

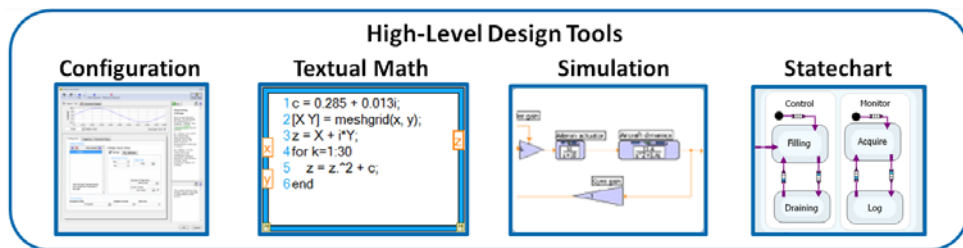


Høgskolen i Telemark

Telemark University College

Department of Electrical Engineering, Information Technology and Cybernetics

LabVIEW Programming Examples



LabVIEW

Graphical Programming



Hans-Petter Halvorsen





Høgskolen i Telemark

Telemark University College

Department of Electrical Engineering, Information Technology and Cybernetics

Table of Contents

1 - Customizing the LabVIEW Environment

2 - Wires and Variables

3 - Strings

4 - Arrays

5 - SubVIs

6 - Clusters

7 - Formula Node

8 - Debugging

9 - Project Explorer

10 - State Machine

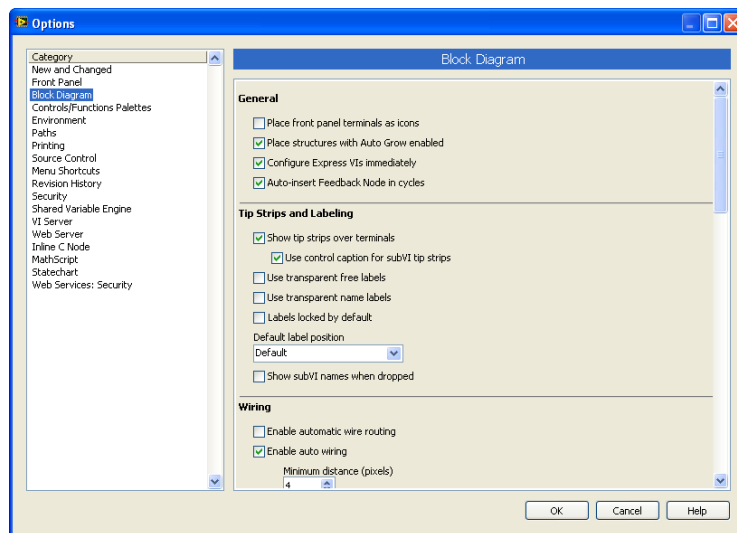




Customizing the LabVIEW Environment

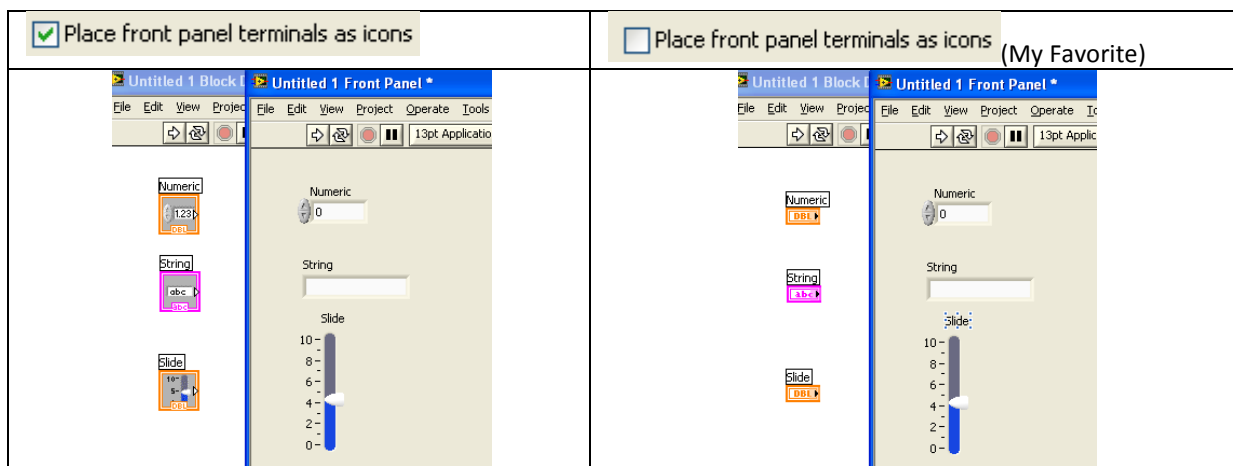
Description: LabVIEW has lots of possibilities for customizing the appearance and the use of the LabVIEW environment. Select “Options...” from the Tools menu.

Requirements: LabVIEW 2009



Task: In this example you will customize the LabVIEW Environment so it best fits your demands. The example will go through the most important settings in the Options window (Select “Options...” from the Tools menu). The default settings is not necessary the best, here are some recommendations for setting up the LabVIEW environment. Try the different settings above and see the difference and make your own personal choice.

Setting: “Place front panel terminals as icons” (Category: Block Diagram – General)



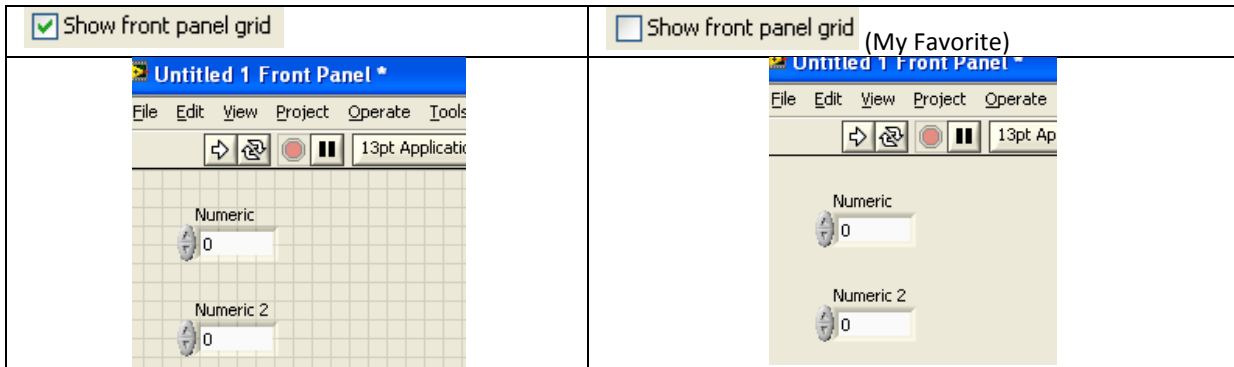
Comments: The setting to the right is my personal (and recommended) favorite. For LabVIEW beginner is the setting to the left easier to understand, but it takes too much space in the Block Diagram!

Setting: “**Enable automatic wire routing**” (Category: Block Diagram – Wiring)



Comments: This prevents LabVIEW from automatically connecting adjacent blocks. The setting to the right is my personal (and recommended) favorite. When you use the setting to the right you have more control and you may easily switch between the tools using the **Tab** key. Although it seems useful to have auto wiring enabled, it is my experience that the auto wiring is a little annoying since it tends to draw wires between blocks when you do not want any wire, etc.

