TechTeach

Introduction to LabVIEW 8.5

by

Finn Haugen

31. August 2008

Contents:

1 Preface

Note: Videos are provided in Section 4 and 7.1 These videos gives essentially the same information as in the text of the respective sections.

If - for some reason - the videos does not display correctly in the current player, try some other. Windows Media Player is probably the default video player on your PC. Alternative players are RealPlayer and QuickTime (both can be downloaded from the Internet for free).

The aim of this document is to give you an introduction to LabVIEW version 8.5. It is assumed that you have LabVIEW 8.5 installed on your computer. The introduction is to "core" LabVIEW. Thus, no particular LabVIEW toolkit or module is covered.

Only a basic introduction is given, enough to make you able to develop your own LabVEIW programs. If you need additional information, use Help i LabVIEW or search for relevant examples in LabVIEW.

This tutorial is self-instructive, having a number of activities that you are supposed to perform. These activities are shown in blue boxes, as here:

> Activities are shown in blue boxes as this one.

Please send comments or suggestions regarding this document via e-mail to finn@techteach.no.

More tutorials that may be relevant for you as a LabVIEW user are available from Finn's LabVIEW page.

There are also lots of tutorials available on the National Instruments home page, http://ni.com. I suggest that you - now - or actually better (because then you will not be distracted :-), *after* you have accomplished my tutorial - browse this LabVIEW Interactive Tutorial which gives you a good overview over plattforms and applications. The present tutorial (that you now read) focuses more on LabVIEW as a programming tool. (I think you quickly meet the wall if you do not master basic LabVIEW.)

[Table of contents]

2 Introduction

LabVIEW is a development system for industrial, experimental, and educational

measurement and automation applications based on graphical programming, in contrast to textual programming - however, textual programming *is* supported in LabVIEW. LabVIEW has a large number of functions for numerical analysis and design and visualization of data.

LabVIEW now has several toolkits and modules which brings the LabVIEW to the same level of functionality as Matlab and Simulink in analysis and design in the areas of control, signal processing, system identification, mathematics, and simulation, and more. In addition, LabVIEW has, of course, inbuilt support for the broad range of measurement and automation hardware produced by National Instruments. Communication with third party hardware is also possible thanks to the availability of a large number of drivers and the support for communication standards as OPC, Modbus, GPIB, etc.

LabVIEW is produced by National Instruments.

[Table of contents]

3 Launching LabVIEW

You can launch LabVIEW on you PC as follows:

> Start LabVIEW via Start / All Programs / National Instruments / LabVIEW 8.5 / LabVIEW.

[Table of contents]

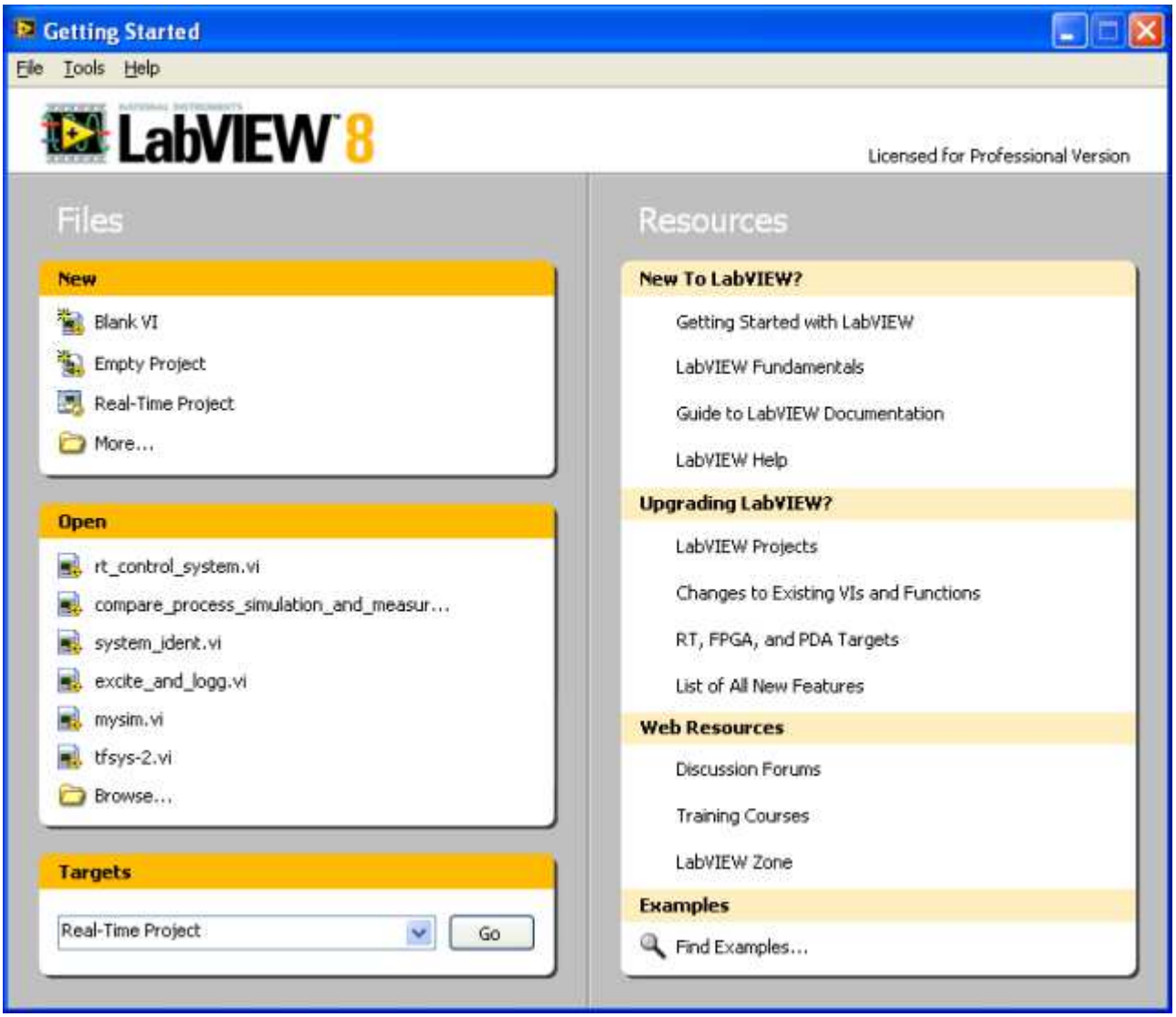
3.1 License Activation

Depending on the way LabVIEW has been installed on your computer, you may be asked by the Activation Wizard to *activate* the LabVIEW license online (via Web). The activation is done only once on a given PC.

[Table of contents]

3.2 Starting using LabVIEW

Assuming that the license activation has run without problems, starting LabVIEW opens the Getting Started dialog window shown below.



Getting Started dialog window

Here are a comments to the most commonly used options in this dialog window:

- **Files / New / Blank VI** opens a new, blank VI. VI is short for *Virtual Instrument*, which represents a program developed in LabVIEW. This is the most commonly used option.
- **Files / New / Empty Project** opens a new project which is a collection of various LabVIEW files and other files making up a project. (A project is a new construct in LabVIEW 8.)
- **Files / Open** shows the recently opened files. It is kind of a history list.
- **Resources / New to LabVIEW? / Getting Started with LabVIEW** links to the Getting Started with LabVIEW PDF document of approximately 80 pages. You may regard it is an alternative to or an addition to the present tutorial. The present tutorial is more directly aimed to developing applications running continuously with a fixed cycle time, as in a control, data acquisition, or a simulation application.
- **Resources / LabVIEW Fundamentals** links to the LabVIEW Help, where you can find information using the Contents, Keywords, and Search options in the usual

way.

- **Resources / Find Examples** links to the library of examples included in LabVIEW. This library represents a large knowledge base. If you have a specific problem, you will probably find an appropriate example here. The examples are also available via the Help menu in any LabVIEW window.

Try the items listed above. However, it is assumed that you do not spend too much time on each of them now. (If you open new files, close them without saving.) We will return to several of these items during this tutorial.

[Table of contents]

4 Looking at an example VI: level_measurement.vi

4.1 Downloading and opening the example

Note: Below are two AVI videos.Playing these videos gives the same information as is provided with the subsequent text in this Chapter.

The video **level_measurement_front_panel.avi** (17 minutes, 180 MB) describes the operation and contents of the *Front panel* of the **level_measurement.vi** example described in Sections 4.1, 4.2, and 4.3 below.

The video **level_measurement_block_diagram.avi** (26 minutes, 310 MB) describes the *Block diagram* of the **level_measurement.vi** example described in Section 4.4 below.

Here begins the text that gives the same information as the videos above.

To see how a LabVIEW program works and how it is constructed, let us study the VI (Virtual Instrument) named **level_measurement.vi** which I have made for the purpose of having a simple yet illuminating introductory example. In Sec. 7 of this document you will learn how to develop this VI. This VI converts an assumed (user-adjusted) level measurement signal, u, in milliamperes into a level value, y, in meters according to the following mathematical formula:

$$y = K*(u-u0)$$

where K is the gain and u0 is the zero of the measurement. K and u0 can be adjusted by the user. By default, K is 0.5m/mA and u0 is The level value is shown in a tank indicator and in a chart. If the level becomes greater than 7 meters og smaller than 1 meter, a proper alarm is displayed in each case. The program runs periodically with a cycle time of 0.1s. A lamp, or a LED (Light Emitting Diode), indicates if the programs runs or not.

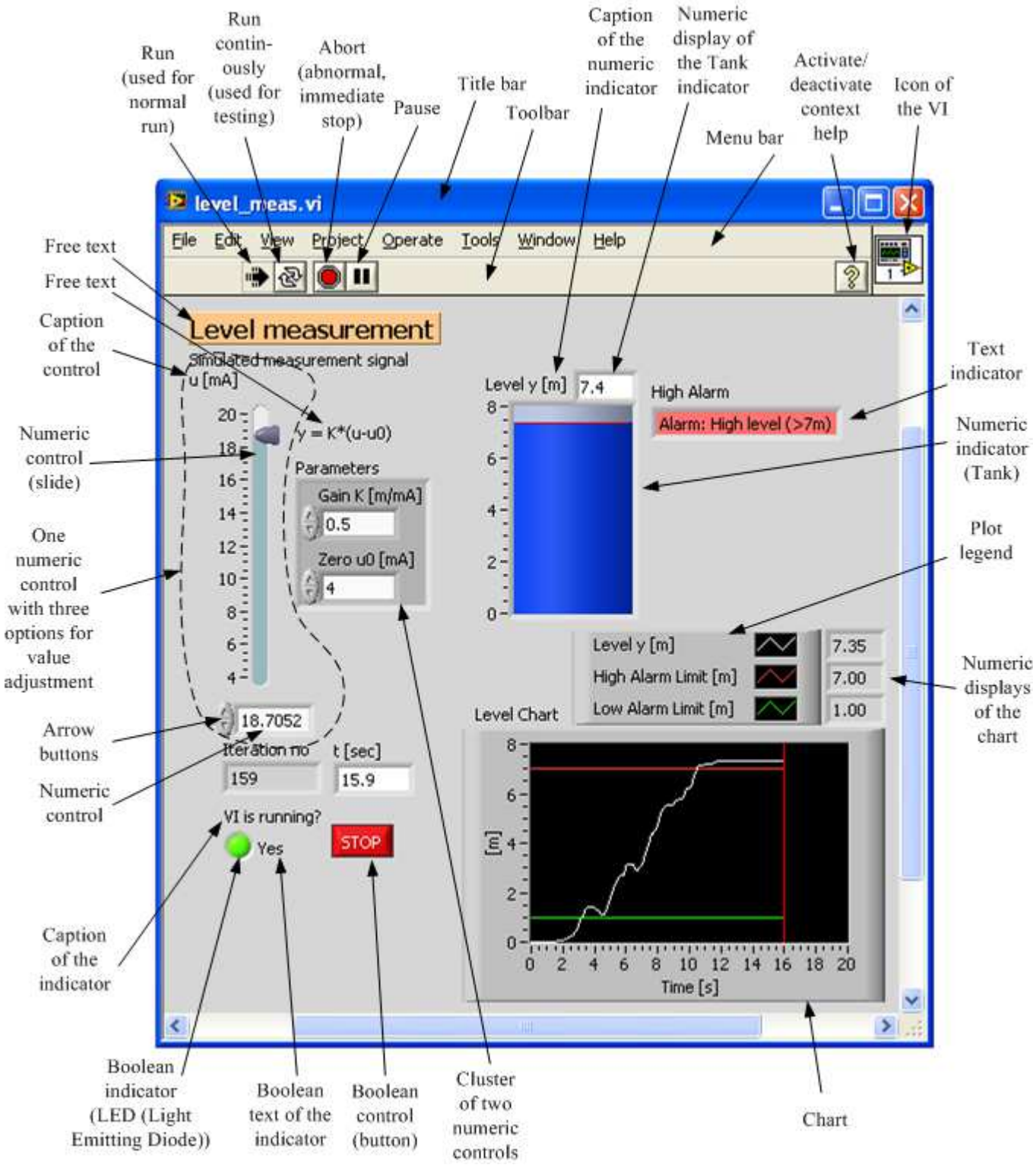
Download level_measurement.vi to any folder (directory) you wish (e.g. C:\temp), or just open it from the default download folder. Then open the VI via **Open /**

Browse in the Getting Started dialog window.

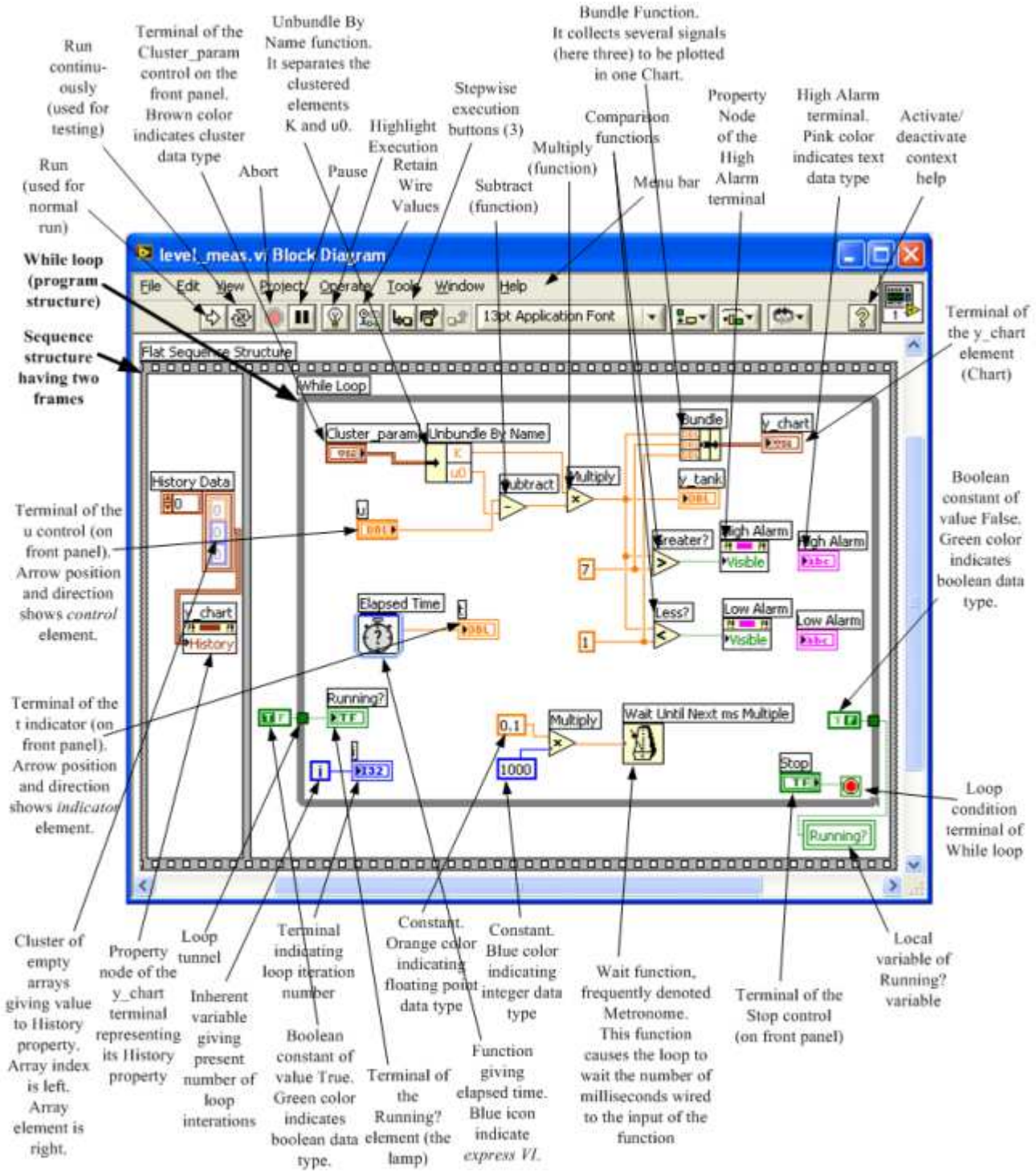
Opening level_measurement.vi opens two windows:

- ***The Front panel***, which constitutes the *user interface* of the VI.
- ***The Block diagram***, which contains the *program code* defining the functionality of the program.

The front panel and the block diagram of level_measurement.vi are shown in the following figures. At the moment, disregard the comments added to the figures. We will soon return to the details.



The front panel of level_measurement.vi



The block diagram of level_measurement.vi

[Table of contents]

4.2 Running the VI

The VI is operated using the buttons on the toolbar, cf. the front panel figure:

- **The Run button** is used to start the VI. This button becomes white when the VI is not running, and black when it is running. If the VI contains errors, the VI does not start, and the Run button is grey and broken.

- **The Run Continuously button** is used only for testing purposes when you want to run the VI over and over again. It works as when you continuously click the Run button after the VI has stopped. (Unfortunately many users erroneously clicks the Run Continuously button to run the VI in normal operations. Actually, you need this button very seldom.)
- **The Abort button** is used to abort the operation of the VI. It is used for abnormal or uncontrolled stop. Typically, the programmer will have put a specific button or switch on the front panel to provide for a controlled stop. In level_measurement.vi there is a red stop button at the bottom of the front panel.
- **The Pause button** is used to pause the VI as it runs. By clicking the Pause button again the VI runs again.

Run the level_measurement.vi. Then play with the VI by adjusting some of the elements on the front panel. To *stop* the VI click the Stop button.

Run the VI again. Try the Pause button and the Abort button.

[Table of contents]

4.3 Studying the Front panel of the VI

Now that you have played with the level_measurement.vi, we will study the contents of the VI in so that you get a more detailed knowledge about LabVIEW. Look at level_measurement.vi. It contains two types of elements:

- **Controls**. You can adjust the value of a control. A control is an input element.
- **Indicators**. These elements are used to indicate values. An indicator is an output element.

State if each of the following elements is a *control* or an *indicator*:

- The element showing time
- The Stop button
- The element where the gain K can be adjusted
- The lamp
- The chart

Below are detailed comments about the elements on the front panel.

- **Description and Tip strip of elements**: The programmer may have added a description and/or a tip strip of a particular element:

  Run the level_measurement.vi.
  - To see the Description and Tip strip in a separate window: **Right-click on the element / Description and Tip**. Close the display using the OK button in

the window that was opened.
- To see the Tip strip: Move the cursor over the vertical pointer slide. The figure below shows the tips strip.

Click the Show Context Help Window button in the front panel toolbar, thereby displaying the description and the tip of the front panel element where the cursor is at the moment. Then, disable the Show Context Help Window button.

- **Data types**: The front panel elements - both controls and indicators - has a specific *data type*. (We will come back to data types in more detail when we study the block diagram of this VI.) In this VI there are elements of the following data types:
  - *Floating point number* (decimal numbers)
  - *Integers*
  - *Boolean or logical elements*, which means that the possible values of the element are either On (or True) or Off (or False)
  - *Text*
  - *Clusters*. They contain one or more elements of possibly different data types.

Find at least one front element for each of the data types listed above. (To see a textual element, run the VI and set the value of u greater than 18.)
Answer:
- Floating point number: The numerical indicator displaying time.
- Integer: The numerical indicator displaying iteration number.
- Boolean: The Stop button.
- Text: The textual alarm indicator displaying an alarm as the level is greater than 7m.
- Cluster: The element containing the gain K and the zero u0.

- **Several options for value adjustment**: For one specific control element there may be several optional ways to adjust the value of the element, cf. the vertical pointer slide for adjusting u on the front panel. In this example there are three ways to adjust the value:
  - Typing the value
  - Clicking the arrow buttons
  - Moving the pointer on the slide

Run the level_measurement.vi. Try the three different ways of adjusting the measurement value, u.

- **Several ways to indicate the value**: For one specific indicator the value may be shown in more than one way.

Run the level_measurement.vi. Observe that the level is indicated in a chart and in an inherent numeric display. The tank indicator is actually one separate level indicator.

- **Plots**: The present front panel contains one *chart* plotting altogether three signals: (1) The level. (2) The High alarm limit, which is 7 (meters). (3) The Low alarm limit, which is 1. In LabVIEW a chart is a continuously updated diagram with time along the x-axis. You can think of a chart as a pen plotter where the x axis develops linearly with time. (LabVIEW also has *graphs* (though not in this example) which is a more general x-y plot where the x-axis may not show time but some other values.) The user can change several properties of the chart even while the VI is running (unless these properties have been programmed to be non-adjustable):

  Run level_measurement.vi.
  1. Change the maximum y scale value from 8 to 10 by double-clicking on 8 and editing. Then, reset the value to 8.
  2. Right-click anywhere on the chart, and select Autoscale Y from the menu that is opened. Then, deselect Autoscale Y, and set the Y axis scaling from 0 to 8.
  3. Right-click on the plot legend above the diagram to open a menu of plot settings, and change line color, line width, line style, interpolation, point style. Also try some of the bar plots.
  4. Try the various options available via **right-click (on the chart) / Visible Items**:
     - Hide/show the inherent digital display
     - Show/hide X scrollbar
     - Show/hide Scale legend
     - Show/hide Graph palette
  5. Try the various options available via **right-click / Update Mode**:
     - Strip Chart
     - Scope Chart
     - Sweep Chart
  6. Take a snapshot of the chart via **right-click / Export Simplified Image** (select the options Emf-file and Save to clipboard in the menu that is opened). Paste the saved image into e.g. a Word document or some photo editor.
  7. Clear the chart via **right-click / Clear Chart**.

- **Free text, captions, and labels**: Anywhere on the front panel some *free text* may be written by the programmer. For a specific element the *caption* of that element may be shown. The caption is a descriptive text for that element. Furthermore, an element must have a *label*, which is the name of the element. When referring to the

element as a program variable, the label is the variable name. The programmer decides wether the caption and/or the label is shown or not on the Front panel. In the block diagram the label is always shown. The caption is not shown in the Block diagram.

> Run the level_measurement.vi. Observe that the tank indicator has a caption (on the Front panel) which is different from the label (the label, not the caption) is always shown on the diagram.

[Table of contents]

4.4 Studying the Block diagram of the VI

The Front panel is the user interface of the VI. However, the Front panel itself contains no program code. The program code defining the functionality of the program is contained in *the Block diagram* of the VI.

> It is assumed here that the level_measurement.vi does not run. Open the block diagram of the VI using the menu selection **Window / Show Block Diagram**. Then use the menu **Window / Show Front Panel** to show the front panel. Also try toggling between the front panel window and the block diagram window using the keyboard shortcut **Ctrl E** (this is a useful shortcut to remember - it will save you a lot of time during a lifetime with LabVIEW).

The following are several comments to the block diagram.

The program representation is *graphical*

A LabVIEW program contains graphical program code, i.e. the code contains various types of elements, blocks and signal wires. (It is however possible to include textual program code in LabVIEW, using e.g. the Formula node or the MathScript node.)

Corresponding elements on the front panel and the block diagram

To any front panel element (e.g. a numeric control or indicator) there is a corresponding *terminal* in the block diagram. The terminals play the same role as *variables* in other programming languages, as C, Delphi, Visual Basic etc., and sometimes it is natural to say variable in stead of terminal. Note that terminals look a little different depending on wether it corresponds to a control or an indicator, cf. the examples indicated in the Block diagram. One example is the u terminal in the block diagram. It corresponds to the slider on the Front panel. The value of the terminal and of the front panel element (control or indicator) is always the same. You can find the corresponding front panel element to a given block diagram terminal by double-clicking on the terminal, or by right-clicking the terminal and selecting **Find Control** - or **Find Indicator** - from the menu that is opened. The reverse operation works, too: You can can find the corresponding block diagram terminal to a given front panel element by double-clicking on the element, or by right-

clicking the element and selecting **Find Terminal** from the menu that is opened.

