ratón sobre el objeto al que se quiere asociar el método.



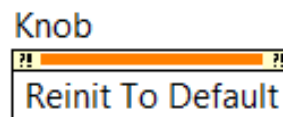
Método que requiere argumentos de entrada/salida



### Invoke Node (Métodos).

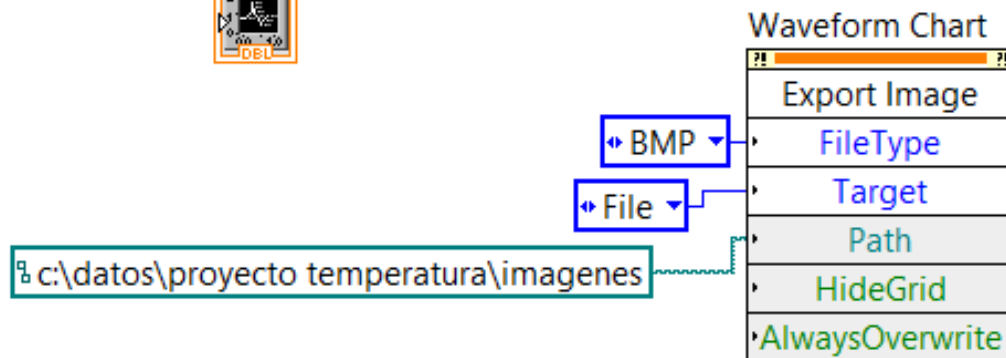
#### Ejemplos:

1. Reiniciar controles y/o indicadores a su valor por defecto → Invoke Node **Reinit to Default**.



2. Exportar un gráfico a una imagen → Invoke Node **Export Chart**.

Waveform Chart

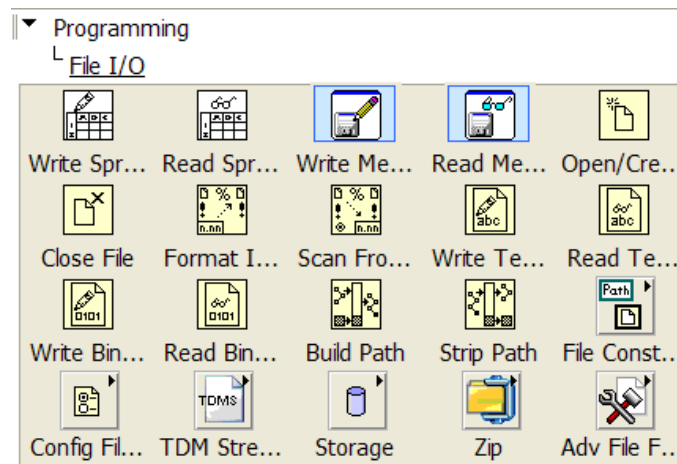




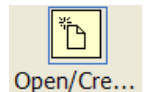
### Ficheros

- Las funciones para manejo de ficheros se encuentran en la paleta de funciones del diagrama de bloques:

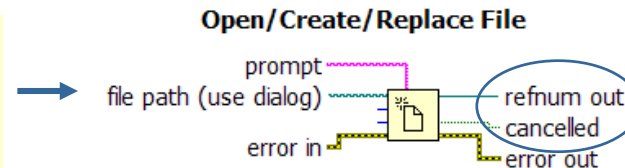
### Functions → Programming → File I/O



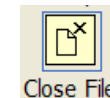
- Para acceder a un fichero, tanto en modo lectura como escritura, lo primero que se debe hacer es abrir el fichero



Al abrir el fichero se obtiene una referencia necesaria para los posteriores accesos (lectura/escritura) al fichero. <sup>0</sup>