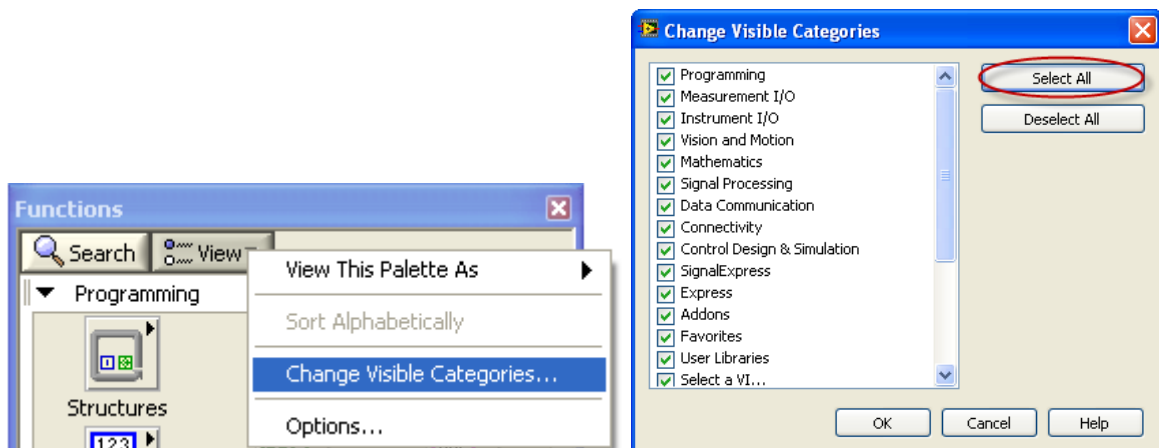
Setting: “**Show front panel grid**” (Category: Front Panel – Front Panel Grid)



Comments: This setting shows a Grid pattern on the Front Panel in “Edit mode”. I think this setting is distracting, but that is my opinion. Note! You may set the same setting for the Block Diagram.

Setting: “**Change Visible Categories**”

The next setting is not located in the Options window, but I think it is worth mentioning in this context. In the Functions or Controls palette, you may select which Categories that should be visible. I recommend that you “Select All”.





Høgskolen i Telemark

Telemark University College

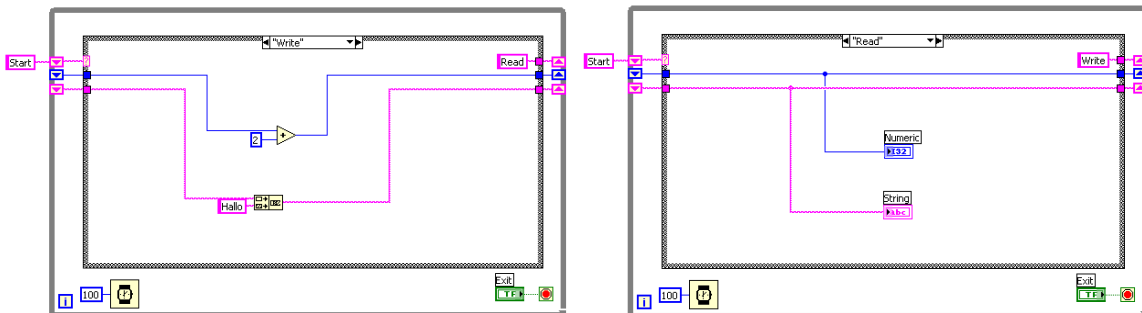
Department of Electrical Engineering, Information Technology and Cybernetics

LabVIEW Wires and Variables Example

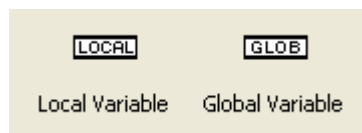
Description: In text-based programming languages, you store and access data with functions through the use of variables. In the LabVIEW graphical programming language, wires implicitly handle all of the data storage and access that are associated with variables in text-based languages. Think of wires as a path for data to flow. Data comes into block diagram objects through a wire and can leave only through a wire. Local (or Global) Variables are used to pass data when a wire in some situations cannot be used.

Requirements: LabVIEW 2009

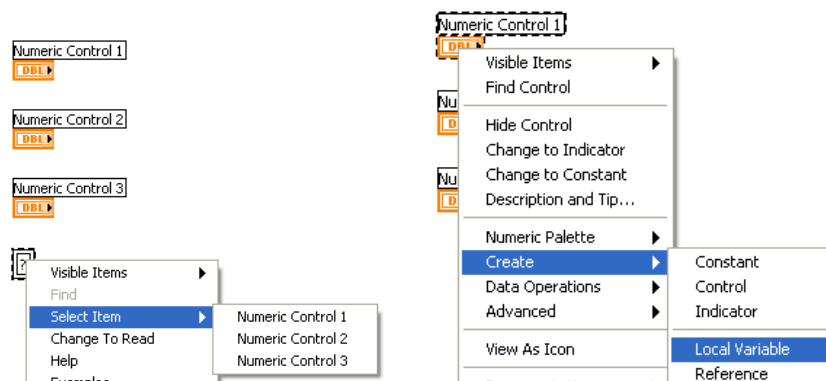
Task: Wires: Create a program where you use **Wires** and **Shift Registers** to update data as shown below.



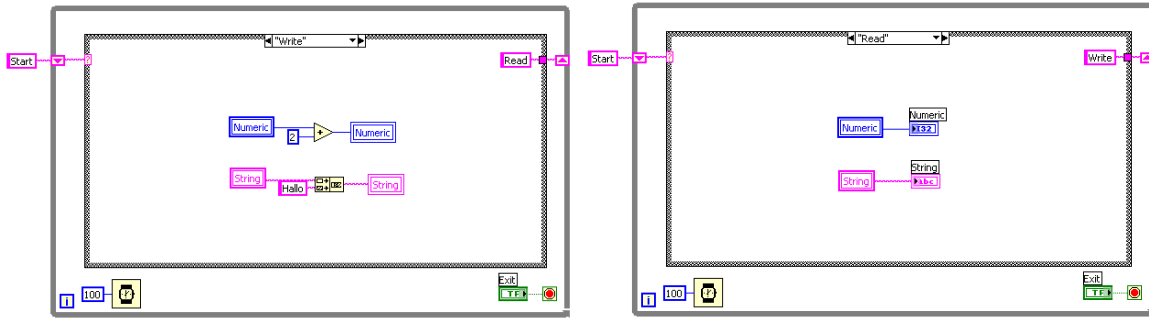
Task: Local Variables: The Local Variable item is located on the Structures palette on the Block Diagram.




When you place a Local variable on the Block Diagram, it looks like a Question mark as seen below. Then you right-click on the Local variable and choose "Select Item" and select which Control/Indicator you want to connect it to. Another way to create a local variable is to right-click on a Control/Indicator either on the Front Panel or the Block Diagram and select "Create → Local Variable".



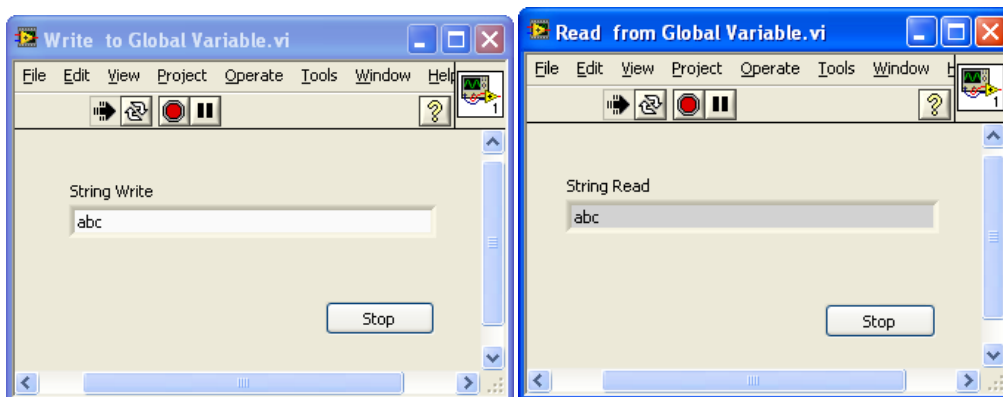
Task: Create the same program as in the previous task and use Local Variables instead.



