



Feasa Enterprises Ltd.
Castletroy,
Co.Limerick,
Ireland. www.feasa.ie
Email: sales@feasa.ie Rev. 2.0
Date: January, 2008

Table of Contents

1. Introduction.....	3
1.1 How to use this document.....	3
2. LabVIEW programming guide.....	4
2.1 Environment.....	4
2.2 Placing items. Wire tool.....	7
2.3 Import vi.....	14
2.4 Types.....	15
2.5 Arrays and clusters.....	16
2.6 Flow Control.....	21
2.7 Property nodes.....	35
2.8 Formula nodes.....	37
3. Programming with the LED Analyser.....	41
3.1 Program structure.....	43
3.2 Open/close ports.....	44
3.3 Capturing LED Color and Intensity data.....	45
3.4 Reading color & Intensity.....	46
3.5 Tools.....	50
3.6 My first vi.....	55

1. Introduction.

This document describes the LabVIEW™ library for the FEASA LED Analyser. The library was developed to support users of LabVIEW™ and allow easy integration of the LED Analyser into systems controlled with LabVIEW™. This library consists of different LabVIEW™ modules that allow the programmer to use the main functions of the LED Analyser very easy and quickly, using the well known concept “function-in-a-box”.

1.1 How to use this document.

The information contained in this document has been divided in two parts:

- The first part of the document is an introduction to LabVIEW for beginners. The readers of this section should have a basic knowledge of programming because some technical concepts are used in the text. Experienced LabVIEW users can skip this section.
- The second part describes how to use the LED Analyser Library for LabVIEW. In this section there is information about all functions, parameters and basic procedures required to develop *Virtual Instruments* (LabVIEW programs) for the LED Analyser.

All sections in this manual are complemented with practical examples and schematics so the user can put in practice almost all the concepts explained.

This guide has been written for LabVIEW 6i, so there may be subtle differences between this version and other versions. However, the user should be able to follow the instructions and concepts given in this document to help program virtual instruments in LabVIEW.

2. LabVIEW programming guide.

There are some important concepts of LabVIEW the user should know before starting the design of a new schematic.

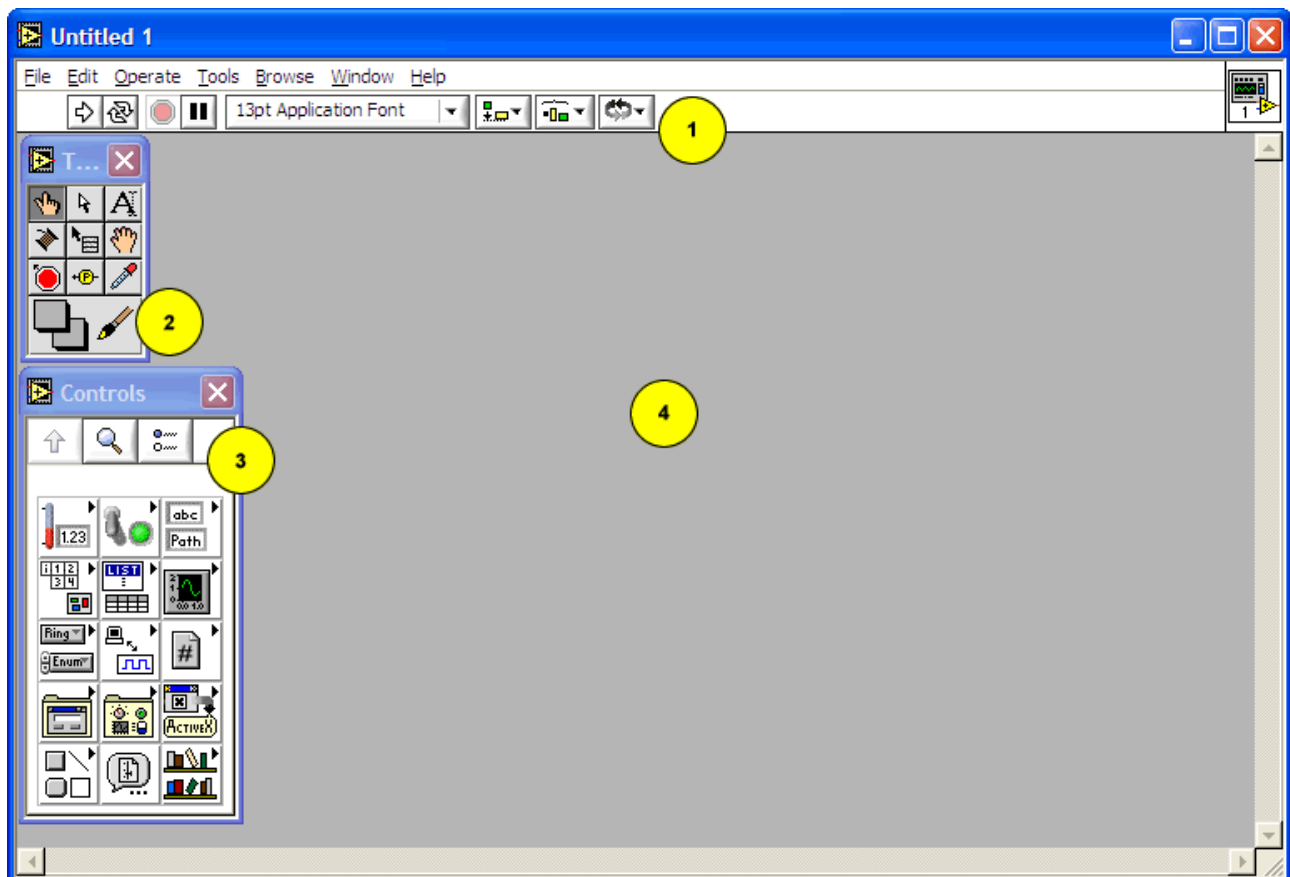
All programs, in LabVIEW are called *Virtual Instruments* and the extension of the files is “vi”. To create a new LabVIEW project run LabVIEW, then click *New VI* and then the LabVIEW editor will appear. Now go to the *schematic mode* (If you are in the front panel mode you can switch to the schematic mode using the Ctrl+E keys).

2.1 Environment.

The LabVIEW environment is divided in two main windows: the Schematics window and the Front Panel window.

Front Panel window

This is the window where the User can interact with the Virtual Instrument, placing all necessary controls to use in it (buttons, knobs, indicators, text-boxes, etc).









In the picture above you can appreciate different numbers:

1. Toolbar & Menu: in the Menu you can find all basic options provided to control the

behavior of LabVIEW as well as useful tools and basic commands such as save, load, import, etc. Some of the basic icons are explained below.

2. This is the Tools Palette, where you can find some basic tools to interact with the elements of the front panel window. You can find this palette in the schematics window as well. Some basic icons are:

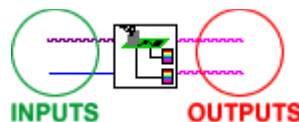
	Operate Value: it is used to change the values of the front panel controls.
	Cursor: it is used to select items, to change their sizes or to move them.
	Edit Text: it is used to add or edit texts in the window.
	Connect wire: it is used to wire all parts of the schematic or to indicate the inputs and outputs when the show connector option is enabled in the front panel window (vi icon in the right-top part of the window).
	Set/Clear breakpoint: it is used to manage breakpoints in debugging method.
	Probe data: it is used to monitor all data transferred through the wires of the schematic window. You can add a probe selecting this tool and then clicking in the wire you want to monitor.

3. This is the Controls Palette, where the user can find all front panel controls and indicators such as knobs, buttons, LEDs, scopes, etc. When a new item is added to the front panel window its equivalent “variable” is automatically generated in the schematics window.

Schematics window

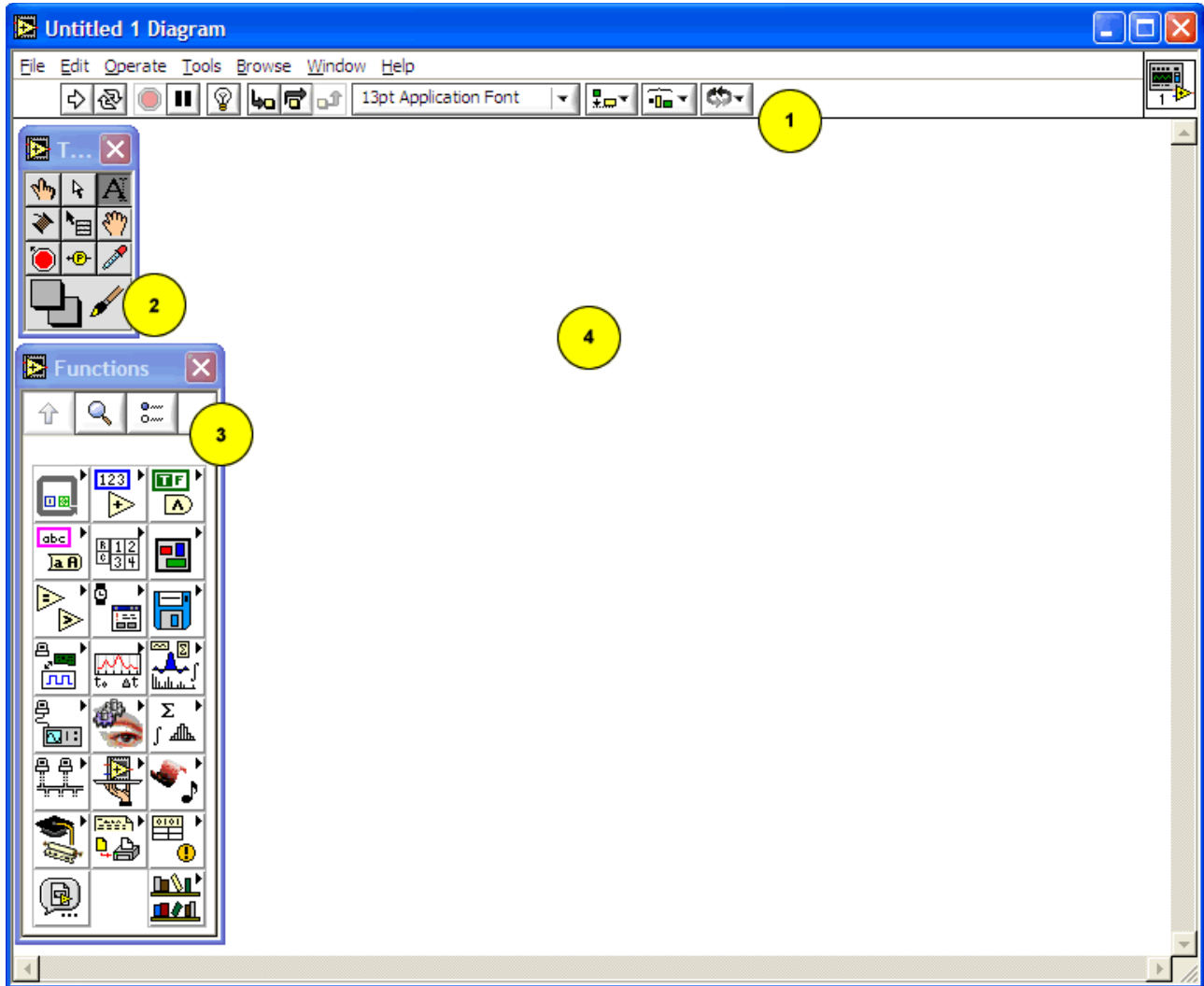
This is the window where the user can define the behavior of the programs and all commands and formulas that are going to be executed. The inputs are commonly taken from the controls of the *Front panel window* and the outputs are sent to the indicators contained in it as well.

The *Schematics window* is the area where you can put all desired functions that are represented as boxes with different icons drawn on them which are representative to its behavior. They have different inputs (on the left) and outputs (on the right).



Normally, all objects (controls/indicators) created in the Front Panel, have their equivalent paired-box in the Schematics window, which allow to manipulate them. If you create an object from the Front Panel, a control box is automatically added to the Schematics window, and also, the control or indicator appear in the Front Panel window, when it is created from the schematics window.

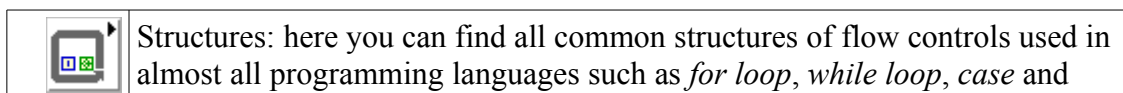
If you have any doubt about how to use a function or about their inputs or outputs you can press the keys “Ctrl+H” to show the Help window, where the description and schema will appear.










In the picture above you can appreciate four numbers:

1. **Toolbar & Menu** window
2. **Tools Palette**: also available in the Front Panel Window
3. **Functions Palette**: this palette is similar to the one of the Front Panel window but instead holds functions, flow control structures, variables, constants, logic & boolean functions, Signal processing functions, timing functions, port I/O functions, etc.

These are shown below:



	other extra items like <i>sequences</i> , <i>formula nodes</i> and <i>variables</i> .
	Numeric: arithmetic functions, numeric constants and numeric conversion functions.
	Boolean: logic functions (AND, OR, NOT, XOR, etc.)
	Strings: functions for creation and string management. You can also find here a function to create string constant or functions for string conversion.
	Arrays: functions to create and manage arrays of different types: numeric, boolean, strings, etc.
	Comparison: functions to compare numbers.
	Instrument I/O: functions to control the Serial and Parallel port, GPIB interfaces, VXI interfaces, etc.
	Import VI: this tool is used to import external vi components to your schematic.

4. **Drawing area:** this is the area where you can place functions, variables, constants and all other necessary components used in programming.

2.2 Placing items. Wire tool.

To start to design your applications the first step would be to place your desired controls and indicators in the Front Panel window, and then place the functions to operate with them in the schematics window. The next step would be to wire them.

There are different things we can place in our *vi* (virtual instrument):

- **Controls:** they are used in the front panel window and are useful to provide an input for the *vi* being developed: Text Boxes, Combo Boxes, Buttons, Knobs, etc.
- **Indicators:** they are also used in the front panel window and they represent outputs for the *vi*: i.e. LEDs, Text Indicators, Digital Indicators, Gauges, Meters, Scopes, etc.
- **Functions:** they are usually found in the *schematics* window and they can be considered as boxes that process an input and give an output as a result of some kind of processing: Additions, Subtractions, Signal Filtering, Array splitting, etc.
- **Constants:** this elements are used in the **schematics** window and are only-read items. They can be strings, integers, etc.

Controls, Indicators & Functions

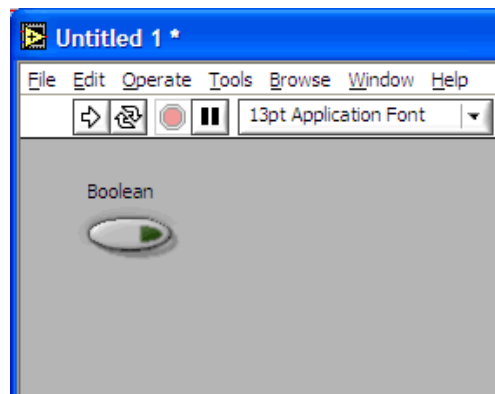
The Controls and Indicators are found in the *Controls palette* of the Front Panel Window. The functions are found in the *Functions palette* of the Schematics window. To place them, navigate through the menus and sub-menus of the *Controls palette*, click in the desired

Function/Control/Indicator and then click in the drawing area to place the element.

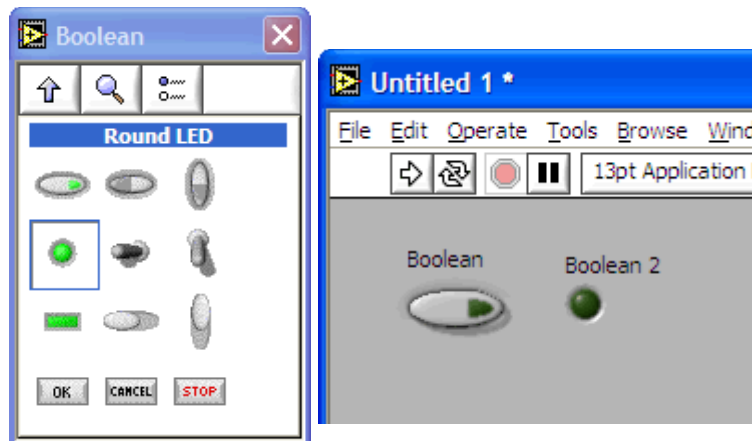
For example.


Example 2.2.a: placing and wiring

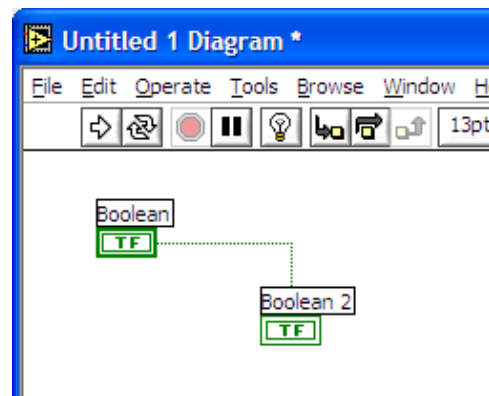
Go to the Front Panel Window (if you are in the schematics window you can switch to the Front panel window pressing the keys Ctrl+E) and click on the Boolean Menu from the Control Palette. Then select the *Push button* and place it in the front panel window:



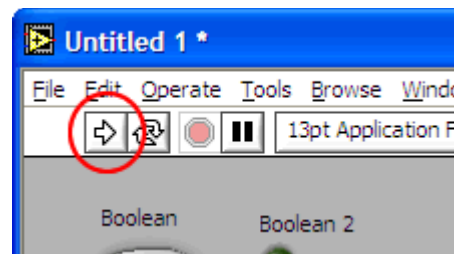
Now select a *round LED* indicator and place it beside the *Push button* control:




Now we want the indicator to show the status of the control, so you will have to put a wire between them in the schematics window (use Ctrl+E to switch between the windows). Use the *wire* tool from the Tools Palette to do the wiring .



Run the *vi* to check the result of the experiment. To run it and see the results go to the Front panel window and select the *Run* icon from the *Toolbar menu*.



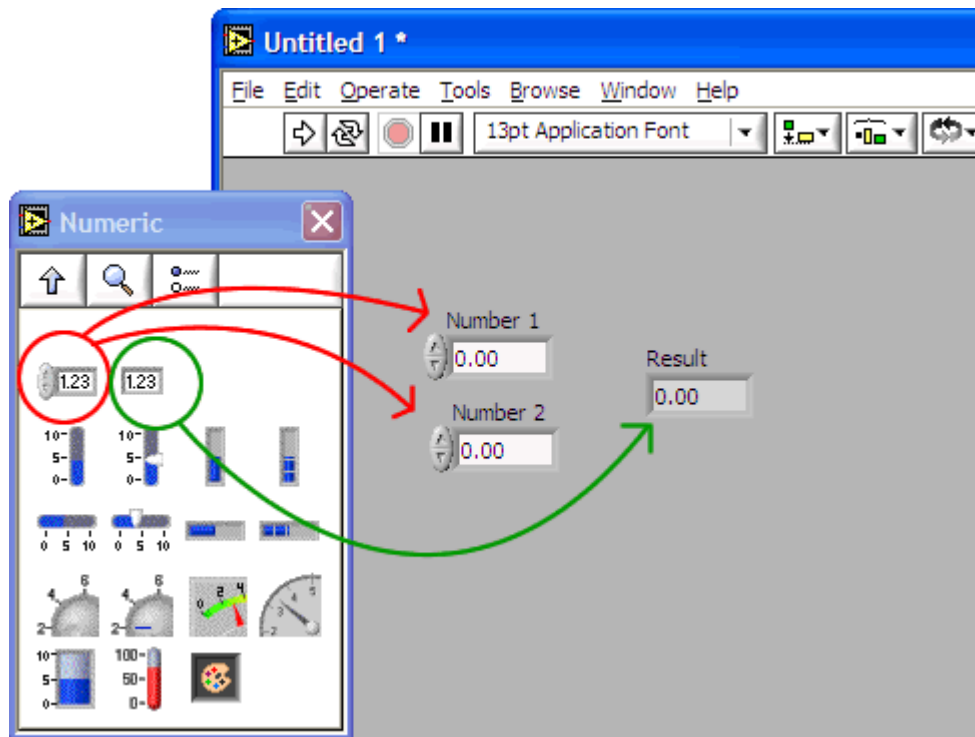
The execution will not run continuously, this means that the *vi* will be executed only once (we will explain later how to get a continuous execution).