> Find the Front panel element corresponding to the u terminal. Is the Front panel element a control or an indicator?
>
> Also find the Front panel element corresponding to the y_chart terminal. Is the Front panel element a control or an indicator?

Data types

Different data types are indicated with different colors. The following data types exist in level_measurement.vi:

- ***Floating point numbers*** (decimal numbers) are shown in orange color. The letters DBL on the icon is for double, which refers to floating point numbers with double precision, which expresses the numerical accuracy of the internal representation of the number in the computer. You can think of floating point numbers as decimal numbers. One example is the u terminal.
- ***Integers*** are shown in blue color. One example is the numeric constant of value 1000.
- ***Boolean*** data are shown in green color. The letters TF on the icon is for True/False, which are the two possible boolean (logical) values of the element. One example is the Stop terminal.
- ***Textual*** data are shown in pink color. One example is the High Alarm terminal.
- ***Cluster*** data are shown in brown color. Clusters are multivariables. They contain one or more elements of possibly different data types. One example is the Cluster_param terminal.

  > Which two elements are contained in the cluster named Cluster_param?
  > (Answer: K and u0.)

Constants

Constants are defined values that exist only in the block diagram (so there are no corresponding front panel element). Constants can have various data types, cf. the description of data types above. One example is the constant of value 7 entering the Greater? function.

> Locate the following elements on the block diagram:
>
> - Floating point constants (you should find 3)
> - Boolean constants (2)
> - Floating point terminals corresponding to indicators on the front panel (3)
> - Textual indicators (2)
> - Boolean indicator (1)

Property node

A property node represents one or more properties of a front panel element. These properties can be either readable or writable. (The latter is typical.) Using property nodes various properties can be set (or retrieved) programatically. Most properties can also be set via the front panel, by right-clicking on the element there.

Locate the two property nodes to the right in the block diagram of level_measurement.vi. To which elements or terminals do these property nodes belong? Which property is set (written a value to) in both theses property nodes? What is the purpose of the property nodes in this case?
(Answers: The property nodes belong to the two text indicators named High Alarm and Low Alarm, respectively. The Visible property is set. The purpose of the property nodes is to show them, i.e. to make them visible, on the front panel when the level passes the respective alarm limits.)

Program execution

A LabVIEW program is executed according to the *data flow principle*: A block or some other program part is executed only if all inputs to the block has valid values. LabVIEW distributes the execution resources equally to the various parts of the program so that no part is left out.

Program structure. Cyclic program execution using the While loop

As seen in the block diagram figure, the progam code is structured using a **Flat Sequence structure**. The (Flat) Sequence structure is in the form of a film strip with frames. Code can be put into the different frames. The code in the *first frame* will be executed first, then the code in the second frame, and so on. In the present VI, the code in the first frame writes an empty array to the History property of the chart, thereby emptying the chart just before the cyclic program execution starts. In the *second frame* is a **While loop**. The program code inside the While loop frame is executed over and over again cyclically, until the stop condition of the While loop is satisfied. It is the value that is wired into the Loop condition terminal (down left in the While loop) that determines if the loop will stop or continue to run. In this VI the Stop terminal is wired to the loop condition terminal, and hence, this switch is used to stop the while loop. And when the loop stops, any program code outside the While loop waiting for data from some tunnels on the While loop frame will be executed. In this VI the Running? local variable will get value False just after the the While loop has stopped, and after the While loop has stopped the program will stop.

The Metronome function, which is actually denoted the Wait Until Next ms Multiple function, ensures that the cycle time of the While loop is 0.1 seconds. The Metronome actually waits the number of milliseconds wired to the input of the function. In our VI the 0.1 constant is multiplied by 1000 constant to transfer from seconds to milliseconds. Alternatively, we could have wired just a constant of value 100 (ms) into the Metronome.

Note that the Metronome does *not* implement a time delay of 100ms (in our VI). Suppose, just to take an illustrative example, that LabVIEW uses 2ms to execute the program code inside the While loop (the actual execution time is smaller for our simple program). Then, if the Metronome implemented a time delay of 100ms, the actual cycle time would be 2ms + 100ms = 102ms which is not as specified. In stead, furtunately, the Metronome implements a time delay of 98ms, so that the cycle time becomes 2ms + 98ms = 100ms, as specified at the Metronome input.

Watching and running the execution step-by-step

LabVIEW shows the process of program execution if you click the Highlight Execution button in the toolbar of the block diagram. Furthermore, you can run the program step-by-step by clicking the Start Single Stepping button. These functions are particularly useful for debugging the program since the program execution is shown clearly and in slow speed.

> Run level_measurement.vi. Open the block diagram of the VI.
>
> Click the Highlight Execution button in the toolbar, and observe how the program execution is highlighted. Then stop the highlighting execution.
>
> Click the Start Single Stepping button in the toolbar to step through the program step-by-step. Stop the single stepping by clicking the Step Out button in the toolbar.

Local variable

A local variable is a *copy of a terminal* (or variable), and the value of a local variable is the same as the value of the terminal. A local variable can be either writable, or readable. One terminal can have several local variables, being readable and/or writeable. Local variables can be used anywhere *within the same VI*, but not in another VI - in such cases, you can use *shared variables*.

> level_measurement.vi has one local variable, namely the local variable corresponding to the terminal named Running?. Locate this local variable. What is its data type? Try to explain how the local variable is used in this program.
> (Answer: Just before the while loop starts, the Running? terminal gets the value of True, thereby lighting the lamp on the front panel. When the user has clicked the Stop button on the front panel, the while loop stops. Just before the while loop stops, the boolean constant of value False is written to the Running? local variable, causing the terminal to get value False, too, thereby turning off the lamp.)

[Table of contents]

5 Help

- **Help about a specific function in the block diagram**: **Right-click on the element / Help**. This opens a Help window for that function.
- **Context help**: By clicking the Context Help button (with symbol ?) at the right side of the toolbar a context help window is opened. By moving the cursor over an element on the block diagram or on the front panel information about that element is shown in the context help window.
- **Help about a topic**: Menu: **Help / Search the LabVIEW Help**.

Various help (technical documents, application notes etc.) can be found on http://ni.com.

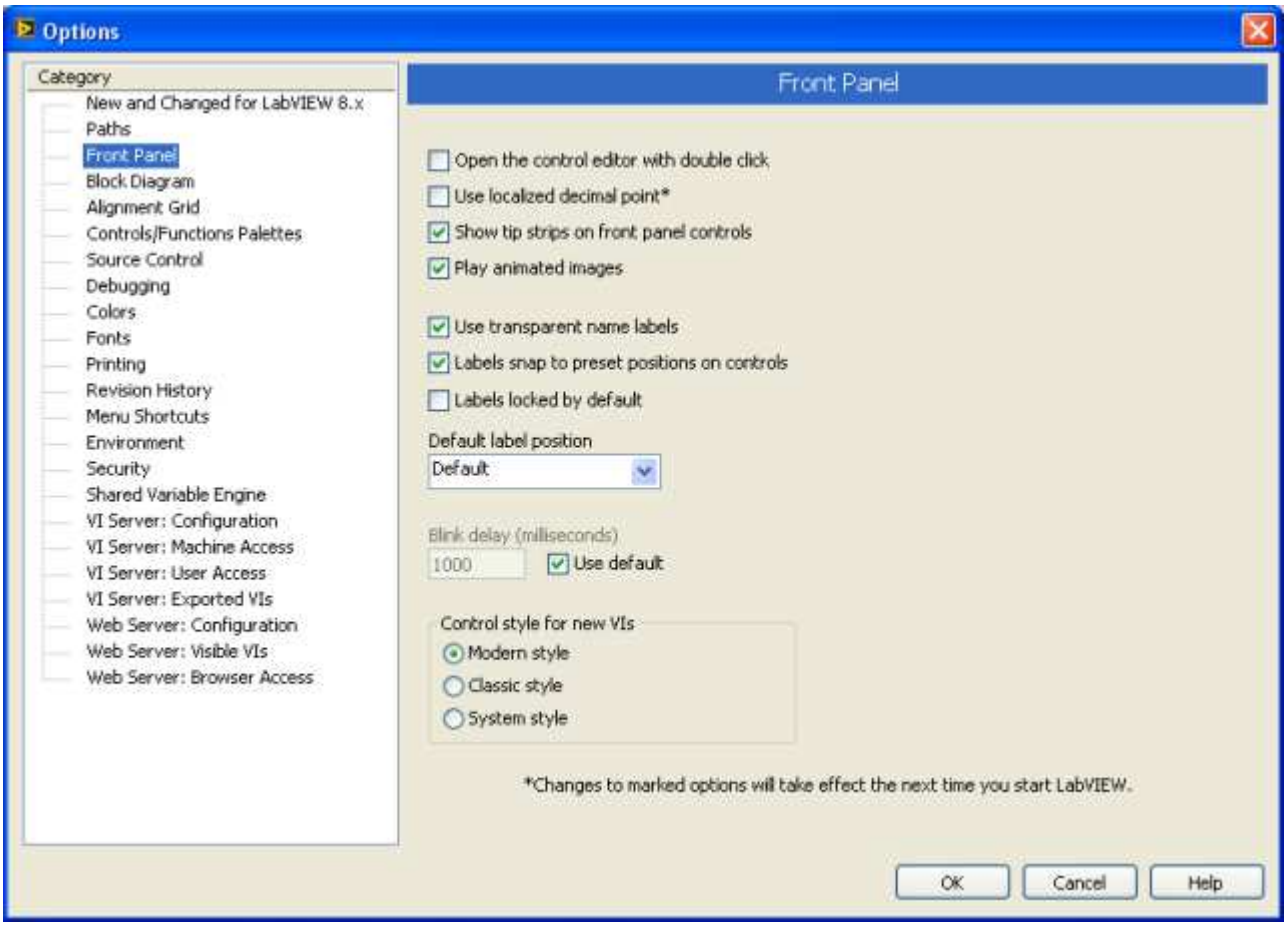
It is assumed that the block diagram of level_measurement.vi is opened. Try the various Help options:

- Get help about the Subtract function in the block diagram (by right-clicking on the function block).
- Activate Context help for the block diagram. Get help about the Subtract function and a number of other blocks as you wish by placing the cursor over the block.
- Search LabVIEW Help for help about the **Subtract** function.

[Table of contents]

6 Customizing LabVIEW

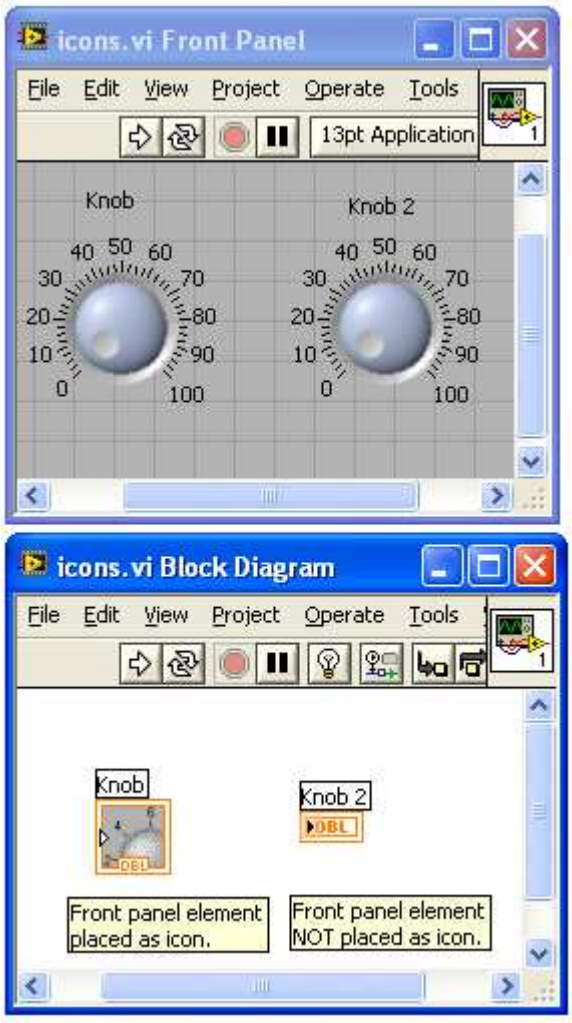
You can customize LabVIEW via the **Tools / Options** menu. This opens the dialog window shown below. Note that some of the options do not take effect until the next time you start LabVIEW (these options are marked with an asterix, cf. the figure below).



The menu Tools / Options opens the Options dialog window

Below are suggested changes of the default settings:

- **Front Panel**:
  - Disable the *Use localized decimal point* option. Disabling makes LabVIEW use *point* as the decimal separator. (If, in stead, this option is enabled, LabVIEW uses the decimal separator as defined in the settings of the Windows operating system.)
- **Block diagram**:
  - Enable *Show subVI names when dropped*. This causes names of subVIs to be shown in a label on top of the subVI icon (block). A subVI is a LabVIEW subprogram which can be used as a function in the parent program to perform a specific task.
  - Disable the *Enable auto wiring* option. This prevents LabVIEW from automatically connecting adjacent blocks. Although it seems useful to have auto wiring enables, 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.
  - Disable *Place front panel elements as icons*. This causes LabVIEW to use small terminal icons on the block diagram. If you, in stead, activate this option, the terminal icons are larger, with a mimic of the element as it appears at the front panel. The figure below shows the difference.



The difference between placing front panel elements as icons (i.e. large icon) and not (i.e. small icon)

As seen from the figure, space is saved in the block diagram by not activating Place front panel elements as icons. (I prefer saving space compared to having the more illustrative icons on the block diagram.)

- **Alignment Grid**: I suggest using alignment grid of 6 pixels on front panel, and no alignment grid on the block diagram.
- **Controls/Functions Palettes**:
  - **The Format pull-down list**: Category (Icons and Text)
  - **The Navigation Buttons pull-down list**: Label All Icons
- **Environment**: Disable the Just-In-Time Advice.

Customize LabVIEW according to the above suggestions or your own preferences.

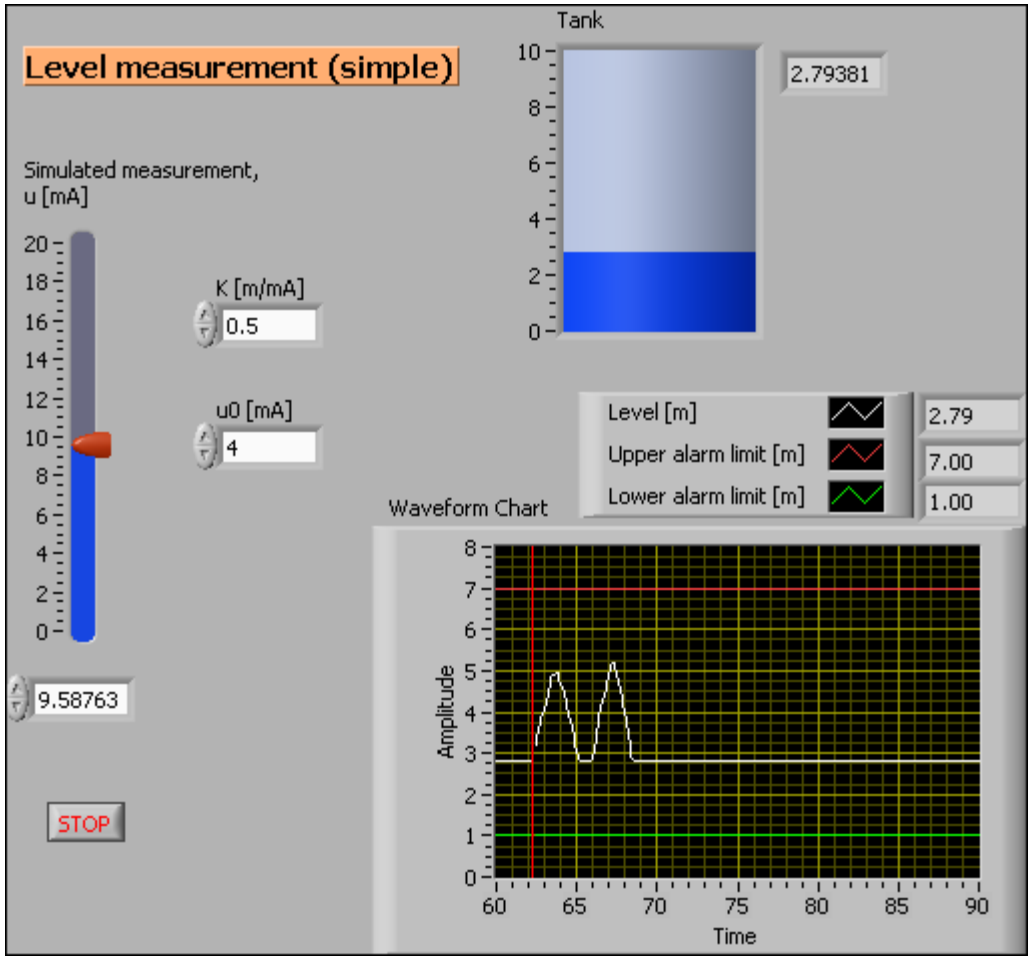
[Table of contents]

7 LabVIEW programming step-by-step

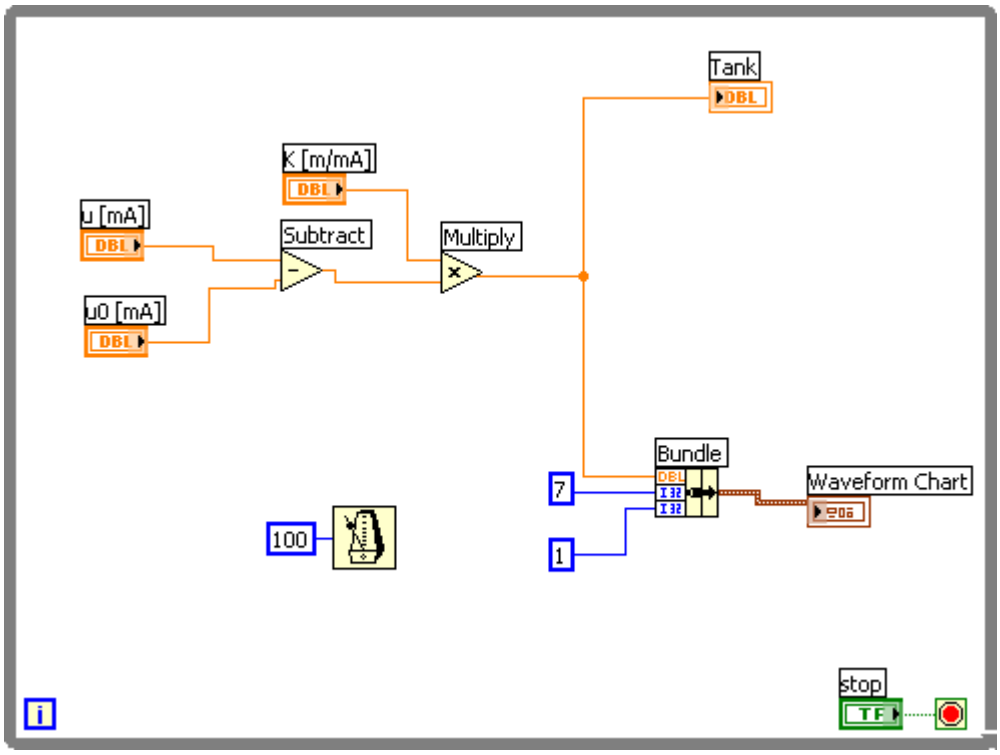
7.1 Two options for learning LabVIEW programming: Textual instructions and video

I have provided two *alternative* options for learning LabVIEW programming:

1. You can follow the detailed **written instructions** in the subsequent sections, starting from Section 7.2. The resulting VI is level_measurement.vi which was presented in Section 4. If you prefer this option, jump to Section 7.2 to continue.
2. You can watch a **video** (see below) which shows how to develop level_measurement_simple.vi which is somewhat simpler than level_measurement.vi. The Front panel and the Block diagram of level_measurement_simple.vi are shown in the subsequent figures. If you prefer this option, read the instructions in the yellow frame just beneath the figures below. (With this option you do not have to follow the descriptions starting in Secton 7.2.)



The front panel of level_measurement_simple.vi



The block diagram of level_measurement_simple.vi

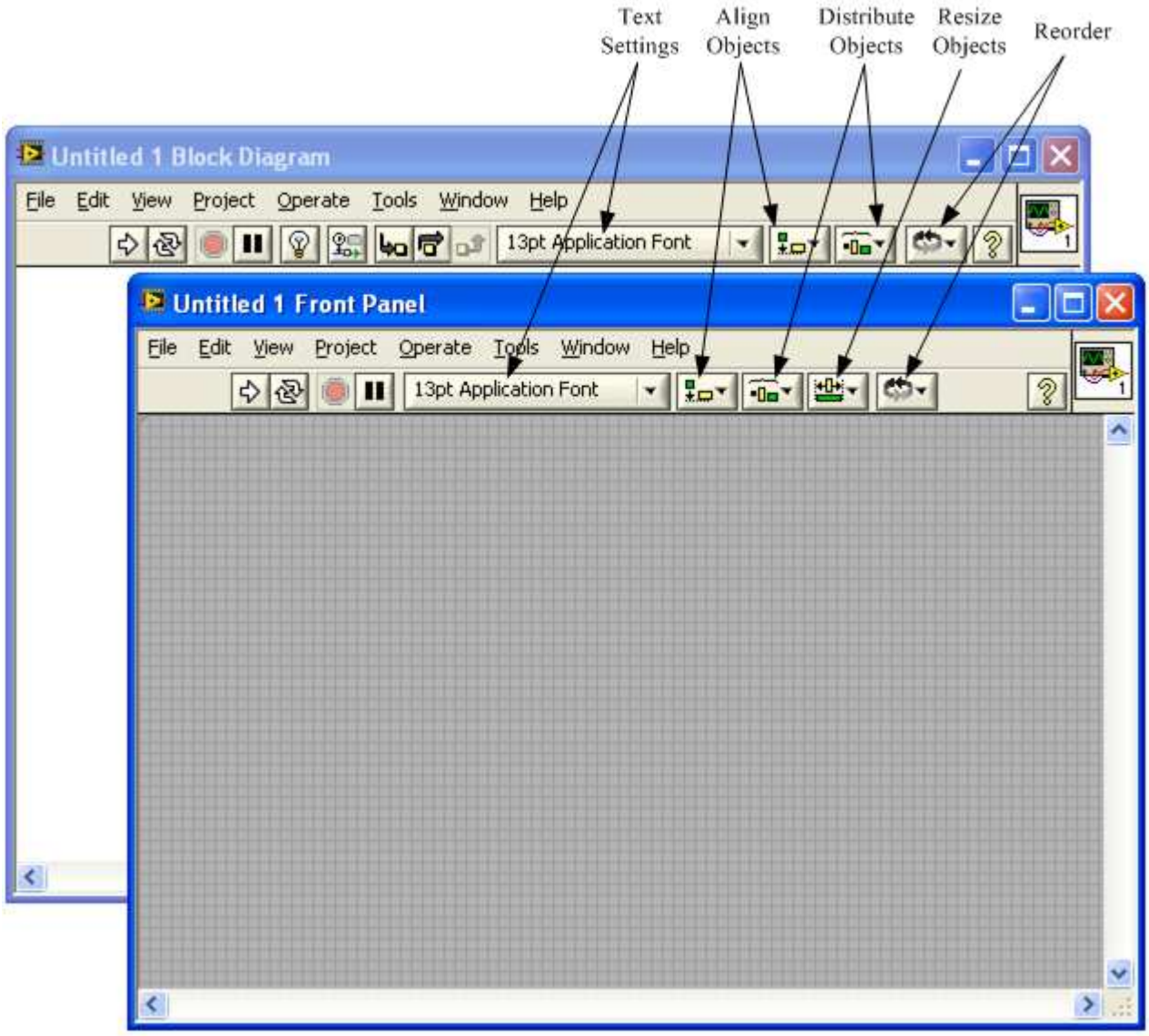
The video **level_measurement_simple.avi** (50 minutes, 500 MB) describes how to program the Front panel and the Block diagram of **level_measurement_simple.vi**.

[Table of contents]

7.2 The programming environment

To start the programming you must open a blank VI:

Open a new (blank) VI via the menu **File / New VI**, thereby opening both a blank Front panel and a blank Block diagram, see the figure below.



The menu selection File / New VI opens both a blank Front panel and a blank Block diagram

The toolbars of both the Front panel and the Block diagram contain buttons which can be used during the programming, cf. the figure above.