Task: Global Variables:  Use global variables to access and pass data among several VIs. When you create a global variable, LabVIEW automatically creates a special global VI, which has a front panel but no block diagram. The Global Variable item is located on the Structures palette on the Block Diagram. When you place a Local variable on the Block Diagram, it looks like a Question mark with a globe, as seen above. Double-click on the item in order to create the Global Variable.



Task: Create 2 VIs that uses a Global variable to exchange data between them. Example:





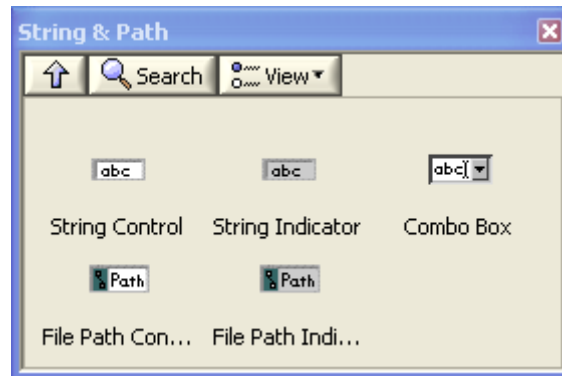
Høgskolen i Telemark

Telemark University College

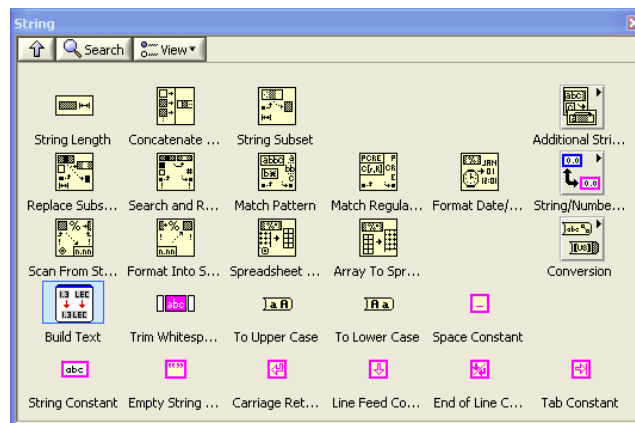
Department of Electrical Engineering, Information Technology and Cybernetics

LabVIEW String Example

Description: Working and manipulating with strings is an important part in LabVIEW development. On the Front panel we have the following String controls and indicators available from the Control palette:



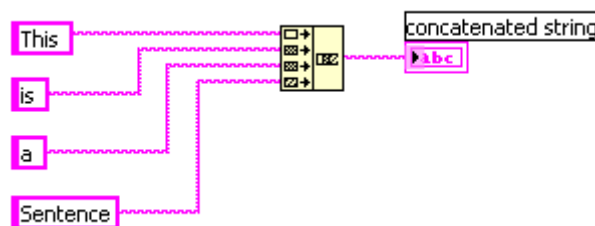
On the Block Diagram we have the following String palette available from the Functions palette in LabVIEW:



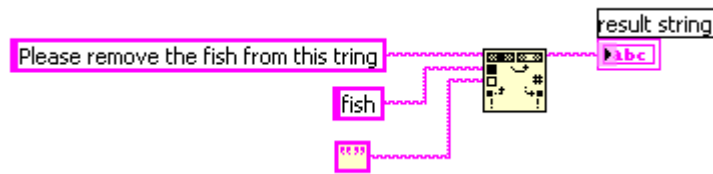
In this Example you will learn how to use strings and string manipulation in LabVIEW.

Requirements: LabVIEW 2009

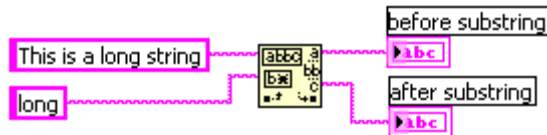
Task: Concatenate Strings. Create the following:



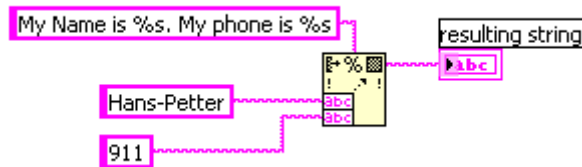
Task: Search and Replace String. Create the following:



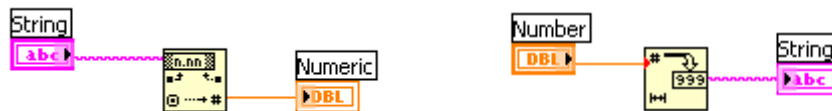
Task: Match Pattern. Create the following:



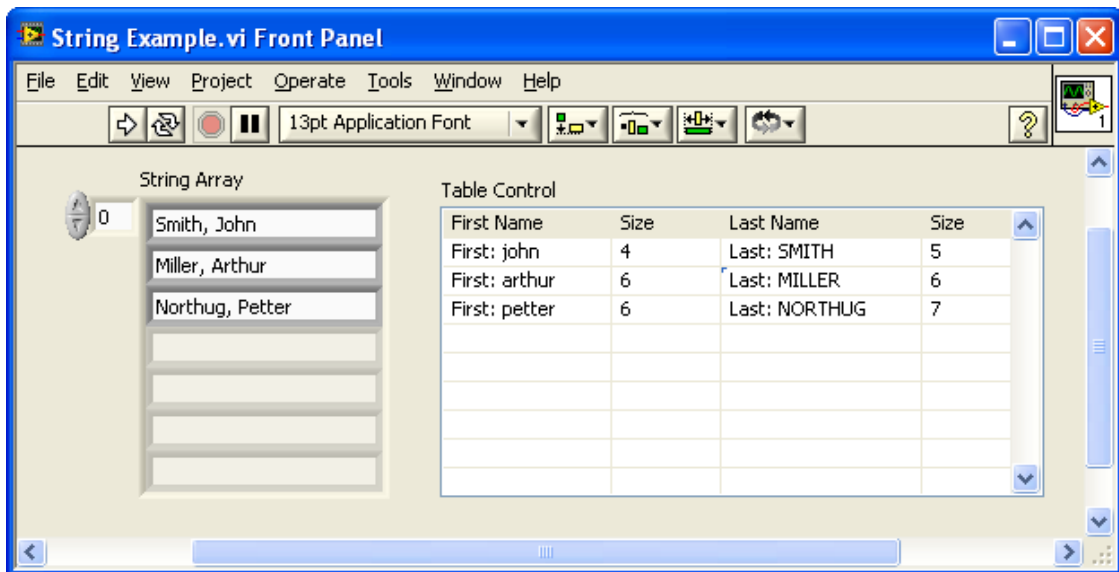
Task: Format into String. Create the following:



Task: String to Number and Number to String. Create the following:



Task: Create the following Example. You will need all you have learned above and more...

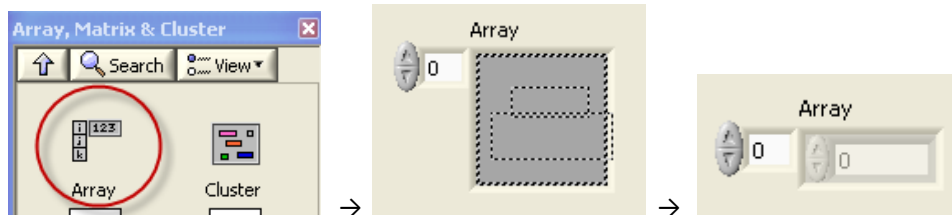




LabVIEW Arrays Example

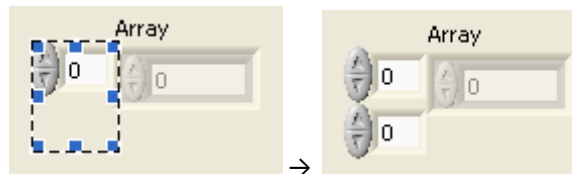
Description: Working and manipulating with Arrays is an important part in LabVIEW development. Arrays are very powerful to use in LabVIEW. In all your applications you would probably use both One-Dimensional Arrays and Two-Dimensional Arrays.