Use the *Operate Value* tool from the Tools palette of the Front panel window to change the status of the Boolean push button . Change the status and then press run again.

Example 2.2.b: calculation and representations

In this example we will develop an application that will calculate a multiplication of two numbers given by numeric controls and the result will be shown in a digital indicator.

Create a new VI and then go to the Front Panel window. Go to the *Functions palette* and select the *Digital control* from the *Numeric menu*. Place the Digital control in the Front panel and then repeat the process to add a new Digital control below the previous one. Now go to the Functions palette, select a *Digital Indicator* and place it in the front panel besides the Digital control.

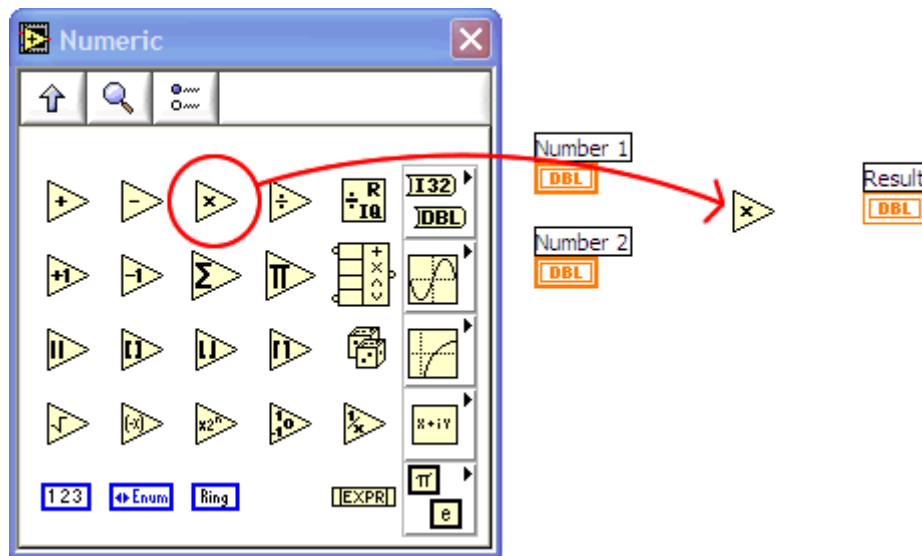


Note: if you want to change the label of controls or indicators labels you have to select the *Edit Text* tool  from the *Tools palette* and then click on the Label you want to modify.

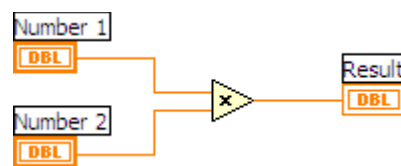
Now switch to the schematic mode to program the mathematical functions.



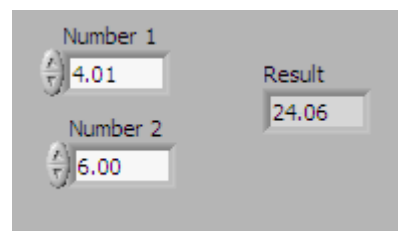
Go to the Functions palette and select the multiply function from the numeric menu, then place the function between the controls and the indicators.



The next step is to wire the controls with the inputs of the multiply function and then the output of the function to the digital indicator.



Now you can test your vi going to the Front panel window and pressing the execute button. Remember that you can change the values using the *Operate value* function of the Tools Palette in the have to execute again the program every time you change a value



Constants

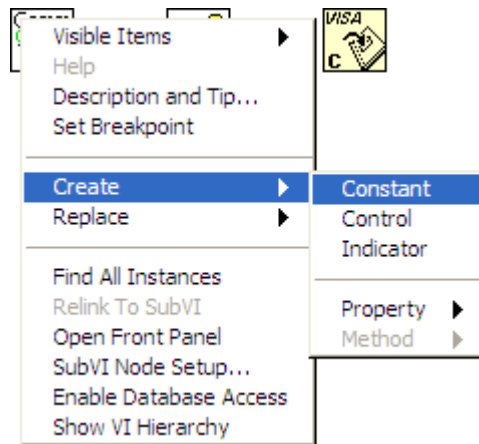
The constants are found in the *Functions palette* of the *Schematics* window. Depending on the type of the constant they are found in one or another menu.

- Numeric constants: Numeric Menu
- Enum constants: Numeric Menu
- Ring constants: Numeric Menu
- True/False constants: Boolean Menu
- String constants: String Menu

- Array constants: Array Menu
- Cluster constants: Cluster Menu
- Visa Resource name constants: Instrument I/O Menu

There are two options to place constants or indicators in the LabVIEW window:

1. Use the functions palette and select the proper constant/indicator type from it: string constant from the string group or numeric constant from the numeric group.
2. Create them directly right clicking in the desired input/output of a module box and select “Create --> Constant” or “Create --> Indicator”. To use this function you have to place the mouse over the input or output pin where you want to place it. If you use this second option, LabVIEW will automatically detect the type of needed variable. Remember to use the active help (CTRL+H) to get information about the pins.



Variables

The variables are used in the schematics window and they hold the value of the element they belong to. Variables can work in read or write mode, depending on whether they are attached to a control or an indicator.



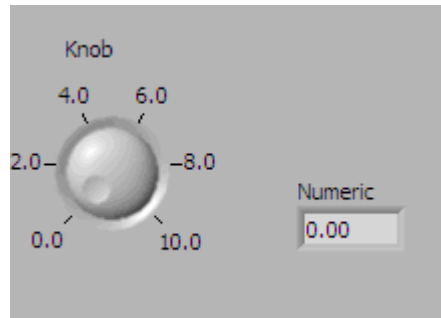
In the picture above you can see a variable box. It is easy to create a variable that belongs to one control or indicator, you only have to right-click in the control and select the option “Create--> Local Variable”.

Example 2.2.c: variable creation:

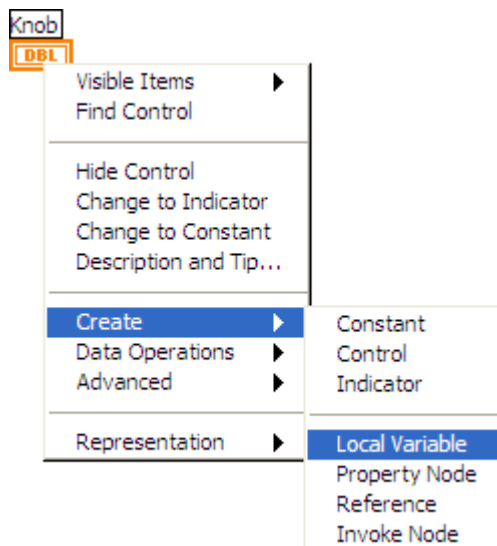
In this example we are going to copy a value from a control to an indicator without wiring them directly. Instead, an intermediate variable will be used.

In this example we need to create a *Knob control* first. To do this you have to go to the Front panel window and then click in the *Knob Control* of the *Numeric menu*. Then place the Knob in the front

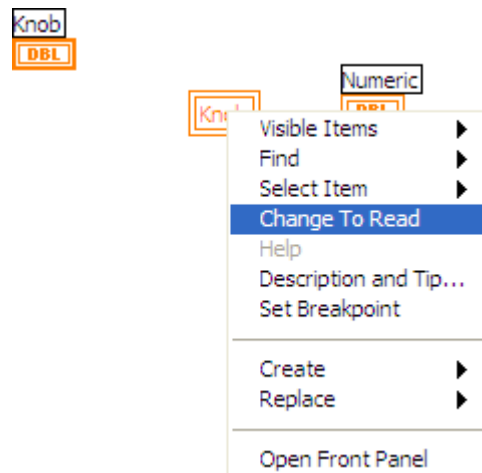
panel and also place a *Digital indicator*.



Switch to schematic mode, right click in the Knob indicator and select “Create--> Local Variable”. Then a box with the variable name will appear.



Now we want the variable to pick the value from the knob control and transfer the value to the Digital indicator. Change the mode of the variable by right-clicking in the variable box and then selecting the option “*Change to Read*” from the pop-up menu.



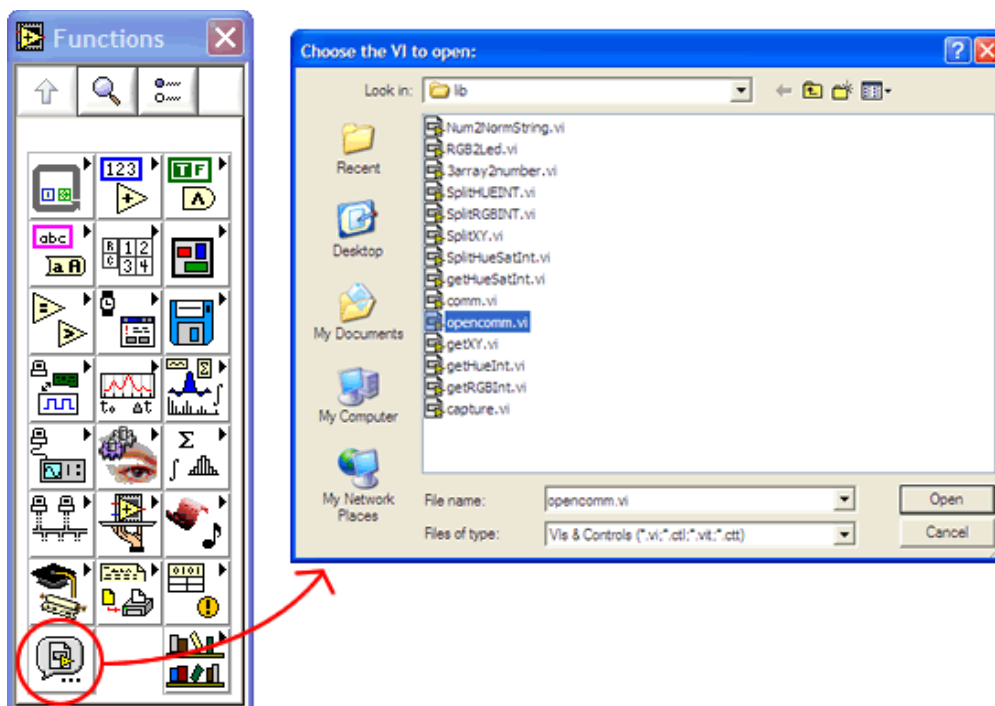
The last step is to wire the Digital indicator to the Knob variable:



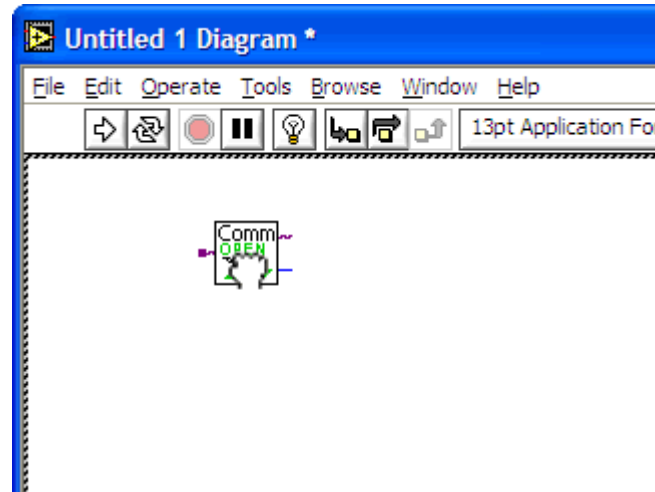
Now you can execute the vi and check how the Knob value is copied into the Digital indicator.

2.3 Import vi.

There is a useful function which can be used to import external vi modules to use them in your current schematic. To import a vi module you can click the “Select vi...” button:



Then locate the file of the module that you want to add to your schematic and click the “Open” button. Then place the module box in the desired location in the schematic.

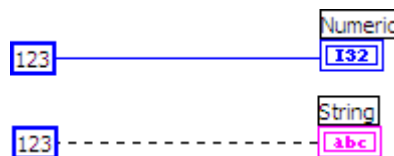


The next step is to wire the box with the surrounding components such as variables, constants, etc. to setup properly the inputs of the box and show or process the output.

2.4 Types.

There are different types of data in LabVIEW and each one is represented in the schematic using different colors. The wires connected to the components uses the same color as the source.

NOTE: if a wire is represented using a black discontinuous line it means that the source data type is different than the destination. You can see an example in the picture below.



In the picture above we have connected a control and an indicator of the same type on the top and a case of different types in the bottom part of the picture.

Let's see the different data types and their associated colors:

- Boolean (green)
- EXT: Extended (orange)
- DBL: Double (orange)
- SGL: Single (orange)
- I32: Long Integer (blue)
- I16: Word Integer (blue)

- I8: Byte Integer (blue)
- U32: Unsigned Long (blue)
- U16: Unsigned Word (blue)
- U8: Unsigned Byte (blue)
- CXT: Complex Extended (orange)
- CDB: Complex Double (orange)
- CSG: Complex Single (orange)

Arrays:

- Strings (pink)
- Paths (dark green)
- Clusters (brown or pink)

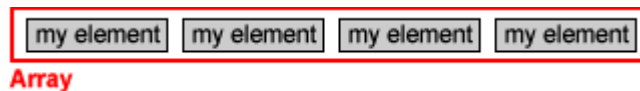
You can change the data type of all controls and indicators by right-clicking in the control (front panel and schematic) and selecting *Representation* from the pop-up menu.

2.5 Arrays and clusters.

The concept of array and cluster is quite similar with the only difference that the *Array* group a number of controls or constants of the same type, and a cluster is commonly used to group different types of controls or variables. These two types of elements are used in both Schematic and Front Panel modes.

Arrays

The arrays are considered as a group of elements of the same data type and each item has an index through which it is accessed.



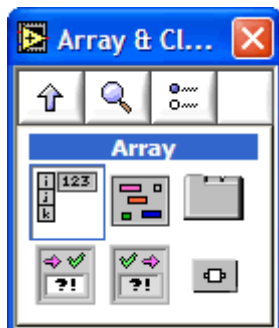
This means that you can group different controls or indicators in the front panel window to act as the same element and access to each contained element using indexing methods. To create arrays in the front panel window you have to use the *Array* option from the *Array & Cluster* menu of the Functions Palette.

In the schematics window the arrays can be managed using the different functions provided in the *Array* menu of the Functions Palette: Create Array, Split Array, Replace Element, Get Element of Array, etc.

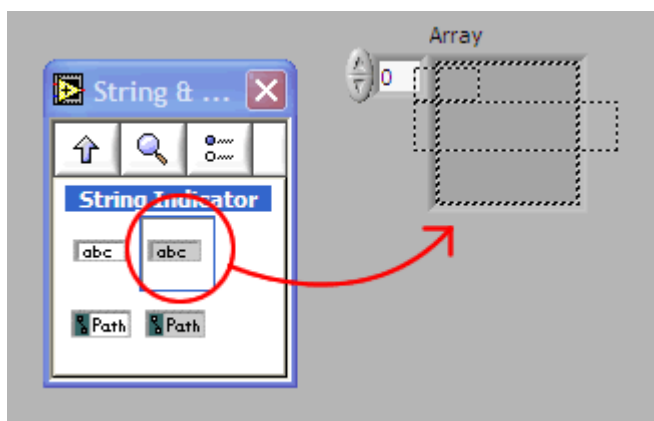
Example 2.5.a: String Array creation and representation

If we want to create a group of different String indicators we need to create an *Array of String*

Indicators. To create it go to the Front panel mode and click in the *Array & Cluster* menu of the Functions Palette and then click in the Array option.

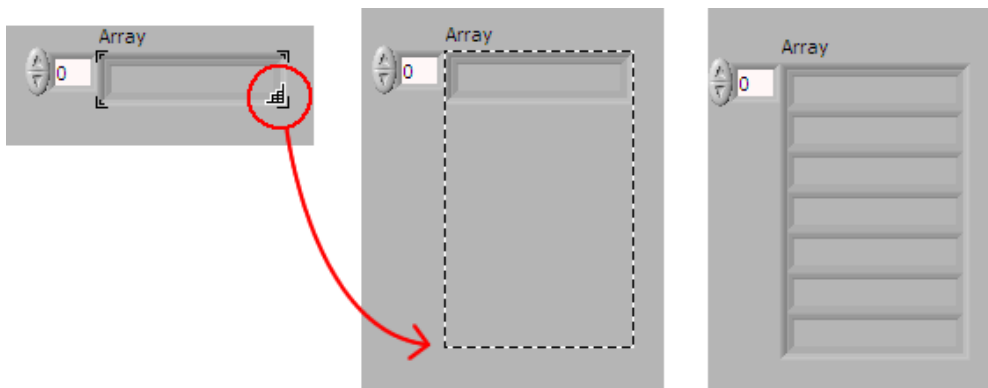


Place the array in the front panel window and then go back in the menu and select the *String Indicator* from the *Strings & Path* menu. Place the String indicator inside the Array.

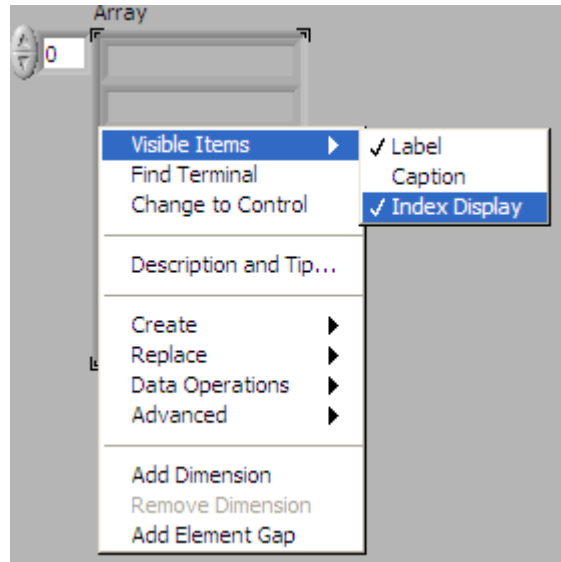


There are two ways to display the Array:

- Show one indicator at the same time and switch between them using the index controller (white box with a number in it and up/down arrows beside it)
- Show all elements at the same time. To do this it is necessary to resize the array by dragging the right-bottom corner with the mouse cursor. Remember to select the Position/Size/Select tool first from the tools palette.

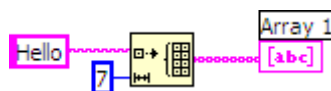


After resizing the array control you can disable the *Index display*, right-clicking in the array edge and unchecking the *Index Display* option from the Visible Items submenu.



Now if we go to the schematic mode we can create an array and represent it in the Array indicator we have just created. Go to the *Functions palette* and select the function *Initialize Array* from the *Array menu*. Place the function in the schematic and insert the parameters as follows.