- Una vez finalizada la lectura o la escritura en el fichero debe cerrarse

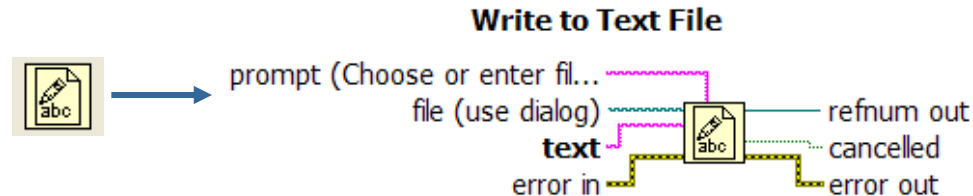






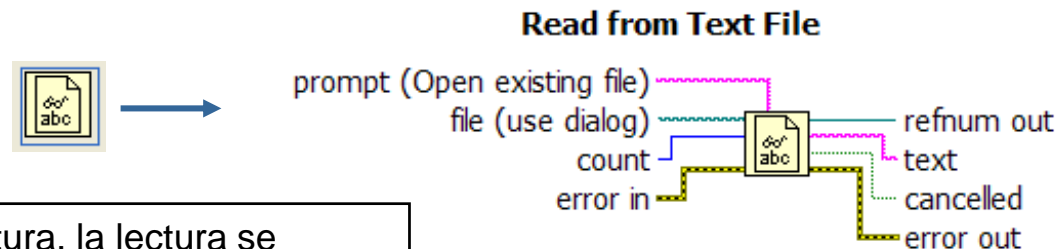
### Ficheros de texto: Lectura y escritura.

- VI para escribir texto en un fichero:



- Si el fichero indicado a esta función proviene de un Path → Se sustituye el contenido de fichero indicado en el Path por el actual.
- Si el fichero proviene de una referencia obtenida al abrir el fichero → La escritura se realiza a partir de una posición determinada.

- VI para leer un un fichero de texto:



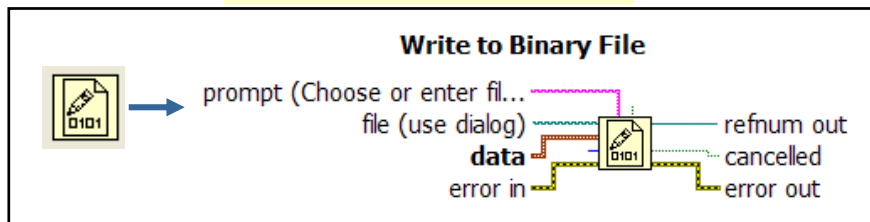
Como en el caso de la escritura, la lectura se llevará a cabo desde el comienzo del fichero o desde la posición que hubiera antes según se le pase la referencia al fichero



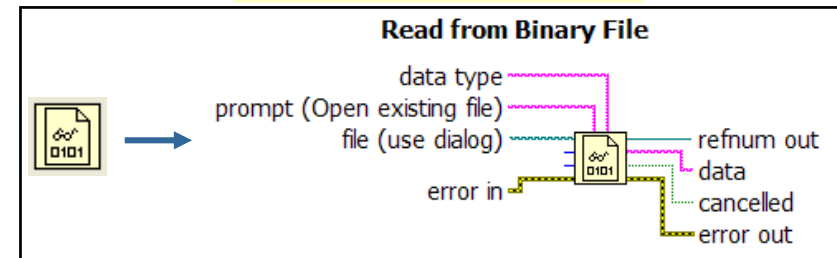
### Ficheros: Binarios y Spreadsheet.

- Existen funciones para escribir/leer **ficheros binarios** muy semejantes a las utilizadas para escritura/lectura de ficheros de texto.