On the Front Panel using the Control palette we can create an array as follows (Array, Matrix & Cluster subpalette):

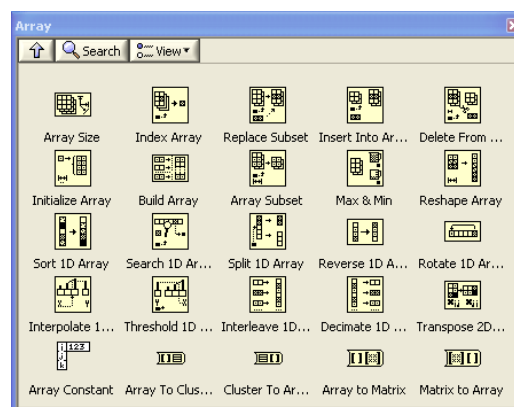


You drag and drop the empty Array on the Front Panel, next you find a Control or Indicator (Numeric, String, Boolean, etc,) and drag it into the empty Array. You can create an Array of (almost) any kind of Control or Indicator.




2D or multidimensional Array? Just drag the mouse in the Index display to the left and increase the dimension.



On the Block Diagram we have the following Array palette available from the Functions palette in LabVIEW:

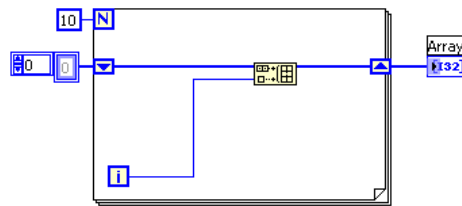
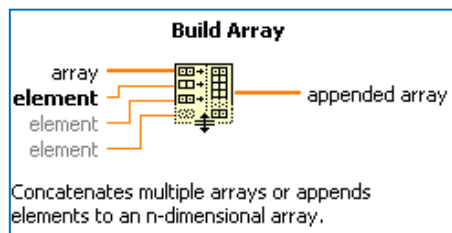


Use the Array functions to create and manipulate arrays. The most useful Array functions are:

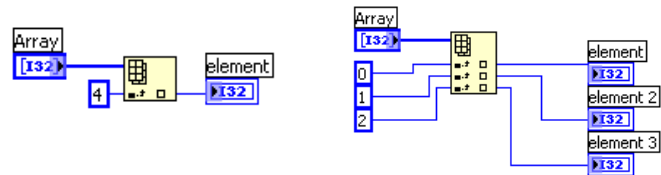
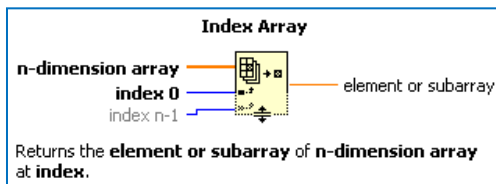
 Array Size	 Index Array	 Delete from Array	 Search 1D Array
 Initialize Array	 Build Array	 Array Subset	 Array Constant

All these functions are basic (but very useful) array functions you will probably be using in all your applications and VIs.

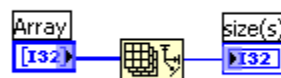
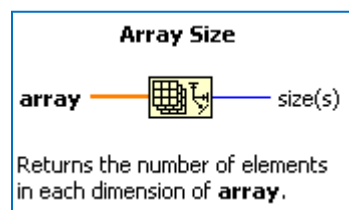
Task: Build Array. This function concatenates multiple arrays or appends elements to an n-dimensional array. Try the simple example below. This example using the Build Array function inside a For loop in order build an array with 10 elements.



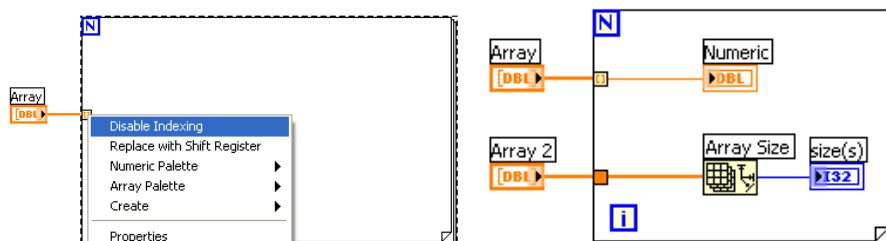
Task: Index Array. This function returns the element or subarray of n-dimension array at index. It is always useful to find a specific value in an array. The Index Array is extendible, so you can drag it out to find more than one elements. Try the simple example below.



Task: Array Size. This function returns the number of elements in each dimension of array. Try the simple example below. The example finds the size of an arbitrary array.



Task Auto-indexing. If you wire an array to a For Loop, you can read and process every element in that array by enabling auto-indexing. You also can enable auto-indexing by configuring a For Loop to return an array of every value generated by the loop. Create a simple example in order to see the difference.





Høgskolen i Telemark

Telemark University College

Department of Electrical Engineering, Information Technology and Cybernetics

LabVIEW Sub VI Example

Description: Sub VIs are very useful in LabVIEW. Using Sub VI helps you manage changes and debug the Block Diagram quickly. You can also easily reuse your code. SubVIs are equal to **functions** in text based languages.

Requirements: LabVIEW 2009

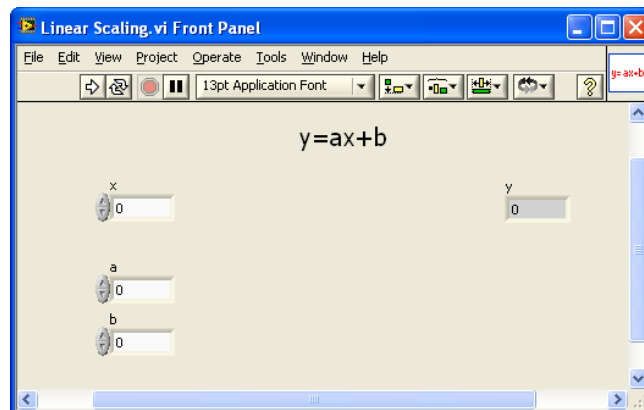
Task: Create a SubVI that performs a linear scaling $y = ax + b$. Where a , b and x are inputs, and y is an output.

The Procedure is as follows:

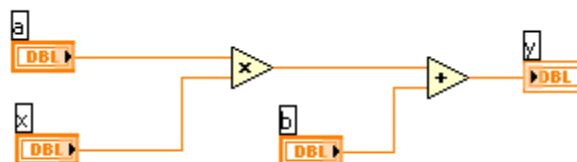
Step 1: Create a New VI (File→New VI) (Blank VI)

Step 2: Give the VI a Name (**Linear Scaling.vi**)

Step 3: Create your Front Panel with your necessary Controls and Indicators



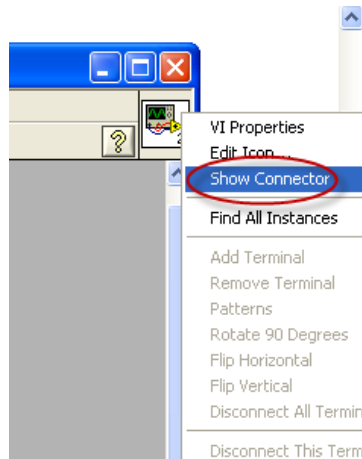
Step 4: Create your Block Diagram. The Block Diagram could look something like this:




Step 5: Create the Input and Output Connectors. Right-click on the little icon in the upper right corner and select "Show Connector".