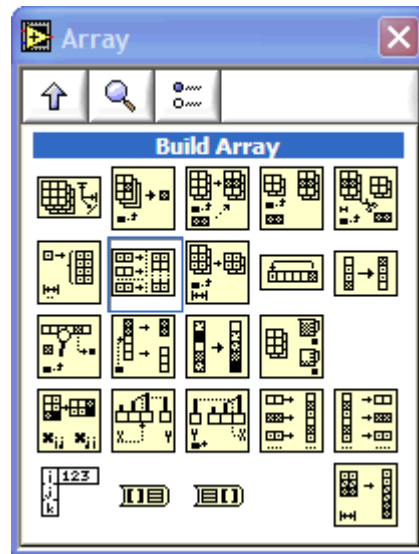
- The first input is the default value for the elements of the array that is going to be created (this function uses the same type for the array elements than the type of the first input).
- The second input is the size for the first dimension. You can check the help (Ctrl+H) to get more information about how to create more dimensions.



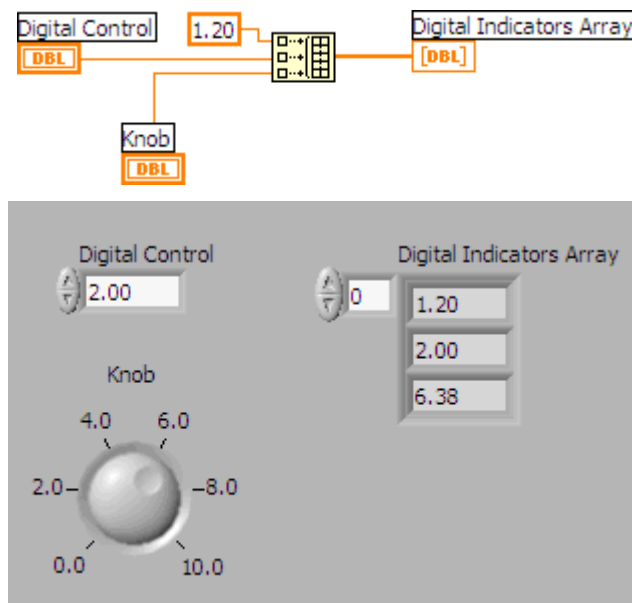
Note that the dimension size should be the same than the destination array indicator. In the schematic above we have created an Array strings of 7 elements (7 strings) and we have wired it to the array indicator. Execute it and see the results.

Example 2.5.b: compound array

You can also compound an array using the function *Build Array* of the Array menu in the functions palette.



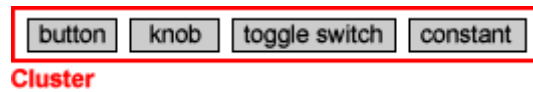
You can select the number of inputs of the array clicking the mouse in the right-bottom corner and dragging down the mouse. You only have to add elements to that array.



In this example we added 3 numeric elements to the array and we displayed it in a Digital indicator array.

Clusters

Sometimes is used to group different Controls or Variables together.



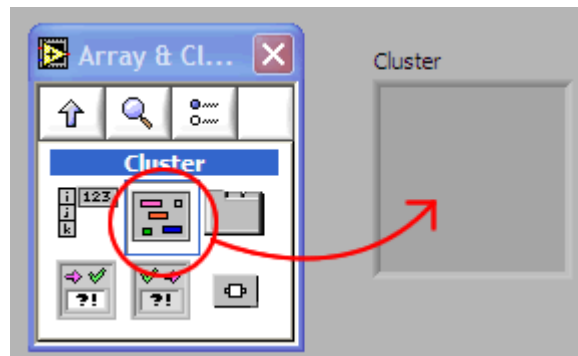
You can understand the cluster structure as a box that holds a lot of different items. You can create a cluster from the front panel window using the *Cluster* function in the *Array & Cluster* menu or can also be created in the schematics side using the *Cluster* menu of the functions palette.

There are specific functions to manage clusters in the menu *Cluster* of the Functions Palette (array-related functions doesn't work).

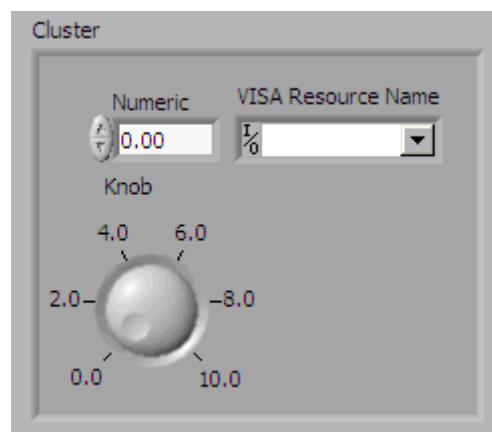
Let's see some examples:

Example 2.5.c: create a Cluster

Now we are going to group different control types in a cluster. To do this you have to select the *Cluster* option from the *Array & Cluster* menu in the *Controls Palette* and place the cluster in the front panel window.

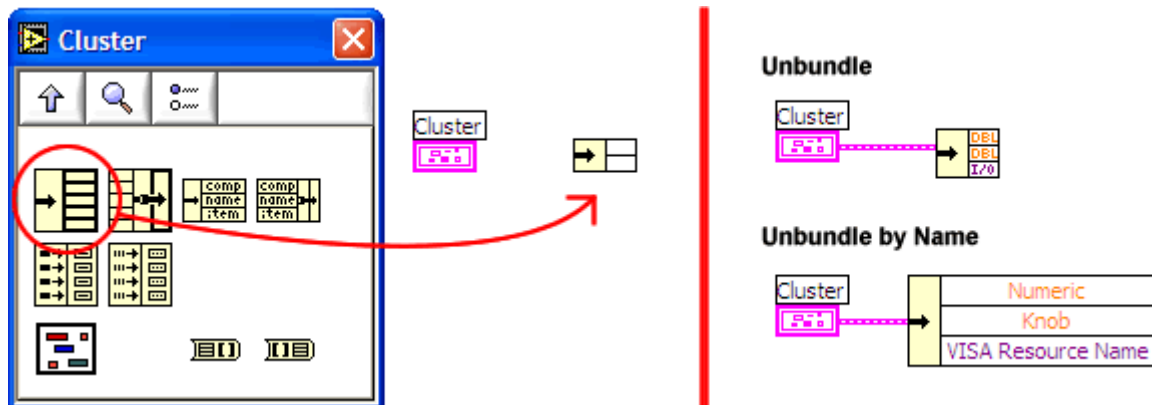


Now you can add different items to the cluster.



But if you switch to the schematic mode you will see only one element.

Despite of having all components grouped, you can access all elements of a cluster at anytime using the function *Unbundle* or *Unbundle by Name* that you can find in the *Cluster* menu of the Functions Palette (schematic window). This function will provide an output for every component in the cluster.



Then, if you wire the cluster with the Unbundle functions, it will automatically read all components of the cluster. If you use the function *Unbundle by Name* you can select what's the item you want to see but if you want to show all components you can do it by clicking in the bottom-left corner of the function box and dragging down it.

2.6 Flow Control.

The flow control structures allow the programmer to intervene in the vi execution process, being able to repeat some parts of the code (schematic) or decide whether to execute specific instructions.

The flow control structures can be found inside the *Structures* menu in the *Functions Palette* of the Schematic window:

- Sequence
- Case
- For Loop
- While Loop

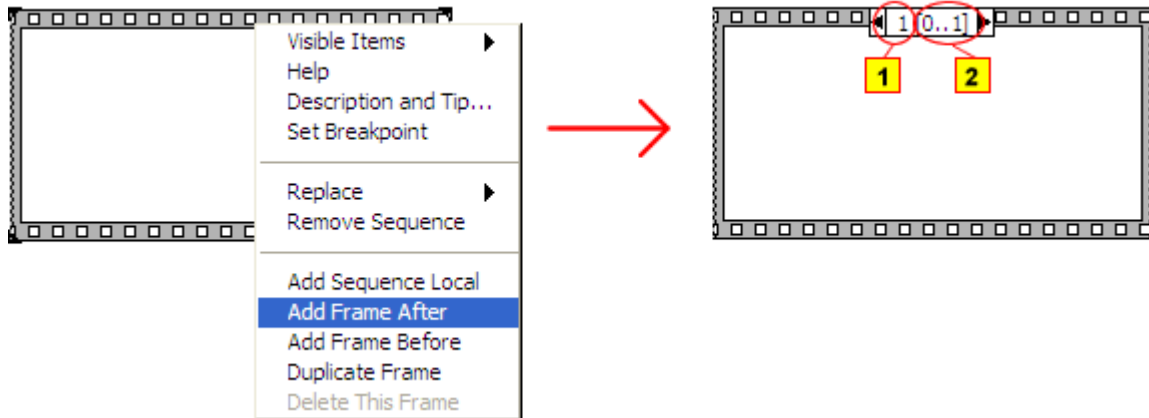
Sequence

This structure allows segments of the code or frames to be executed in a sequential manner. This is analogous to the frames in a movie film running sequentially.



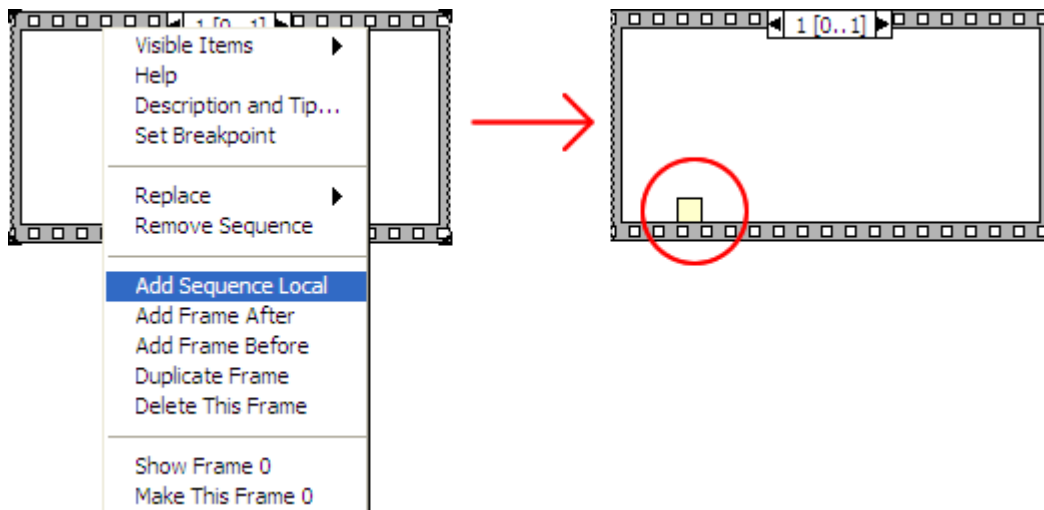
The sequence is a control in which you can create frames and each one contains pieces of source code inside. The frames are executed sequentially starting from frame 0 to the last one and you can

create as many as you want by right-clicking on the top edge of the frame and selecting the option “Add frame” from the pop-up menu.



Now in the top part of the sequence will appear an indicator giving information about the current frame (number 1 in the picture above) and information about the available frames (number 2 in the picture above). You can navigate between frames using the left and right arrows.

You can also share information between frames using *Local variables* or “Sequence locals”, so a data generated by one frame can be read by the following frames. To create Local variables you must right-click on the edge of the sequence structure and then press the option *Add Sequence Local*.

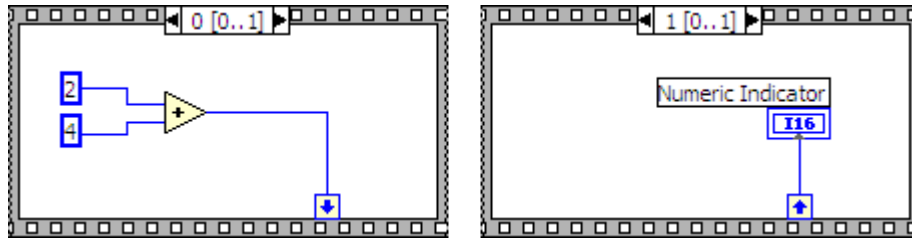


The created Sequence Local will take the type from its source object once it has been wired. If you connect a String constant to the sequence, then the sequence will become a string. If it is wired to a numeric control it will become numeric.

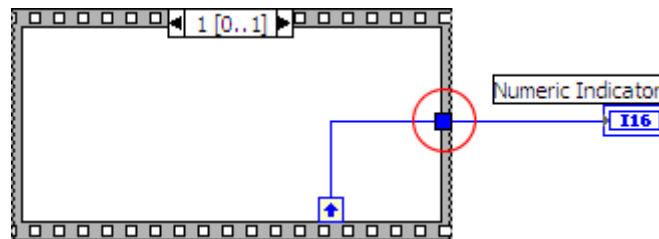
Example 2.6.a: Creation of sequences

In this example we are going to create a sequence with two frames, the first frame will calculate an addition of two numbers and the result will be transmitted to a following frame using a *Sequence*

Local. You can see the schematic below:



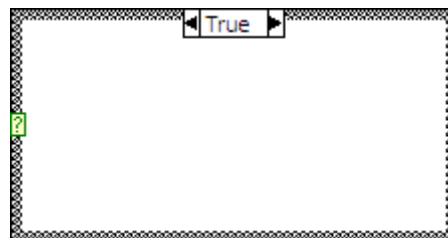
To show the results, we have created a numeric indicator and we have placed it inside the second frame, so the value will be transferred when this frame is completed. If the Numeric indicator is placed outside the sequence structure and connected through a **tunnel** (circle in the picture below), the value will be transferred to the indicator when the last frame of the sequence is executed.



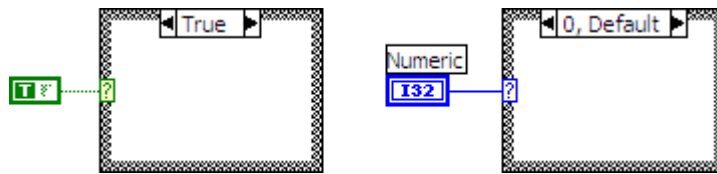
The sequence will help to save space in the schematic because sequential logic processes can be grouped into one sequence structure.

Case

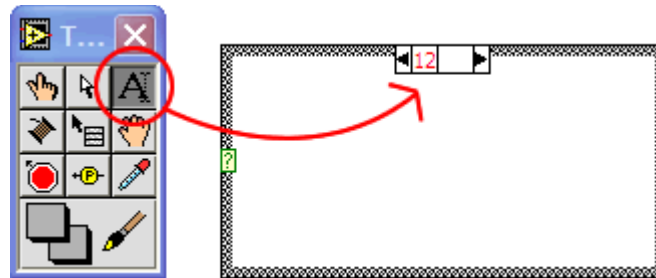
The case sequence will allow us to change the behavior of a part of the program depending of one condition. It is similar to the last structure, but only one of the frames is going to be executed, this means that we can write different sections of code in the same structure, but only one section will be executed. To determine which frame will be executed, this structure will check the *condition* given by an input line -in the left part of the structure (small box with a question tag inside)-. The cases are shown in the top part of the case structure.



Once we connect the input condition to its source, the *Case structure* will take the same type as it. This means that if we connect the input condition to a boolean indicator, the cases of the case structure will be True and False. If we connect it to a numeric control, the structure will cases will be 0 and 1 by default.



The conditions can be changed by editing the text label of each case using the *Text tool* of the *Tools Palette*.



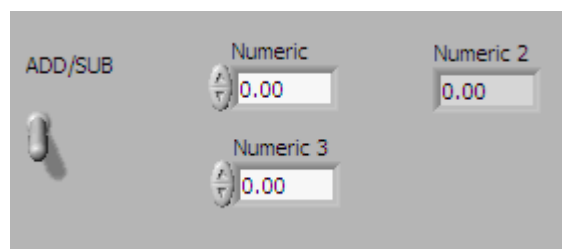
In this structure you can add and delete cases similarly as in the previous structure (frames), clicking in the *case box* on the top of the structure (the one with the two arrows). You can view or edit all cases by selecting them first using the arrow.

Note: if you use an output tunnel in the case structure to transfer data from inside the case structure to outside you will need to connect the tunnel to a data in all cases. Otherwise if you leave the tunnel unconnected in a case, you will get an error.

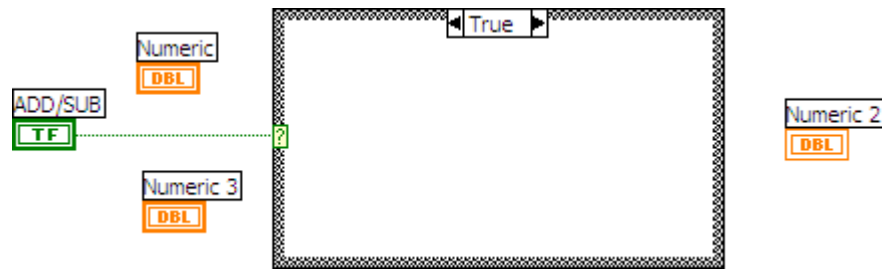
Example 2.6.b: case sentence

In this example we are going to create a case structure to decide the mathematical operation that we are going to use with two numbers to calculate the result.

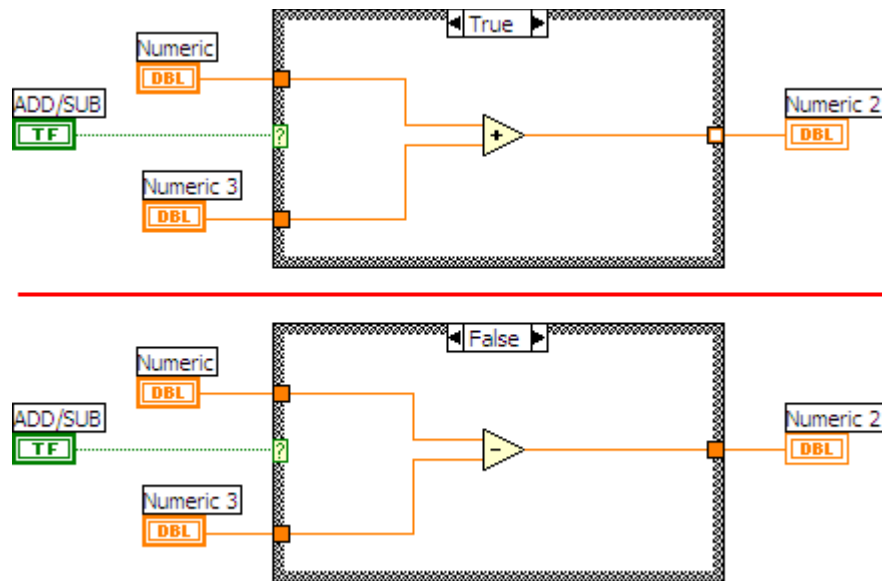
Go to the Front panel window and add a *Vertical Toggle Switch* (Controls palette--> Boolean), then add two *Digital controls* and an Indicator. Now rename the Toggle switch to "ADD/SUB".



Now go to the schematics window, create a *Case structure* and the connect it to the toggle switch control. After this, place the Digital controls on the left of the case structure and the Digital indicator on the right.



Now, we are going to place an *Add* function inside the True case, and a *Sub* function inside the False case.



Note that before completing the False case, the output tunnel box is filled in white instead of orange: this indicates an error caused because not all possible cases return a value. So once False is filled in and wired to the output, the tunnel turns to orange, indicating that everything is correct now.

Now try to change the toggle switch in the front panel window and execute the program to see what happens.