During the programming you will be using the following *palettes*:

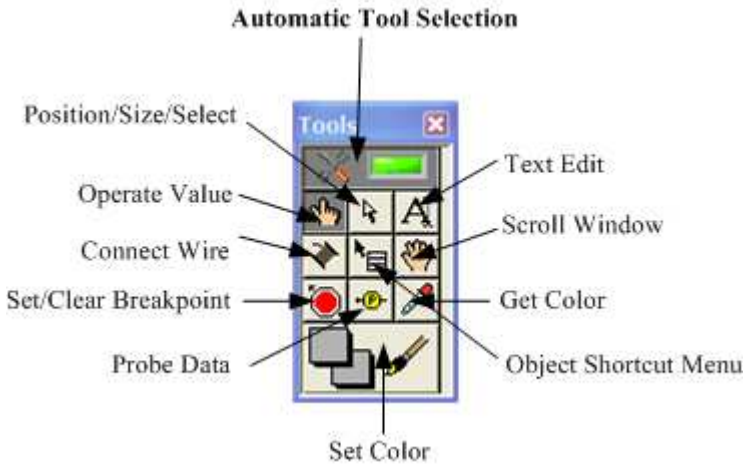
- **Tools palette**
- **Controls palette**, which may be opened *only when the Front panel is in front of the PC desktop*
- **Functions palette**, which may be opened *only when the Block diagram is in front of the PC desktop*

These palettes are described below:

Tools Palette

The Tools Palette can be opened by the menu **View / Tools Palette** (if it has not already

been opened). The figure below shows the Tools Palette.



The Tools Palette, which can be opened by the menu View / Tools Palette

Open the Tools Palette (menu **View** / **Tools Palette**).

The Tools Palette contains buttons which you can use to set the cursor in various modes:

- **Automatic Tool Selection** (default) which lets LabVIEW automatically select the cursor mode, depending on the present position of the cursor. In most cases the automatic tool selection works very fine, hence you do have to care about selecting any of the other buttons in the Tools Palette. (Except when you are going to set colors to elements. Then you will have to select the Set Color button.)
- **Operate Value**, to changing the value of elements on the front panel or the block diagram
- **Position/Size/Select**
- **Edit Text**, to enter text
- **Scroll Window**
- **Object Shortcut Menu**, to open a context dependent menu. This button is equivalent to right-clicking on the element.
- **Probe Data**, to insert probes into the block diagram. A probe is a small vindow showing the value of the wire (line) on which you have clicked. This may be useful in debugging (detecting errors).
- **Get Color**, to snap the color where you have clicked. This color can then be found in the History list in the color chart window which can be opened using the Set Color button, see below.
- **Set Color**, to change the color of elements on the front panel or the block diagram.
- **Set/Clear Breakpoint**, to set and to clear breakpoints in the block diagram. This may be used in a debugging process.
- **Connect Wire**, to connect elements in the block diagram by drawing wires (lines) between them.

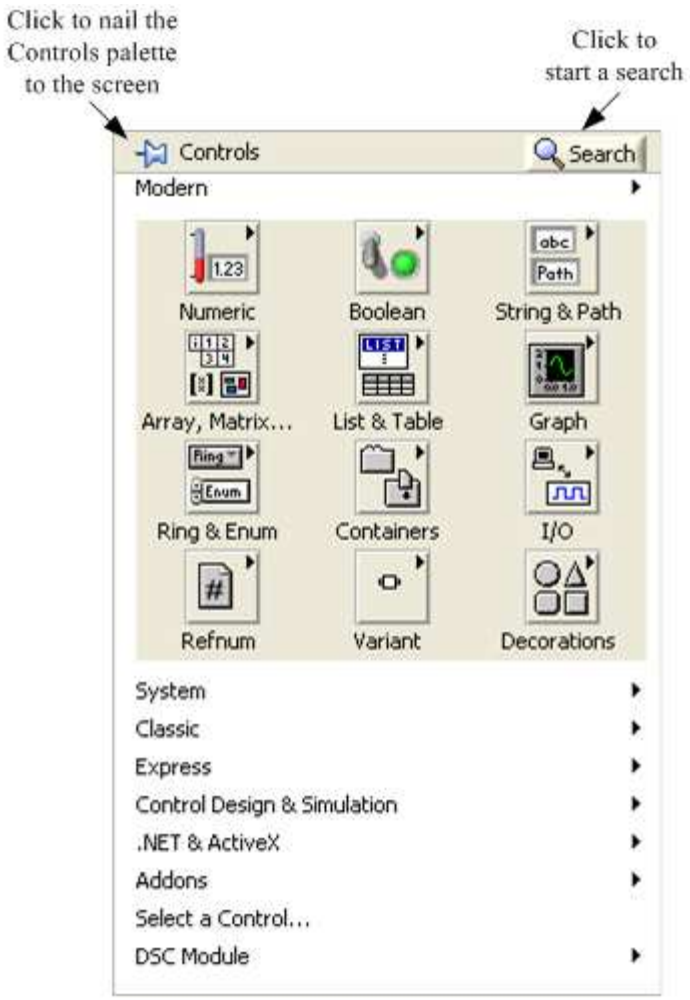
The Controls Palette

The *Controls Palette* contains sub-palettes which contains the elements that you can put on the front panel. (The Controls Palette actually contains both control elements and indicator elements. Hence, a more accurate, but longer, name would be Controls and Indicators Palette.) The Controls Palette can be opened only when the front panel is in front of the computer desktop. You can open the palette in two ways:

- By the menu **View / Controls Palette**
- By right-clicking somewhere on the front panel.

Open the Controls Palette (menu **View / Controls Palette**). To get a glimpse of the contents, browse the palettes in the two upper rows (these palettes are probably the most frequently used).

The figure below shows the Controls Palette. Note that new palettes may be added to the Controls Palette due to installation of additional modules or toolkits. Therefore, the Control Palette on your computer may look a somewhat different from the Controls Palette shown in the figure below.



The Controls Palette, which can be opened by the menu View / Controls Palette

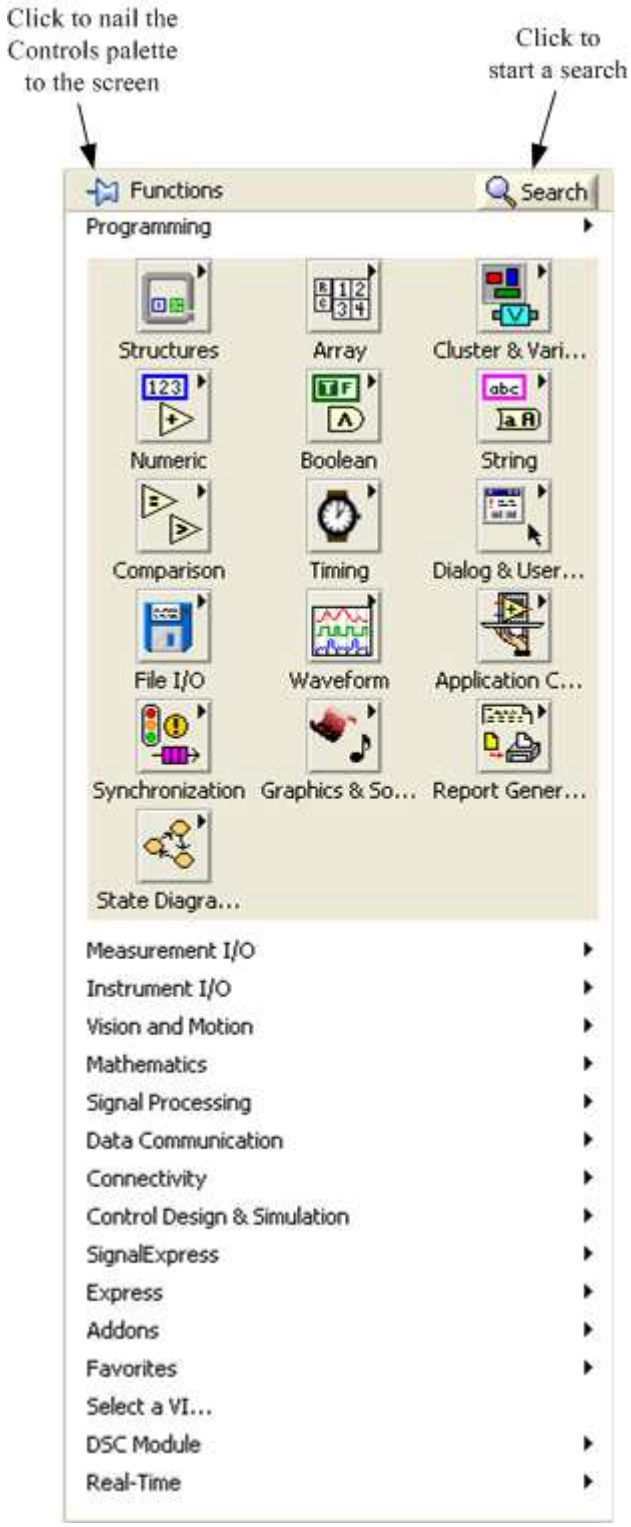
The Functions Palette

The *Functions Palette* contains sub-palettes with functions (blocks) that you can put on the block diagram. The Functions Palette can be opened only when the block diagram is in front of the computer desktop. You can open the palette in two ways:

- By the menu **View / Functions Palette**
- By right-clicking somewhere on the Block diagram.

> First, put the *Block diagram* of your VI in front (e.g. by shortcut Ctrl+E). Then open the Functions Palette. To get a glimpse of the contents, browse the palettes in the three upper rows (these palettes are probably the most frequently used).

The figure below shows the Functions Palette. Note that new palettes may be added to the Functions Palette due to installation of additional modules or toolkits. Therefore, the Functions Palette on your computer may look somewhat different from the Functions Palette shown in the figure below.



The Functions Palette, which can be opened by the menu View / Functions Palette

[Table of contents]

7.3 General programming guidelines

Suggested procedure

The programming of a LabVIEW program (a VI) may follow this procedure:

1. Elements are placed on the front panel, and then configured by setting various properties (e.g. labels are defined, scales are defined, etc.).
2. The functionality of the VI is implemented in the block diagram.
3. Additional front panel elements may be added, and additional block diagram code may be added. Note that it is possible to enter front panel elements via the block diagram. This may be quite useful since LabVIEW then automatically creates front panel elements of the correct data type. (This will be demonstrated later.)

Remember to *save your work frequently*. It may be wise to save the VI under slightly different names as the program evolves, thereby making it easier to retrieve earlier version of the program if the development has taken an adverse direction.

Testing

I have seen many complicated VIs that does not work correctly when they are run the first time, and due to the complexity, it may be very difficult and time consuming to find the errors. In some cases, as if the VI is to control some physical systems, it may be *dangerous* to run a VI that has not been tested appropriately.

A VI that you develop should not be regarded as complete (finished) until it has been tested! You should test the VI continuously, i.e. at the different stages of the development. The final test should, if possible, be accomplished by some other person than you. (When I work on projects, I always let the user run the VI to give me feedback before I regard the VI finished.)

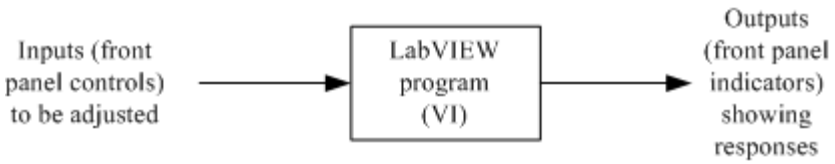
How do you actually test a VI? It depends on the nature of the VI:

- The VI does not involve communication with any external system
- The VI does involve communication with an external system

These two cases are deescribed below.

The VI does not involve communication with any external system

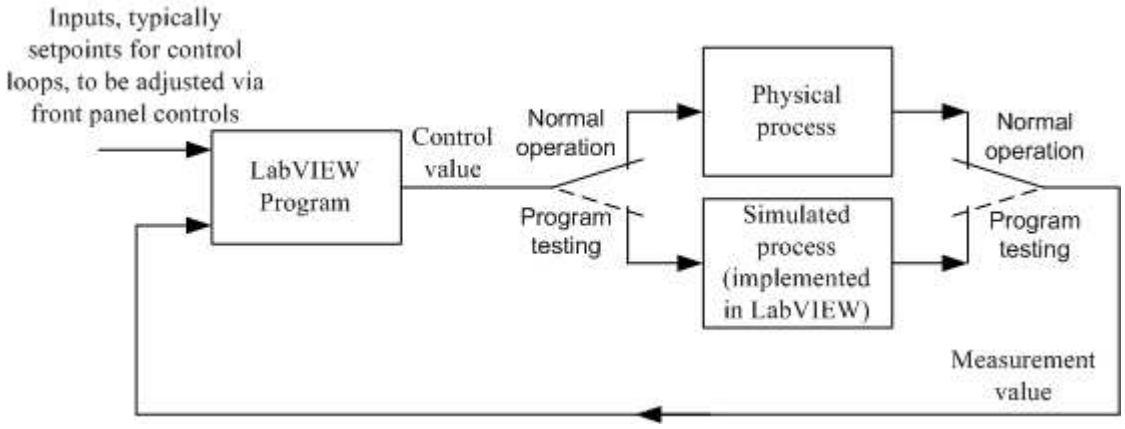
To test the VI, adjust the front panel elements over all of their ranges of values, including extremal values, so that alarm setting etc. may be triggered. From the observed responses, conclude if the VI, or the part of the VI under concern, works correctly. This testing principle is illustrated in the figure below.



Testing a LabVIEW program can be done by adjusting the inputs over all theirs ranges of values

The VI does involve communication with an external system

Examples are VIs involving I/O (input/output) with physical processes in control applications where the VI calculates control signals as functions of process measurements. When testing the VI you substitute the physical process by *a simulated process*. The controller function controls the simulated process, and the simulated process feeds measurements back to the controller. This testing principle is illustrated in the figure below. The simulated process can be in the form of a transfer function model or a state-space model representing the differential equations describing the physical process. To learn more about creating simulators and using simulations for testing control systems, see Introduction to LabVIEW Simulation Module.



Testing a LabVIEW program for control using simulation

[Table of contents]

7.4 Developing the VI

7.4.1 Well-known editing tools apply!

The editing tools that you probably are familiar with from text editing etc. also apply in LabVIEW. Here is a list over these tools:

- **Undo**: Menu **Edit** / **Undo**, or keyboard shortcut **Ctrl**+**z**.
- **Redo**: Menu **Edit** / **Redo**, or keyboard shortcut **Ctrl**+**Shift**+**z**.
- **Cut**: Menu **Edit** / **Cut**, or keyboard shortcut **Ctrl**+**x**.
- **Copy**: Menu **Edit** / **Copy**, or keyboard shortcut **Ctrl**+**c**.
- **Paste**: Menu **Edit** / **Paste**, or keyboard shortcut **Ctrl**+**v**.
- **Delete**: Menu **Edit** / **Delete**, or Delete key. (Using Delete removes the element without saving it on the clipboard, while Cut removes the element while saving it on the clipboars for a possible later reinsertion.)
- **Fine positioning** of elements one pixel at a time can be made using the arrow buttons on the keyboard.

Now, let us create level_measurement.vi!

[Table of contents]

7.4.2 Starting with the Front panel

It is assumed that the front panel of a blank VI is in front of the desktop of your PC.

Save the blank VI with name **my_level_meas.vi** in any folder you prefer.

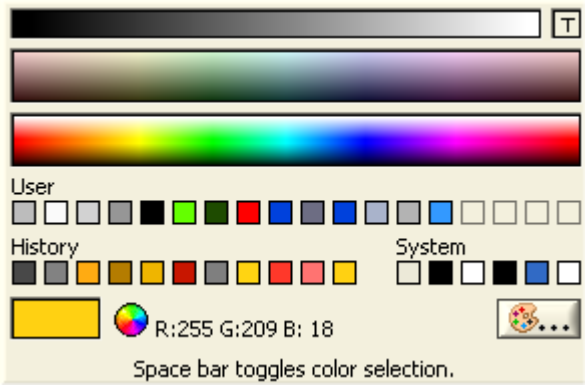
We will now insert elements on the Front panel and configure them according to level_measurement.vi.

Free text on the Front panel

Starting with the free text "Level Measurement":

- *Add* the free text "Level Measurement" on the front panel as follows: **Double-click at an appropriate place on the front panel / Type the text "Level Measurement"**.
- *Configure* the text element: **Select the text with the cursor / Click the Text Settings button in the toolbar (cf. this figure) / Select Size / Select 18.**
- **Text Settings button / Style / Bold.**
- Set the background color of the text field: **Open the Tools palette (menu Tools Palette) / Select the Set Color button on the Tools palette thereby opening the Color palette shown in the figure below / Right-click on the field behind the text / Select a proper color (for example an orange color as in level_measurement.vi).** *After you have finished color setting click the Automatic Tools Selection button on the Tools palette*.

Save the VI.



Color palette opened via the Set Color button on the Tools palette

Note: You can set the color of every part of an element on the front panel the same way

as you set the color of the text field above, i.e. **Tools palette / Color button / Right-click on the part on which you want to change the color**.

You must also add the free text y = K*(u-u0) to the front panel:

Enter the free text "y = K*(u-u0)" on the front panel (with no particular configuration of the text).

Save the VI.

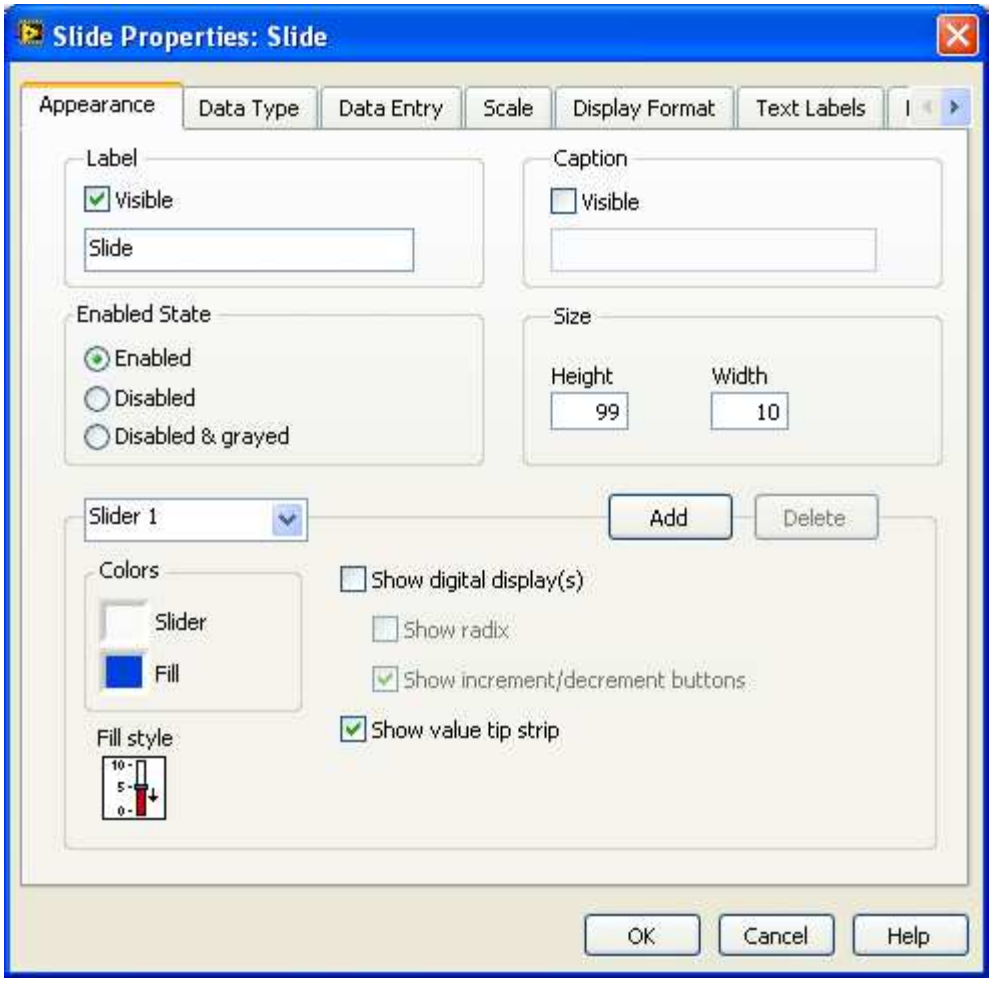
The Vertical pointer slide

Next, we insert and configure the Vertical pointer slide of the simulated measurement signal, u, cf. level_measurement.vi:

- *Insert* the Vertical pointer slide of the simulated measurement signal, u: **Right click at an appropriate place on the front panel (causing the Control palette to appear) / Select Numeric palette / Select Vertical Pointer Slide**, and drop it on the front panel.
- *Configure* the element (Vertical Pointer Slide): **Right-click on the element / Select Properties in the context menu thereby being opened, thereby opening the Properties window which is shown in the figure below.** Make the following settings (keeping other settings unchanged):
  - **Appearance tab / Label**: Enter following text: "u". Disable (uncheck) Visible.
  - **Appearance tab / Caption**: Enable (check) Visible. Enter following text: "Simulated measurement signal u [mA]". (You can break a text line by clicking Shift + Enter.)
  - **Appearance tab**: Enable Show Digital Display(s).
  - **Appearance tab / Color / Slider**: Select any color you want.
  - **Appearance tab / Color / Fill**: Select any color you want.
  - **Data Entry tab / Uncheck Use Default limits / Minimum** = 4. This limits the adjustable value.
  - **Data Entry tab / Maximum** = 20.
  - **Data Entry tab / Increment** = 0.1, thereby setting the incremental change as the arrow buttons of the front panel element are clicked.
  - **Scale / Scale Style**: Select lower left option.
  - **Scale / Scale Style / Select Range / Minimum**: 4. This sets the maximum value of the scale, but this setting is independent of the Data Range / Minimum setting above.
  - **Scale / Scale Style / Select Range / Maximum**: 20.
  - **Scale / Format and Precision**: You can keep the default settings here, but, to see the options, open the tab.
    - Select Scale in the pulldown list. Ensure that the following settings are made:

        - Automatic formatting
        - Digits: 6
        - Precision Type: Significant digits
        - Hide trailing zeroes
      - Select Digital Display 0 in the pulldown list. Then do the same settings as above.
    - Save the settings by clicking the OK button in the Properties window.
- Adjust the size of the Vertical pointer slide by moving the cursor over the element and dragging one of the squared handles that appear.
- Move the Digital display to a position under the slide. (This is just one of my own habits. You may of course place the Digital display wherever you want :-). *Note*: You can fine-position any element using the keyboard arrows. Try it!
- Move the Caption so that is appears nicely situated relative to the slide.

Save the VI.



Property window of the Vertical Pointer Slide

About the default value of an element

The default value of an element is the value that the element has just after the VI is loaded into the RAM (random access memory) of the computer, that is, just after you have opened the VI from the permement memory, e.g. the hard disk. Thus, the default value is a start value when the VI is run the first time after being opened. But is the default value also the starting value of en element if you stop the VI and then start it again without closing it? No! In that case the starting value is the value that the element had at the end of the previous run. (So, LabVIEW remembers the latest value.) LabVIEW uses zero as the default default value! Default values can be set in couple of ways:

- Adjust the value of u to some value, say 10. To define the *current value* as the default value: **Right-click on the element / Select Data Operations / Make Current Value Default**.
- To define the current values as default values for *all elements on the front panel* simultaneously: **Menu Edit / Make Current Values Default**. (Try this now, although the front panel contains only one element, so far.)

Save the VI.

You can at any time *reinitialize* the value of an element to its default value:

- Adjust the value of u to some other value than the default value, e.g. 5. To reinitialize the value of u to its default value: **Right-click on the element / Data Operations / Reinitialize to Default Value**.
- Adjust the value of to some other value than the default value, e.g. 5. To reinitialize *all elements* to their default values simultaneously: **Menu: Edit / Reinitialize Values to Default**. (Try it now, although the front panel contains only one element, so far.)

Save the VI.

The Gain K element

Next, we insert and configure the Gain K element, cf. level_measurement.vi:

- Insert the Gain K Numeric control: **Right click at an appropriate place on the front panel (causing the Control palette to appear) / Select Numeric palette / Select Numeric control**, and drop it on the front panel.
- Configure the element (numeric control): **Right-click on the element / Select Properties**. Make the following settings in the Properties window (keeping other settings unchanged):
  - **Appearance tab / Label**: Enter the following text: "K". Disable (uncheck) Visible.
  - **Appearance tab / Caption**: Enable (check) Visible. Enter the following text: "Gain K [m/mA]".

Save the VI.

The Zero u0 element

Next, we insert and configure the Zero u0 element, cf. level_measurement.vi:

- Insert the Zero u0 Numeric control: **Right click at an appropriate place on the front panel (causing the Control palette to appear) / Select Numeric palette / Select Numeric control**, and drop it on the front panel.
- Configure the element (numeric control): **Right-click on the element / Select Properties**. Make the following settings in the Properties window (keeping other settings unchanged):
  - **Appearance tab / Label**: Enter following text: "u0". Disable (uncheck) Visible.
  - **Appearance tab / Caption**: Enable (check) Visible. Enter following text: "Zero u0 [mA]".