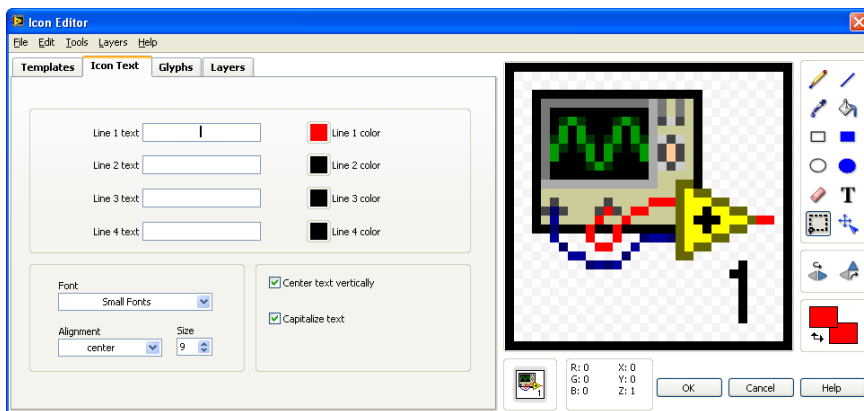
Faculty of Technology, Postboks 203, Kjølnes ring 56, N-3901 Porsgrunn, Norway. Tel: +47 35 57 50 00 Fax: +47 35 57 54 01



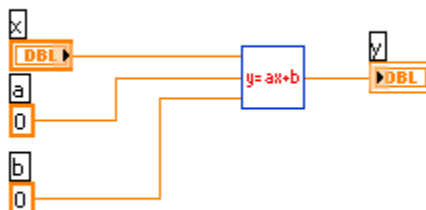
 Select the Wire tool and click on the wanted connector, then click on the Control or Indicator on the Front Panel you want to connect to this connector.



Step 6: Create an Icon using the Icon Editor. Right-click on the little icon in the upper right corner and select “Edit Icon...”.



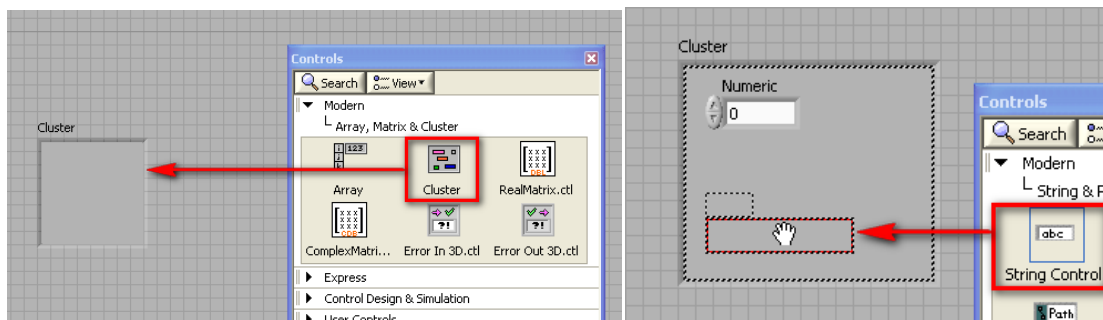
Step 7: Create a new VI that you use to test your Sub VI. Example:



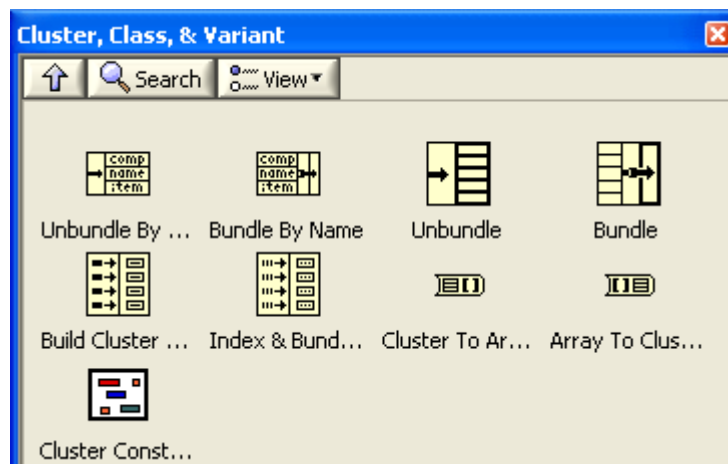


LabVIEW Cluster Example

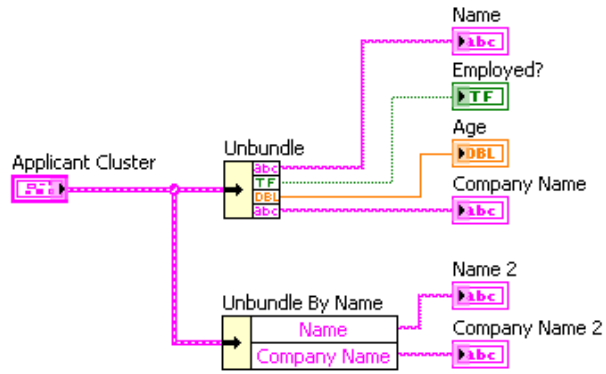
Description: Clusters group data elements of mixed types, such as a bundle of wires, as in a telephone cable, where each wire in the cable represents a different element of the cluster. A cluster is similar to a record or a struct in text-based programming languages. Bundling several data elements into clusters eliminates wire clutter on the block diagram and reduces the number of connector pane terminals that subVIs need. Like an array, a cluster is either a control or an indicator. A cluster cannot contain a mixture of controls and indicators.



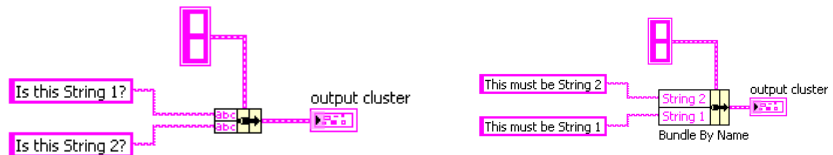
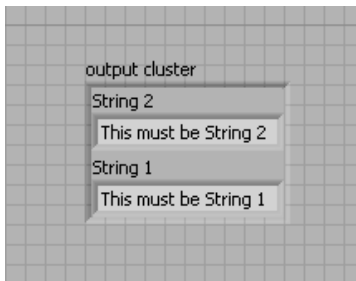
Cluster functions: In the Cluster, Class & Variant subpalette on the Block Diagram we have the following Cluster functions we may use to manipulate and get data in or out of a cluster. In this example we will create clusters and use these functions.



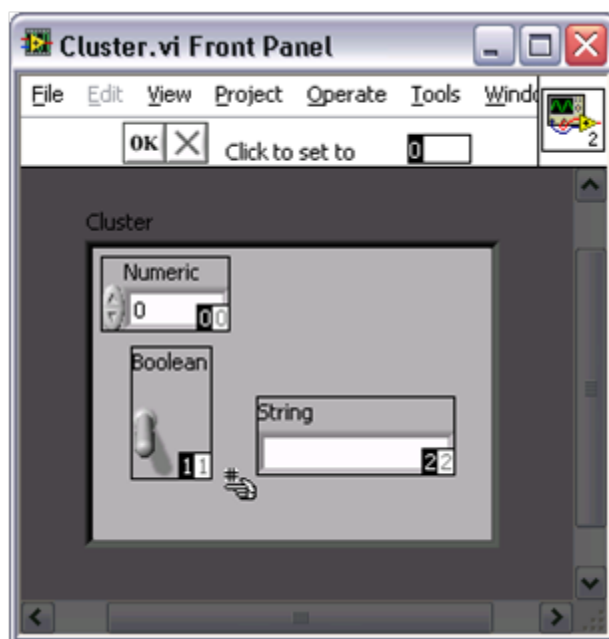
Task: Unbundle/Unbundle By Name. Use the **Unbundle** functions to disassemble a cluster into its individual elements. Use the **Unbundle by Name** function to return specific cluster elements you specify by name. You can also resize these functions for multiple elements using the mouse. Create the following code:



Task: Bundle/Bundle By Name. Use the **Bundle** function to assemble a cluster from individual elements. To wire elements into the Bundle function, use your mouse to resize the function. Create the following code:



Task: Cluster Order. Cluster elements have a logical order unrelated to their position in the shell. The first object you place in the cluster is element 0, the second is element 1, and so on. If you delete an element, the order adjusts automatically. The cluster order determines the order in which the elements appear as terminals on the Bundle and Unbundle functions on the block diagram. You can view and modify the cluster order by right-clicking the cluster border and selecting Reorder Controls in Cluster from the shortcut menu.