#### Escritura fichero binario

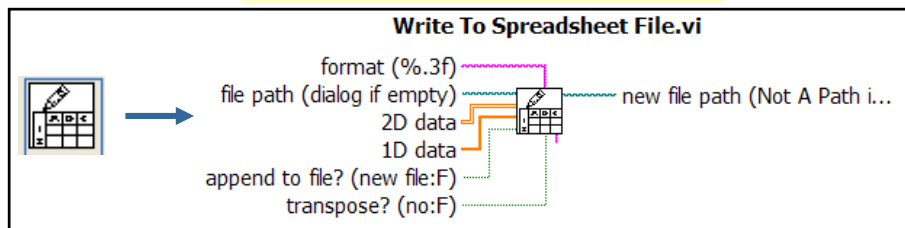


#### Lectura fichero binario

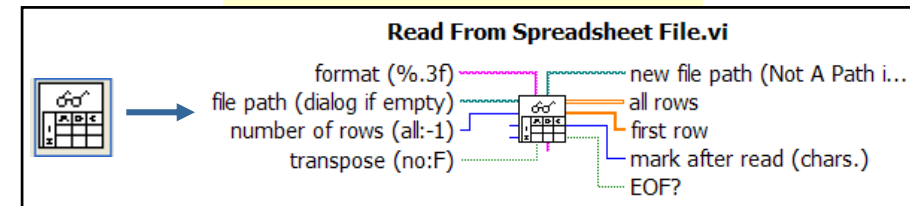


- Ficheros spreadsheet:** Estos ficheros tienen extensión **.csv** y se utilizan para almacenar tablas.
  - Cada línea del fichero es una fila de la tabla
  - Las columnas pueden separarse mediante tabuladores o el carácter “;”.
  - Se caracteriza por que pueden ser leídos por otras aplicaciones como por ejemplo Excel.

#### Escritura fichero spreadsheet



#### Lectura fichero spreadsheet

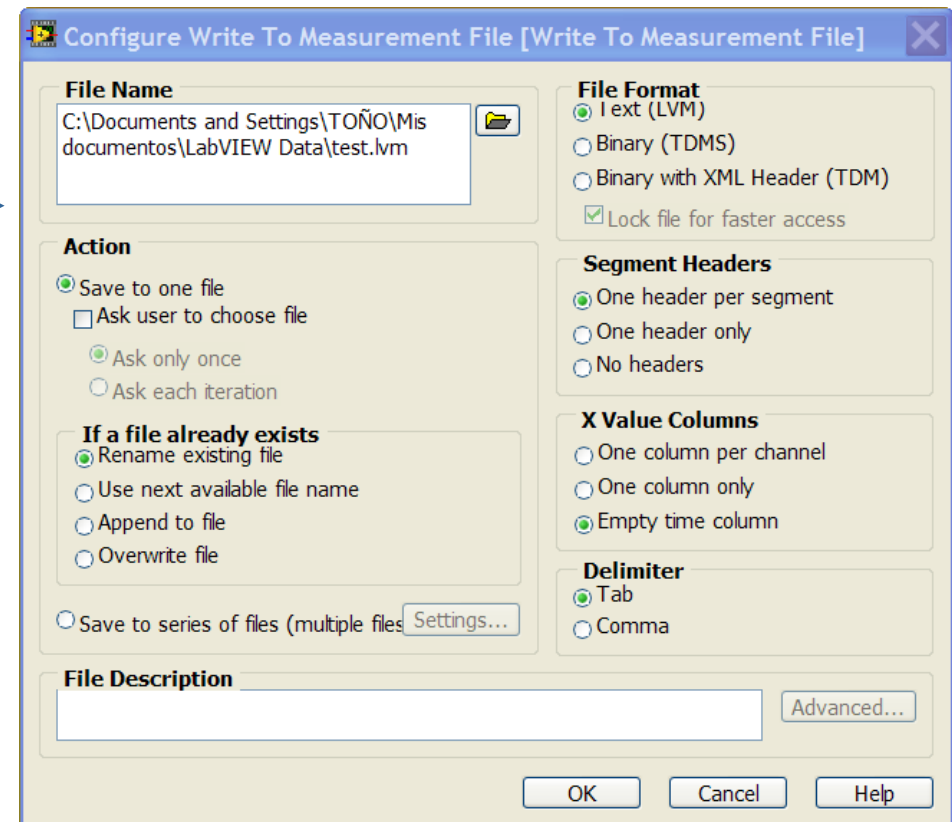
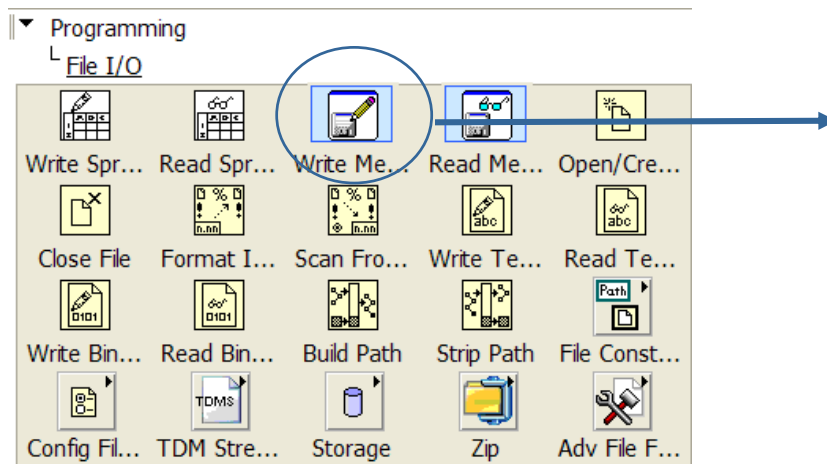