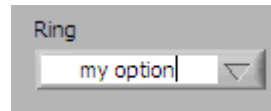
Example 2.6.c: Multiple option selector

In this example we will learn how to use the *Ring control* and the case structure to select between the different options given. We will show a different message box to the user depending on the selected option.

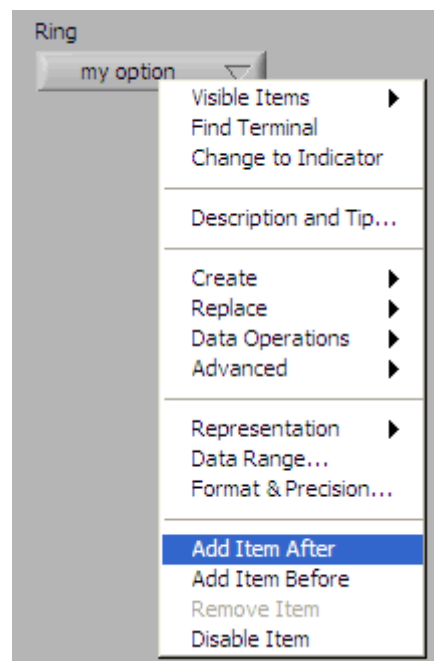
You can find the *Ring control* in the *Ring & Enum* menu of the Controls palette of the Front panel window.



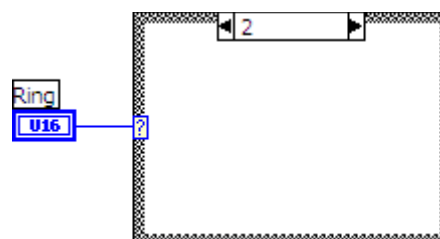
This control provides a drop-down menu in which you can add different options that could be selected by the user. A default option is created initially with a blank text, that can be edited using the *Edit Text* tool from the *Tools palette* (click inside the control to write your text).



You have the possibility to add more options if you right-click in the Ring control and select the option *Add Item*. So now let's to add 2 new options.



Then go to the schematics window and place a *Case structure* and wire it with the Ring control. Notice that the data type for the Ring indicator is numeric in this case (blue color), this means that it will not provide the text of the options but the index of them: option 1 (index 0), option 2 (index 1), option 3 (index 2), etc. Check the number of cases of the Case structure because we have 3 options and maybe you have only two cases, so you would like to add a new case for the third option. If you don't add a third case and this has been selected, the default case will be executed instead.

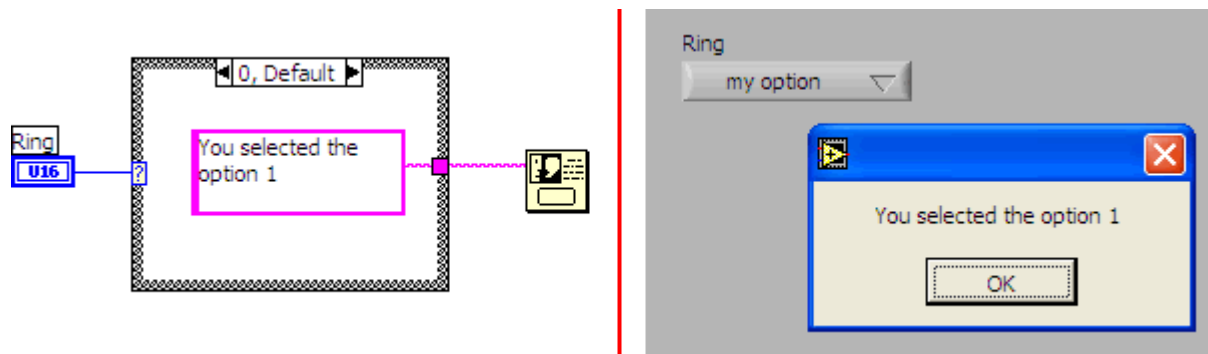


One button dialog

Now I will introduce you a new function, the *One button dialog* that can be found in the *Time & Dialog* of the functions palette which is used to show message dialog boxes. Select and place it on the right of the case structure.



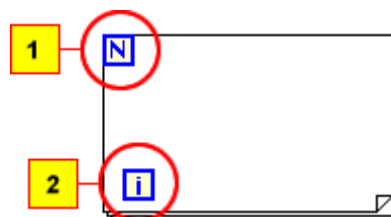
Now we will place a *String constant* (it can be found in the *String menu* of the *Functions palette*) inside each case and we will wire them to the string indicator. The target of this is to show a different message in the dialog box, depending on the selected option in the Ring control.



Now experiment with the VI and check the result.

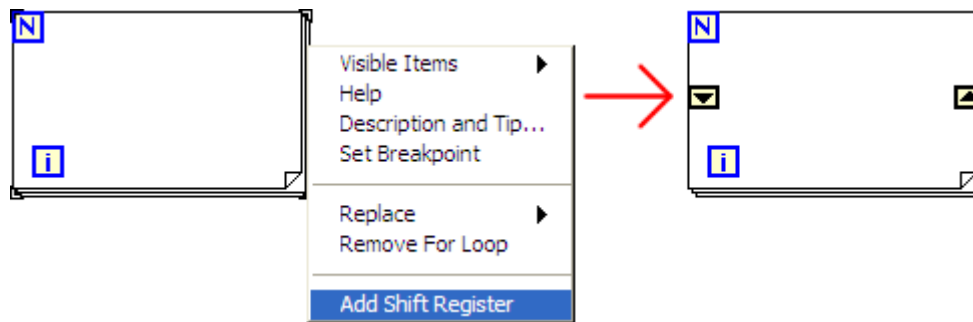
For Loop

This structure is not divided in frames like the previous structures. It only contains one place to write code, but has the property that the code will be repeated a number of times.



If you see the picture above, the object with the mark 1 is the number of times that the code will be repeated, so you have to wire a numeric constant to this point. The number 2 is an item that indicates the current loop count that is being executed (the first loop is 0).

In this structure you can create some type of local variables called *Shift registers*, that are used to transfer a value between one loop and the next one. You can create as many *Shift registers* as desired right-clicking in the left or right edge of the For-Loop structure.

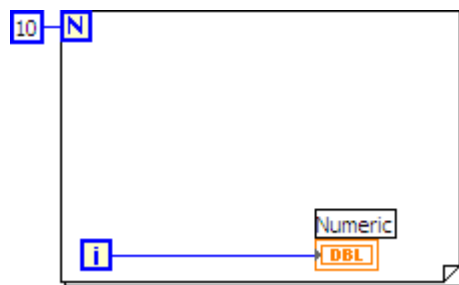


In this structure you can also transfer values from inside to outside using tunnels with the only difference that the value will not be transferred out of the structure until the end of the last loop. There are two types of tunnels: indexed and not indexed.

- Not indexed: In this value we are transferring the original data type without modifications. If the tunnel is an input, the value will be taken in the first loop. If the tunnel is an output, the value will be transferred in the last loop.
- Indexed: in this mode we are not managing single numbers but arrays. If we have an input tunnel with an array and don't enable indexing, the whole array will be transferred to the structure in the first loop; If we enable indexing, the loop will take a different index from the array every loop. If the tunnel is an output it works similarly: if indexing is not enabled, the original data type will be transferred to the output otherwise an array will be transferred to the output at the end of the last loop, but a different index will be written in the array every loop.

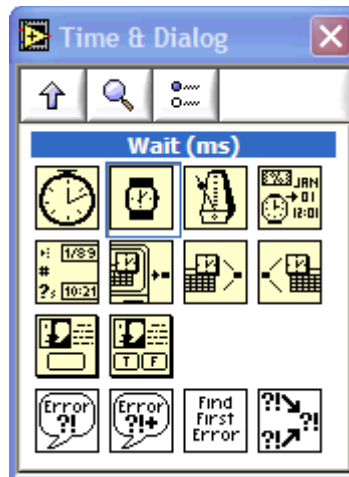
Example 2.6.d: Iterative numbers

In this example we are going to create a *For loop* and see the iterations of the loop in real time. To do this go to the front panel mode and add a *Digital Indicator*. Then switch to the schematics mode and place a *for loop*; then create a *Numeric constant* with a value of 10 and wire it to the loop counter. Now place the digital indicator inside the For loop.

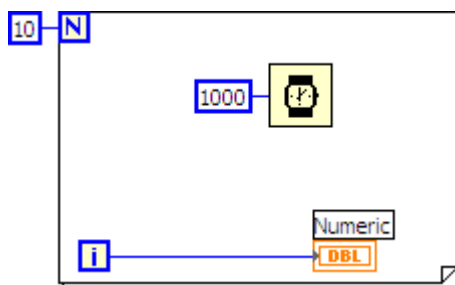


This program will do 10 iterations and display in the digital indicator the current iteration every loop. If you execute this program you won't be able to see anything because the program runs too fast so you can use the step-by-step execution mode or you can do a small modification in the program to show all iterations: we are going to introduce you the timing functions.

Go to the *Functions palette* and select the *Wait (ms)* function from the *Time & Dialog* menu. Place the box inside the For loop. Now add a numeric constant with the value 1000 to the input of the Wait function.



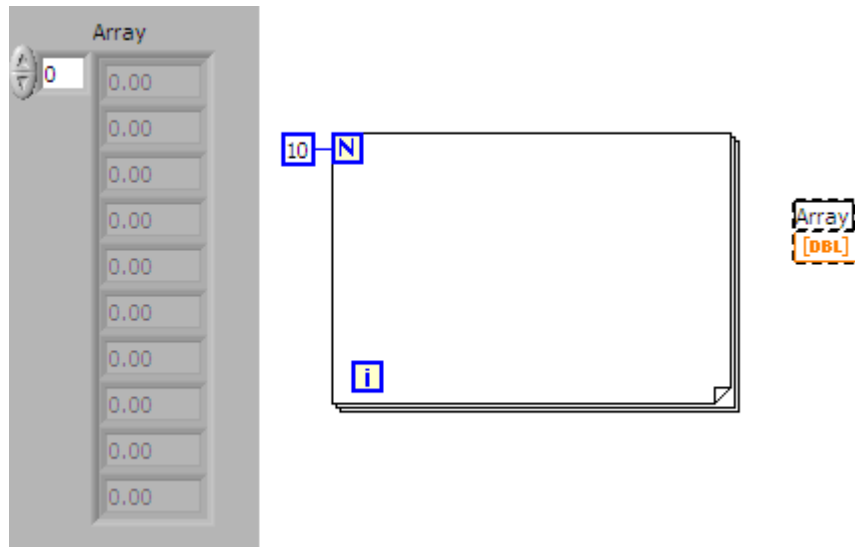
These timing functions only affect to the environment in which they are. If you put a 1 sec Wait function inside a frame of a sequence the program will stop the execution for one second in that frame and then it will go on normally. So if you put the Wait function inside the For loop the program will stop the execution for one second each iteration of the loop.



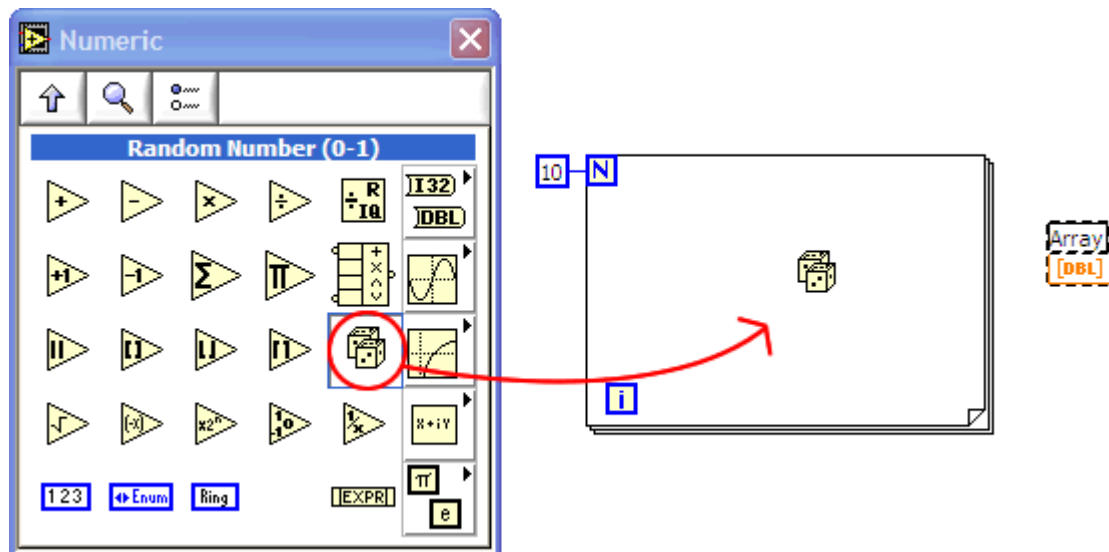
Now if you execute the vi you will see how every 1 sec the value increases in the digital indicator, from 0 to 9.

Example 2.6.e: Random numbers array

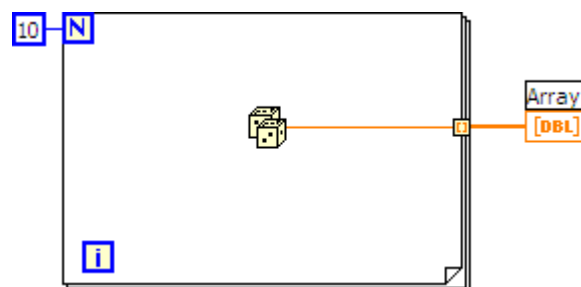
In this example we are going to generate an array of 10 random numbers so we are going to use *Indexing* and a new function to generate random numbers.



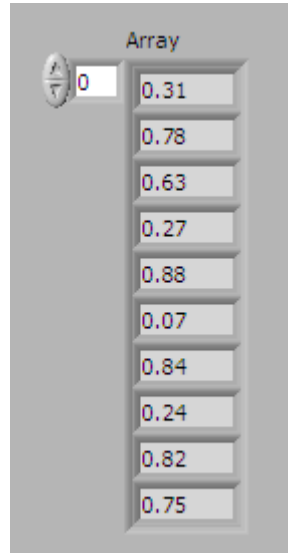
The first step will be to create an array of 10 *Digital indicators* in the *Front panel* window. Then go to the *Schematics* window and create a 10-iterations *For loop*. Now select *Functions palette*→ *Random Number* function from the *Numeric* menu and place it inside the *for loop*.



Connect the random number generator function to the Digital indicator array through an indexed tunnel.



If we execute the vi, the *for loop* will save a new random number in the *array* on each iteration, so you should see 10 different numbers in the *Digital indicator array* in the front panel window.



While loop

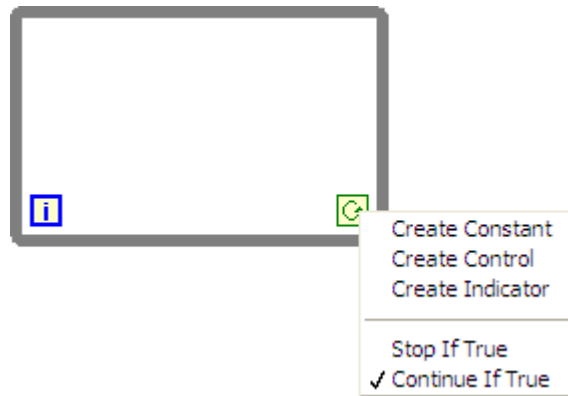
The while loop is a continuous loop with only a stop condition, so once the loop is running it can only be stopped when the stop condition is true.



In the picture above you can see the while loop structure in which you can find two numbers: the number 1 is the current number of iteration and the number 2 is the boolean stop condition. This last one can switch between 2 conditions:

- Stop If True: the loop will continue the execution until the condition is True.
- Continue If True: the loop will continue if the condition is true and it will stop if the condition is false.

To change the condition type you have to right click in the Stop condition and select the new condition.



The way of transfer data between the inside of the loop and the environment is exactly the same as previous structures, using tunnels (indexed or non indexed).

It is also possible to create *Shift registers* to transfer data between one iteration and the next one, like the *for loop*.

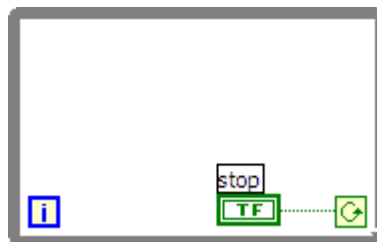
This type of loops are very useful because they are commonly used as a **main loop** of the vi. This means that until this moment, all vi we have developed were not able to run continuously. It was necessary to press the run key every time we wanted to execute them. So we can solve this situation using the while structure, because we will be able to run the program continuously until an *Exit* button is pressed or all measurement process's has finished, etc.

Let's see some examples:

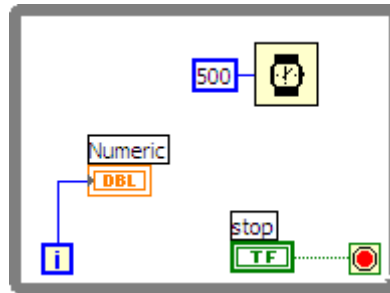
Example 2.6.f: Exit button

In this example we are going to create a vi that will run continuously. We will include a Stop button that will stop the application when it is pressed.

Go to the front panel window and select a *Stop Button* from the *Boolean* menu of the Controls Palette, place it in the panel and then add a *Digital indicator*. Now go to the schematics window and add a *While loop* to the schematics. Then put the Stop control inside the while loop and wire it to the *Stop condition* but you have to change the stop condition to Stop If True, because the Stop button generates a True condition when it is pressed.



Now put a *Wait (ms)* function inside the while loop, setup it with a time of 500 ms. Place the digital indicator inside the *while loop* and wire it to the current count number.

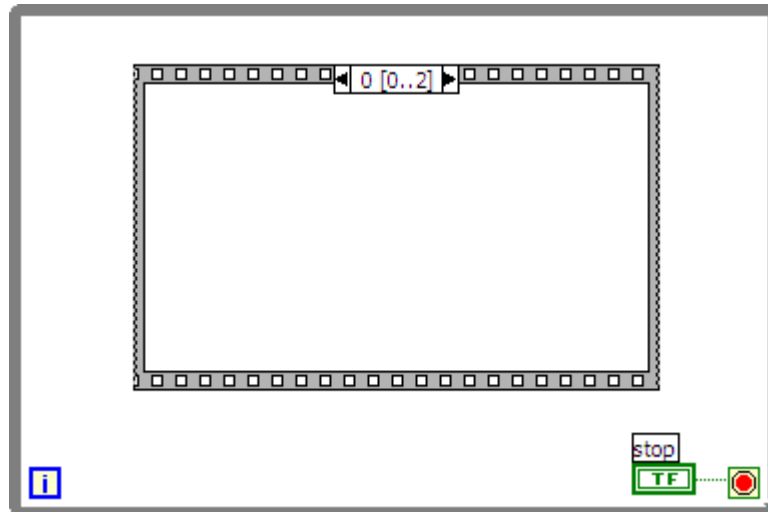


If you run the application you will see how the numeric counter increases by one every 500 ms infinitely until the Stop button is pressed.