Save the VI.

Aligning, distributing, resizing objects

Sometimes you want to align a specific edge of front panel elements to e.g. a common bottom line or to distribute elements with equal space between them or to resize elements so that they get equal size. This can be done using the Align Objects button, the Distribute Objects button, or the Resize Objects button, respectively, on the front panel toolbar.

Align the left edges of the Gain K numeric control and the Zero u0 numeric control: Select both these controls (using the mouse). Then click the Align Object button on the toolbar and select Left Edges in the palette that is opened..

Save the VI.

The Tank indicator

Then, we insert and configure the tank indicator, cf. level_measurement.vi:

- Insert the tank indicator: **Right click at an appropriate place on the front panel (causing the Control palette to appear) / Select Numeric palette / Select Tank , and drop it on the front panel**.
- Configure the element (numeric control): **Right-click on the element / Select Properties**. Make the following settings in the Properties window (keeping other settings unchanged):
  - **Appearance tab / Label**: Enter following text: "y_tank". Disable (uncheck) Visible.
  - **Appearance tab / Caption**: Enable (check) Visible. Enter following text: "Level y [m]".
  - **Appearance tab**: Enable Show Digital Display(s).
  - **Scale tab / Scale Range / Minimum**: 0.
  - **Scale tab / Scale Range / Maximum**: 8.
  -

Save the VI.

What about the Stop button?

Looking at the Front panel of level_measurement.vi, there is still one element on the front panel that we have not put on the Front panel of my_level_meas_simple.vi, namely the Stop button. Although we could very well have placed it on the Front panel as with the other elements, we will in stead place it there via the Block diagram. It is generally useful to be aware of this possibility.

Now, it's time to make the VI *work* properly! The functionality of the VI is implemented in the *Block diagram*.

[Table of contents]

7.4.3 Then developing the Block diagram

The Block diagram so far

The Block diagram of your **my_level_measurement.vi** should be similar to the Block diagram shown in the figure below. So far, the Block diagram just contains the terminals of four Front panel elements. Of course, the Block diagram is completely undeveloped.

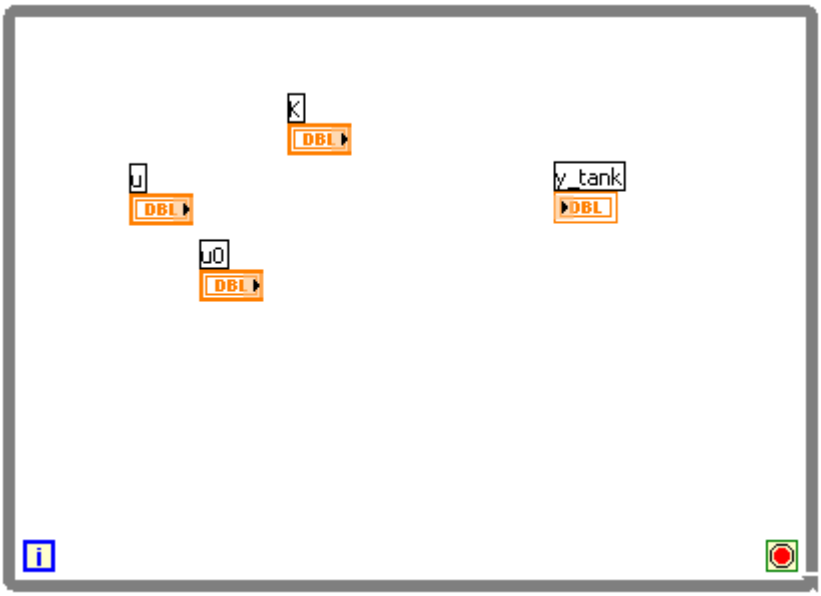
The (completely) undeveloped Block diagram of my_level_measurement.vi after (almost complete) Front panel development

Adding a While loop

Our VI shall run continuously with a cycle time of 0.1 s. Therefore, we need to put the code inside a While loop.

Insert the While loop: **Right-click somewhere up to the left in the Block diagram (causing the Functions palette to appear) / Select Structures / Select While Loop**, and click to drop it on the Front panel and with the mouse button still down expand the While loop so that it embraces the four terminals.

The result should be as in the figure below.



The Block diagram after the While loop has been inserted

Save the VI.

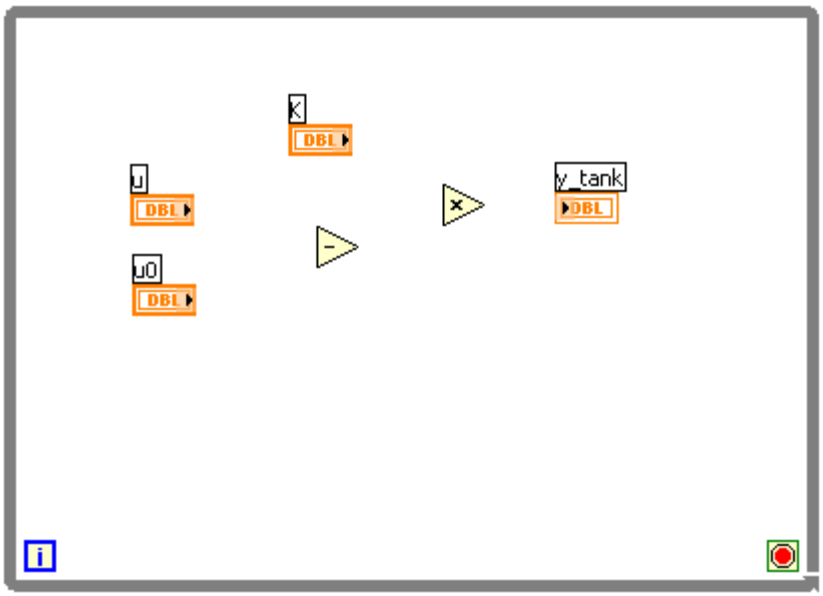
Note: It is essential that code that shall be inside a While loop, programmatically, actually *appears* (is seen) inside the loop.

Inserting functions

Next, we insert two mathematical functions - the **Subtract** and **Multiply** functions into the Block diagram.

- Insert the Subtract function: **Right-click at proper position in the block diagram (causing the Functions palette to appear) / Select Numeric / Select Subtract, and click to drop the block on the block diagram**.
- Insert the Multiply function: **Right-click at proper position in the block diagram (causing the Functions palette to appear) / Select Numeric / Select Multiply, and click to drop the block on the block diagram**.

The result should be as shown in the figure below.



The Block diagram after the Subtract and the Multply functions have been inserted

Save the VI.

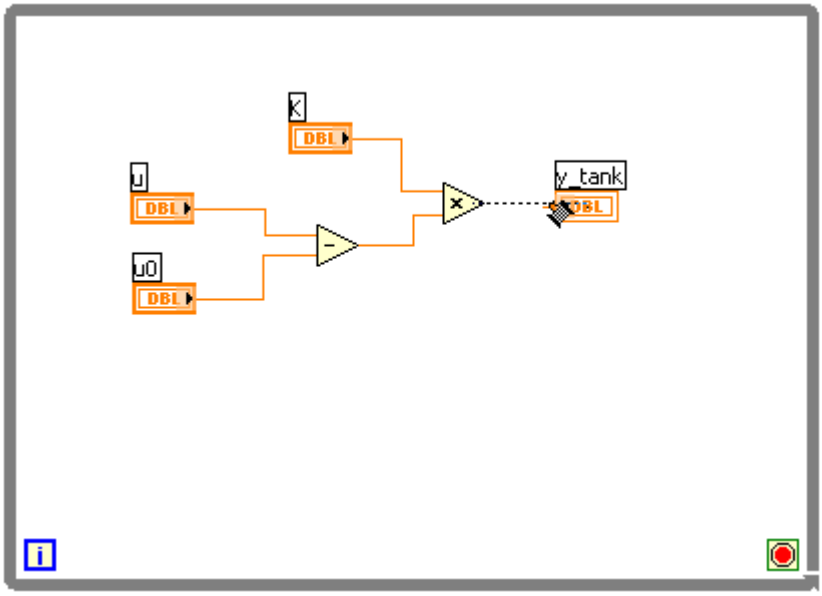
Wiring together elements

Next, we draw wires between the blocks on the Block diagram.

Draw the wires in the Block diagram: Move the cursor to a connection point of either a terminal or a function block, causing the cursor icon to be a spool. Then (left-)click the mouse, and draw a wire between the elements that you want to

connect.

The result should be as shown in the figure below.



The Block diagram after wires have been drawn using the Connect wire tool

Save the VI.

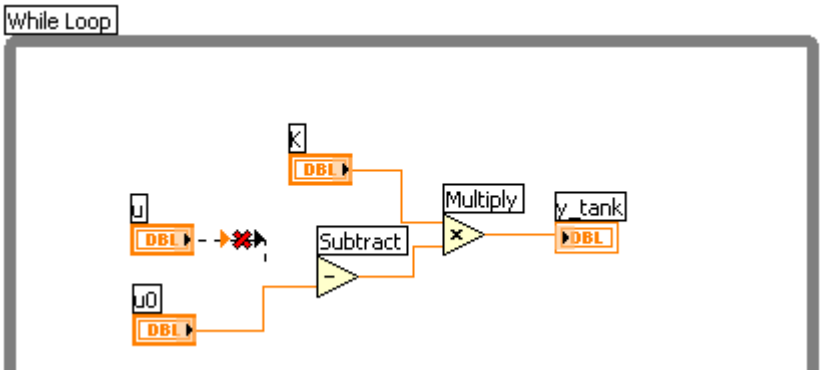
Wiring tips

Here are a number of useful tips about drawing wires:

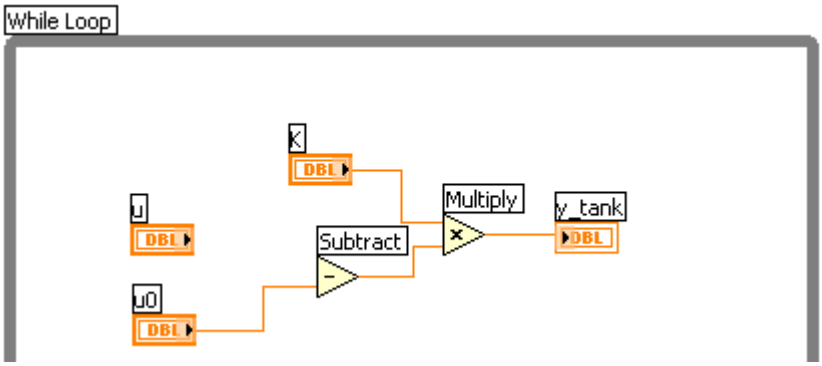
- You can draw a wire of any pattern by releasing the mouse button and then moving the cursor before again clicking the mouse and continuing drawing.
- You can delete a wire or a part of a wire in a normal way, i.e. by selecting it with the mouse pointer (single- clicking selects one line segment of the wire, while double-clicking selects the whole wire) and pressing the Delete key.
- If you delete a part of a wire, a *broken wire* remains. You can delete a broken wire using the Delete key, but you can also use the menu **Edit / Remove Broken Wire**. It turns out that removing broken wires is a frequent operation, and therefore you can save yourself some time by using the shortcut **Ctrl + b**.
- If you are not happy with the wire route you have drawn, you can let LabVIEW redraw the wire automatically with a resulting "optimal" path: **Right-click on the wire / Clean Up Wire**.

Let us try out these tips!

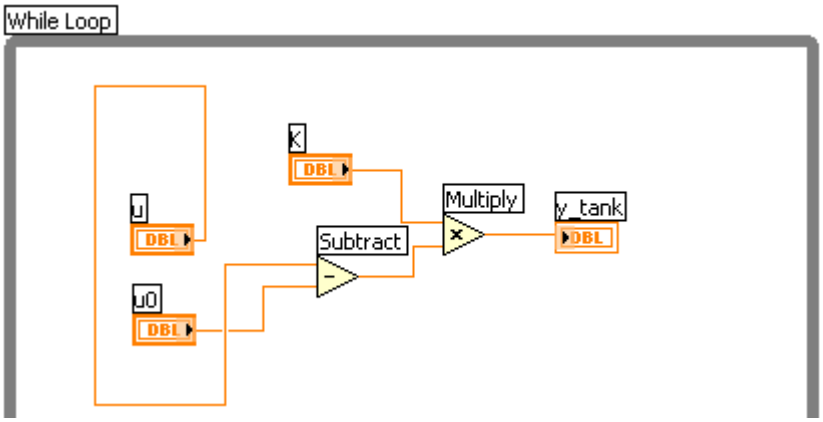
Delete the part of the wire between the u terminal and the **Subtract** function that enters the latter function. The resulting Block diagram is shown in the figure below.



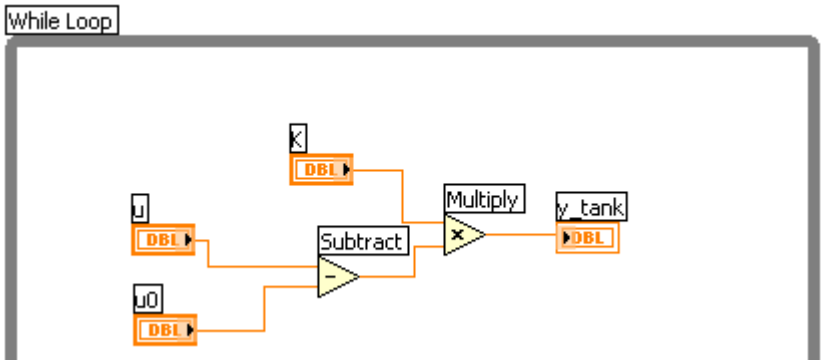
Remove the broken wire going out from the u terminal using the shortcut Ctrl + b. The resulting block diagram is shown in the figure below.



Draw a "crazy" wire from the u terminal to the **Subtract** function, see the figure below.



Make LabVIEW clean up the long wire between the u terminal and the **Subtract** function by right-clicking the wire and selecting **Clean Up Wire** in the menu that is opened. The resulting block diagram is shown in the figure below.



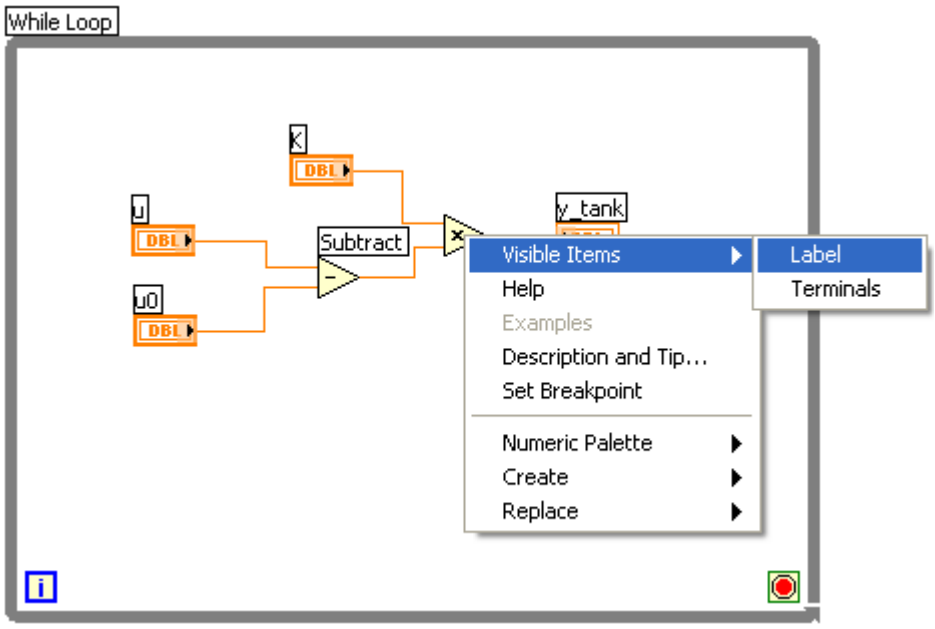
Save the VI.

By now, your VI should be similar to level_measurement_1.vi. If you want, you may proceed with level_measurement_1.vi as the starting point (saving it as my_level_meas.vi in a proper folder).

Making labels visible

To make your VI even more self-documenting *labels* of functions and program stucture elements can be made *visible*.

Make the labels of the While loop frame, the **Subtract** and the **Multiply** functions *visible*: **Right-click on the element / Visible Items / Label**, see the figure below.



The Block diagram after wires have been drawn using the Connect wire tool

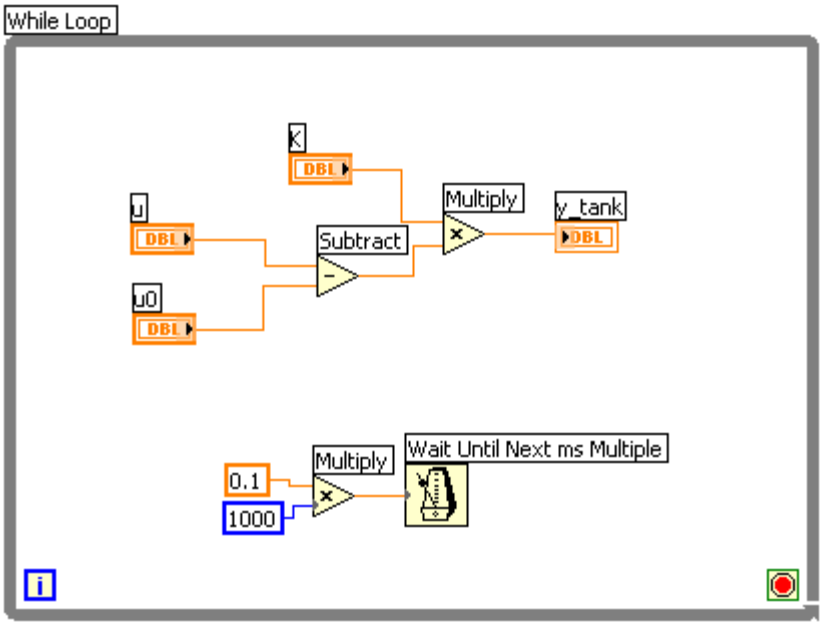
Save the VI (my_level_measurement.vi).

Adding the Metronome (the Wait Until Next ms Multiple function)

Next, we add the **Wait Until Next ms Multiple** function, which we for simplicity denote the **Metronome** function, to implement a cycle time of 0.1s of the While loop. (The **Metronome** function was explained here.) Since the **Metronome** must have the cycle time in units of millisensonds at its input, 0.1 is multiplied by 1000, and the product is wired to the **Metronome**.

- Insert the **Metronome** function (the **Wait Until Next ms Multiple** function) from the **Functions palette / Timing subpalette**.
- Insert the **Multiply** function from the **Functions palette / Numeric subpalette**.
- Insert two numeric constants from the **Functions palette / Numeric subpalette**. Then enter values for the constants by double-clicking them and typing values.
- Wire together the constants and the functions according to the figure below.
- Make the labels of the functions visible.

The result should be as shown in the figure below.



The Block diagram including the Metronome function (Wait Until Next ms Multiple function)

Save the VI.

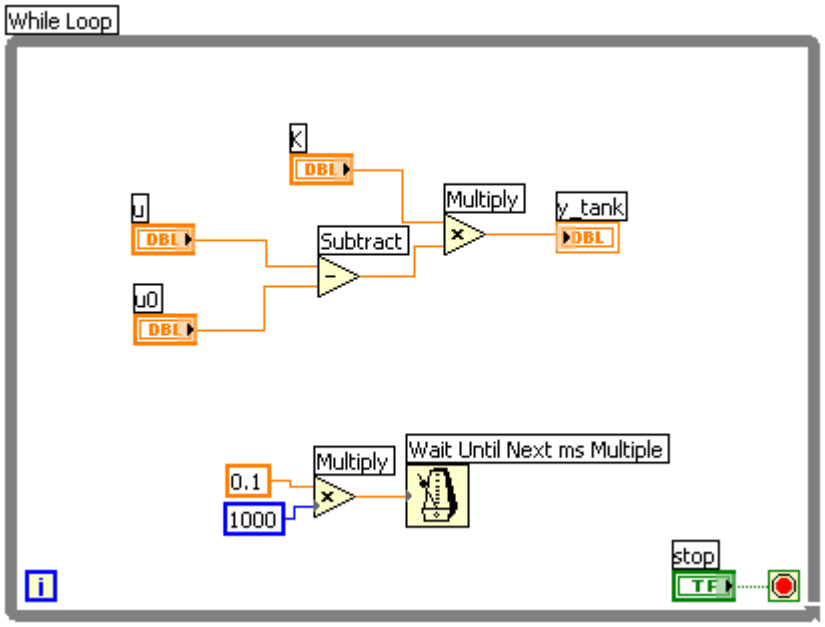
Creating the Stop button via the block diagram

Now we will create a **Stop** button on the Front panel. As mentioned earlier, we could have put a Stop button on the front panel by selecting from the **Controls palette** /

Boolean subpalette, and then wire its terminal to the **Loop condition** terminal in the Block diagram. In stead we will create the button *from the Block diagram*. It is my experience that this is very useful in many cases, since LabVIEW then automatically selects the correct data type (it may be more cumbersome to create manually e.g. a cluster of arrays of numeric element at the Front panel than to let LabVIEW create a correct Front panel element).

Create a boolean Front panel element whose terminal will be wired to the Loop condition terminal: **Right-click on the Loop condition terminal / Create Control**.

The resulting Block diagram is shown in the figure below.

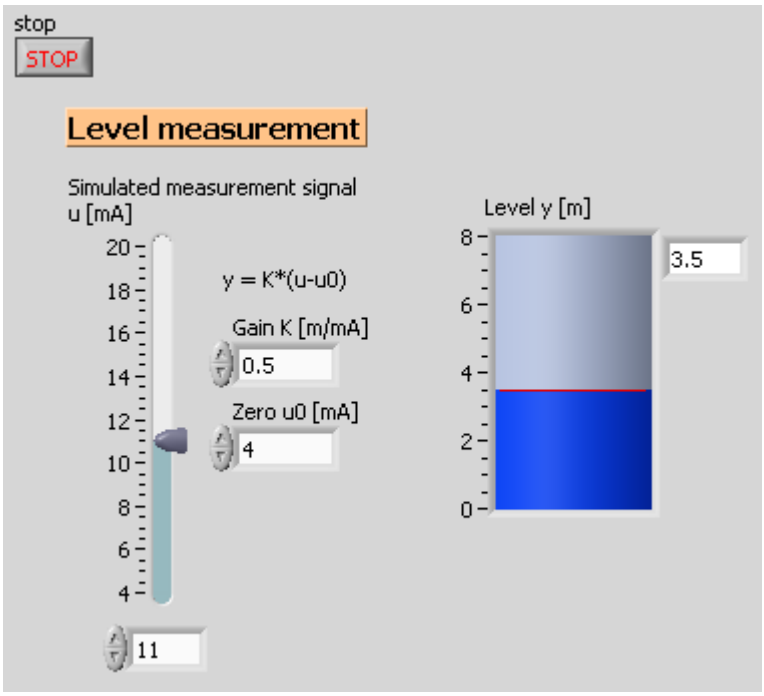


The Block diagram including the boolean stop terminal wired to the Loop condition terminal

What was actually created on the Front panel?

Open the Front panel of the VI (Ctrl+E on the keyboard).

The figure below shows the Front panel.