Høgskolen i Telemark

Telemark University College

Department of Electrical Engineering, Information Technology and Cybernetics

LabVIEW Formula Node Example

Description: A Formula Node in LabVIEW evaluates mathematical formulas and expressions similar to C on the block diagram. In this way you may use existing C code directly inside your LabVIEW code. It is also useful when you have “complex” mathematical expressions.

Requirements: LabVIEW 2009

Task: Create a simple SubVI where you use the Formula Node to calculate a (slope) and b (intercept) in the equation $y = ax + b$ when you have two points (x_1, y_1) and (x_2, y_2) .

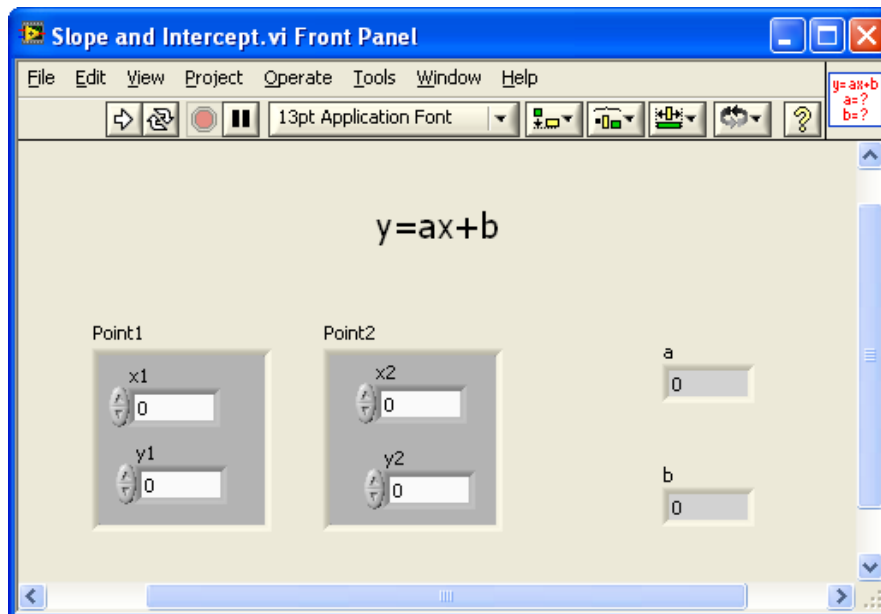
The Procedure is as follows:

Step 1: Create a New VI (File→New VI) (Blank VI)

Step 2: Give the VI a Name (**Linear Scaling.vi**)

Step 3: Create your Front Panel with your necessary Controls and Indicators.

Example:

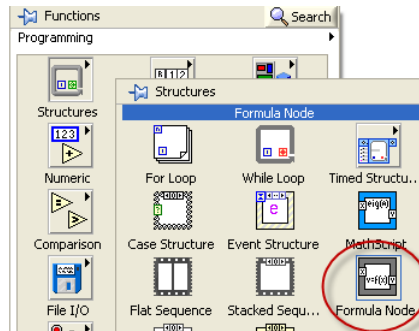


Step 4: Switch to your Block Diagram (**Ctrl+E**).

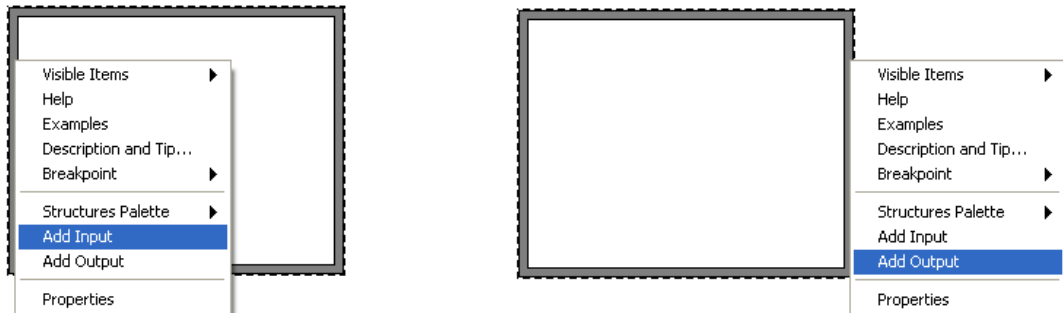
Step 5: Add the **Formula Node** to you Block Diagram:



Faculty of Technology, Postboks 203, Kjølnes ring 56, N-3901 Porsgrunn, Norway. Tel: +47 35 57 50 00 Fax: +47 35 57 54 01



Step 6: Add Inputs and Outputs:

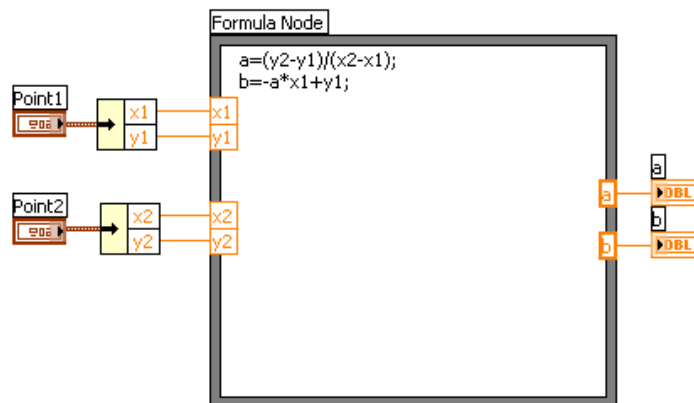


Step 7: Create your C-code inside your Formula Node.

The formula for finding the slope (a) and intercept (b) is as follows:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1), \text{ where } a = \frac{y_2 - y_1}{x_2 - x_1}$$

The Block Diagram could look something like this:



Step 8: Create the Input and Output Connectors. Right-click on the little icon in the upper right corner and select “Show Connector”.

Step 9: Create an Icon using the Icon Editor. Right-click on the little icon in the upper right corner and select “Edit Icon...”.

Step 10: Create a new VI that you use to test your Sub VI.



Høgskolen i Telemark

Telemark University College

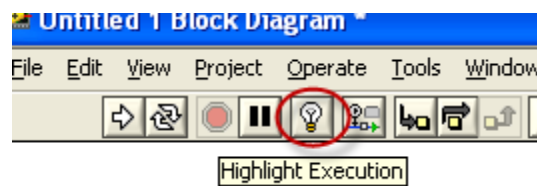
Department of Electrical Engineering, Information Technology and Cybernetics

LabVIEW Debugging Example

Description: LabVIEW offers different debugging techniques, such as **Highlight Execution**, **Probes**, **Breakpoints** and **Single Stepping**. In this example we will use these debugging techniques on an existing VI.

Requirements: LabVIEW 2009

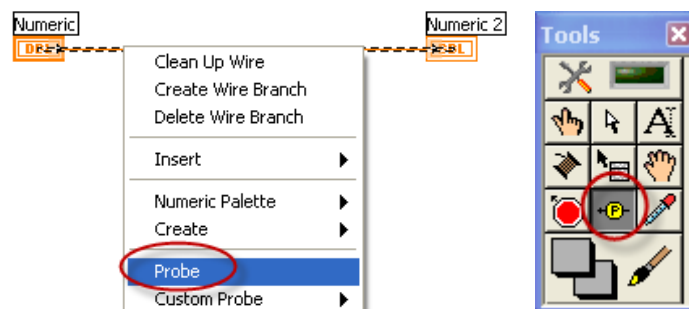
Task: Highlight Execution: View an animation of the execution of the block diagram by clicking the **Highlight Execution** button.