### Ficheros: Funciones Express.

- Hay dos funciones Express relacionadas con ficheros que están especialmente diseñadas para almacenar (*Write Measurement*) y recuperar información (*Read Measurement*) de señales, por ejemplo información de adquisiciones de datos.

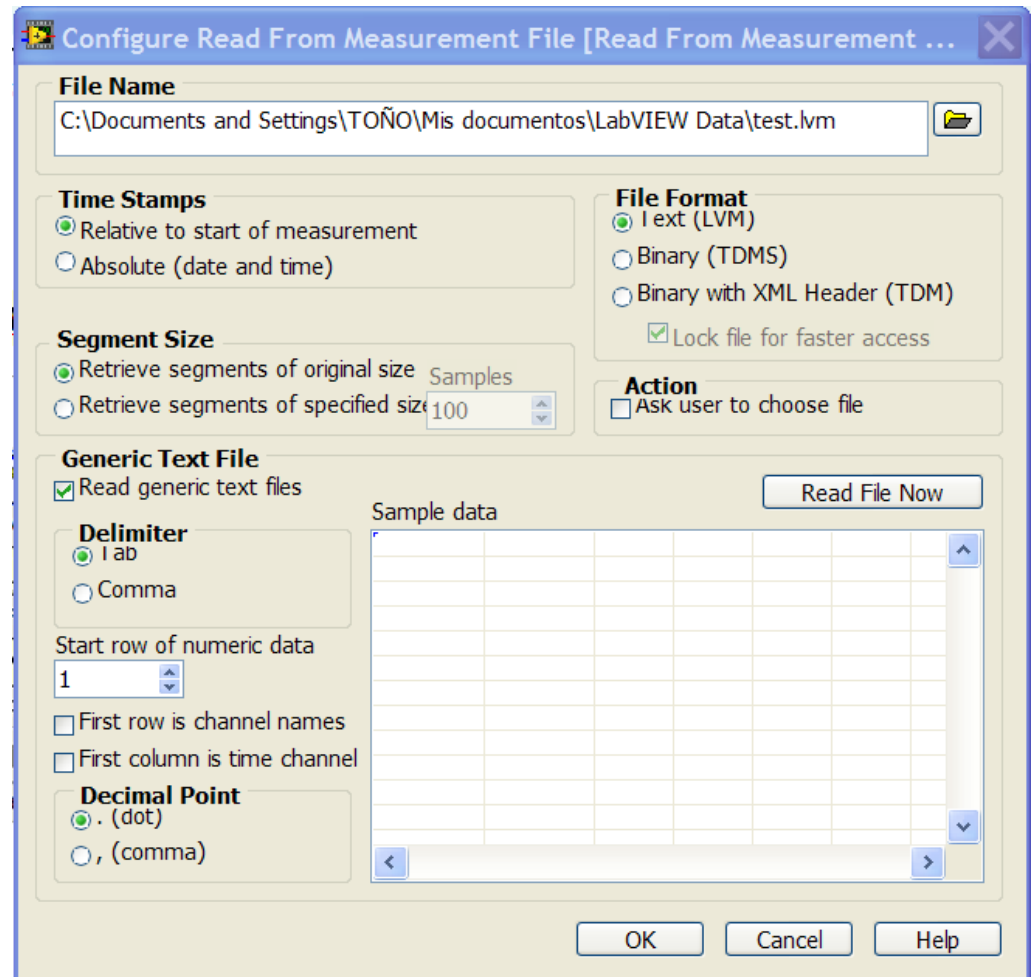
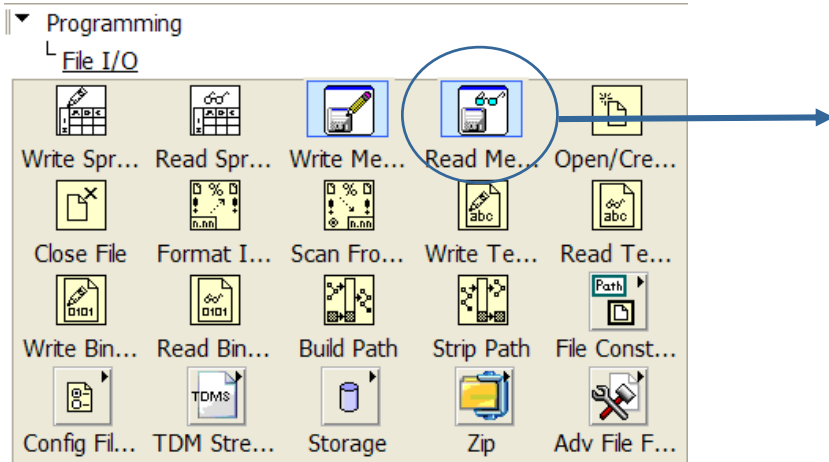
#### Escritura: *Write Measurement*