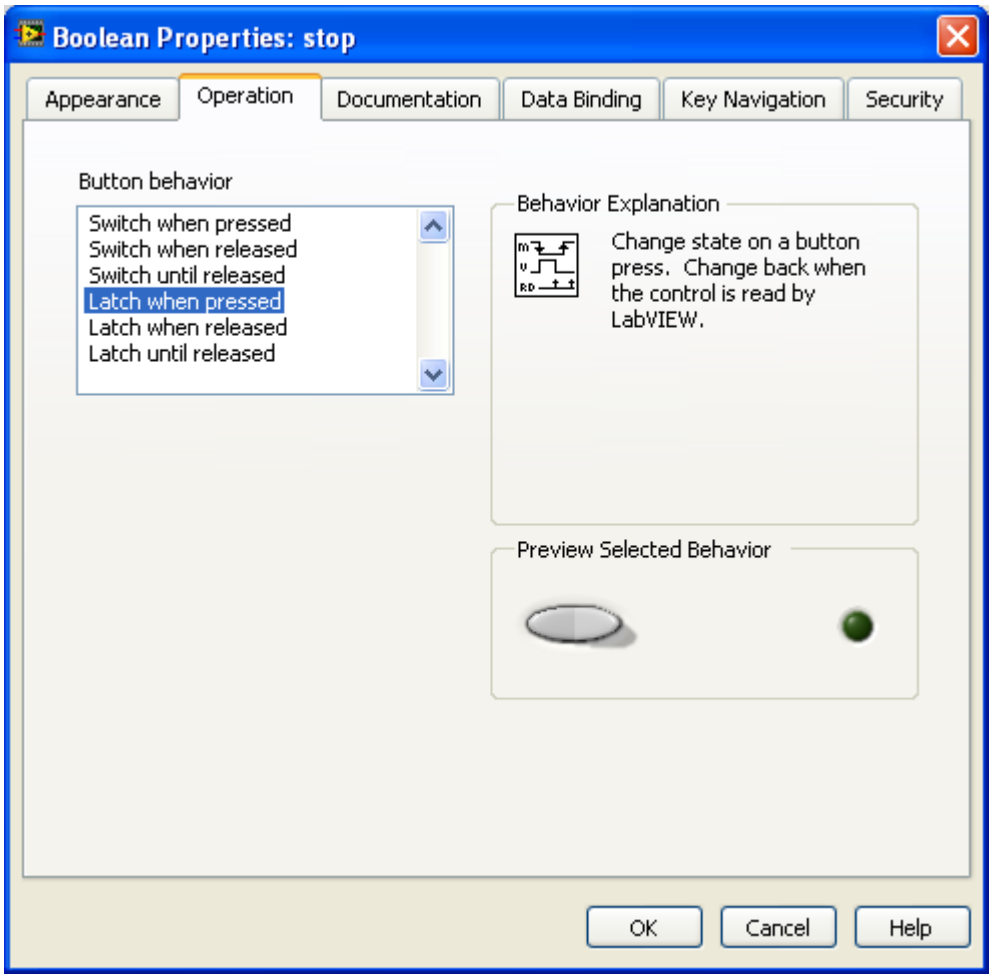


The Front panel containing the Stop button automatically created by LabVIEW

So, the Stop button was placed in the upper left corner of the Front panel. (This is the default position of Front panel elements that are automatically created.) Let us do a few cosmetic changes on the button. Furthermore, we have to make the button *operate correctly* - a button is not just a button! There are a number of ways a button can operate: It may change its state immediately after having been pressed, or when the press is released. And it may remain in the down-position, or it may pop up as if it is forced by a spring.

Move the Stop button to the bottom of the Front panel.

Do the following changes to the appearance of the button via its Property window which you open by **right-click on the Stop button / Selecting Properties in the menu**. The Property window is shown below (with the Operation tab selected, however it is the Appearance tab that is selected by default).



The Operation tab of the Property window of the Stop button

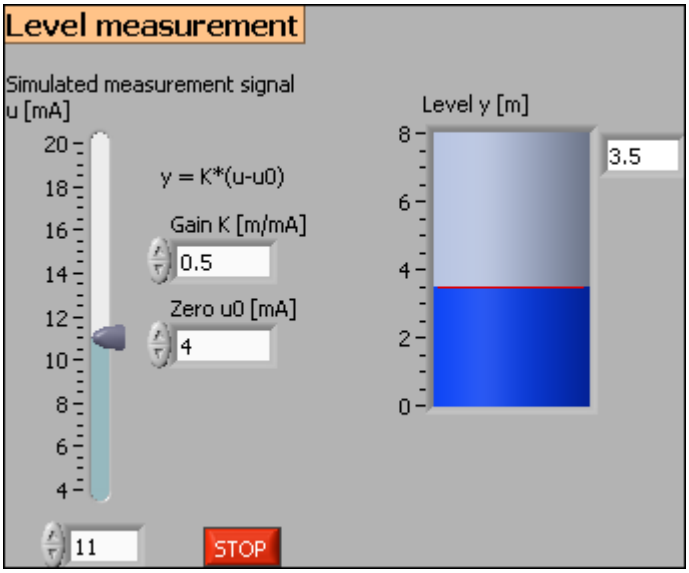
- Hide the label.
- Set both the On color and the Off color to red.
- Set the color of the text on the button (boolean text) to white.
- Set the operation mode to "Latch when pressed", see the figure below. Preview the operation of the button by clicking the Preview Selected Behaviour button in the Property window.
- A question to you: Why is not a good idea to select the "Switch when pressed" operation mode or the "Switch when released" operation mode? (Try the preview function.)
  Answer: The Stop should pop out automatically after the button has been clicked, so that the button will not stay in the stop state if the VI is run again (it would then not start).

[Table of contents]

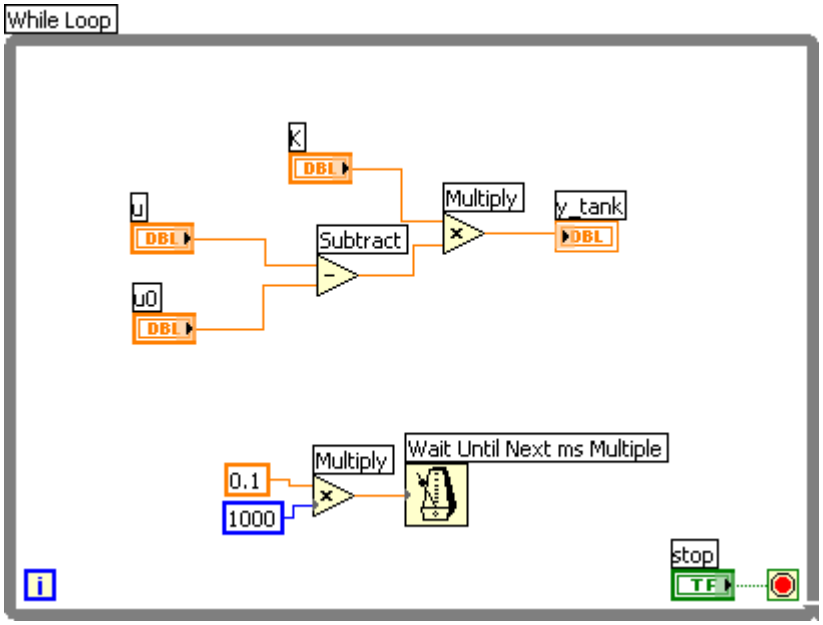
7.4.4 Debugging and testing

By now your **my_level_measurement.vi** should similar to **level_measurement_2.vi**. The figures below show the Front panel and the Block diagram of

level_measurement_2.vi. (If you want, you may proceed with **level_measurement_2.vi** as the starting point, starting by saving it as my_level_measurement.vi in a proper folder.)



Front panel of level_measurement_2.vi



Block diagram of level_measurement_2.vi

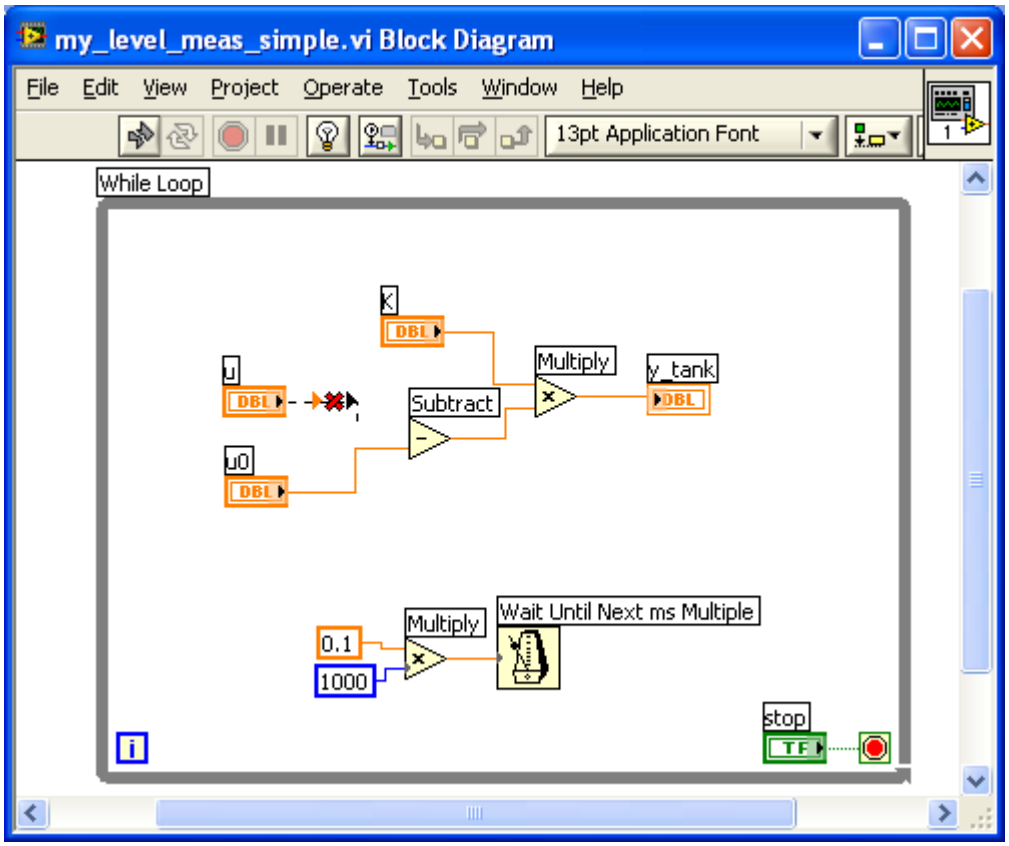
Hopefully, the VI - **my_level_meas.vi** - works as assumed. But the only way to confirm it, is by testing it, and of course correcting the errors, which is denoted *debugging* in computer science.

Detecting and correcting syntax errors (debugging)

Before you can test the VI it must be free of *syntax errors*, that is, it must have no technical errors. Actually, a VI will not run if it has syntax errors. LabVIEW helps a lot in

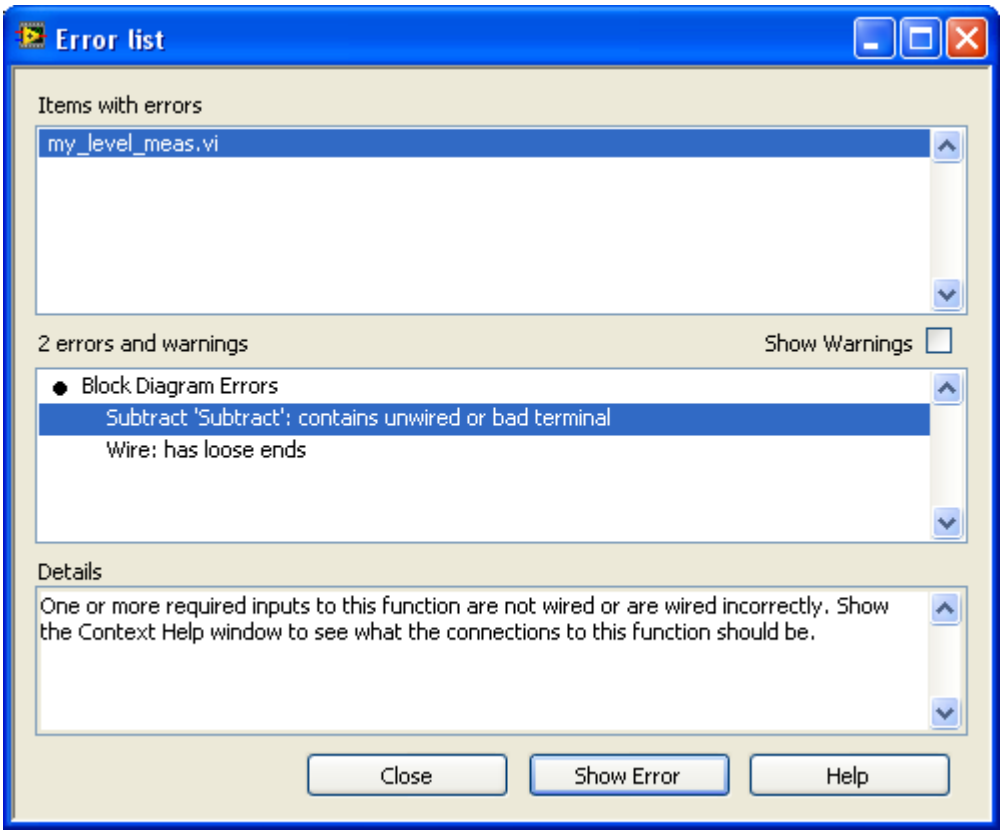
finding syntax errors. Let us try!

Introduce a syntax error in the VI by deleting a part of the wire between the u terminal and the **Subtract** function, see the figure below. Observe that the Run button is *grey and broken*!



The Run button is grey and broken due to a syntax error

Click the Run button to try to run the VI. Since the VI has an error the Error List window is opened, see the figure below.



Error List window

Now, double-clicking the first of the two errors in the error list (see figure above), causing LabVIEW to zoom into the error in the block diagram. Fix the error (by completing the wire)! Observe that the Run button now again is normal (i.e. white and not broken).

Testing the VI to check its functionality

Even though the VI has no syntax errors it may, of course, still not work correctly due to *functional errors*. It is necessary to test the VI you should operate it through all possible operating ranges and under the various operation conditions while observing the responses and judging if the VI behaves correctly.

With our simple VI we can perform the test as follows:

Start (run) and stop the VI a couple of times to check that these operations work correctly.

Run the VI. Adjust the values of u, K and u0 to some integer values (so that it is easy to check the calculations by hand), e.g. u = 10, K = 0.5, and u0 = 4. The VI then produces y = 3. Check this by manual calculations according to the implemented formula y = K*(u-u0).

With complicated VIs you should test the VI at several stages of its development. (It is like checking that the lower part of the building is safe and strong before you start building the next part upon it.)

While performing functional tests probes, execution highlighting, and stepwise execution may be useful tools to detect errors. These tools are described (and practiced) earlier in this document.

[Table of contents]

7.4.5 VI development continued

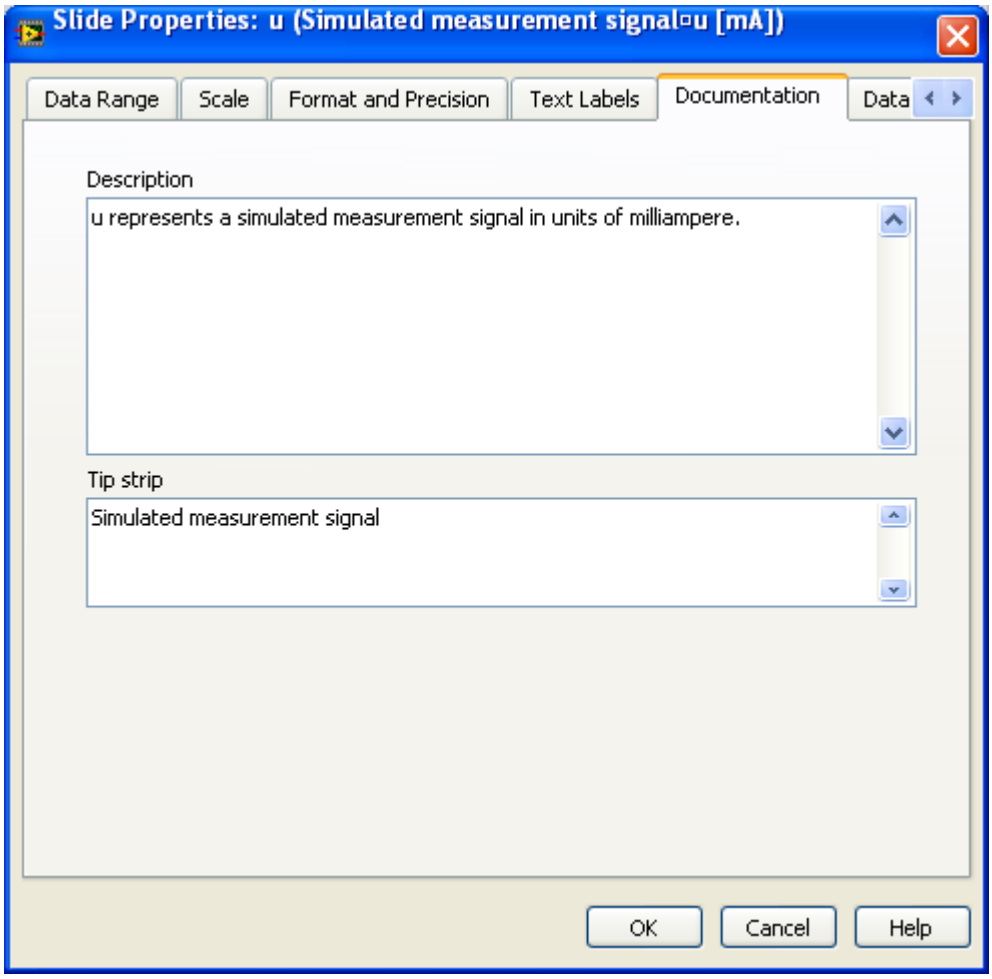
Now that you have got some experience in debugging and testing a VI, we will continue developing the complete level_measurement.vi. The following elements will be added to the VI:

- **Description and Tip strip** of an element
- **While loop iteration number** (to be displayed in a numeric indicator on the Front panel)
- **Clock** (for showing elapsed time)
- **Cluster** (for collecting the K and u0 elements into one cluster element)
- **Local variable**
- **Boolean indicator**
- **Comparison functions**
- **String indicators**
- **Chart** (which is a continuously updated plot)
- **Property node** (for configuration the chart)

Description and Tip strip

We will add a Description and a Tip strip for the u element (the simulated measurement signal):

> Open the Property window of the vertical pointer slide of u (right-click on the element, and select Properties). Open the Documentation tab, and enter the Description and Tip strip text shown in the figure below.

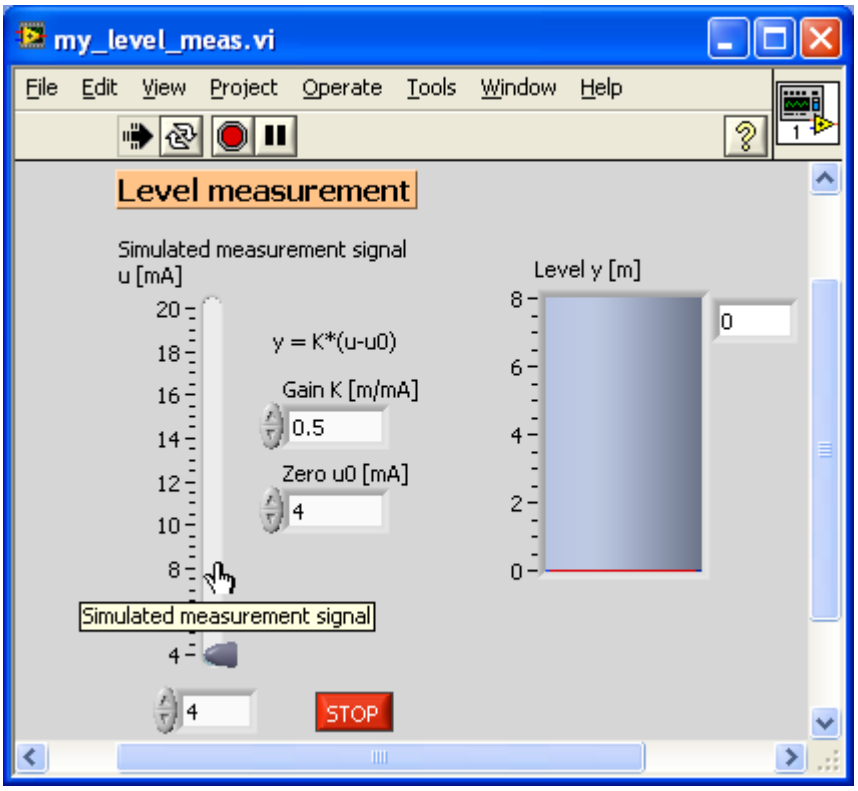


Description and Tip strip

Save the VI.

Run the VI and check if the Description and Tip strip are effective:

- To see the Description: **Right-click on the element / Description**.
- To see the Tip strip: Move the cursor over the vertical pointer slide. The figure below shows the tips strip.



Tip strip for the vertical pointer slide

While loop iteration number

To display the Loop iteration number of the While loop, we can connect a numeric indicator to the Loop iteration terminal, see the lower left corner of the block diagram shown in this figure.

Open the block diagram of my_level_meas.vi. **Right-click on the Loop iteration terminal / Create Indicator**, thereby creating a numeric display indicator at the upper left part of the front panel.

Open the front panel of my_level_meas.vi, and move the indicator to a new position, as shown in this figure. Give the element Label "i" (invisible), and Caption "Iteration no." (visible).

Run the VI. Does it seem from the continuously increasing value of the Iteration i element that the While loop runs ten times per second?

Stop the VI. Save it.

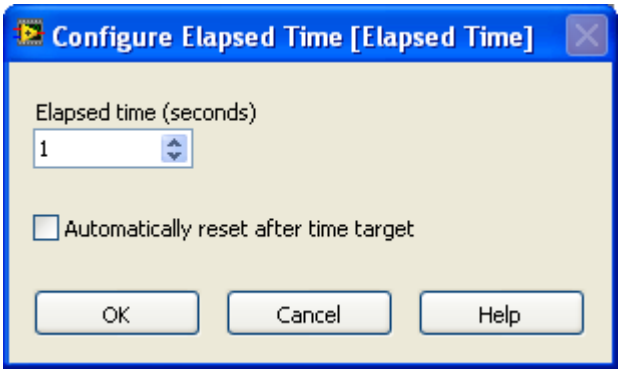
[Table of contents]

Clock

Often we want to have the elapsed time displayed at the front panel. This can be implemented by adding the an Elapsed Time function on the block diagram and displaying the **Elapsed Time** output of the function in a numeric display on the front panel.

Open the block diagram of my_level_meas.vi. Make the While loop somewhat larger, to make space for the **Elapsed Time** function to be added, cf. this figure.

Locate the Elapsed Time function on the Timing subpalette on the Functions palette, and add it to the block diagram, cf. this figure. As you drop the block a large blue icon of the function is displayed, and a Configuration window is automatically opened. In this window *disable* Automatically reset after time target, see the figure below. Then click OK.



Configuration window of the Elapsed Time function

Create a numerical indicator to the Elapsed Time output of the Elapsed Time function: **Right-click on the Elapsed Time output / Create Indicator**.

Open the front panel of my_level_meas.vi, and move the Elapsed Time indicator to a new position, as shown in this figure. Give the element Label "t" (invisible), and Caption "t [sec]" (visible).

Open the block diagram. The icon of the Elapsed Time function block is large. To use a small icon in stead: **Right-click on the block / View as icon**. The resulting icon is as shown in this figure.

Open the front panel. Run the VI. Does the t element display the elapsed time?

Stop the VI. Save it.

Express VIs

The **Elapsed Time** function block has a light blue color, indicating it is an *Express VI*. Express VIs, or Express functions, are actually made of elementary LabVIEW functions. You can configure Express VIs by double-clicking on them. There are many Express VIs

on the various subpalettes of the Functions palette.

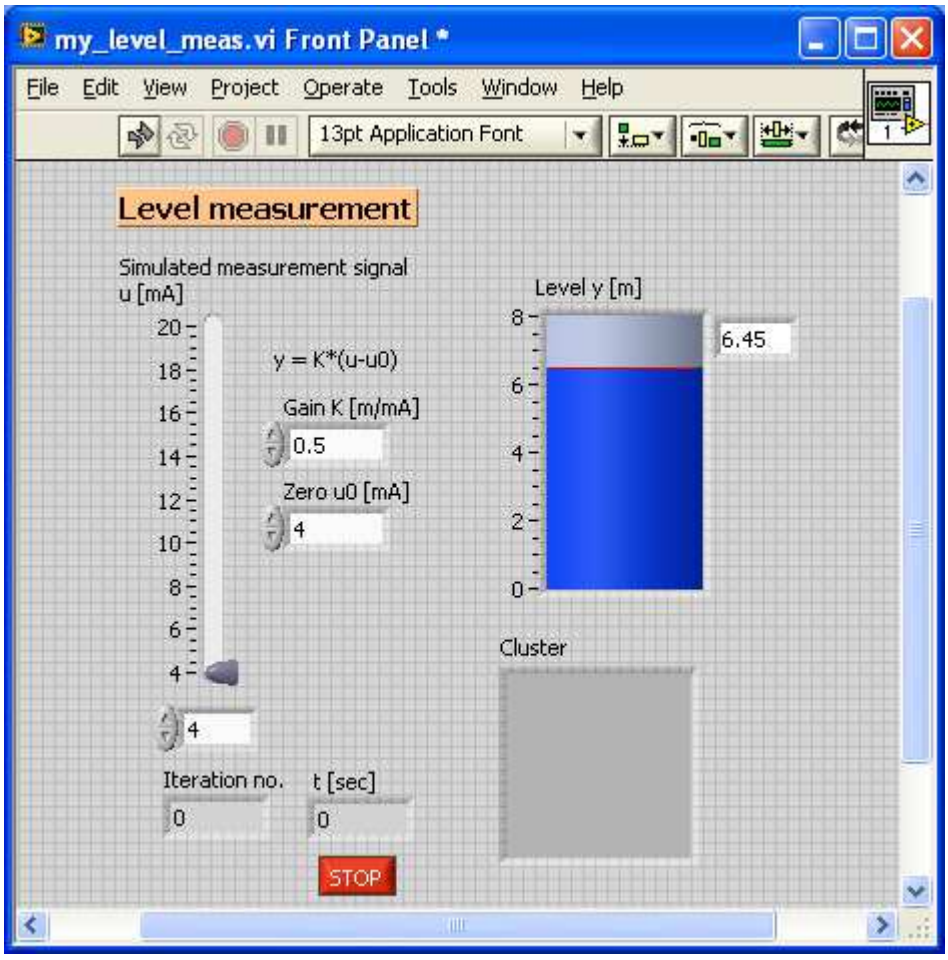
Open the block diagram of a new VI (you do not have to save it, as we will not use it later). Open the following subpalettes, and locate the Express VIs there. You can, to the extent you have time and interesest, drop the Express VIs onto the block diagram to see the configuration window of the Express VI (the configuration window is automatically opened).