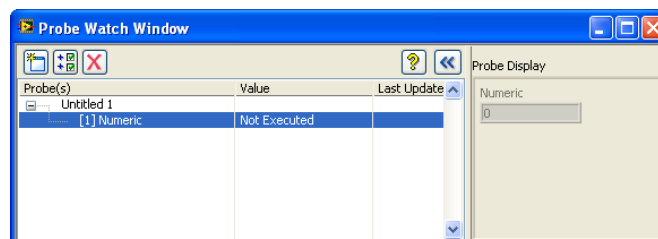
Execution highlighting shows the movement of data on the block diagram from one node to another using bubbles that move along the wires. Use execution highlighting to see how data values move from node to node through a VI.

Task: Probes: Use the Probe tool to check intermediate values on a wire as a VI runs. Use the Probe tool if you have a complicated block diagram with a series of operations, any one of which might return incorrect data. If data is available, the probe immediately updates and displays the data during execution.

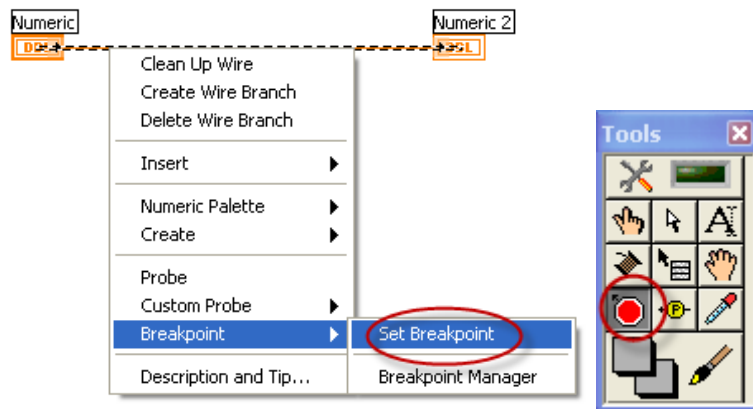
Add a Probe by right clicking a wire and select "Probe". Probes are also available from the Tools Palette.



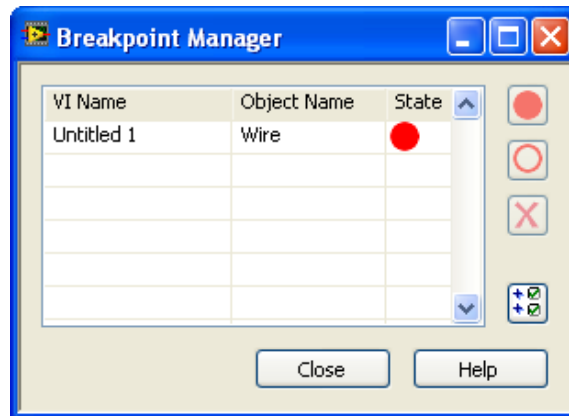
Use the **Probe Watch** window to manage and watch values in all your probes:



Task: Breakpoints: Use the Breakpoint tool to place a breakpoint on a VI, node, or wire and pause execution at that location. When you set a breakpoint on a wire, execution pauses after data passes through the wire and the Pause button appears red. Right-click on a wire to set a breakpoint or use the Tools Palette as shown below.



The **Breakpoint Manager** (available from the View menu) lists all your breakpoints.



Task: Single Stepping: Single-step through a VI to view each action of the VI on the block diagram as the VI runs. The single-stepping buttons, shown as follows, affect execution only in a VI or subVI in single-step mode.



When you single-step through a VI, nodes blink to indicate they are ready to execute. If you single-step through a VI with execution highlighting on, an execution glyph appears on the icons of the subVIs that are currently running.



Høgskolen i Telemark

Telemark University College

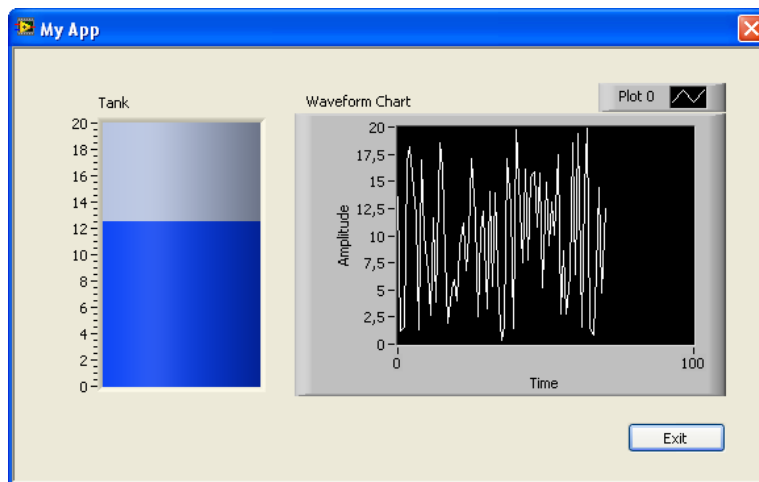
Department of Electrical Engineering, Information Technology and Cybernetics

LabVIEW Project Explorer Example

Description: Projects in LabVIEW consist of VIs, files necessary for those VIs to run properly, and supplemental files such as documentation or related links. Use the Project Explorer window to manage projects in LabVIEW. In the Project Explorer window, you can use folders and libraries to group together items, and you can use a list of VI hierarchies called Dependencies to keep track of items a VI depends on.

Requirements: LabVIEW 2009

Task: Create a new Project in LabVIEW. Create several folders in order to organize your project. Create a folder for your Main VI and a folder for your Sub VI, etc. Create a “Dummy” Application with some VIs that you insert into these folders. At the end create an **executable** file (.exe) of your main VI, e.g., “MyApp.exe”. The Example below simulates a process system using a Random generator inside a While Loop:



The Procedure is as follows:

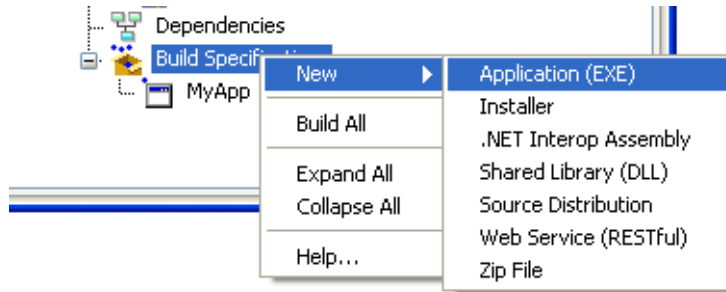
Step 1: Create a **New Project**. Use the Project Explorer window to create and edit LabVIEW projects. Select File»New Project to display the Project Explorer window, or select “Empty Project” in the “Getting Started” window.



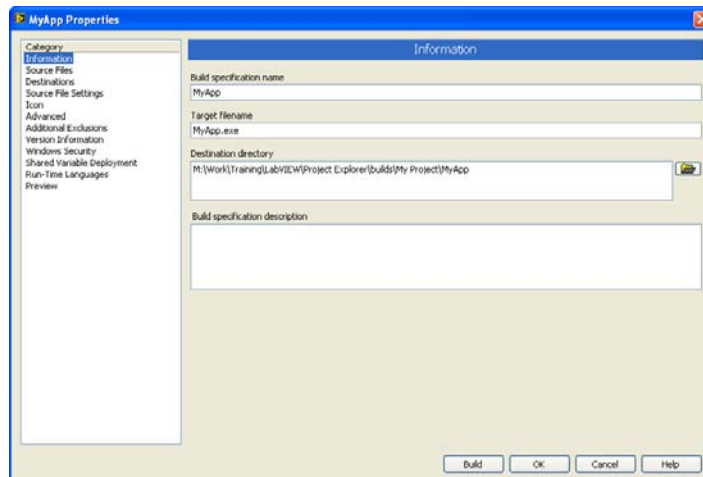
Faculty of Technology, Postboks 203, Kjølnes ring 56, N-3901 Porsgrunn, Norway. Tel: +47 35 57 50 00 Fax: +47 35 57 54 01

Step 2: Create Folders and Files (VIs) and test your application.