### Ficheros: Funciones Express.

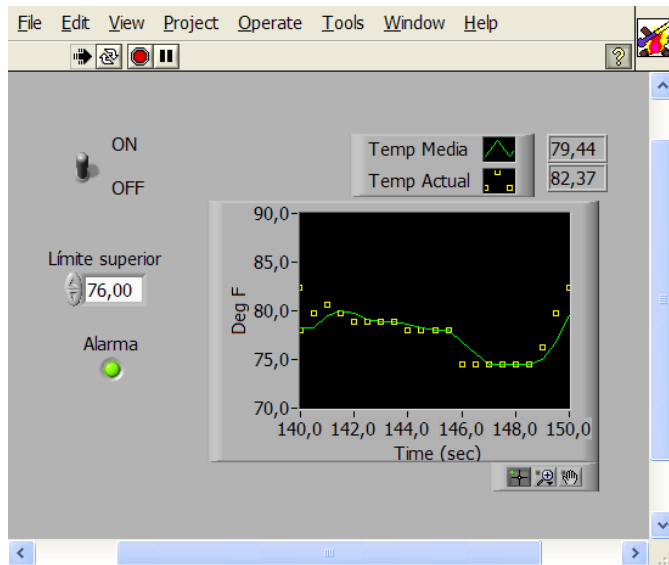
Lectura: **Read Measurement**









### Ejercicio 13: Ficheros.

Modificar el código del ejercicio 7 para de forma que los datos de temperatura adquiridos se almacenen en un fichero de texto junto con la hora a la que se ha adquirido cada uno de ellos. Al arrancar la aplicación debe solicitar el nombre y localización del fichero en el que se deseen almacenar los datos.