- Comparison subpalette. (The Express VI is the Comparison VI.)
- Timing subpalette
- Dialog & User Interface
- File I/O
- Mathematics / Fitting

Cluster

Clusters are elements that contain one or more elements of possibly different data types. We will create a cluster containing the Gain K element and the Zero u0 element. What is the benefits of using clusters? On the Front panel clusters visually groups elements that are somehow related, e.g. controller parameters. On the Block diagram clusters may simplify the code since one cluster wire represents several single wires.

Open the front panel of **my_level_meas.vi**. Add an empty cluster from the Array, Matrix & Cluster palette to the front panel. The empty cluster may intermediately be placed anywhere on the Block diagram since we will move it soon. The figure below shows the Front panel with the empty cluster.

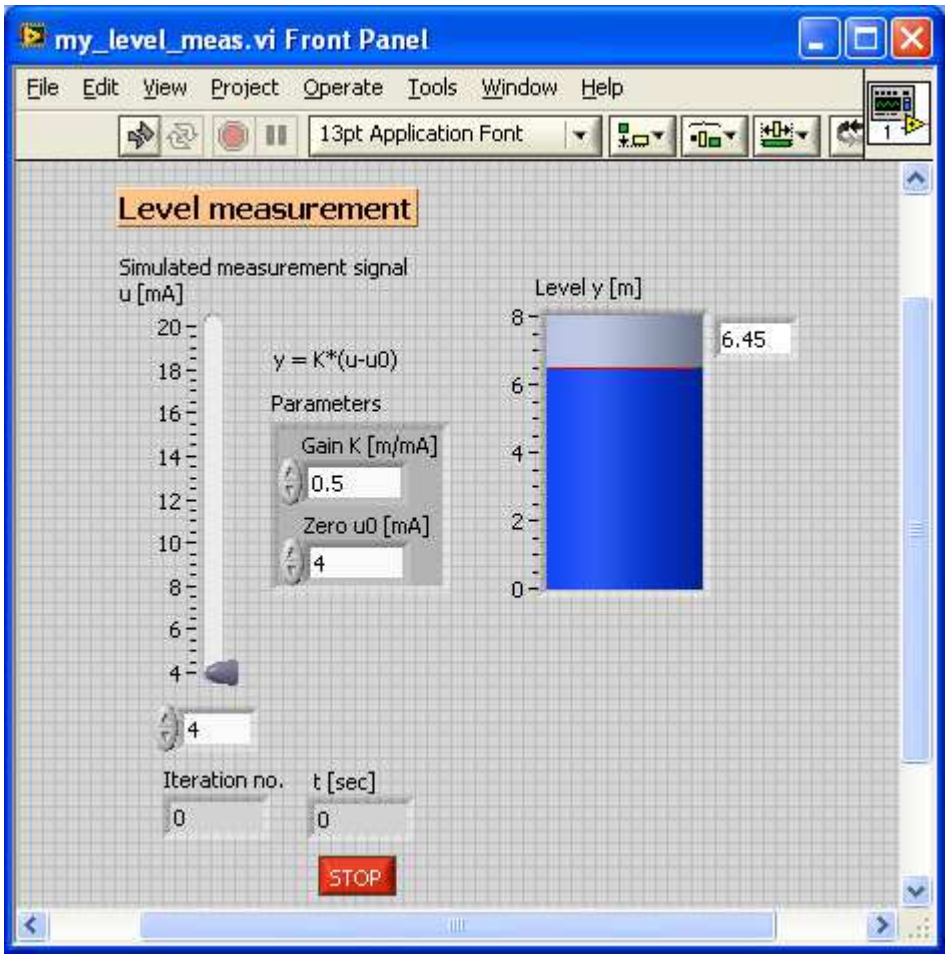


Front panel with empty cluster

Now, first drag the Gain K element and then the Zero u0 element into the cluster (you may fine-position the elements using the keyboard arrows). Right-click on the cluster boarder and select **Autosizing / Size to fit**.

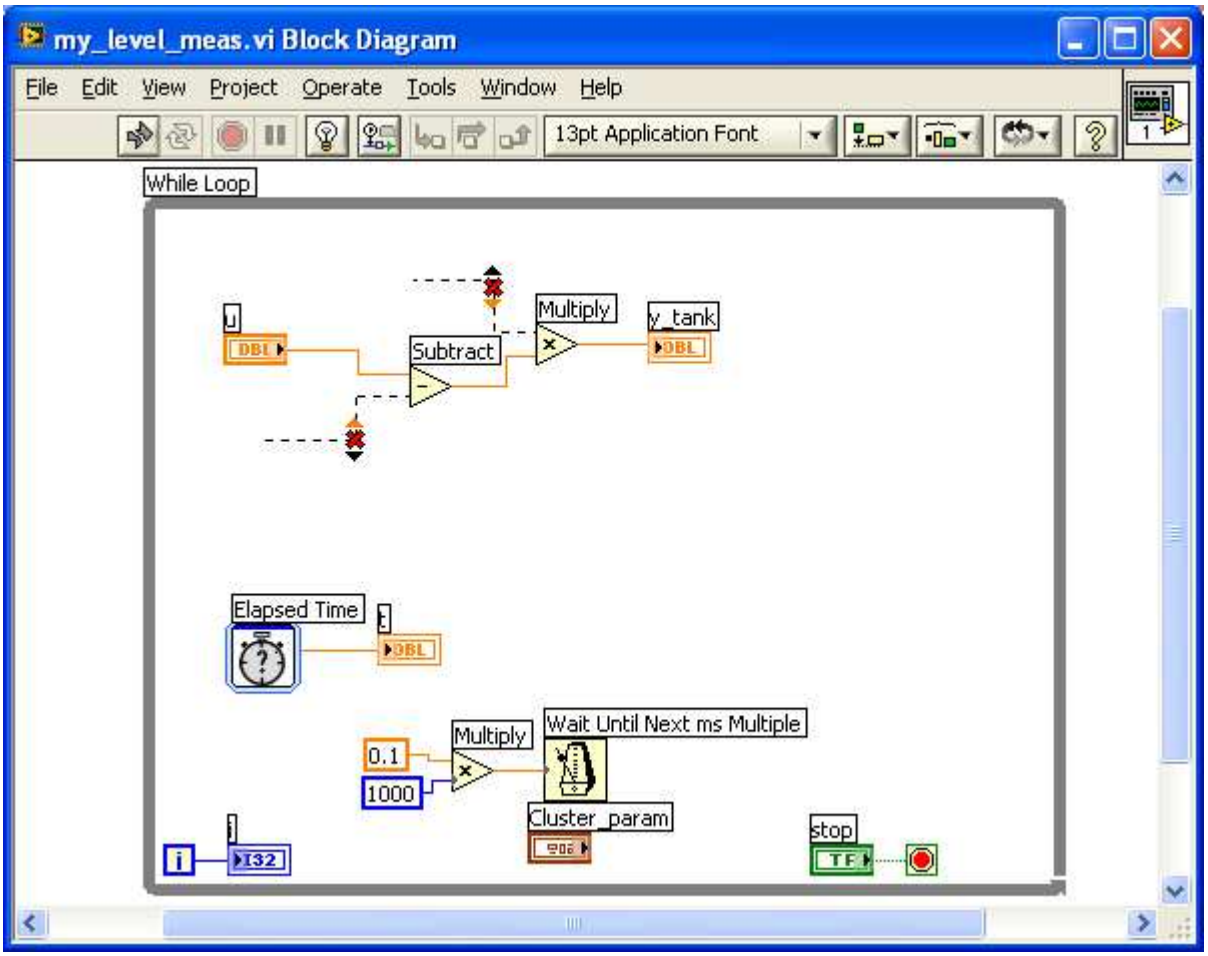
Set the label and caption of the cluster via the Property window of the cluster (to be opened via right-click on the cluster boarder) as follows: Label "Cluster_param" (invisible). Caption: "Parameters" (visible).

Move the cluster to the original position of the Gain K and the Zero u0 elements, see the figure below.



The Front panel with the cluster containing the Gain K and the Zero u0 elements

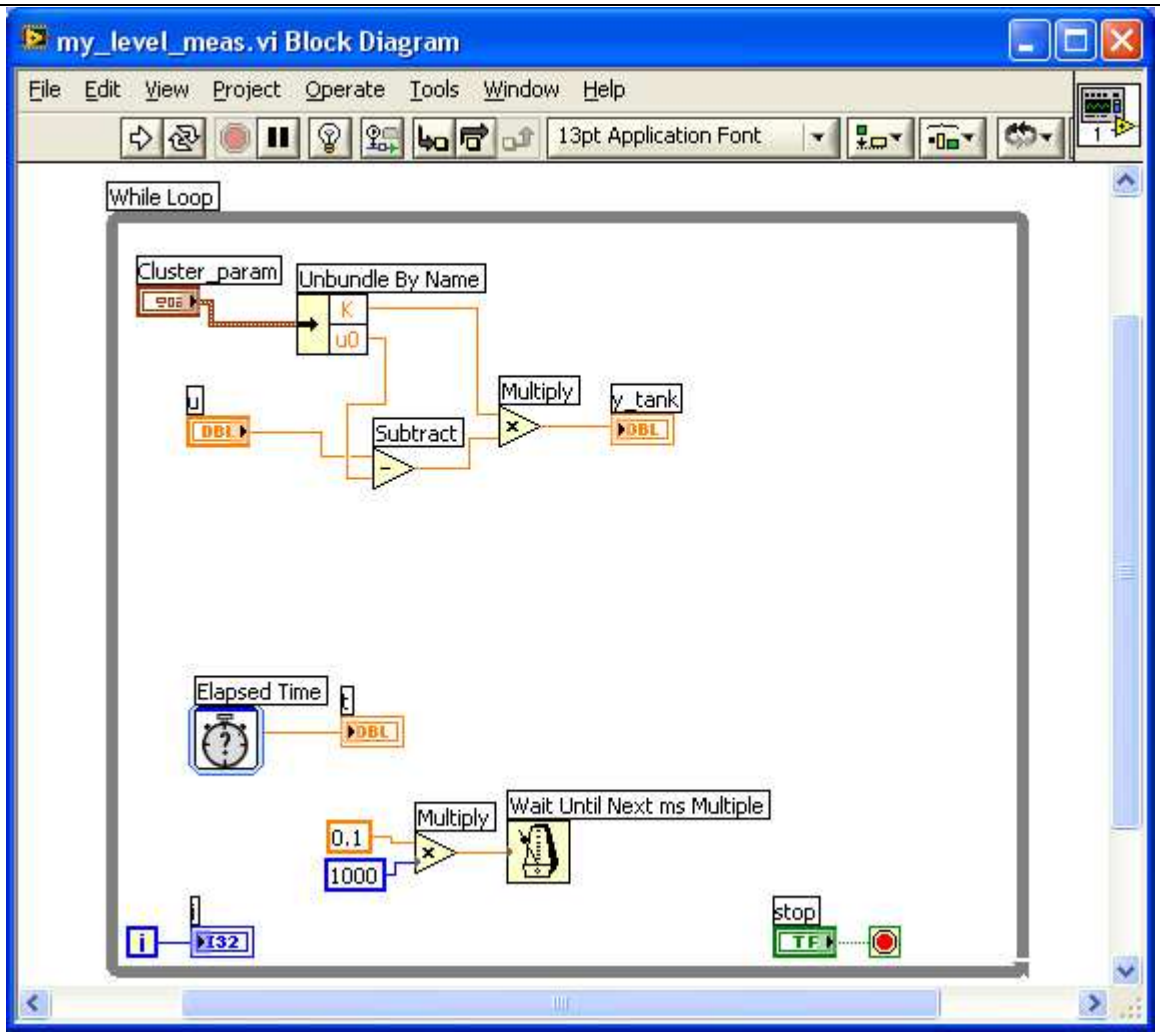
Now, open the Block diagram, see the figure below. The **Cluster_param** terminal has been placed at an more or less arbitrary position.



The Block diagram with the Cluster_param terminal situated (arbitrarily) below the Metronome

Next, do as follows. The final result is shown in the figure below.

- Move the **Cluster_param** terminal to a position upper left in the While loop. It is essential that the terminal is *inside* the While loop.
- Add an **Unbundle By Name** function to the right of the cluster terminal. This function is on the Cluster & Variant subpalette of the Functions palette.
- Connect the **Cluster_param** terminal to the **Unbundle By Name** function.
- Expand the **Unbundle By Name** function one step down by dragging the bottom boarder of the function downwards, thereby showing terminals of both K and u0.
- Finally, wire the K element to the **Multiply** function and the u0 terminal to the **Subtract** function.



Block diagram with the cluster terminal and the Unbundle By Name function.

Save the VI.

Boolean indicator. Local variable

We will now add a boolean indicator in the form of a LED (Light Emitting Diode) which will be lightening while the VI runs. Local variables will be used in the implementation.

Open the block diagram of **my_level_meas.vi**. Add a True constant and a False constant from the Boolean subpalette on the Functions palette, see the figure below. (Comment: By clicking a boolean constant the value changes from True to False or from False to True.)

Open the front panel of my_level_measurement.vi. Add a LED from the Boolean subpalette on the Controls palette at the position shown here (the LED labeled Running?). Set the properties of the LED as follows (in its Property window):

- Label "Running?" (invisible)
- Caption "VI is Running?" (visible)

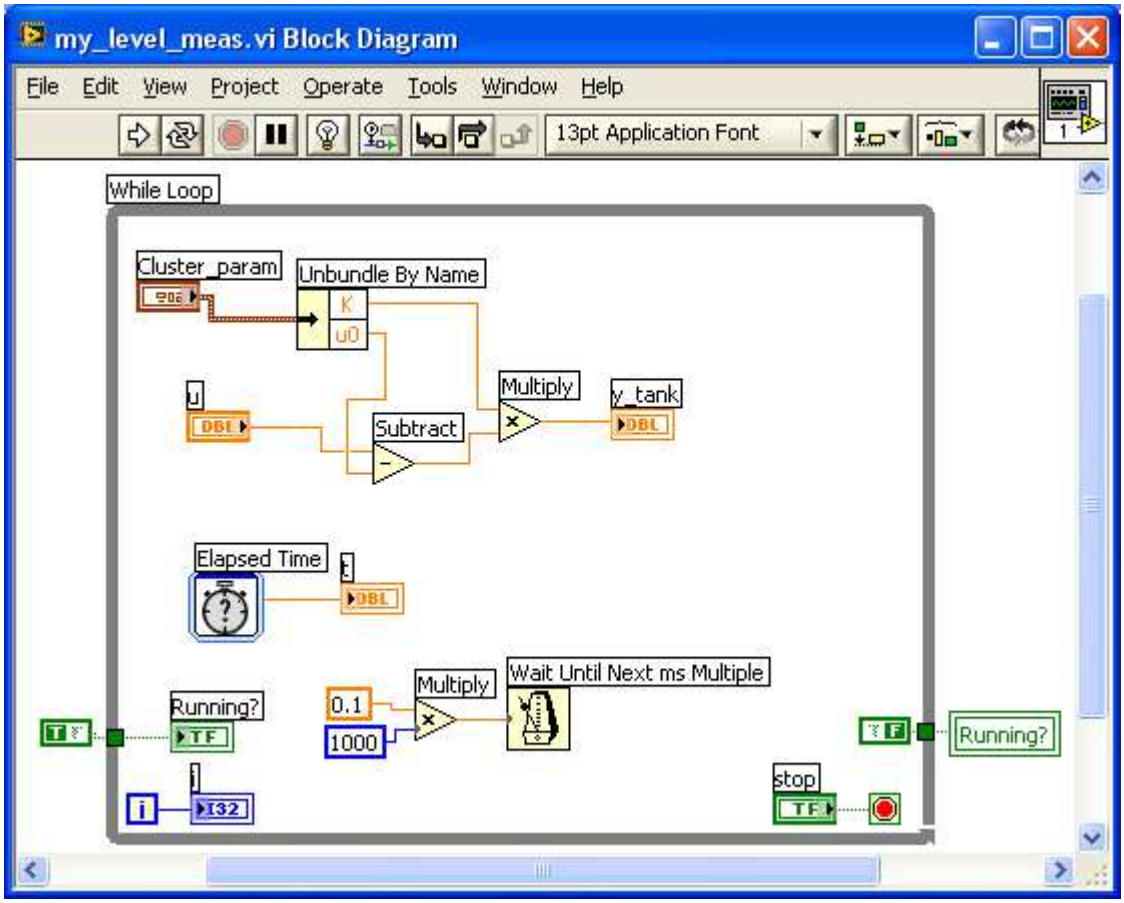
- Check for Show Boolean Text
- Uncheck for Lock text in center
- On text: "Yes"
- Off text: "No"

Note: It may happen that the Caption has a black background color. By double-clicking the Caption field the background color becomes white.

Open the block diagram of the VI. Locate the boolean Running? terminal and move it to the position shown in the figure below. Then wire the True constant to the Running? terminal.

Create a Local variable belonging to the Running? variable (terminal): **Right-click on the Running? terminal / Create Local Variable**. Then, place the Local variable just outside the While loop, and connect it to the False constant, see the figure below.

Note: By default a local variable is readable. You can set it to be writable by right-clicking on it and then selecting **Change to Write** in the menu that is opened.



Block diagram with a True constant connected to the Running? terminal and a False constant connected to the Local variable belonging to the Running? terminal (or variable)

Open the Front panel, and run the VI. Observe that the LED is turned on. Stop the VI, and the LED should turn off.

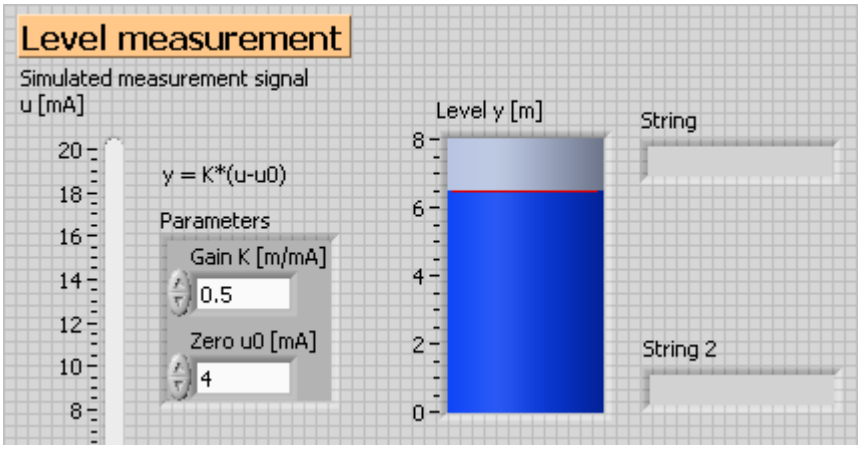
Save the VI.

Comparison functions. String indicators. Property nodes

Now we will implement the alarms. A High Alarm text will be shown on the front panel if the level is greater then 7 meters, and a Low Alarm text will be shown if the level is less than 1 meter.

We start by adding String indicators which will be used to display the alarms.

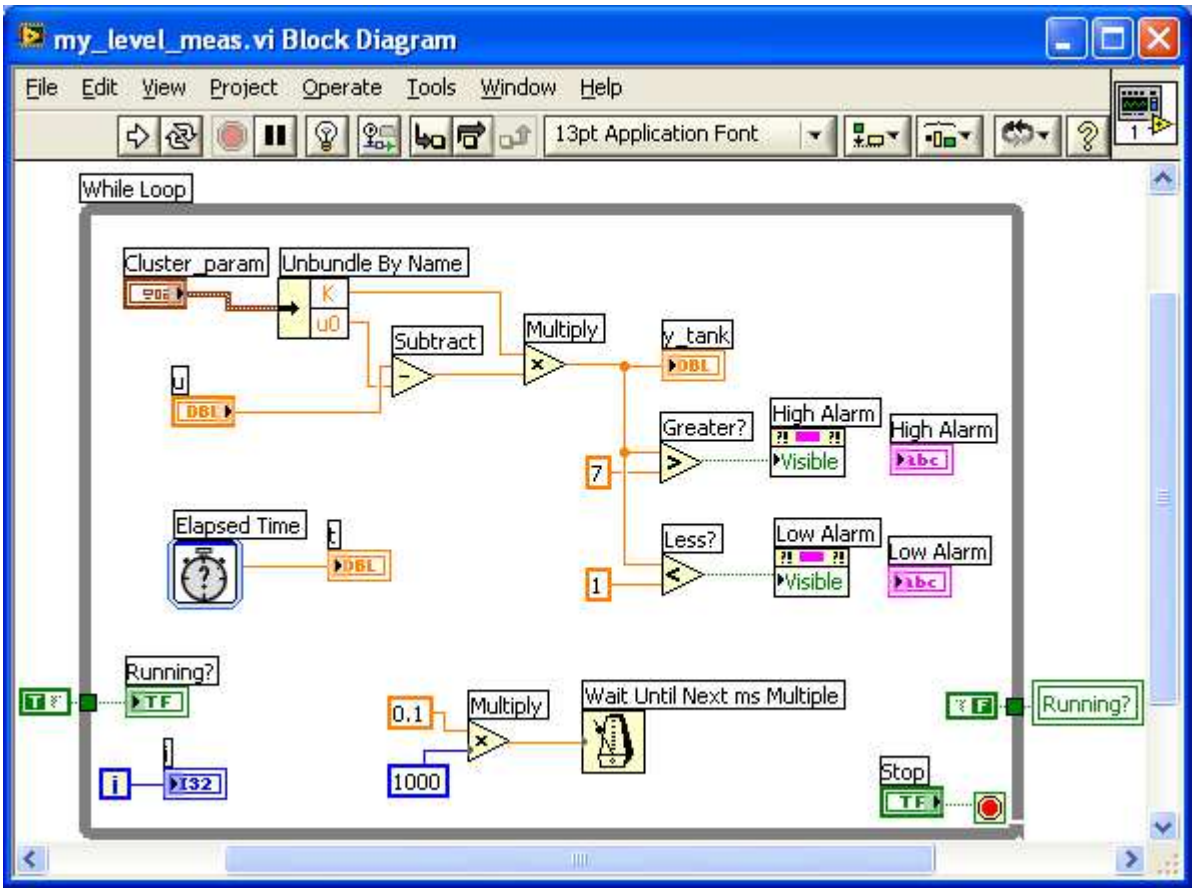
- Open the front panel of **my_level_measurement.vi**.
- Add two String indicators from the String & Path subpalette (on the Controls palette). Place them as shown in the figure below.