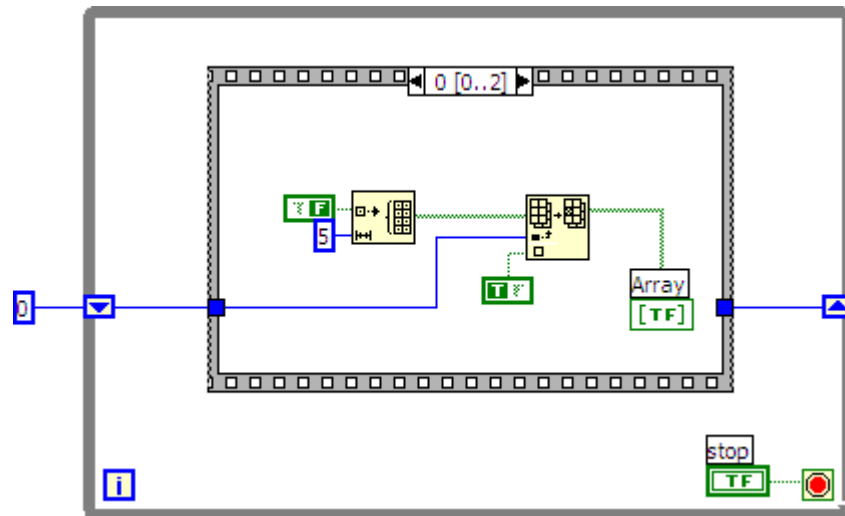
Example 2.6.g: fun lights

In this example we are going to design a set of 5 lights that will turn on and turn off consecutively. Only one light will be turned on at time, and the others will be off.

To do this you need to go to the Front panel window and create a Stop button and an Array of 5 rounded LEDs. Then go to the Schematics window, create a while loop and put the stop button inside. After this you have to create a sequence inside the while loop with 3 frames. See the picture below to understand the structure.

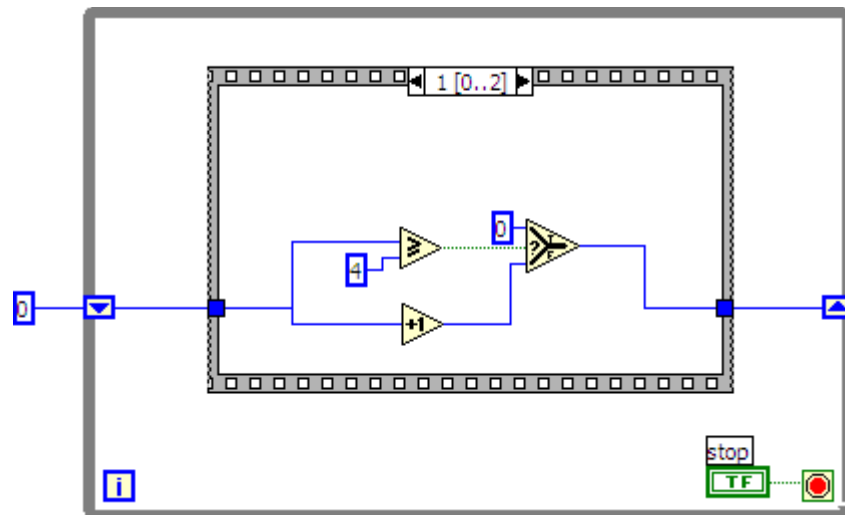


Now, add a shift register in the while structure, that will be the index of the switched-on LED. In the first frame we will build an array of 5 boolean numbers, initialized to False, which is wired to a function that change the status of the current LED to True in the array.

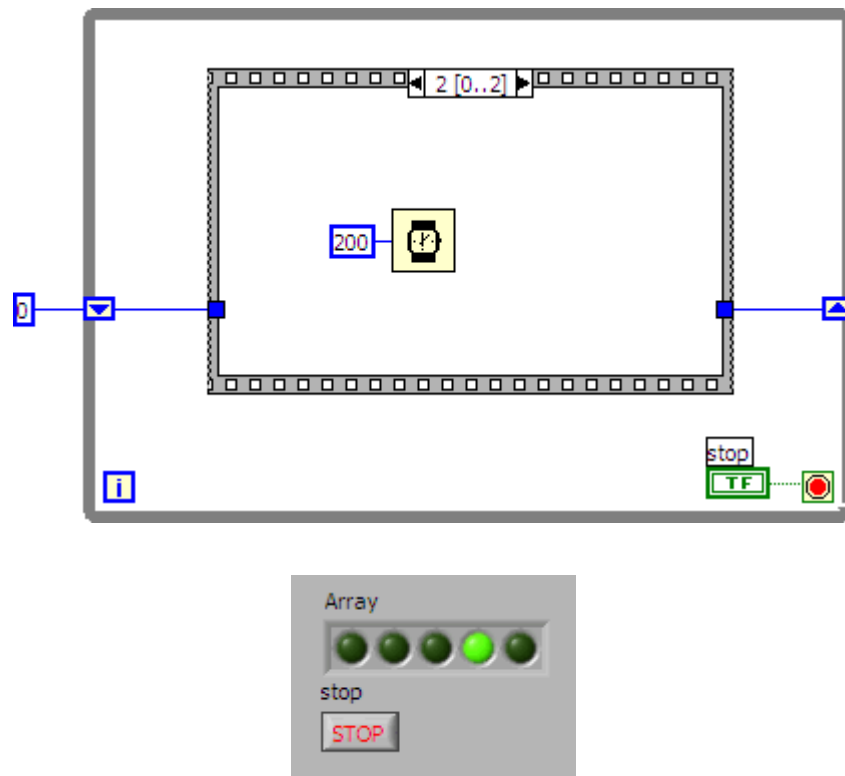


The picture above is the schematic of the process that creates an array of five items with the function *Initialize Array* and replace the current index to turn on with the function *Replace Array Subset*. This process is calculated every loop.

In the frame number 1 you have to increase the index by one and then check the overflow because when this occurs you have to reset the value to 0. This is made using a comparison, an increment function and a boolean *Select* (remember to use Help in case you have any doubt about how using those functions).



The third frame consists on a delay of 200ms.



2.7 Property nodes

Properties define the behavior or appearance of Controls and Indicators and it's possible to change them using *Property Nodes*. Depending on the control used there can be different many properties or even none, and those properties are handled through the Schematics window.

You can create a property node by right-clicking the mouse over the desired control or indicator (in schematics mode) and then select Create->Property Node.

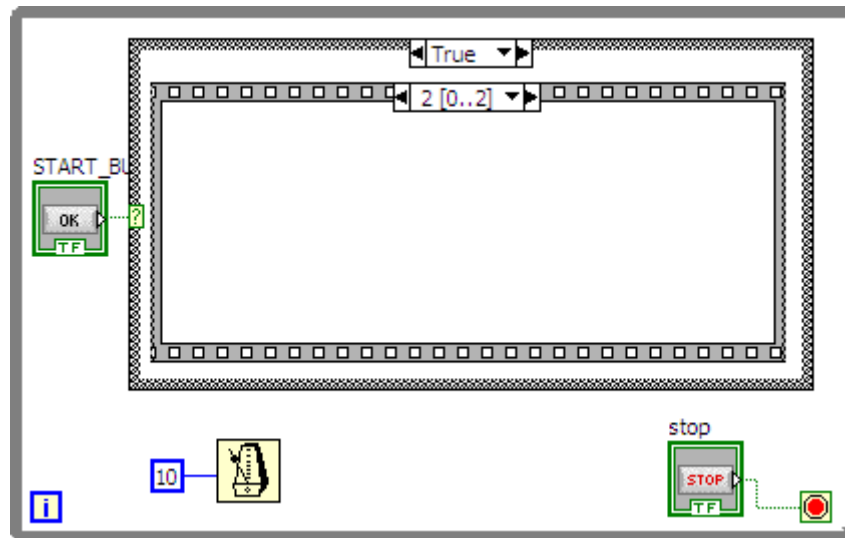
Let's see how to work with property nodes through an example:

Example 2.7.a: Enable/disable button.

The target of this exercise is to create a sequence that disables a button and enables it two seconds later. For this purpose we will create two buttons first: a *OK button* to start the process a *Stop button* to stop the program execution.

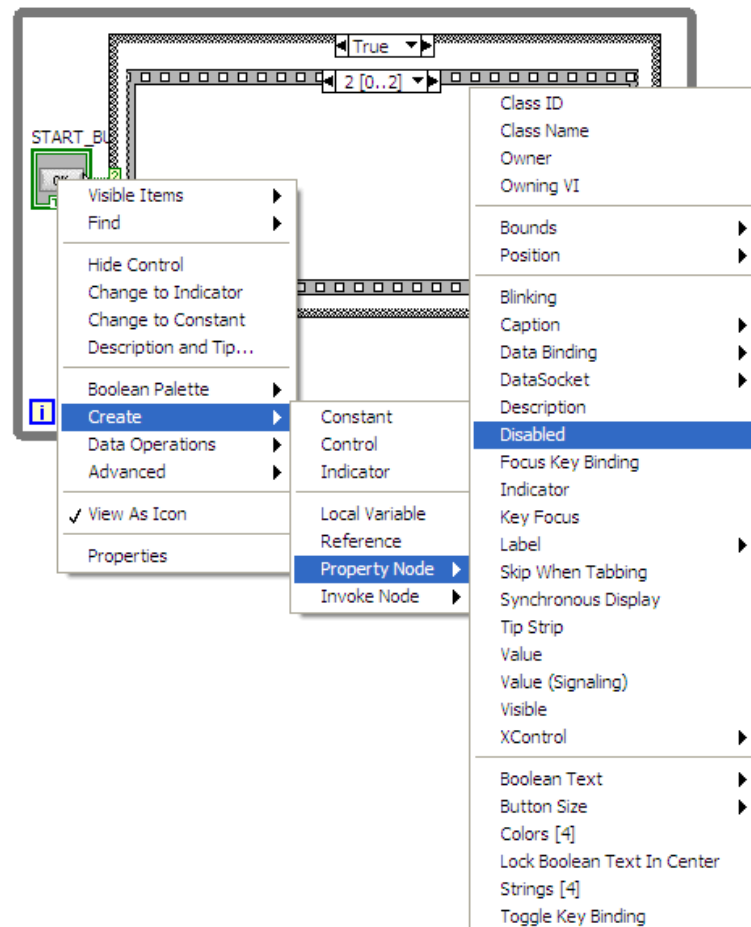


Then, in the schematics window, a basic while-loop program structure will be created in order to keep the program working when the execution is started. We have already experimented with this structure in previous examples.

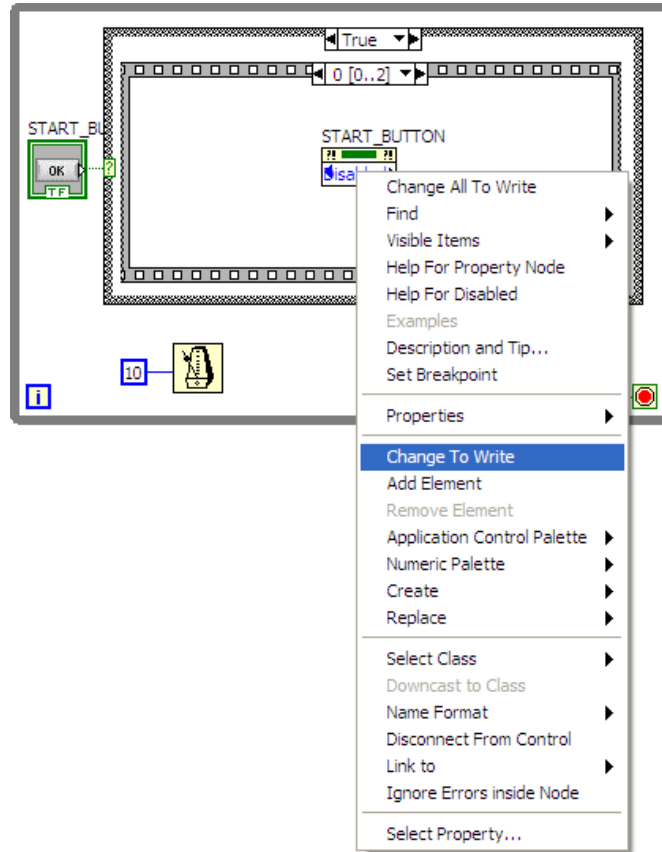


The start button is wired to a Case structure, allowing the sequence, which have been placed inside of it, to be started when the button is pressed. This sequence will contain 3 frames: the first one to disable the button, the second one to delay the execution and the last one to re-enable the button.

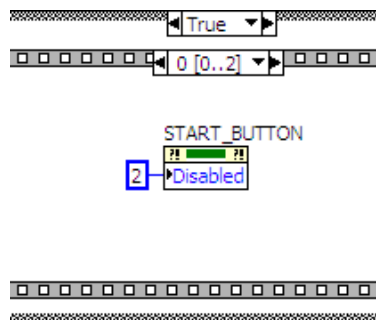
To achieve this target, a property node has to be created first:



By default, the node has a read-only behavior, but we want to set its value, so it has to be changed to *Write*.



Now, we have to set the value: 2 for grayed out & disabled and 0 for Enabled (check help for more info).

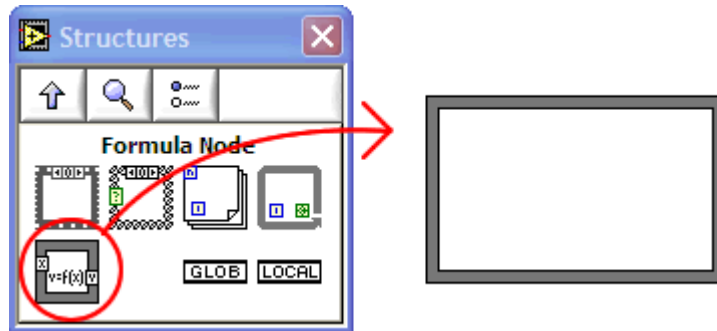


Now, after completing the sequence 1 (Wait ms function) and 2, the program should operate as expected.

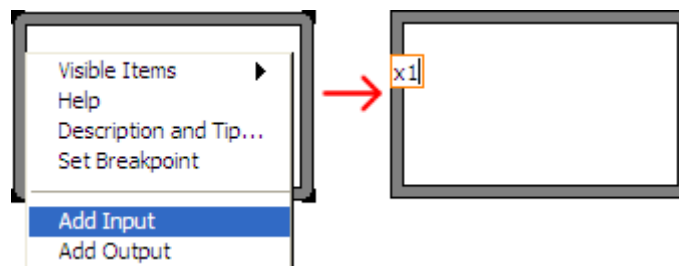
2.8 Formula nodes

The *Formula node* is a structure that allows the programmer to write formulas inside, providing the parameters as input tunnels and get the results as output tunnels. They can only be used in the

schematics window and can be found in the *Structures* menu of the *Functions Palette*.

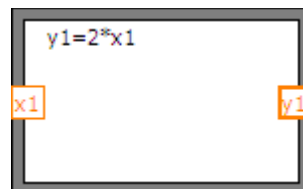


You can add inputs or outputs right-clicking in the edge of the Formula node and selecting the proper option from the pop-up menu.



Once you click in the *Add* function, a box will appear in which you have to write the name of the Input or the Output.

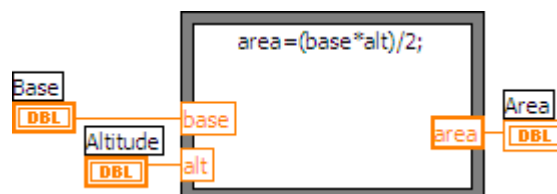
Write the formula inside using the *Edit text* tool of the *Tools palette* and clicking inside the formula node.



Now let's see some examples.

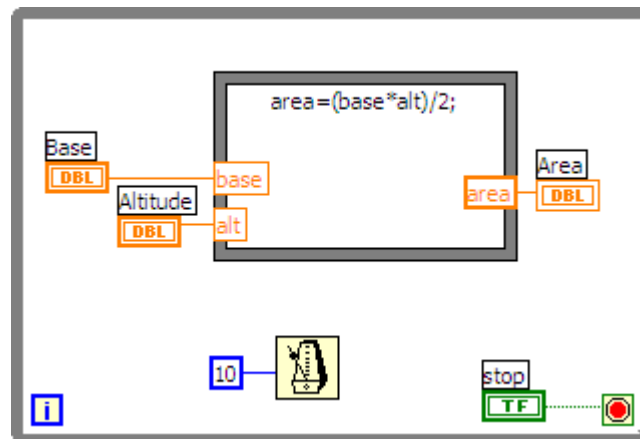
Example 2.8.a: Calculate the area of a triangle

In this example we are going to calculate the area of a triangle. So we need 2 digital controls (altitude & base) and a digital indicator (area) in the front panel and then a formula node in the schematics window.



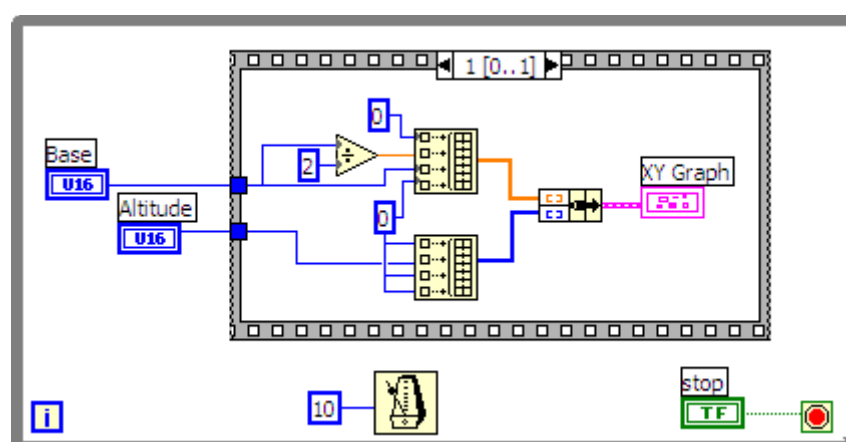
Note that all lines of the formula node are finished with a semicolon character. So if you want to put 2 different mathematical expressions inside the formula node, they should be finished and separated by semicolons.

Now we will add an external while loop and a stop button to make the vi work without interruption.

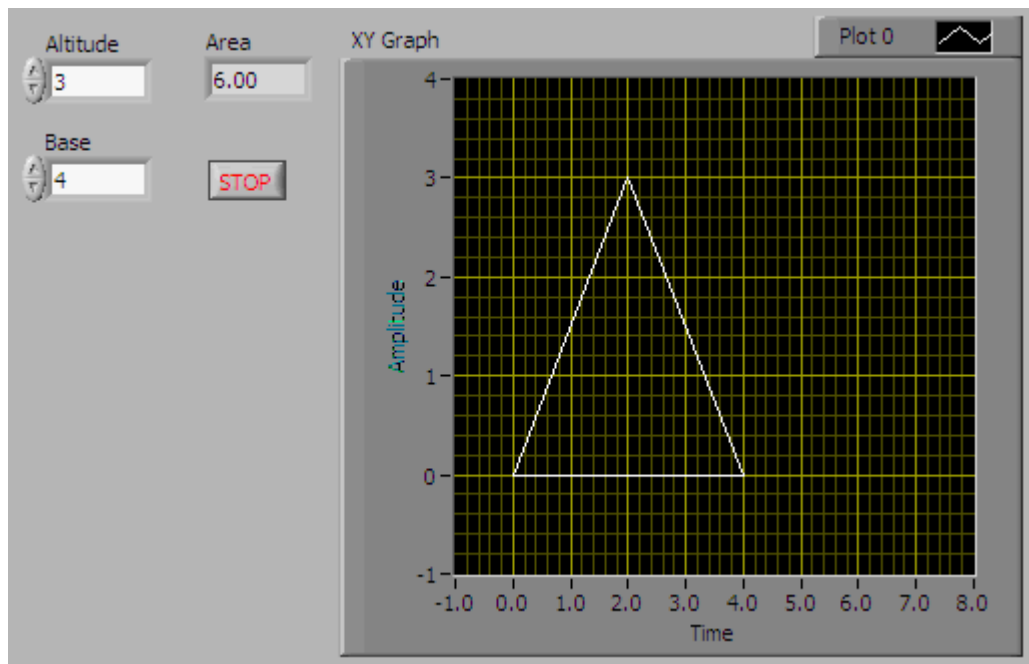


You can see in the picture above that we have included a timing function called *Wait Until Next ms Multiple* that stops the program execution until the internal computer time match as a multiple of the number given. In this case the program will stop every loop until a multiple of 10 ms is detected. This method will save us a lot of CPU processing time, because the program will not run completely continuous.