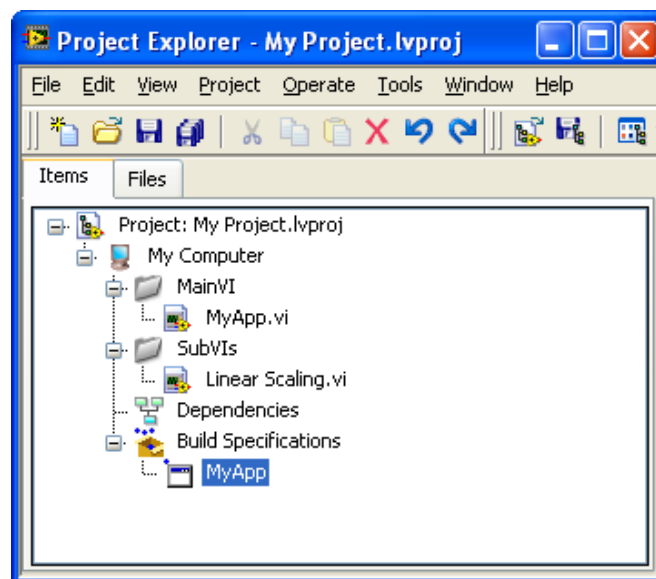
Step 3: Create an **Executable** file (.exe) of your Application.



Step 4: Configure the **Properties** for your Executable Application:



Step 5: Finished. The Project Explorer could look something like this:





Høgskolen i Telemark

Telemark University College

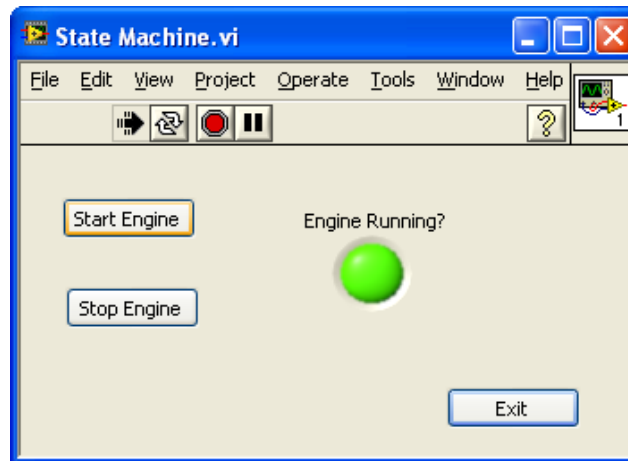
Department of Electrical Engineering, Information Technology and Cybernetics

LabVIEW State Machine Example

Description: The **state machine** is one of the fundamental architectures LabVIEW developers frequently use to build applications quickly. The State Machine approach in LabVIEW uses a **Case structure** inside a **While loop** to handle the different states in the program, and the transitions between them. A **Shift Register** is used to save data from and between the different states.

Requirements: LabVIEW 2009

Task: Create a simple program in LabVIEW using the State Machine approach. The User Interface (Front Panel) should look something like this:



When the user push the “Start Engine” button, the light gets on and when the user push the “Stop Engine” button, the light gets off. When the user pushes the “Exit” button, the program will stop.

The Procedure is as follows:

Step 1: Create a **New VI** (File→New VI) (Blank VI)

Step 2: Create your **Front Panel** as in the Example above.

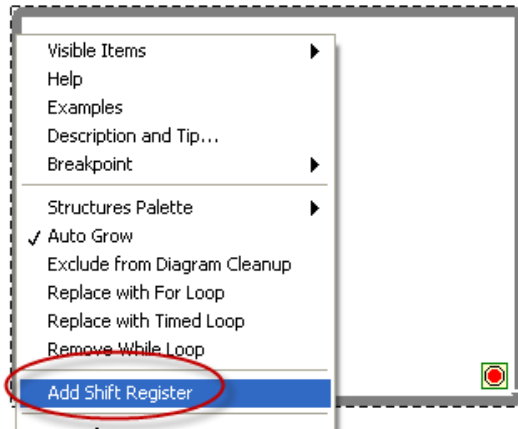
Step 3: Add a **While Loop** to your VI



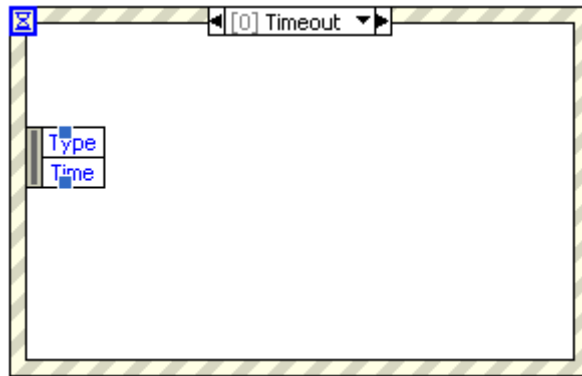
Step 4: Add a **Case Structure** to your VI inside the While Loop



Step 5: Add a **Shift Register** to your While Loop by right-click on the While Loop and select “Add Shift Register”.

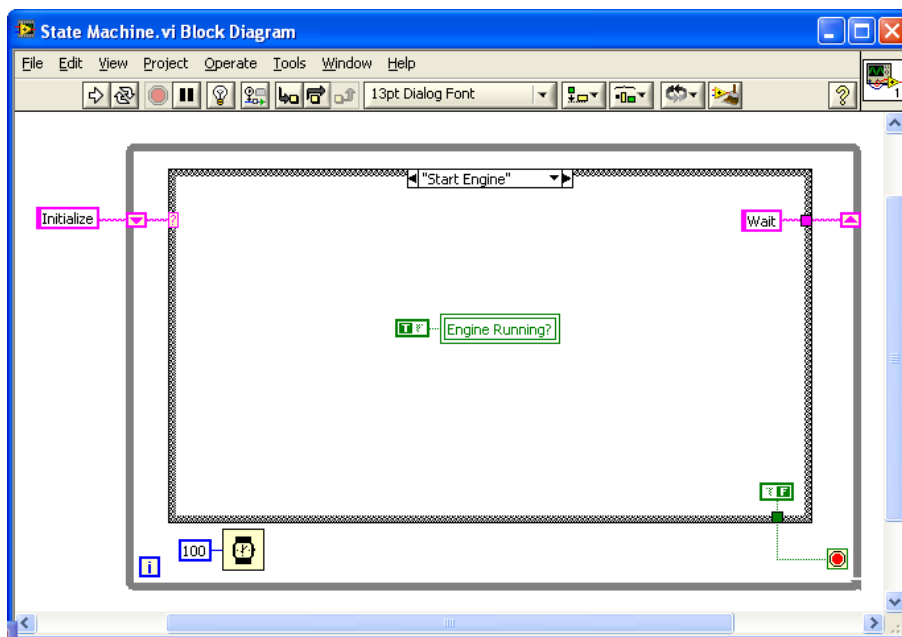


Step 6: Add a Case called Wait in your Case Structure. Create an **Event Structure** inside this case. Add Event for your 3 different buttons; “Start Engine”, “Stop Engine” and “Exit”.



Step 7: Finish the Code.

Here is an extract of the program:





Høgskolen i Telemark

Telemark University College

Faculty of Technology

Kjølnes Ring 56

N-3914 Porsgrunn, Norway

www.hit.no

Hans-Petter Halvorsen, M.Sc.

Telemark University College

Department of Electrical Engineering, Information Technology and Cybernetics

Phone: +47 3557 5158

E-mail: hans.p.halvorsen@hit.no

Blog: <http://home.hit.no/~hansha/>

Room: B-237a