Two String indicators from the String & Path subpalette

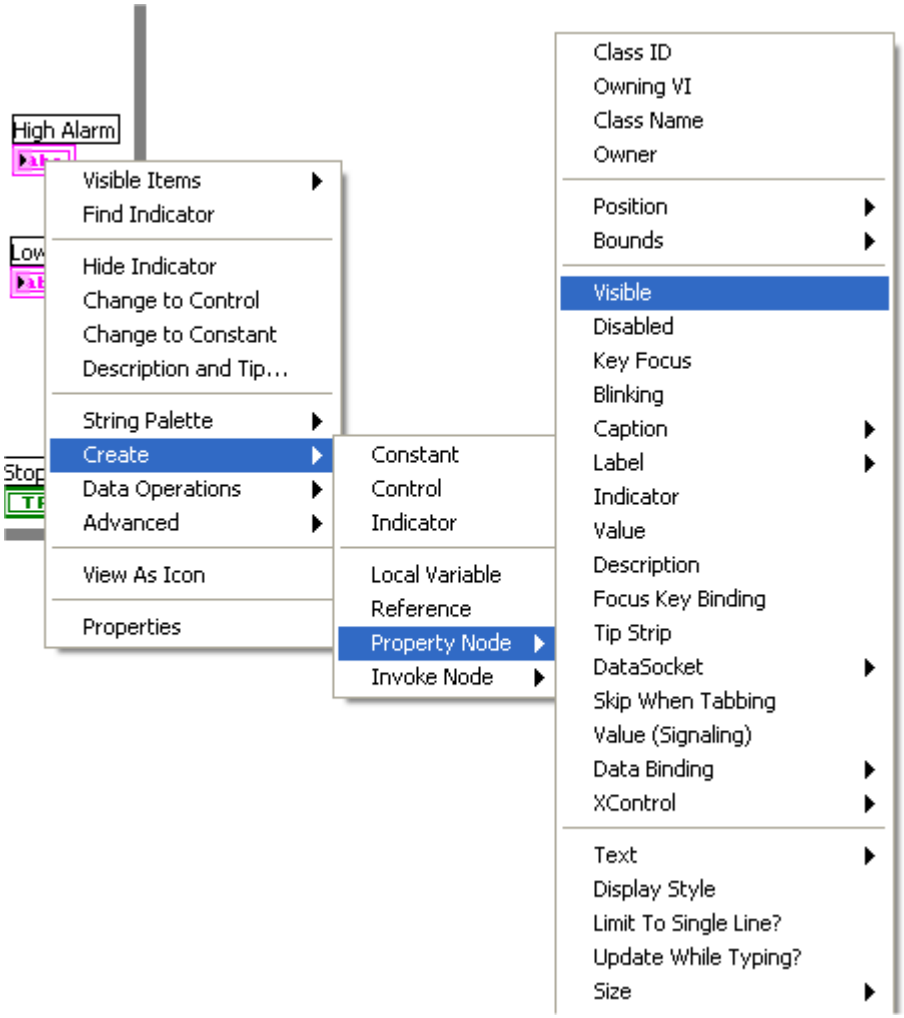
- For the upper String indicator:
  - Enter the text "Alarm: High level (>7m)". Then, to define this string as default: **Right-click on the indicator / Data Operations / Make Current Value Default**.
  - Label (to be set via the Property window): "High Alarm".
  - Hide the indicator (it will be displayed during the program execution if the level exceeds the upper alarm limit): **Right-click on the element / Advanced / Hide Indicator**.
- Repete the above steps, but for the lower String indicator, but enter the string "Alarm: Low level (<1m)", and set the label to "Low Alarm".

Now we will add the necessary functionality to the block diagram. While following the instructions below, refer to the figure just below showing the resulting block diagram.



Block diagram with Comparison functions, text indicators and property nodes

- Open the block diagram.
- Add a **Greater?** function and a **Less?** function from the Comparison subpalette to the block diagram.
- Add two numeric constants, and give them values 7 and 1, respectively (by double-clicking and typing the numbers).
- Wire the various signals to the **Greater?** function and the **Less?** function, respectively.
- Locate the High Alarm indicator and the Low Alarm indicator, and move them inside the While loop to positions shown in the figure above.
- Create a Property node for the High Alarm element: **Right-click on the High Alarm indicator / Create Property Node / Visible**. This opens a list of properties, see the figure below, in which you select Visible, thereby dropping Property node on the block diagram.



With Right-clicking on the High Alarm indicator / Create Property Node a list of properties of the indicator is displayed.

A property is either *readable* or *writable*. In our case it must be writeable, so that the indicator will be visible or not visible on the front panel depending on the output of the **Greater?** function. Thus, **Right-click on the Property node / Select Change to Write**.

- Repeat the step above (creating a property node) for the Low Limit indicator.
- Finally, wire the **Greater?** function ouput to the input of the High Alarm Property Node, and the **Less?** function ouput to the input of the Low Alarm Property Node, cf. this figure.

Open the front panel, and run the VI. Adjust the value of u to the maximum value and then to the minimum value. Are the alarms displayed correctly?

Save the VI.

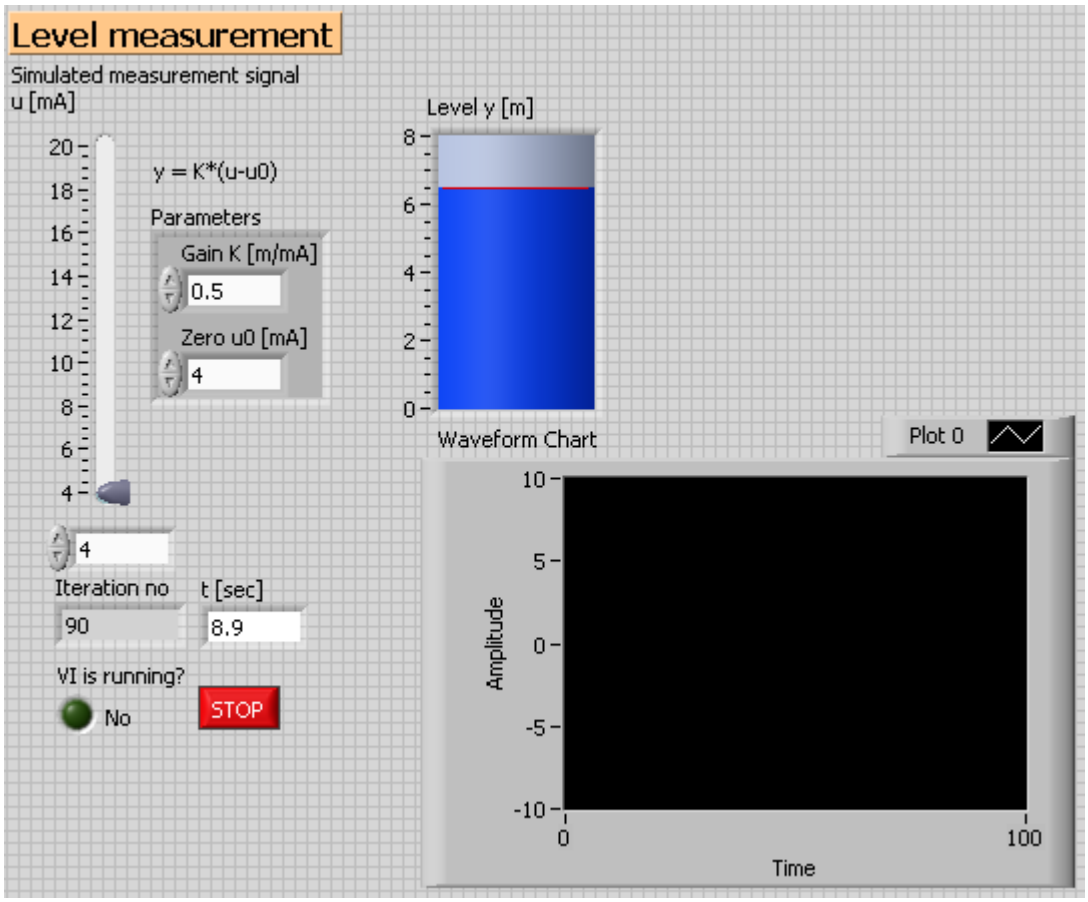
Chart

We will now add a *chart* to our VI. A chart is a continuously updated diagram for plotting variables with time along the x-axis. In our VI the chart will plot three signals:

1. The level.
2. The High alarm limit, which is 7 (meters)
3. The Low alarm limit, which is 1

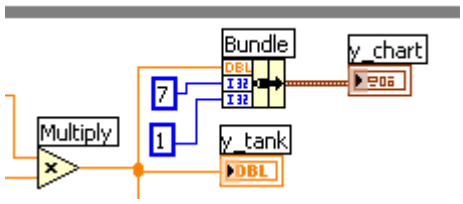
Adding a chart

Open the front panel of **my_level_meas.vi**. Add a chart at the position shown in the figure below. The chart is found at **Controls palette** / **Graphs** / **Waveform Chart**.



A (waveform) chart is added to the Front panel

The (three) signals to be plotted in the chart are collected using a **Bundle** function in the Block diagram. The **Bundle** function produces a *cluster* of signals, and this cluster is then wired to the chart. The relevant resulting part of the Block diagram is shown in the figure below.



Using a Bundle function to collect the signals to be plottet in the chart

- Open the Block diagram of the VI.
- Move the y_chart terminal to a position shown in the figure above.
- Insert a **Bundle** function from the / **Cluster & Variants** palette, as shown in the figure.
- Expand the **Bundle** function to have three inputs by dragging the bottom edge of the function downwards.
- Add two numeric constants from the Numeric palette, and give them values 7 and 1, respectively.
- Wire the output from the **Multiply** function, which is the level signal, and the two constants to the Bundle function.
- Wire the **Bundle** output to the y_chart terminal (which thereby changes color and icon, automatically).
- Open the front panel. The chart now have three digital displays. You can move them outside to the right of the chart, cf. this figure showing the front panel of the VI.

Now, let us do some changes to the chart on the Front panel:

- Open the Front panel. The chart has got three digital displays. Move them outside to the right of the chart, cf. this figure showing the front panel of the VI.

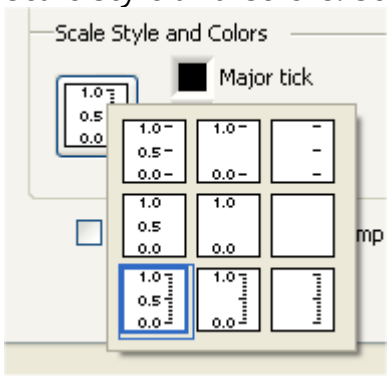
There is a large number of options when *configuring* a chart. We will now add a chart with typical basic settings. We will mostly make the settings in the *Property window* of the chart. Later you will see how to configure the chart programmatically, i.e. by wiring values (constants) to a *Property node* of the chart in the Block diagram.

Basic settings via the Property window of the chart

To configure the chart, open its Property window (by right-clicking somewhere on the chart). Then do the following setting in the different tabs:

- **Appearance tab**:
  - Label: "y_chart". Invisible. Caption: "Level Chart". Visible
  - Update Mode: Sweep chart (causing the pen to plot on a fixed paper)
  - Show Digital Display(s)
- **Scales tab**:

- Select Time (x-axis) in the axis list at the top of the page of this tab.
  - Name: "Time t [s]"
  - Scaling factors: Multiplier: 0.1, corresponding to the 0.1 second cycle time of our program (it is the cycle time of the While loop). (With the default value of Multiplier of 1, the x-axis increment would be 1, so in one second the x-axis value would increase by 10, which is not what we want.)
  - Minimum: 0
  - Maximum: 20
  - Scale Style and Colors: Select as shown in the figure below:



  - Grid Style and Colors: Click the Button, and select the option to the right. (If you want to change the colors of the grid, you can click the color buttons.)

- Select y-axis in the Scales tab.
  - Name: "[m]".
  - Grid Style and Colors: Click the Button, and select the option to the right. (If you want to change the colors of the grid, you can click the color buttons.)

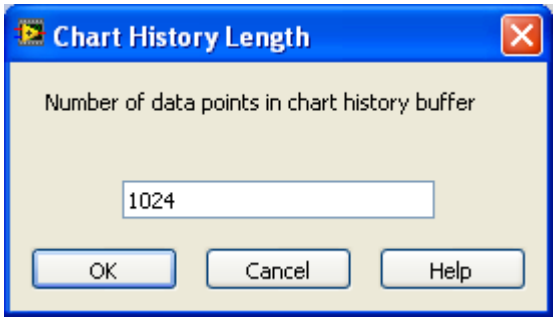
To set the Plot legend (above the diagram, to the right) properly:

- Drag the bottom edge of the Plot legend downwards to expand the legend to show information for *three* plots. Move the Plot legend to a proper (nice) position just above the chart.
- The Plot legend text: Double-click each of the three Plot legend texts areas, and type the following texts, respectively: "Level y [m]"; "High Alarm Limit [m]"; "Low Alarm Limit [m]".
- The line color: **Right-click on the display to the right of the text field / Select Color**. Set a proper color to each of the three lines.

Open the front panel. Run the VI, and play with the u value. The y-axis has automatic scaling, which is the default setting. Change to manual scaling while the VI runs by right-clicking on the chart, and selecting AutoScale Y.

The chart has an inbuilt *data buffer* which stores previous (historical) data so that the previous signal values are shown in the chart. The default length of this buffer is 1024. Thus, at each instant of time the most 1024 recent samples of the signal are stored in this buffer. In our VI, with a sampling time of 0.1s and a range of 50s along the x-axis, the buffer must have a length of 50s/0.1s = 500 samples. Therefore the default buffer length is ok in our case. But in other cases it may be necessary to increase the buffer length.

To see/set the buffer length: **Right-click on the chart / Select Chart History Length**, see the figure below. However, as argued above, it is not necessary for us to change the buffer length, so you can just close the dialog window.



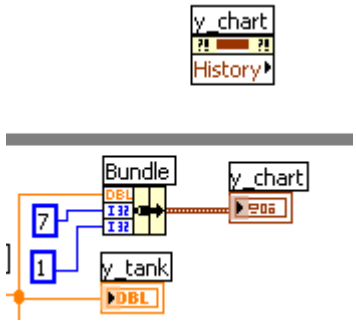
The Chart History Length can be seen/set via Right-click on the chart / Chart History Length

Programmatic configuration of the chart

As an example of programmatic configuration of the chart, we will set the History property of the Property node of the chart in the Block diagram so that the chart is emptied as the VI starts.

Open the block diagram of **my_level_meas.vi**. Do as follows:

- Open the *History property* of the Property node as follows: **Right-click the y_chart terminal / Create Property Node / History property (at the bottom)**. See the figure below.

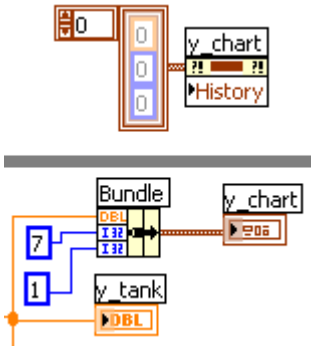


The Property Node with the History property of the y_chart element

- Change the History property from *readable* to *writeable*: **Right-click on the**

Property Node / Change to Write. (The arrow then points into the Property node at its left side.)

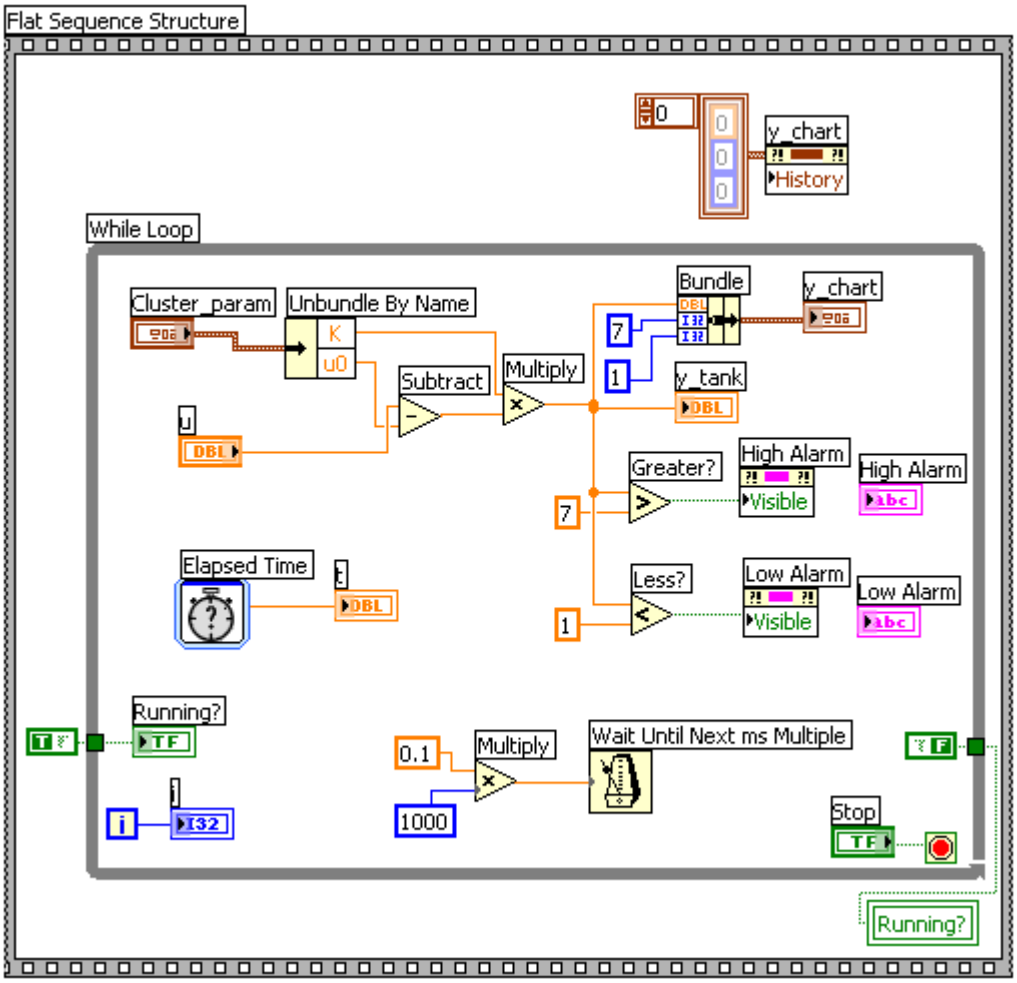
- To add a proper constant to the History property node: **Right-click on the node / Create Constant**. (The constant automatically becomes a cluster of three empty arrays (one array for each plot line), causing the History buffer to be emptied.) See the figure below.



A constant wired to the History property of the Property Node

Next, we have to ensure that the added code is executed *before* the While loop starts. This can be done using a Sequence structure which is in the form of a film strip with frames. Code can be put into the different frames. The code in the first frame will be executed first, then the code in the second frame, and so on.

Add a Flat Sequence from the Structures palette (on the Functions palette), and ensure it embraces the While loop, see the figure below.

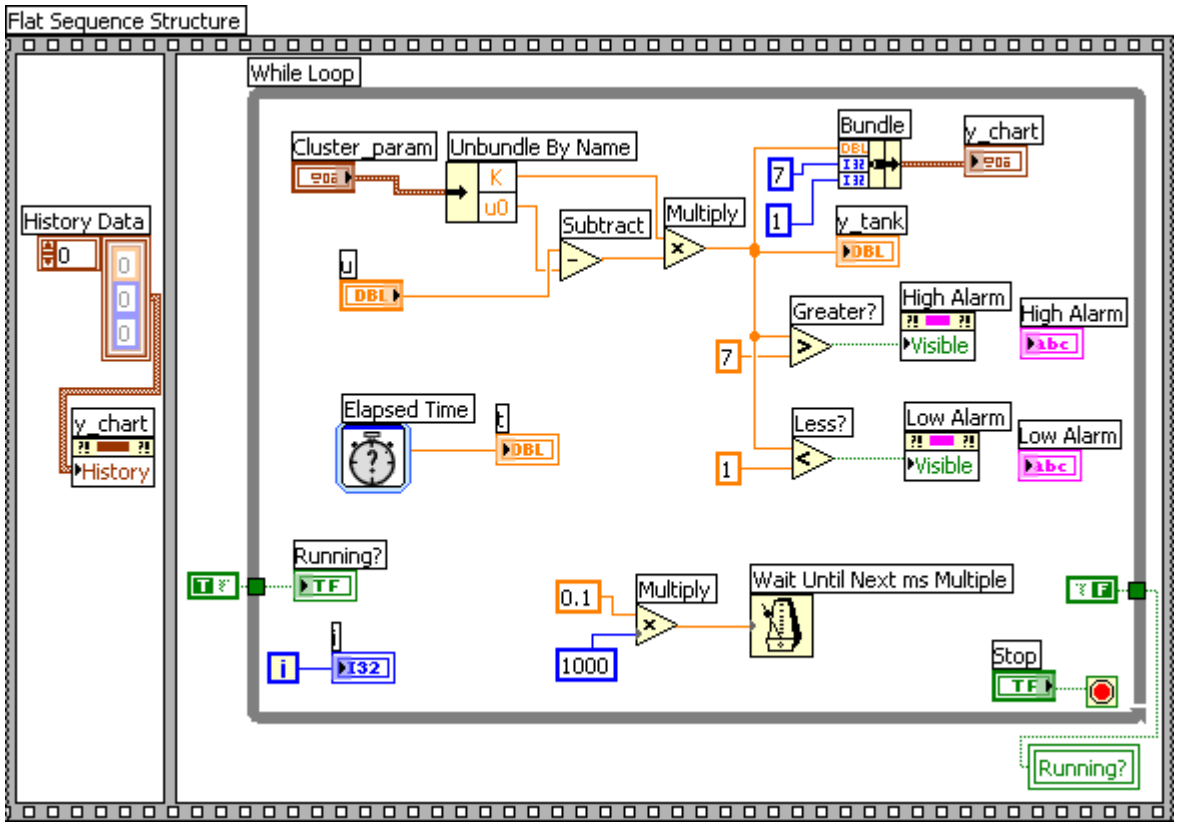


A Flat Sequence structure embracing the While loop

Then, do as follows:

- Add one frame *before* the present single frame: **Right-click on the top border of the frame / Add Frame Before**.
- Move the Property node (with the empty array wired to it) inside the empty frame (to the left).
- Make the label of the Sequence Structure visible: **Right-click on the border / Visible Items / Label**.
- Make the Label of the array entering the Property node visible.
- To save some space, move the array above the Property node, as shown in the figure.

The resulting block diagram is shown in the figure below.



Block diagram with a Flat Sequence structure

Open the Front panel, and Run the VI. Adjust the input signal u. Stop the VI, and run it again. Is the chart emptied at the moment the VI is started, as expected?

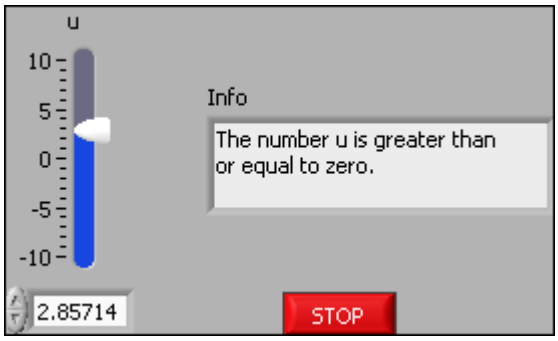
[Table of contents]

8 Additional topics

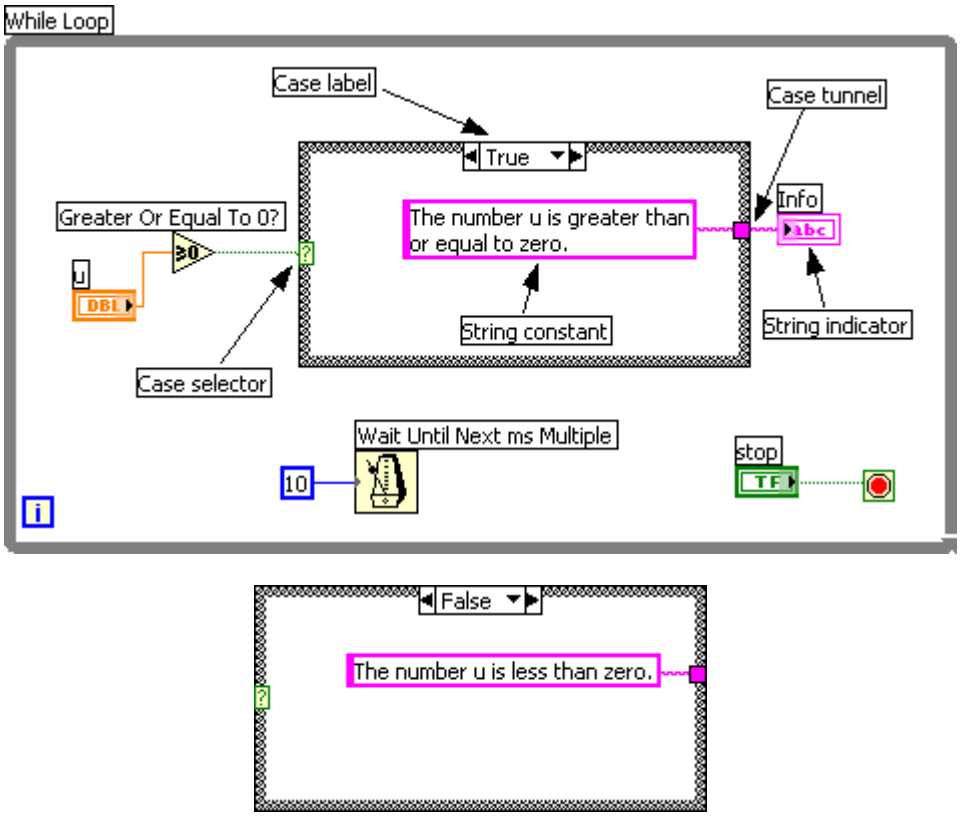
8.1 Case structure

A *Case structure* consists of a number of different *cases* or *frames* containing their individual LabVIEW code. Only one of the cases is active at any instant of time. The Case *selector* selects or controls which one among the cases is active. So, a Case structure implements a way of selecting among alternative portions of program code. The Case structure in LabVIEW si similar to the case or switch structure in other programming languages

case.vi is a simple example of a Case structure. The Case structure is used to display alternative texts in the String indicator, depending on wheter the value of u adjusted with the vertical slider is greater than (or equal to zero) or less than zero. The figures below show the Front panel and the Block diagram of this VI.