Now we can improve our vi including a graphical representation. To do this we suggest you to use the *XY graph*, that can be found in the *Graph* menu of the *Controls palette* in the Front panel window. We have created a sequence for a better understanding, including the formula created previously in the first frame and the graphic management in the second frame.



In the picture above you can see the schematic of the representation: the 4 X coordinates of the lines that compose the triangle are in the top array, and the 4 Y coordinates are in the bottom array, then we join them in a cluster, that is given as an input of the *XY graph*. Now let's see the front panel.



3. Programming with the LED Analyser

In this section we are going to learn how to use the LED Analyser Library for LabVIEW and how to develop LabVIEW VI's to communicate with it. We are going to learn methods to measure *color and intensity* from the LED Analyser (to capture and read back the color data).

Refer to the *LED Analyser User Manual* for a detailed description of the commands and functions available in the unit.

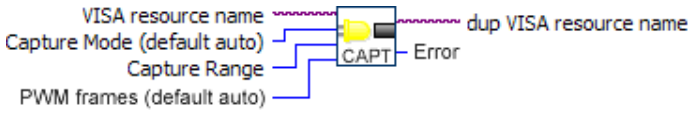
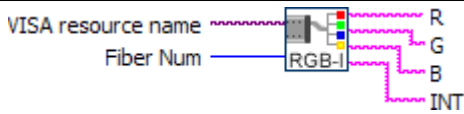
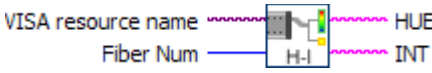
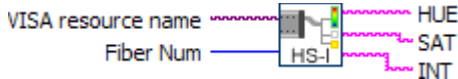
Before start to use the *LED Analyser Library* for LabVIEW, we suggest you to copy all files of the Library provided by Feasa Enterprises to a folder in your hard drive. Then try not to move this folder in order to preserve the consistence of the vi paths of all applications you are going to create.

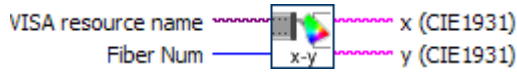
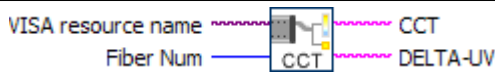
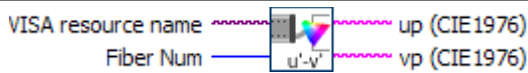
The LED Analyser Library for LabVIEW consist on different VI modules used to develop LED Analyser VI's easily. There are different types of functions depending on the purpose:

- Acquisition & Measure
- Communication control
- Tools

Acquisition & Measure

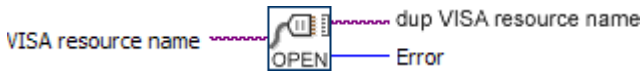
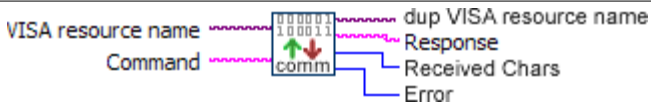
Here you can find a summary of all vi used to instruct the LED Analyser to Test LED's for Color and Intensity.

<p>Name: capture.vi</p> <p>Description: makes a capture for the LED Analyser with different capture modes: auto, manual, PWM. The inputs are a Visa resource name, the Capture Mode (0=auto, 1>manual, 2=PWM capture), the Capture Range (from 1 to 5) and the number of PWM frames (for PWM capture only). The outputs are a duplicated VISA resource and an error output (0=no error, 1=error).</p>	
<p>Name: getRGBInt.vi</p> <p>Description: read back the RGB Color and Intensity information of a fiber from the LED Analyser. The inputs are the VISA resource name and the number of the fiber to read. The outputs are strings with the read RGB and Intensity values.</p>	
<p>Name: getHueInt.vi</p> <p>Description: read back the RGB Color and Intensity information of a fiber from the LED Analyser. The inputs are the VISA resource name and the number of the fiber to read. The outputs are strings with the read Hue and Intensity values.</p>	
<p>Name: getHueSatInt.vi</p> <p>Description: read back the RGB Color and Intensity</p>	

information of a fiber from the LED Analyser. The inputs are the VISA resource name and the number of the fiber to read. The outputs are strings with the read Hue, Saturation and Intensity.	
<p>Name: getxy.vi</p> <p>Description: read back the Color Chromaticity values (CIE1931) of a fiber from the LED Analyser. The inputs are the VISA resource name and the number of the fiber to read. The outputs are strings with the read x-y values.</p>	
<p>Name: getCCT.vi</p> <p>Description: read back the Correlated Color Temperature information of a fiber from the LED Analyser. The inputs are the VISA resource name and the number of the fiber to read. The outputs are strings with the read CCT and the u-v distance.</p>	
<p>Name: getupvp.vi</p> <p>Description: read back the u'-v' Color information (CIE1976) of a fiber from the LED Analyser. The inputs are the VISA resource name and the number of the fiber to read. The outputs are strings with the read CCT and the u-v distance.</p>	

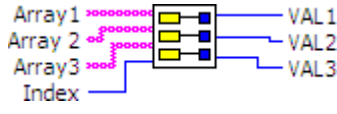
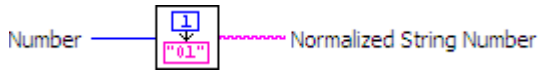
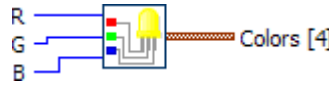
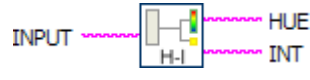
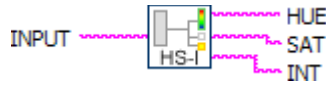

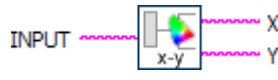
Communication control

Here you can find all VI modules used to establish a communication with the LED Analyser.

<p>Name: opencomm.vi</p> <p>Description: Executes all instructions to open the LED Analyser port properly. The input line is a VISA resource name var, where the user selects the input port. The outputs are a duplicated VISA resource (the same as input) and an error flag (0: no error, 1:error).</p>	
<p>Name: comm.vi</p> <p>Description: it is used to send commands to the LED Analyser. The input line is a VISA resource name that is the same that the one used to open the communication. The other input is the command to send to the LED Analyser (if the command does not contain the CR +LF characters, this VI append them to the command automatically). The outputs are a duplicated VISA resource, the response string from the LED Analyser and the received Chars.</p>	

Tools

Here you can find some useful tools that will help you format strings, arrays or retrieving and decoding data.

<p>Name: 3array2number.vi</p> <p>Description: given 3 array (i.e. RGB) this module will get a given index from them: if Index is 1 the output will be Array1[1], Array2[1] and Array3[1].</p>	
<p>Name: Num2NormString.vi</p> <p>Function: This vi converts a 1 or 2 digit number in the input to a 2 digit decimal string. This is useful to give the fiber number when you make loops to read back the color information of all the fibers from the LED Analyser.</p>	
<p>Name: RGB2Led.vi</p> <p>Function: This vi is used to set the Color property of a light indicator using the RGB components values. The inputs are the RGB numeric components and the output is a Color array property.</p>	
<p>Name: SplitHUEINT.vi</p> <p>Function: This vi extracts the Hue and the Intensity values from the string resulting from the execution of the command GETHSI## (where ## is a fiber/LED number from 01 to 20).</p>	
<p>Name: SplitHueSatInt.vi</p> <p>Function: This function extracts the Hue, Saturation and the Intensity values from the string resulting from the execution of the command GETHSI## (where ## is a fiber/LED number from 01 to 20).</p>	
<p>Name: SplitRGBINT.vi</p> <p>Function: This function extracts the RGB and Intensity values from the string resulting from the execution of command GETRGBI## (where ## is a fiber/LED number from 01 to 20).</p>	
<p>Name: SplitXY.vi</p> <p>Function: This function extract the XY values from the string resulting from the execution of the command GETXY## (where ## is a fiber/LED number from 01 to 20).</p>	

3.1 Program structure

The LED Analyser is a hardware device used to measure Color and Intensity, that has a port which is connected to the computer. This port can be RS232 or USB and it is used to establish a communication and retrieve all required data about LED's being tested.

When we want to develop a new LabVIEW vi to communicate with the LED Analyser we should follow some basic steps:

- Copy the library files to the same drive and path where the VI project is located.
- If you have a USB LED Analyser you have to connect the USB cable before running LabVIEW; Otherwise the port may not be detected in some versions of LabVIEW.

- The first necessary function to establish a communication with the LED Analyser is *Opencomm* (*opencomm.vi*), which is used to setup and open the communications port. In order to select the port is necessary to connect a *VISA resource name* constant or control to the *Opencomm module* function, then you can use this purple line or the duplicate VISA resource in the function box to connect the remaining functions.
- The second step is to use the function *Comm* (*comm.vi*) to send commands to the LED Analyser or use other functions like *Capture*, *getHUEINT*, etc. which sent the convenient command and provide the expected output already parsed.
- Once the communication process is finished it is necessary to *close the port* (*VISA close function*), before the VI or LabVIEW are closed, otherwise the port could be locked up and you may need to reboot the PC.
- All the LED Analyser Library modules that use a *VISA resource name* should share the same line, connecting together to the same resource or implementing a daisy chain through the VISA-duplicated VISA I/O. We suggest you use the last method and the reason of this is that LabVIEW will not execute the next function until the previous one has finished so if you want to execute a capture and then read back the color, you can connect the duplicated VISA output from the capture box to the VISA input of the next box, so LabVIEW will not execute the second box until the first one has finished.

3.2 Open/close ports


Open port

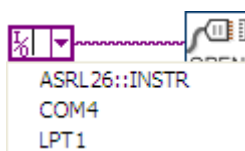
To establish a communication with the LED Analyser, a port must be opened. Import the “opencomm.vi” module which is used for this purpose.



Then create a *VISA resource name* that will provide a way to select the port of the LED Analyser (Functions Palette--> Instrument I/O--> Visa Resource Name Constant).



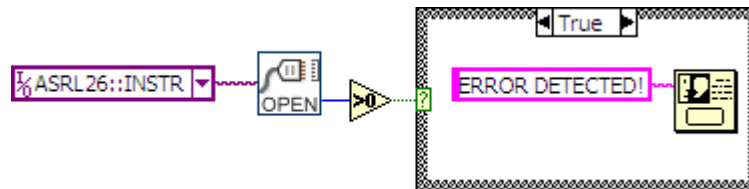
Wire the VISA constant/control to the *opencomm* module and then select the port to open using the *Operate value* tool from the tools window .



All physical serial ports are normally listed as COMxx and all USB LED Analysers or USB to RS232 converters are listed as ASRLxx::INSTR.

The opencomm module provides 2 outputs, a duplicated VISA name, that is internally wired

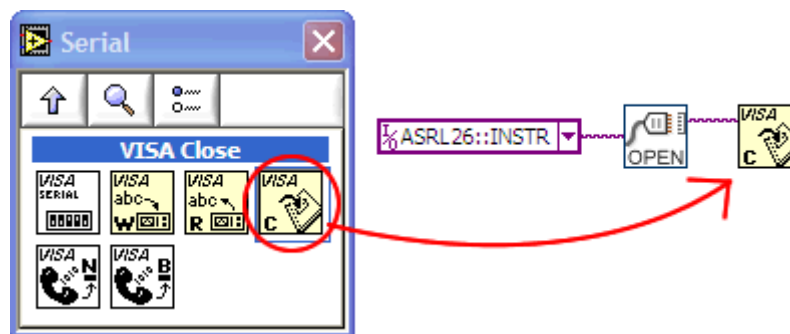
directly to the VISA input, and the other output is an Error output. The *Error output* is numeric and it gives a “0” when no error is detected or “1” when an error has been detected. In the schematic below we have implemented a simple error handling method that shows a dialog when an error occurs.



Close port

It is always necessary to close the communication when the program finishes or the execution is stopped, otherwise could be impossible to open the port again until you restart LabVIEW or even the PC.

To close the port you need a function called *VISA close* that you can find in the *Functions palette* in Instrument I/O --> Serial.

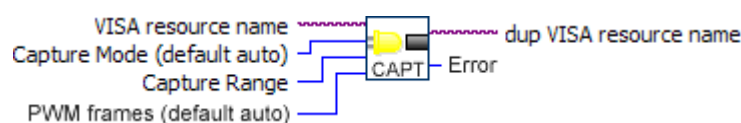


In the picture above we made a vi that open the port “ASRL26” and then closes it.

If you are going to execute a sequential process, we suggest you to use a sequence structure: put the *VISA resource name* outside the structure and connect it to the structure using a tunnel, place the *opencomm* function in the first frame, your desired actions in the following frames and then close the port in the last frame.

3.3 Capturing LED Color and Intensity data

After the port has been opened, it's necessary to execute a capture to read back valid LED data. When the capture command is executed, the LED Analyser will read and store the Color and Intensity data of each LED in a fiber. To use the *capture* module you have to import the file “capture.vi”.



This box has 4 inputs and 2 outputs:

- Outputs: Duplicated Visa resource name, Capture Error line (0: no error, 1: error).
- Inputs: Visa resource name, Capture Mode, Capture Range and PWM frames (for PWM capture only).

The Visa resource name comes from the open port function or other previous communication functions.

The *Capture Mode* is an input used to set the type of capture that is going to be executed in the LED Analyser: Auto (0), Manual (1) and PWM (2).

The *Capture Range* input is used only in the Manual and PWM modes and allow the user to input the range (numeric) that will be used to take the measurement, depending on the intensity of the LED under test: Low (1), Medium (2), High (3), Super (4) and Ultra (5).

PWM frames is an input used to specify the number of frames taken for averaging when it's being measured an PWM modulated LED. If this parameter is not given, then the averaging factor will be adjusted automatically.

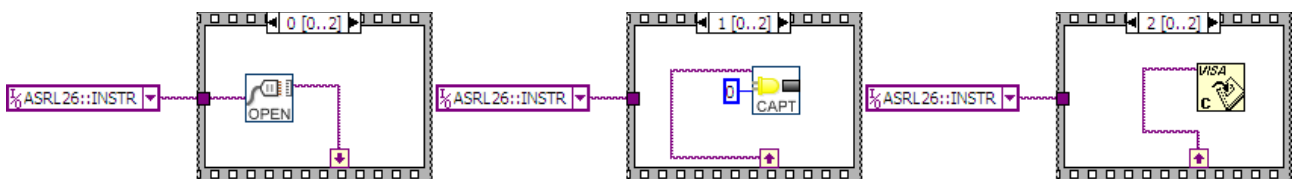
When PWM capture mode is used, if Capture Range is not specified, an Auto-range Capture will be triggered.

Example 3.3.a: Auto capture

To execute an *auto-capture* you need to open the port, then to execute the capture and finally close the port.



You can also implement this schematic using a sequence structure with 3 frames: one frame to open the communications, the next one to do the capture, and the last one to close the communications.



3.4 Reading color & Intensity

After the capture has been executed the results are stored in the LED Analyser memory. The next step, is to read back valid data about the Color and Intensity data from the LED Analyser through the opened port. This data is provided in different formats:

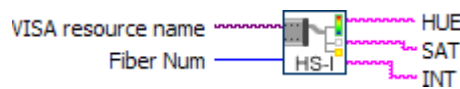
- RGB
- Hue

- Saturation
- x-y (CIE1931)
- u'-v' (CIE1976)
- Relative Intensity
- Absolute Intensity (Feasa LED Spectrometer needed to adjust the values)
- CCT (Correlated color temperature) & u-v distance

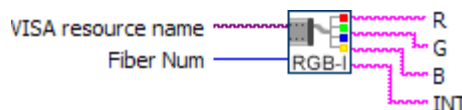
Refer to the *User Manual* for a detailed description of these commands.

There are different functions available to access to this data:

- “**getHueSatInt.vi**”: used to read back the Hue, Saturation and Intensity of a fiber



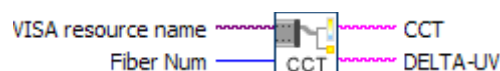
- “**getRGBInt.vi**”: used to read back the RGB color components and Intensity of a fiber



- “**getxy.vi**”: used to read back the chromaticity x-y components (CIE1931) of a fiber



- “**getCCT.vi**”: used to read back the Correlated Color Temperature of a fiber



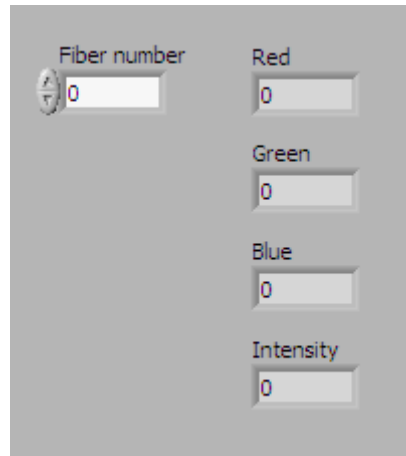
- “**getupvp.vi**”: used to read back the u'-v' color components (CIE1976) of a fiber



Note that all function boxes have the same inputs but different outputs. The inputs are always the *VISA resource name* and the *Number of the Fiber* which is in the range 1 to 20. The outputs are the data read from that fiber formatted as string (ASCII characters), so if you want to get a numeric value you have to use the *String/Number Conversion* functions that can be found in the *Strings menu* of the *Functions palette*.

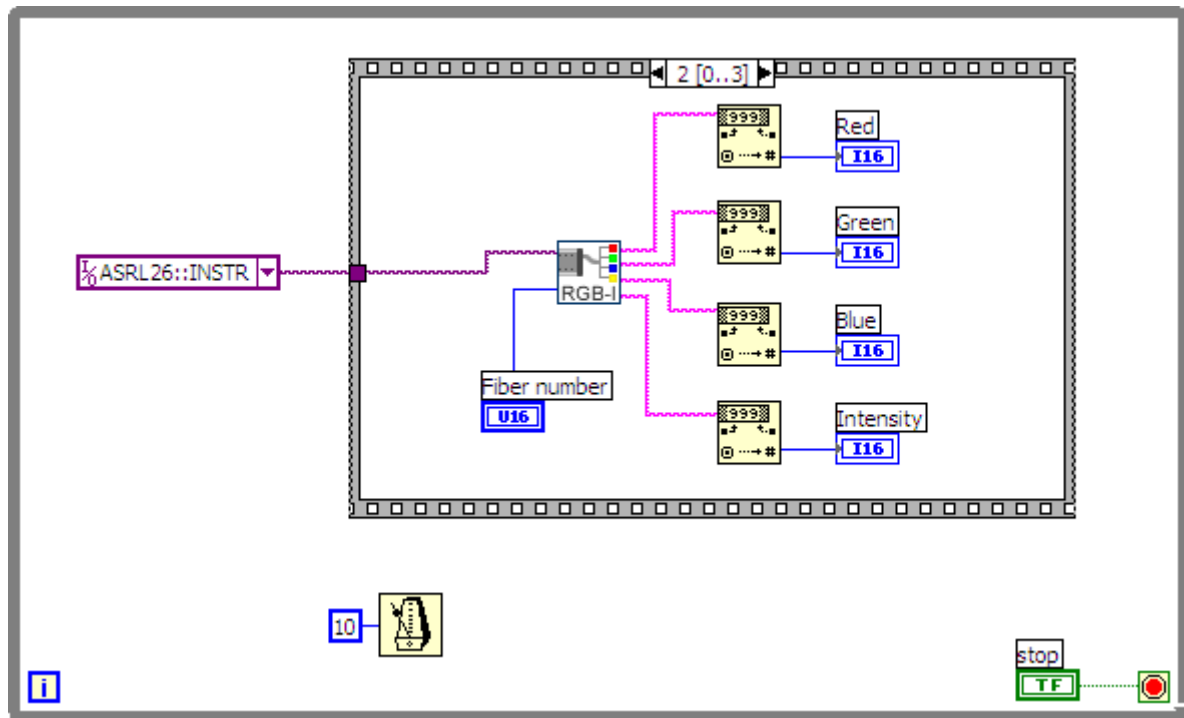
Example 3.4.a: Auto capture & RGB read

To implement this example you need to create a digital control and four digital indicators in the front panel window. Put the name “Fiber number” to the digital control and “Red”, “Green”, “Blue” and “Intensity” to the digital indicators. You can also create a *Stop* button to put the schematic into a while loop and run the vi continuously.