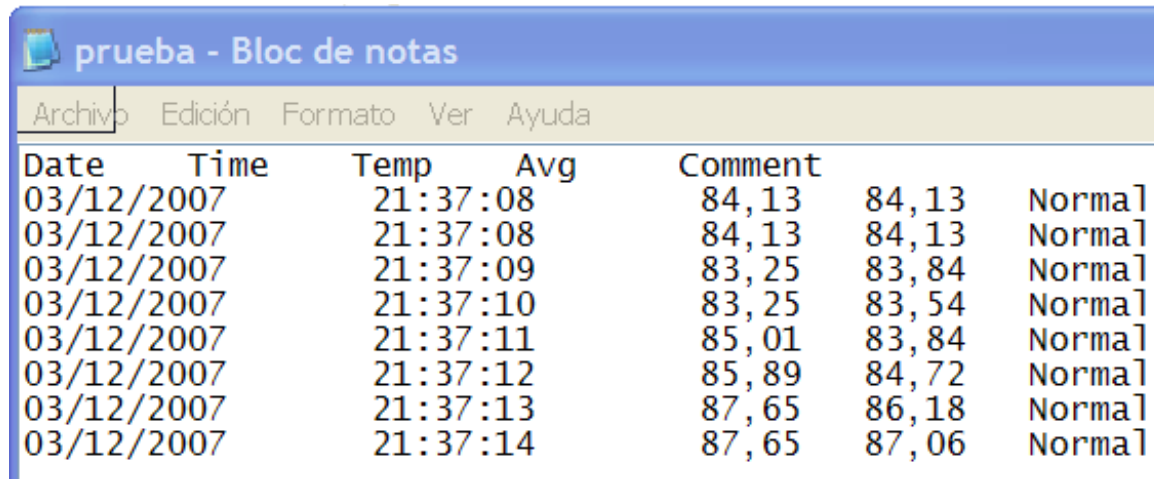
datos - Bloc de notas				
Archivo	Edición	Formato	Ver	Ayuda
12:44:45		74,461719		
12:44:45		74,461719		
12:44:46		74,461719		
12:44:46		74,461719		
12:44:47		74,461719		
12:44:47		76,219531		
12:44:48		79,735156		
12:44:48		82,371875		
12:44:49		83,250781		
12:44:49		82,371875		

- Para obtener la hora utilizar la función  de **Functions** → **Time** → **Timing** → **Get Date/Time String**
- Para formatear los datos utilizar la función  de **Functions** → **File I/O** → **Format Into String**
- Para tabular las columnas de hora y temperatura e introducir los finales de línea utilizar las constantes tabulador  y fin de línea  respectivamente. (**Functions** → **String**)



### Ejercicio 14: Ficheros.

13.1.- Modificar el código del ejercicio 12 para que el fichero de texto presente el aspecto de la siguiente figura.



Date	Time	Temp	Avg	Comment	
03/12/2007		21:37:08	84,13	84,13	Normal
03/12/2007		21:37:08	84,13	84,13	Normal
03/12/2007		21:37:09	83,25	83,84	Normal
03/12/2007		21:37:10	83,25	83,54	Normal
03/12/2007		21:37:11	85,01	83,84	Normal
03/12/2007		21:37:12	85,89	84,72	Normal
03/12/2007		21:37:13	87,65	86,18	Normal
03/12/2007		21:37:14	87,65	87,06	Normal

13.2.- Añadir una tabla al interface de usuario para que puedan visualizarse las 5 columnas del fichero de texto.

Para escribir los datos numéricos a la tabla es necesario convertirlos a texto con la función:

***Functions → String → String/Number Conversion → Number to Fractional String***



### REFERENCIAS.

1. [www.ni.com](http://www.ni.com)
2. <http://www.ni.com/academic/students/learn/esa/>
3. NI Technical Symposium, National Instruments, 2006.
4. Introduction to LabVIEW™ Graphical Programming Hands-On Seminar. Customer Manual, Edición Agosto 2010.
5. Curso básico Labview 6i, Germán A. Holguín, Sandra Milena, Alvaro A. Orozco, Universidad Tecnológica de Pereira.