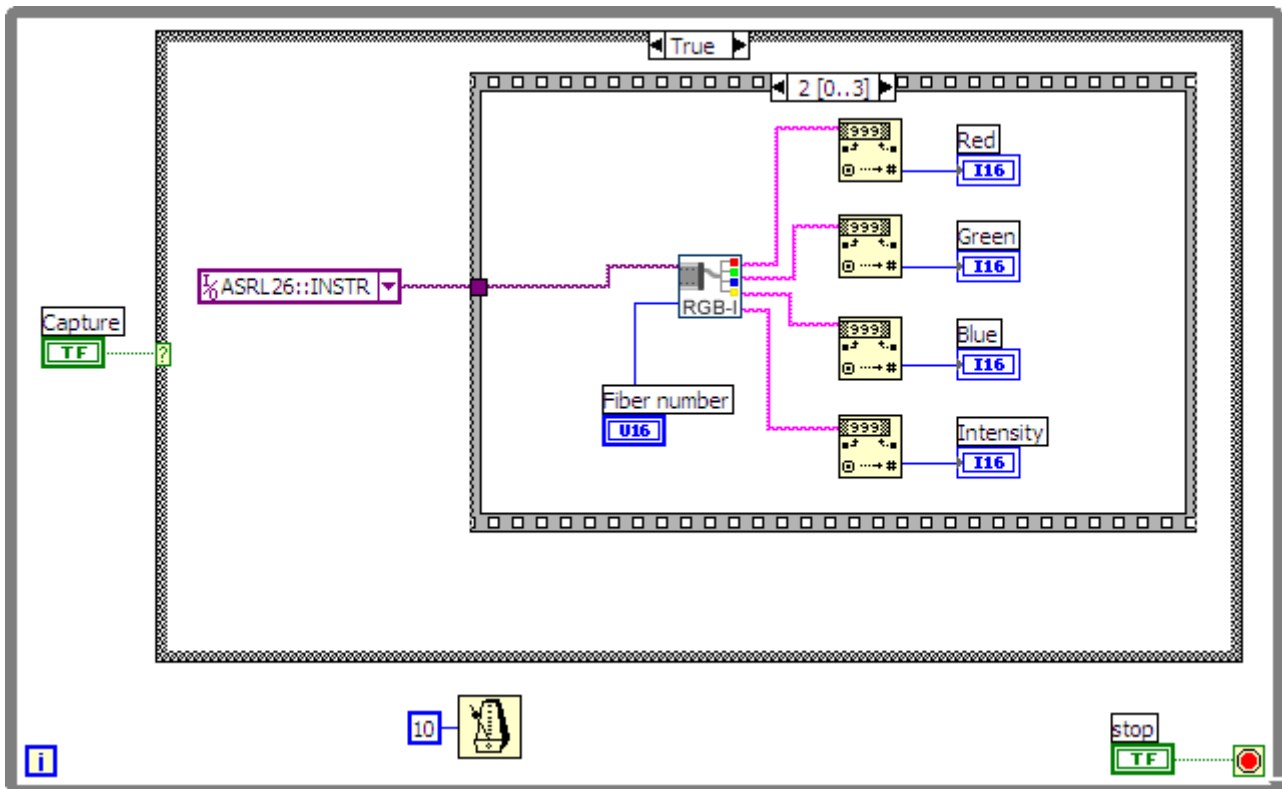
Remember to change the representation of the control to U16 and the representation of the indicators to I16. Now go to the schematics and place an *Open comm* function a *Capture* function a *GetRGBInt* function and a close function. The Fiber number control (digital control) is wired to the *Fiber number* input of the *GetRGBInt* function.

The structure selected to implement the schematic is a main while loop with a sequence (open port + capture + read + close port).



Remember to include a *Wait until next ms Multiple* function to save CPU processing time.

The first frame in the sequence contains the *Opencomm* function, the second one contains the *Capture* function, the third one contains the read commands you can see in the picture above and the last frame contains the *Close* communications function. This program will capture and read continuously so if you want to include a button to capture when you click on it, you can place an *OK* button in the *Front panel* window and a *Case* structure inside the while loop in the schematics window. You have to put the main code inside the True case of the Case structure and wire the condition of the case to the OK button.

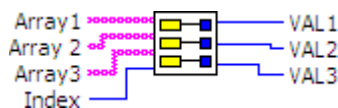


3.5 Tools

In the LED Analyser Library for LabVIEW you can find some different tools that can be useful when developing your own vi.

3array2number.vi

This function is used to extract an index value from 3 arrays simultaneously. This means that we

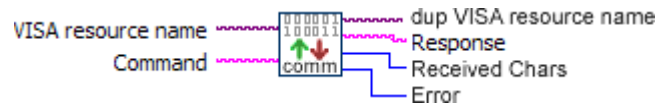


have an input called Index, in which we give the index of the array value we want to get, so if Index=3 the outputs will be Array1[3], Array2[3], Array3[3]. Index starts from 0 to the number of elements -1 (if the array has 5 values the maximum index is 4).

As an example it can be used to get the RGB values of a fiber from the RGB arrays.

comm.vi

This function is used to communicate with the LED Analyser to send commands and receive the response. See the *User Manual* for a detailed description of all the commands available.



In the picture above you can see that there are different inputs and outputs. The inputs are the *VISA resource name* of the port through which data is transmitted, the *Command* you want to send and the *Number of characters* of the response. In the outputs you have *duplicated VISA resource name*, a *Response* string, a count with the number of *Characters received* and a numeric error flag (0: no error, 1 error).

Remember that you must open the port using the function *opencomm.vi* before send any command, otherwise you will get an error. Let's see an example.

Note: when communicating to low-level with the LED Analyser, characters CR+LF (ASCII 13 + 10) need to be sent at the end of the command to validate the command (to say to the LED Analyser that the command has been entered), however in this module there is no need to do it because it's done automatically inside of *comm.vi*. You can optionally include the CR + LF characters and, in this case, the module won't add them internally.

Example 3.5.a: Capture

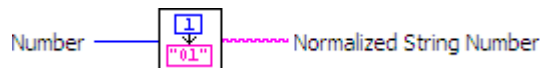
In this example we are going to send the command “CAPTURE” or “C” (equivalent) to the LED Analyser. Once the capture is executed, the LED Analyser will return “OK” so the number of received characters will be 4 (consider CR + LF too), but *Comm.vi* filters control characters internally and only the string “OK” will be available at the output. In this example, the outputs *Error* and Number of received characters are not used.



Remember to select the proper port before executing the vi.

Num2NormString.vi

This function is used to convert a 1 or 2 number value to a 2-digit fixed character string value (adding a 0 as a first digit if it's necessary). This function is normally used to compose serial commands for the LED Analyser that need the fiber number to be specified.



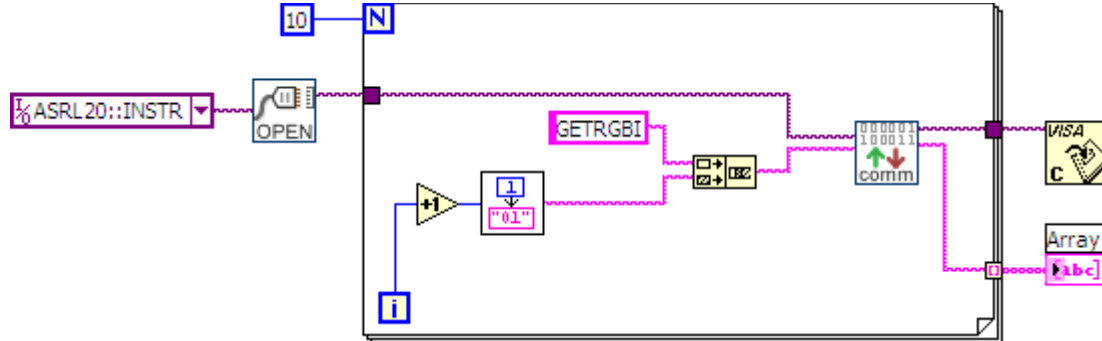
For example:

- The number 1 will be converted to the string “01”
- The number 12 will be converted to the string “12”

Let's see how to compose a command using this function.

Example 3.5.b: Read back color & Intensity

In this example we are going to build a loop that will retrieve the data for Color and Intensity of 10 fibers. The schematic is shown in the picture below.



In this schematic you can see the function box *Num2NormString*, used to compose a read command “GETRGBIXX”, in which XX is the number of the fiber to read. The number of fiber to read is provided by the loop count that is passed to an *Add 1* function because the number provided starts by 0. Then this number converted to ASCII and added to the command string that composed using a *Concatenate Strings* function. Then the command is given to the *Comm* function that will send it through the port to the LED Analyser. The result is represented by an Array of string indicators.

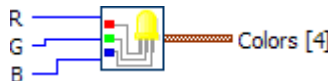
SplitHUEINT.vi, SplitHueSatInt.vi, SplitRGBINT.vi, SplitXY.vi

When you use the Comm function to send a command to get the data for Color and Intensity from the LED Analyser you will get a string with all information enclosed in it. You can easily access to this information using the functions of this section, that will split the information for you.

- SplitHueSatInt.vi Used to split the data provided by the command “GETHSIxx”
- SplitRGBINT.vi Used to split the data provided by the command “GETRGBIxx”
- SplitXY.vi Used to split the data provided by the command “GETXYxx” or “GETUVxx”

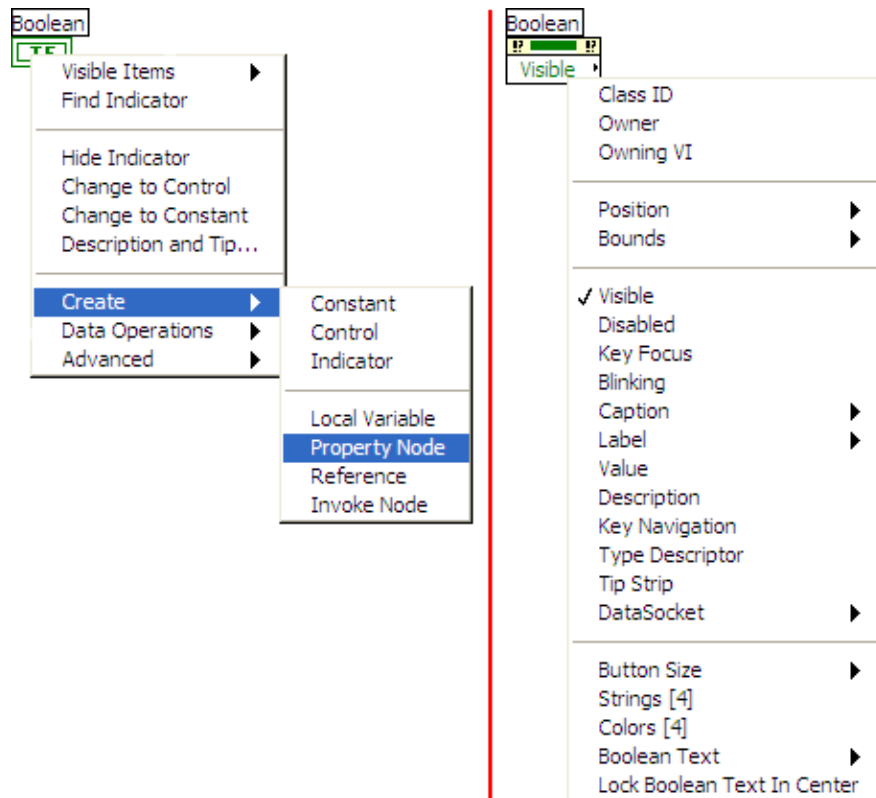
RGB2Led.vi

This function is used to change the *Color property* of a LED using the 3 RGB components as parameters. So you can give the 3 RGB components of the Color you want to be represented in the LED indicator and it will light with the color given.



You should notice that the output is not a direct line connected to the LED, because the input of the LED indicator is boolean and we provide a property value, so you will need to create a property node to the LED to transmit the parameters.

To create a property node you have to right-click in the LED box of the *schematics* window and then select the option “Create --> Property Node” from the pop-up menu.

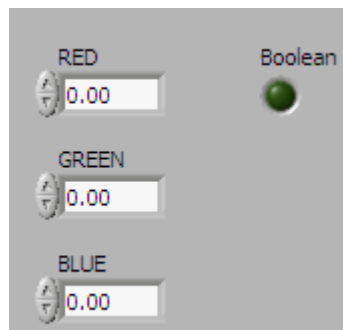


Once the property node is created select the desired property clicking on it (using the *Operate Value* tool from the *Tools Palette*).

You can change the operate mode of the property by right-clicking in the property element and selecting the options *Change To Read/Change to Write*.

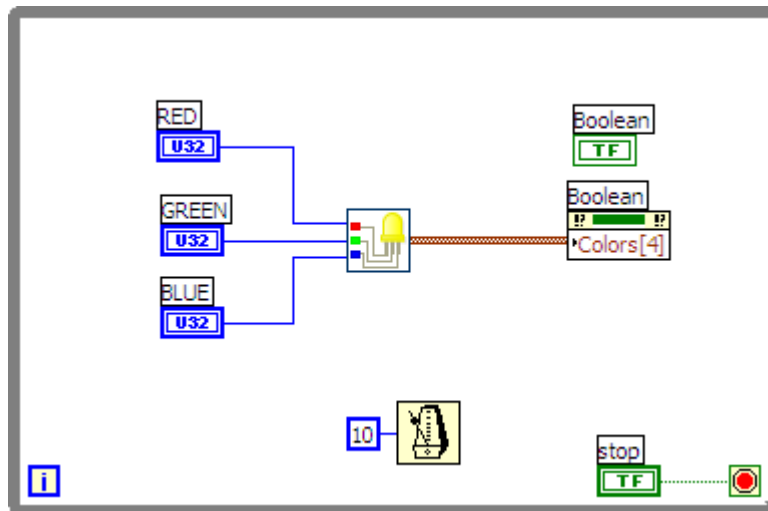
Example 3.5.c: Setting a color of an LED

In this example you have to create three Numeric controls, a LED indicator and a Stop button in the Front Panel Window. Name the controls as RED, GREEN and BLUE.



Now add a *RGB2Led* function and place it with all controls and the LED inside a while loop. Create a property node for the LED indicator and change the property to “Colors [4]”.

After this, wire the numeric controls to the *RGB2Led* function and the output to the property node.



If you execute the program you will be able to change the color of the LED by changing the numeric values of the 3 RGB components through the Numeric controls.



3.6 My first vi

Once you have learned how to use the different functions of the LED Analyser Library for LabVIEW you can already develop your own applications to work with the LED Analyser. So in this section we will propose you an exercise to put in practice all you have learned but without giving you the complete solution step by step. Instead of this we will give you different targets to get and you should be able to do them and after this, compare them with the provided schematics to check whether your progress is good or not.

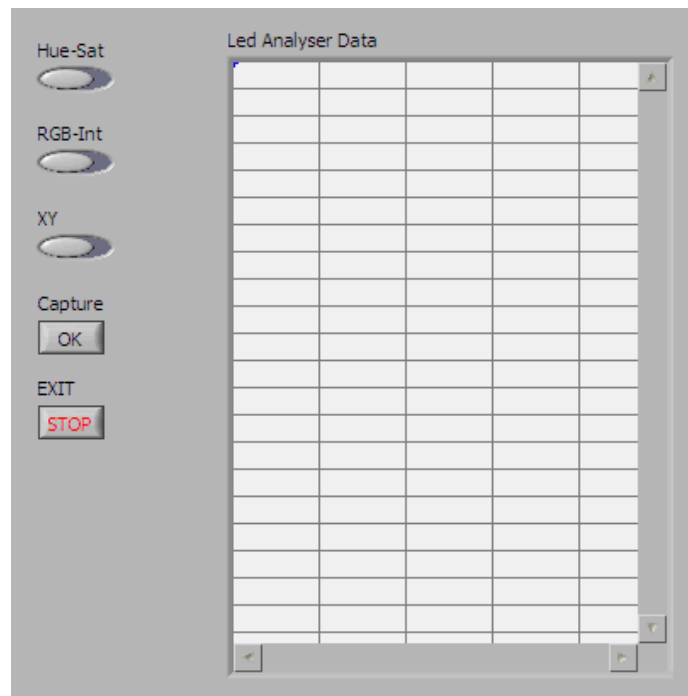
Let's start to design...

Definition: This program will execute a capture in the LED Analyser and will read back the color and intensity data. The information will be represented in a grid and we will be able to enable or disable the acquisition of some data types.

Objective No 1: design the front panel.

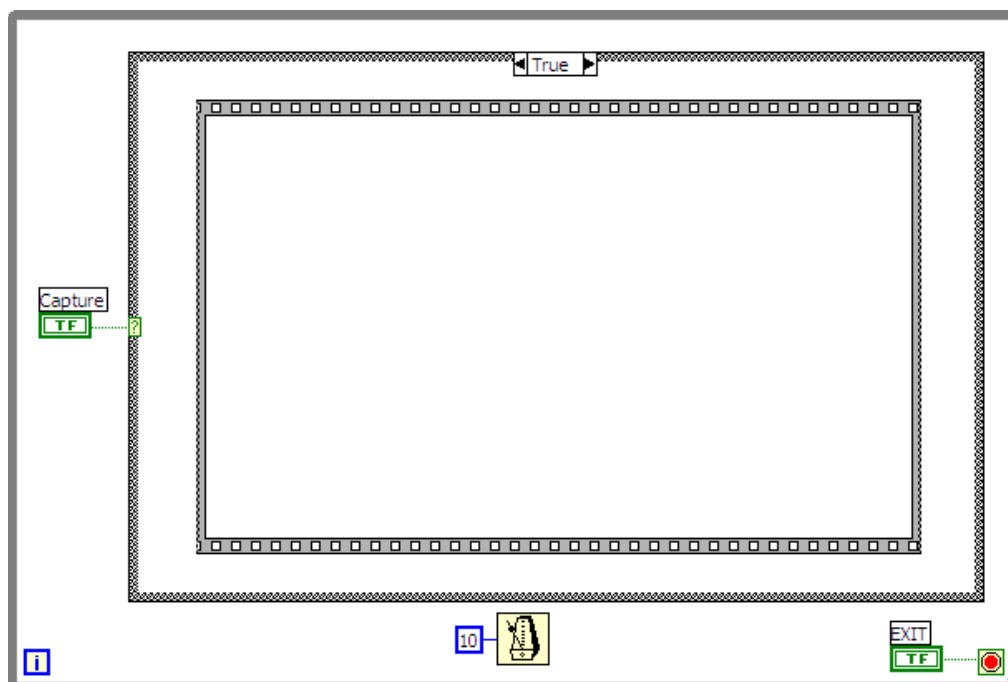
The front panel window should contain a *Table*, a *Stop* button, 3 *Slide switch* to enable/disable the data types and an *OK* button. We suggest you to put names in the controls and indicators to

recognize them better.



Objective No 2: schematics, main structure.

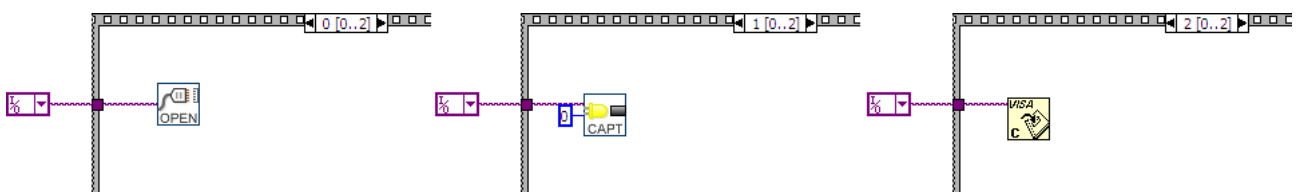
At this point you have to make a timer controlled main loop and a structure to control the execution of the capture. Then check your designed schematic with the solution printed in the picture below.



In the picture you can see a main while loop that provides a continuous execution of the main loop connected to the Exit/Stop button. Inside the while loop there is a *case* structure that is connected to the Capture button. When the Capture button is pressed, the *True* case will be executed so we have to put the Capture code inside.

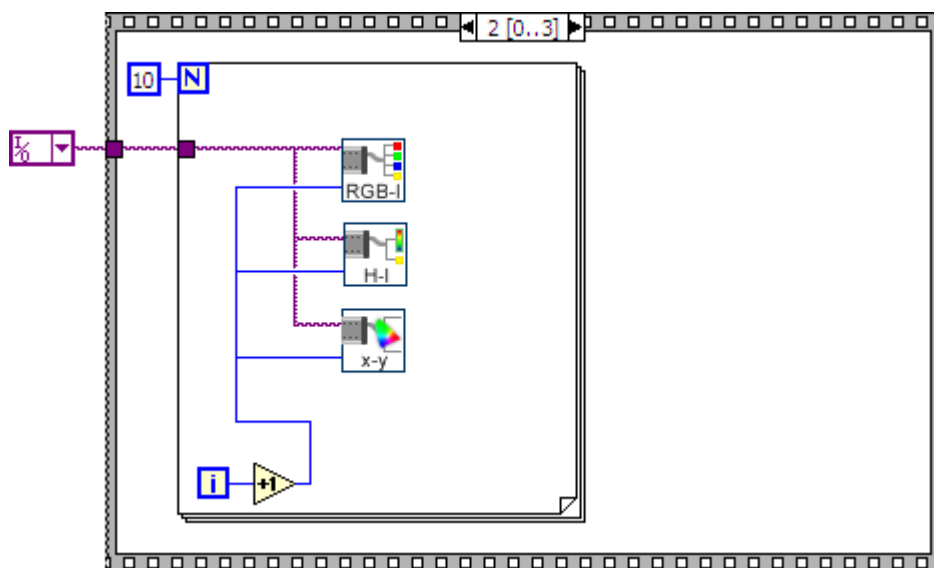
Objective No 3: capture sequence.

Now you have to create the Capture sequence (auto by default), but not the Read process yet. Remember to change the VISA resource name variable in order to select the proper port of the LED Analyser.



Objective No 4: read sequence.

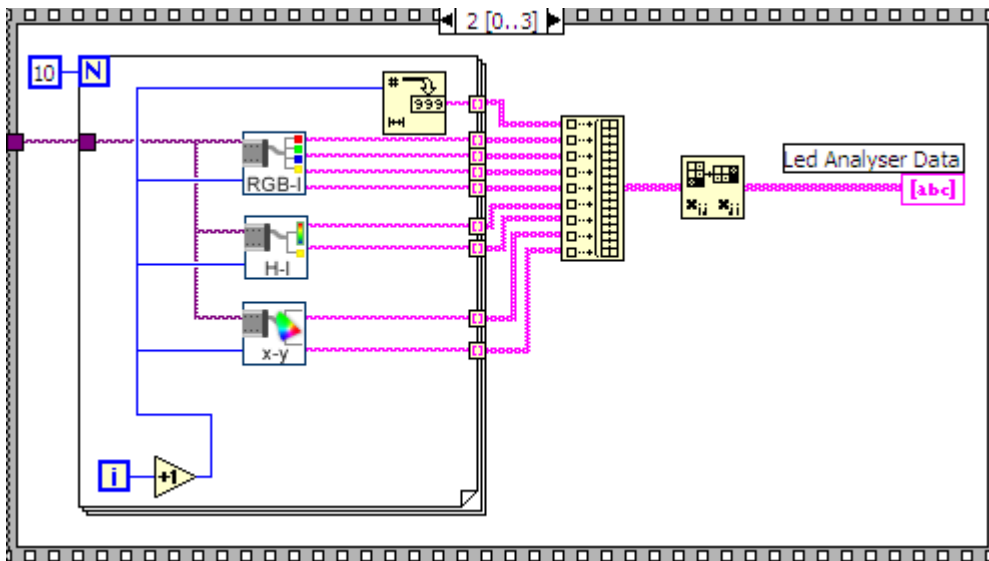
Now we will include a frame in the sequence, between the frame 1 and 2, after the capture to execute the acquisition process. We will only read 10 values (if your device has 10 or more fibers).



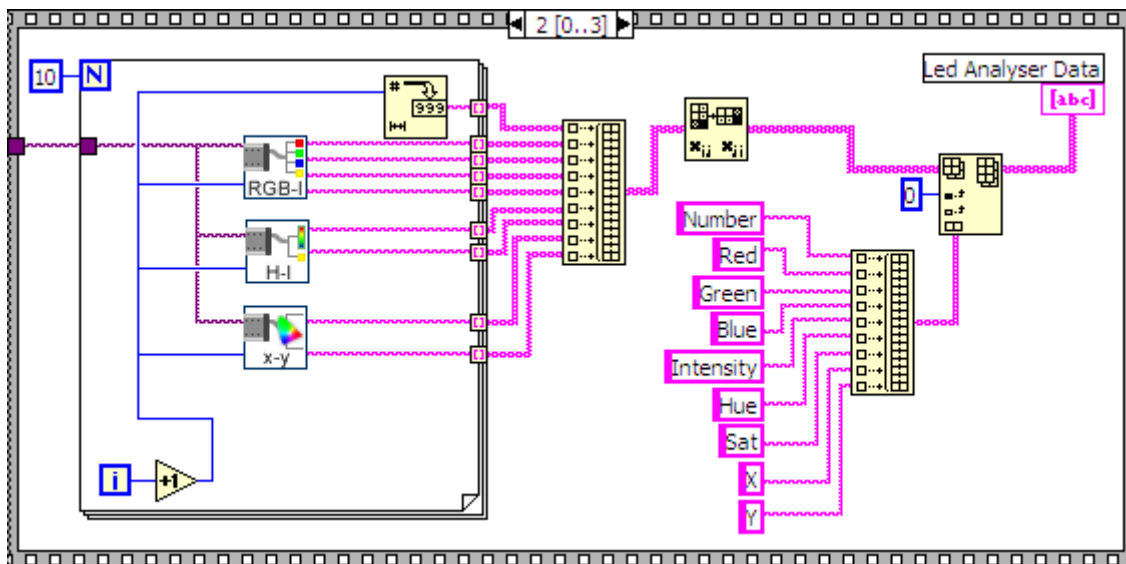
Objective No 5: Representation.

To print all read values to the table and the number index, first we have to build indexed tunnels in the for loop and then join all data into an array (build array tool). What you are really doing is building an array of arrays, this means a matrix. The problem with the data is that it will be stored in rows and we want to print it in columns, so you have to transpose the matrix (10 rows and 9 columns).

You can see a schematic of the representation process in the next picture.



The problem of this schematic is that if we execute it, the data in the table could be confusing because there are no labels. So the next step is to insert labels in the first row of the table (you can use the functions *Build array* and *Insert into array*).

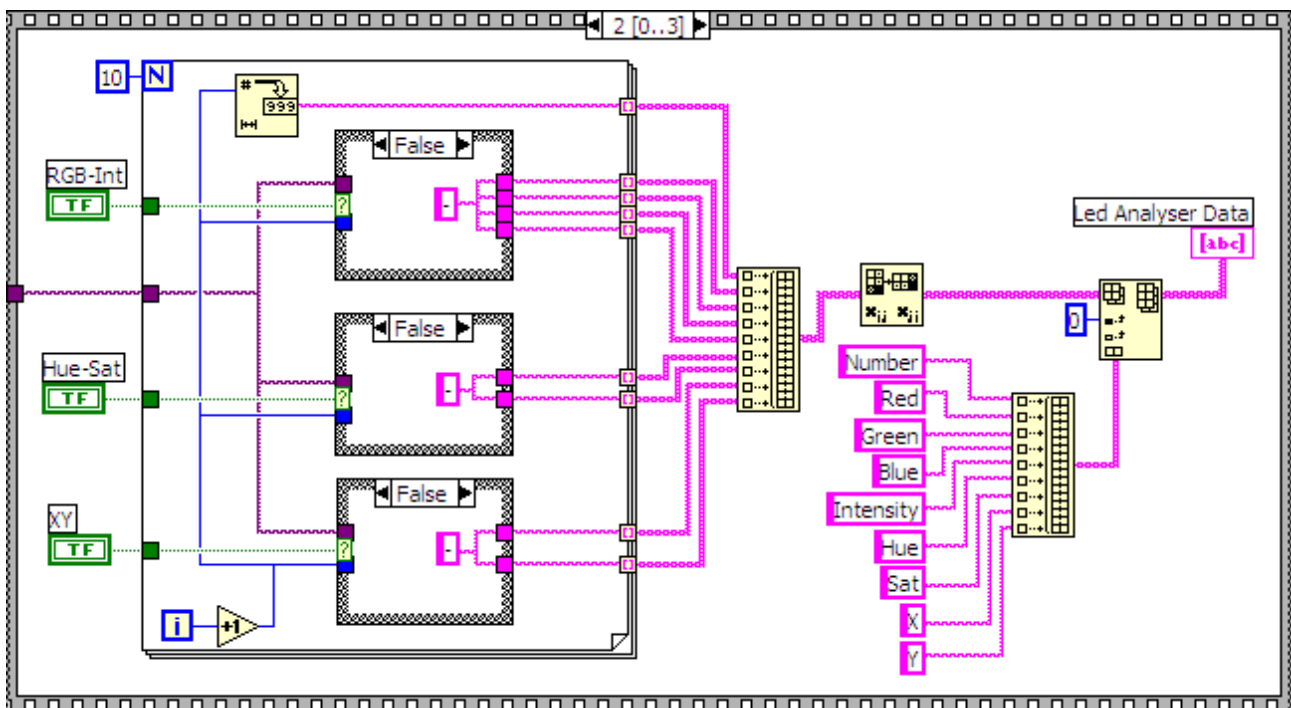
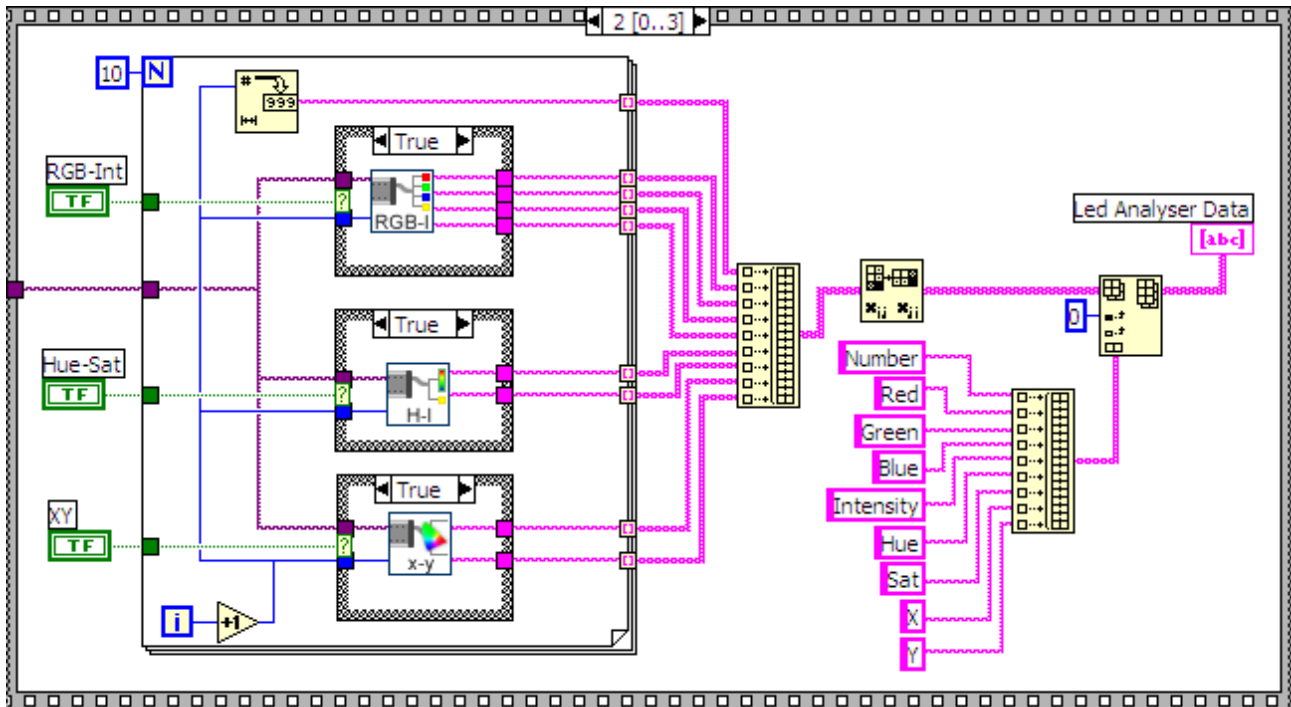


Now you can execute it from the Front panel and see the results of the capture & read process.

Objective No 6: Working with the switches.

At this point we have already developed an application able to capture and read back all measured values from the LED Analyser but we have not used the switches yet.

The requirements to work with the switches is that when the status of a switch is True the data corresponding to that switch will be read but if the status is False the data will be ignored. The case sequences can help you to do it.

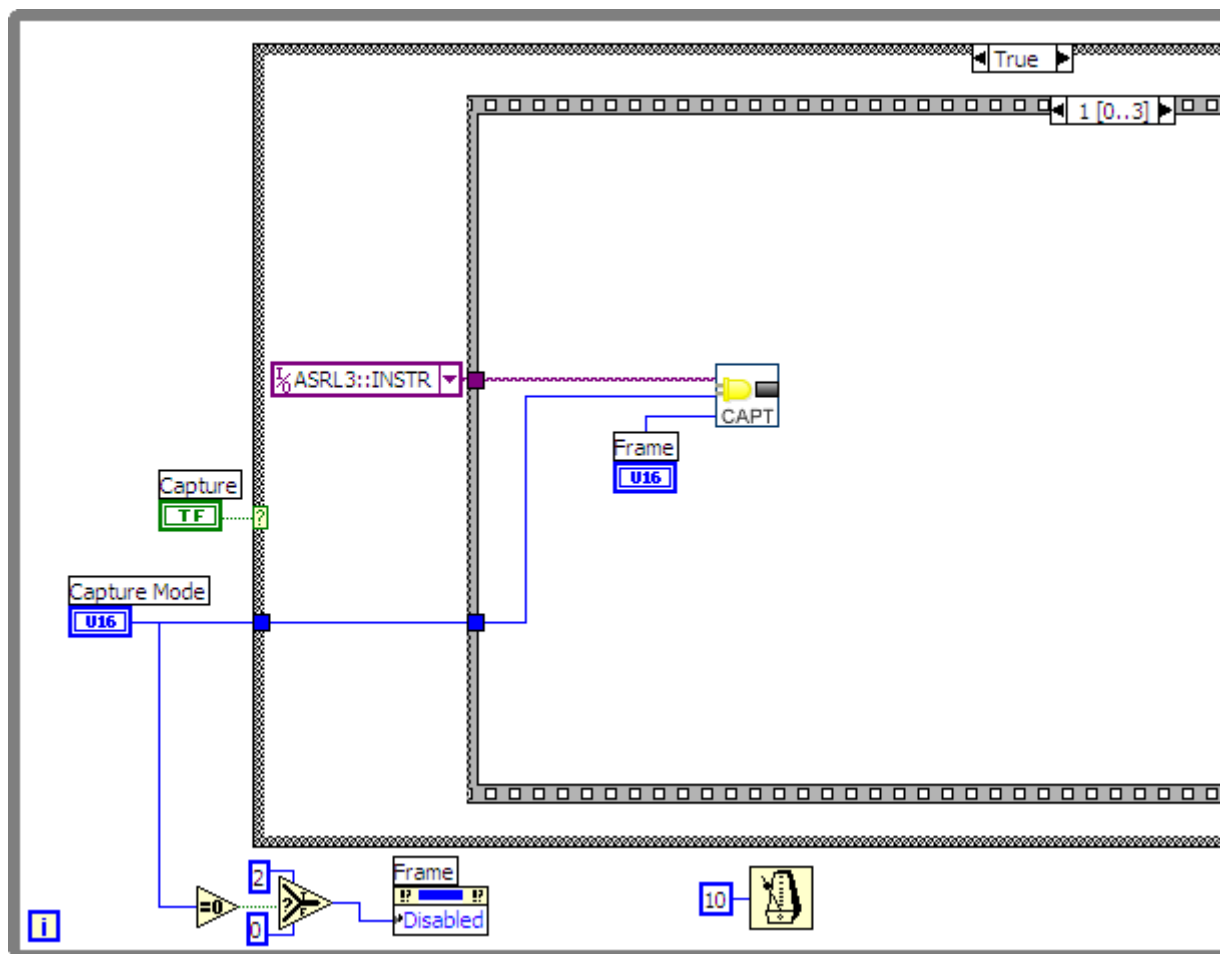


The true and false switches have been placed outside the *for loop* to avoid changes when the loop is running. Notice that we have placed a *dash* character inside the False case)case structure that

enables or disables the data acquisition) to indicate that a Data type is disabled.

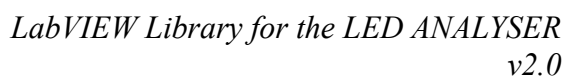
Objective No 7: Capture mode.

After finish a basic capture & read application you could add and option to select the capture mode in the front panel window. You can add a Ring Menu control to select the capture mode and a Numeric Control to select the Range for the Manual capture mode or the number of frames for the PWM capture mode. As an additional improvement we will ask you to limit the numbers of the Numeric indicator from 0 to 8 and to gray out the numeric indicator when not used (use properties). As a help we will tell you that the gray out functions should be inside the main while loop. You suggest you to use decorations in the front panel in order to clarify the screen and group items.



What we have done is to check the Capture mode Ring Menu and gray out (disabled=2) the *Frame Numeric control*.

So now let's take a look to the final front panel window.

Page 60 of 61



Release date: 6 November 2007
Current version: 8th May 2014
Author: Carlos Martínez Rius
Copyright Feasa Enterprises Ltd.
Holland Road
National Technology Park
Castletroy (Co. Limerick)
Ireland

<http://www.feasa.ie/>
sales@feasa.ie