The Front panel of case.vi



The Block diagram of case.vi. The two alternative cases (frames) are shown. The True case is shown in the upper part of the figure, and the False case is shown in the lower part.

Comments to case.vi:

- The comparison function named **Greater Or Equal To 0?** is used to generate a proper Boolean value - True or False - which is wired to the Case selector.
- The two Cases are identified by the labels True or False, respectively.
- String constants contains the alternative strings to be displayed in the String indicator on the Front panel.
- Values out from (or into) Case frames are transferred via *tunnels*. Tunnels are automatically created if as you draw a wire across a Case frame. All output tunnels, i.e. tunnels for signals *leaving* frames, must be wired to some signal. This is the case in the Block diagram shown above. A String constant is wired to the output tunnel

of each of the two frames. (Honestly, you may drop wiring a value to an output tunnel, but if so, you have to tell LabVIEW to use the default value of the indicator to which the tunnel is wired in stead. This is done via right-click on the tunnel.)

You may want to try adding a Case structure yourself:

- Save case.vi as my_case.vi (in any folder you prefer).
- Open the block diagram of my_case.vi.
- Remove the existing Case structure: **Right-click anywhere on the Case structure frame / Select Remove Case Structure**. Then LabVIEW asks you if you want to remove the contents of all other cases than the one shown presently, and you can answer Yes to this question.
- Remove broken wires using the keyboard shortcut Ctrl + B.
- Add a Case structure to the Block diagram: **Right-click where you want the upper left corner of the Case structure to appear / Open the Structures palette on the Functions palette / Select Case Structure on the palette**. Then drag the Case structure so that it gets a proper size, cf. the figure above showing the Block diagram. Note: Ensure that all elements of interest actually appears, i.e. are visible, inside the Case structure. If an element is not visible inside the Case frame, the element *is not functionally inside* the frame!
- Ensure that there is a String constant with proper content in each of the two frames. Wire signals according the the Block diagram.
- Open the front panel of your VI. Save, and run the VI. Does it work?

Finally, some general comments to the Case structure:

- In our example, case.vi, the Case structure Labels have Boolean values, True and False, and the Selector must then recieve a Boolean value, too. However, you may use other data types for the Labels and the Selector, for example an integer type or an enumerate type, making it possible to select among any number of alternative cases. Once you wire say an integer signal to the Selector, the Label is automatically adopted to that data type, getting Labels 0, 1, 2.
- You can edit the Case structure (e.g. removing cases, adding cases, swapping cases, defining a given case to be Default, which means that this case is active if the there is no Label match the present Selector value.

[Table of contents]

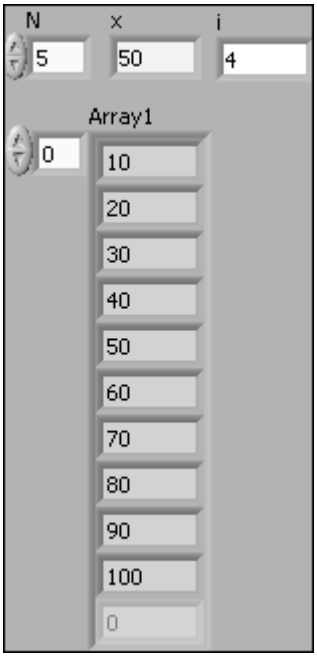
8.2 For loop

The *For loop* is a program structure which can be used to run a certian amount of program code over and over again a *predefined number of times*. The For loop is quite similar to the While loop, however, as said, with the For loop the number of loop iterations is fixed (predefined) while with the While loop that number is not predefined.

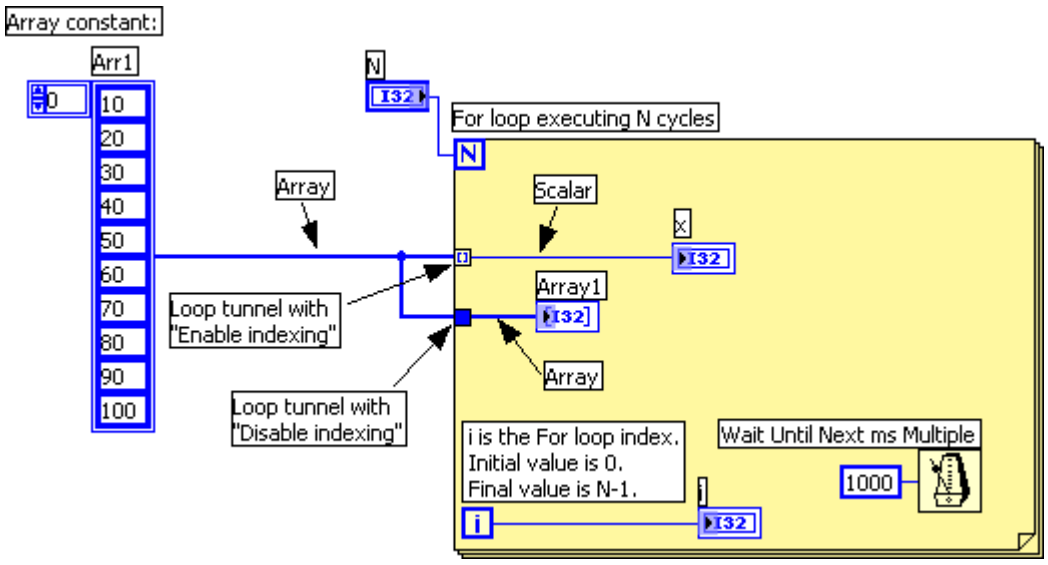
Let us look at an example of a For loop. The program reads successively new values from an array according to the following pseudo code:

```
Arr1 = [10,20,30,40,50,60,70,80,90,100];
N = 5;
for i = 0:N-1
 {
  x = Arr(i);
 }
```

The VI for_loop.vi implements this code in LabVIEW. The Front panel and the Block diagram of this VI are shown in the figures below.



The Front panel of for_loop.vi



The Block diagram of for_loop.vi

Comments to for_loop.vi:

- The N terminal of the For loop defines the number of execution cycle times. N is set to 5 on the Front panel.
- For demonstration purposes I have placed a Metronome (the Wait Until Next ms Multiple function) with Wait time of 1000ms = 1s inside the For loop to make it possible for the user to follow the changes of the values on the Front panel as the VI runs.
- I have put yellow colour to the For loop just to highlight the loop.
- The i terminal is the For loop index. Its value increases by one, starting from zero, each time the For loop executes. The i terminal is available for you to use in your code.
- Arr1 is an array constant which I have manually fed with values.
- The array enters the For loop via two Loop tunnels which are set up in different ways, just to demonstrate important features of For loops:
  - Via a Loop tunnel with setting *Enable indexing*. In this case, what comes out from the tunnel is the value (a scalar) of the array element having the same index number as the present value of the For loop index. Thus, in our VI the For loop automatically reads successive array elements as the For loop executes and displays the element in the x indicator on the Front panel. Since the
  - Via a Loop tunnel with setting *Disable indexing*. In this case, what comes out from the tunnel is the entire array. In our VI the entire array is displayed in the Array1 indicator on the Front panel.

You may want to try adding a For loop to a VI yourself:

- Save for_loop.vi as my_for_loop.vi (in any folder you prefer).
- Open the block diagram of my_for_loop.vi.
- Remove the existing For loop: **Right-click anywhere on the For loop border / Select Remove For Loop**.
- Remove broken wires using the keyboard shortcut Ctrl + B.
- Add a For loop to the Block diagram: **Right-click where you want the upper left corner of the For loop to appear / Open the Structures palette on the Functions palette / Select For Loop on the palette**. Then drag the For loop so that it gets a proper size, cf. the figure above showing the Block diagram. Note: Ensure that all elements of interest actually appears, i.e. are visible, inside the For loop. If an element is not visible inside the loop, the element *is not a part of* the loop!
- Wire signals according the the Block diagram. Set the upper Loop tunnel with Enable Indexing and the other tunnel with Disable Indexing by right-clicking on the tunnel.
- Open the front panel of your VI. Save the VI. Run the VI. Hopefully, the VI works correctly.

[Table of contents]

8.3 Shift register. Feedback node

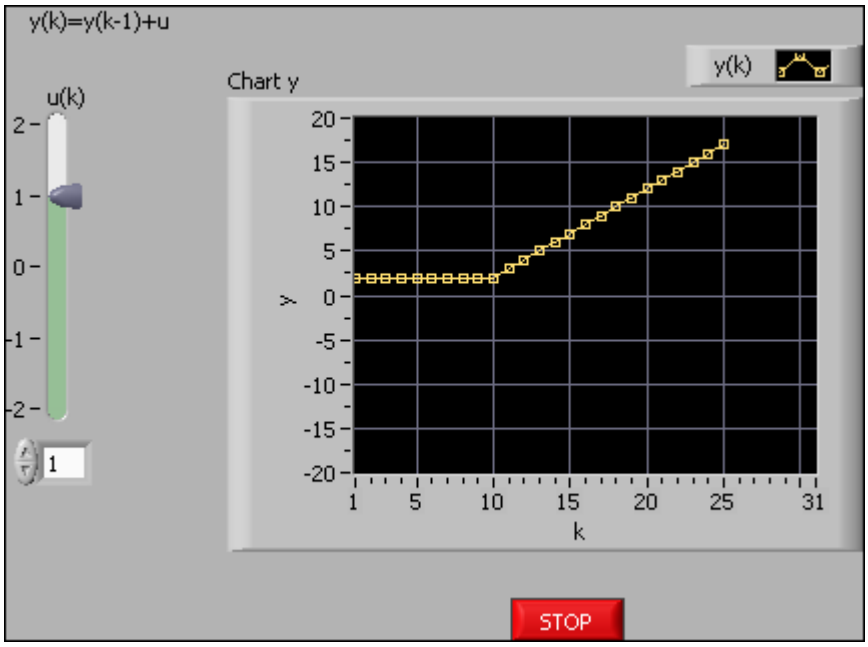
In LabVIEW a **Shift register** is a memory element storing values from previous executions of loops, e.g. While loops and For loops. A Shift register can store values of any type, e.g. integers, decimal number (floating point numbers), arrays, clusters. A **Feedback node** is (almost) equivalent to a Shift register. It has less features than a Shift register, but appears somewhat different in the Block diagram. Both Shift register and Feedback node will be described in the following.

Shift register

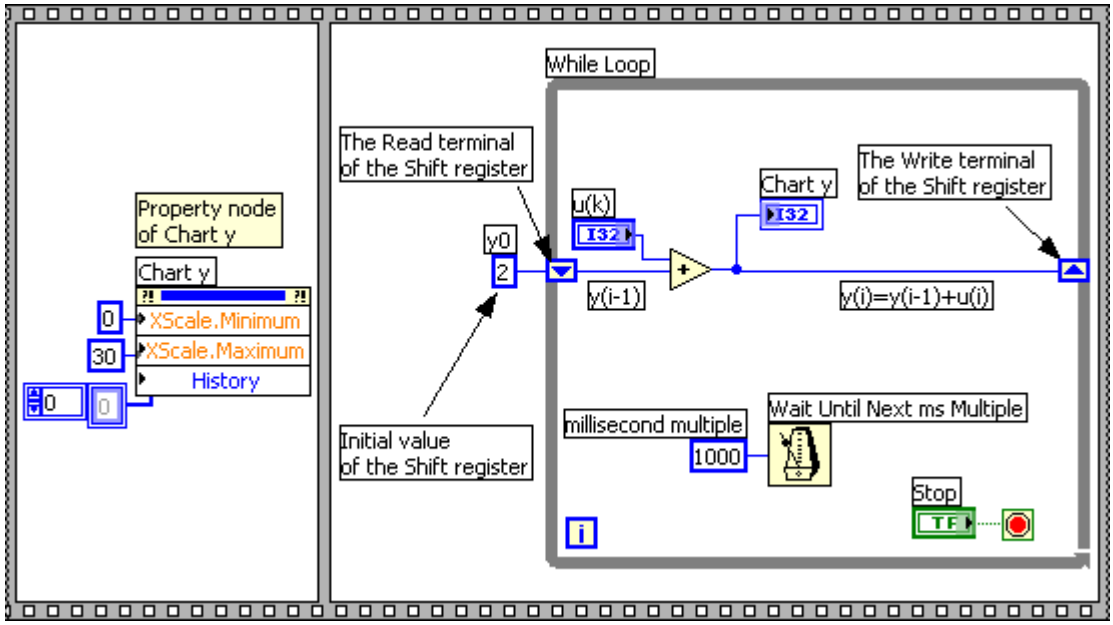
Let us look at a simple example where a Shift register is used to implement the recursive formula

$$y(i)=y(i-1)+u(i), \text{ with initial value } y(-1)=2$$

where i is the time index (which counts the discrete time steps), which starts on zero. The figures below show the Front panel and the Block diagram of shiftregister.vi.



Front panel of shiftregister.vi



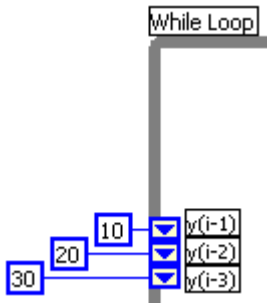
Block diagram of shiftregister.vi

The user can adjust the control u (an integer) while the VI is running. A Shift register is used to store y(i) so that it becomes available as y(i-1) in the subsequent execution (or cycle) of the While loop, and y(i-1) is used in the calculation of y(i), cf. the formula given above.

You may want to try create a Shift register yourself (by first removing the existing Shift register and then immediately creating a new one):

- Save shiftregister.vi as my_shiftregister.vi (in any folder you prefer).
- Open the block diagram of my_shiftregister.vi.
- Remove the existing Shift register: **Right-click on the right (or the left) Shift register terminal on the While loop / Select Remove All from the menu**.
- Remove broken wires using the keyboard shortcut Ctrl + B.
- Create a Shift register: **Right-click on the right (or the left) border of the While loop / Select Add Shift Register from the menu**.
- Wire signals to the Shift register, cf. the Block diagram of shiftregister.vi.
- Open the front panel of your VI. Save the VI. Run the VI. Hopefully, the VI works correctly.

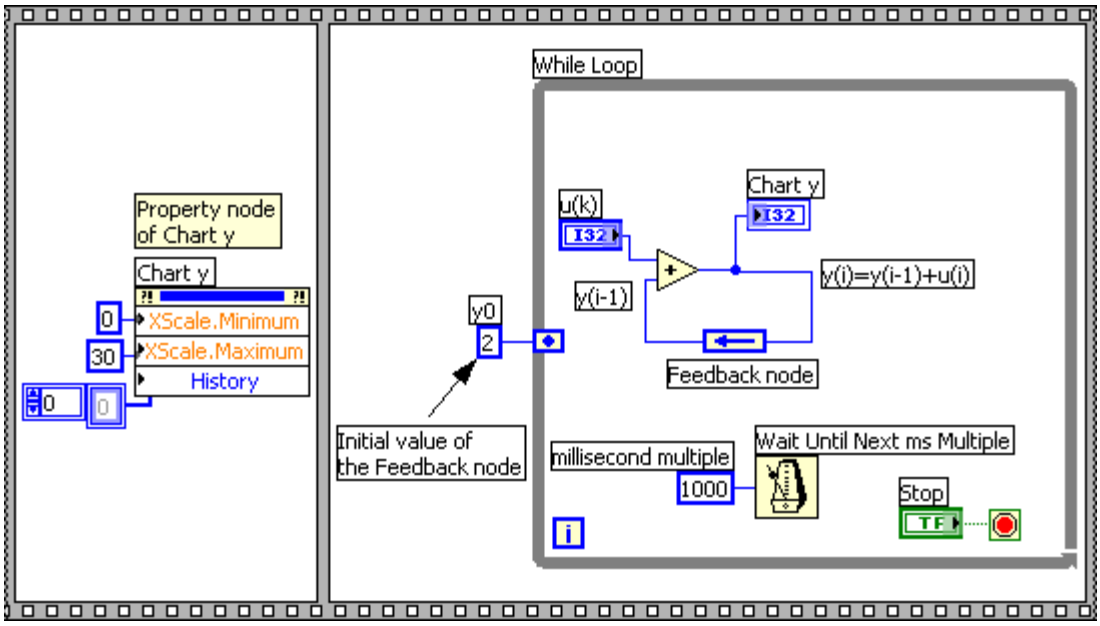
The Shift register in shiftregister.vi stores a value created in the previous cycle of the While loop. What if you want to store *older* values than just the previous value? No problem - just expand the left Shift register terminal by dragging the bottom line of the terminal downwards. And you can assign initial values to each of the elements of the Shift register, see the figure below.



The Shift register may contain values from older cycles of the While loop

Feedback node

The figure below shows a Feedback node. It can store the value of an element from one previous loop cycle (whereas a Shift register may store values from even older cycles). A Feedback node appears somewhat simpler in the block diagram.



A Feedback node, which is (almost) equivalent to a Shift register

You can create a Feedback node in two ways:

- By replacing an existing Shift register with a Feedback node
- By inserting a Feedback Node from scratch

Let us try both ways.

Creating a Feedback node by replacing an existing Shift register:

- Save shiftregister.vi as **my_feedback_node.vi** (in any folder you prefer).

- Open the block diagram of **my_feedback_node.vi**.
- Replace the existing Shift register with a Feedback node: **Right-click on the right (or the left) Shift register / Replace with Feedback node**.
- The wires may overlap. To make the individual wires more visible, move the Feedback node and the wires into and out from the Feedback node somewhat.
- Open the front panel of your VI. Save the VI. Run the VI. Hopefully, the VI works correctly.

Creating a Feedback node from scratch:

- Save shiftregister.vi as **my_feedback_node_scratch.vi** (in any folder you prefer).
- Open the block diagram of **my_feedback_node_scratch.vi**.
- Remove the existing Shift register: **Right-click on the right (or the left) Shift register terminal on the While loop / Select Remove All from the menu**.
- To create a Feedback node from scratch: Open the Functions palette. Copy a Feedback node from the Structures palette into the Block diagram and place it as in shown in this figure.
- Wire a constant of value 2 to the Initializer terminal to the left.
- Open the front panel of your VI. Save the VI. Run the VI. Hopefully, the VI works correctly.

[Table of contents]

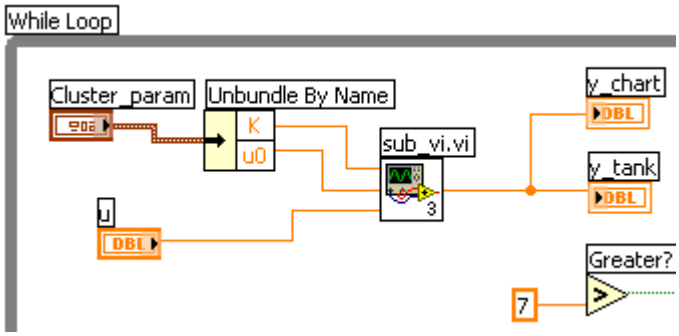
8.4 SubVIs

What is a SubVI?

In all programming languages, e.g. Visual Basic, C++, Delphi, MATLAB, you can create your own *functions* which can be used in - or be called by - the main program. Similarly, in LabVIEW you can create your own *SubVIs*. A SubVI is a special kind of VI as it has input and/or output *connectors*. SubVIs are used in the same way as function blocks in the Block diagram and they communicate with the rest of the Block diagram via the connectors on the SubVI block.

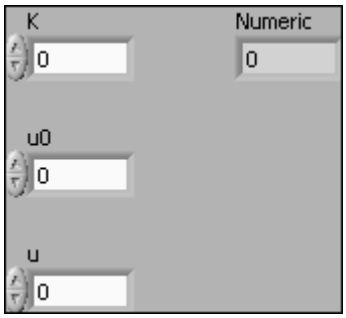
The figure below shows an example of a Block diagram containing the SubVI named **sub_vi.vi**. The Block diagram belongs to a VI similar to level_meas.vi. The SubVI implements the mathematical formula

$$y = K*(u-u0)$$

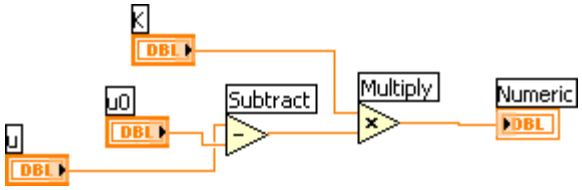


Block diagram containing the SubVI named sub_vi.vi

A SubVI has a Front panel and a Block diagram. By double-clicking on the SubVI block, the Front panel is opened, and then the Block diagram can be opened in the usual way. As you see in the figure below, the Block diagram implements the formula y = K*(u-u0).



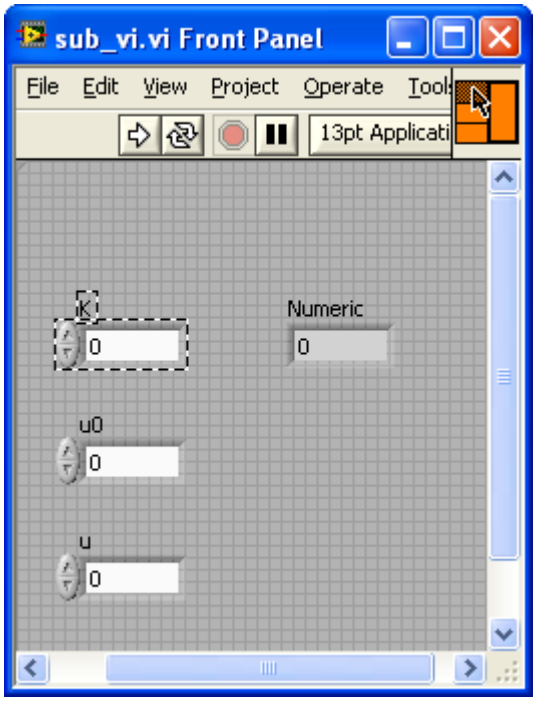
Front panel of the SubVI sub_vi.vi



Block diagram of the SubVI sub_vi.vi

The Front panel *controls* (and their corresponding Block diagram terminals) represent *input connectors* on the SubVI. The Front panel *indicators* (and their corresponding Block diagram terminals) represent *output connectors* on the SubVI.

Where are the *connectors* of the SubVI? You can see them by right-clicking the SubVI icon which is the icon at the upper right corner of the Front panel window, and selecting **Show Connector** in the context menu, see the figure below. By clicking one of the connectors in the Connector pane, the corresponding element on the Front panel is highlighted. In the figure below the connector corresponding to the K element is selekcted and the K control element is thereby highlighted.



The Connector pane of the SubVI sub_vi.vi

Why would you use a SubVI? You can use a copy of the SubVI in several places in one or more Block diagrams, thereby *reusing* code effectively. For example, you may implement a mathematical formula in a SubVI. Furthermore, you can also make your Block diagram *look simpler* by putting code into a SubVI.

How to add an existing SubVI to the Block diagram

As an example, let us add an existing SubVI to the Block diagram of an existing VI. (The application is based on level_meas.vi.)

- Download **main_vi.vi** to any folder you prefer. (This VI misses some code in the Block diagram, but we will now add what is missing.)
- Open the block diagram of **main_vi.vi**.
- Download the SubVI sub_vi.vi to the same folder as above (however, you may download the SubVI to some other folder if you prefer).
- Add **sub_vi.vi** to the Block diagram of **main_vi.vi** as follows: **Right-click in the Block diagram where you want to place the SubVI, see the figure of the resulting Block diagram above / Choose Select a VI in the Functions palette / Browse to downloaded sub_vi.vi, and drop it into the Block diagram**.
- Connect wires to the SubVI connectors as shown in the figure of the resulting Block diagram above.
- Save the VI. Open the front panel. Run the VI. Let's hope it works...

How to create a SubVI

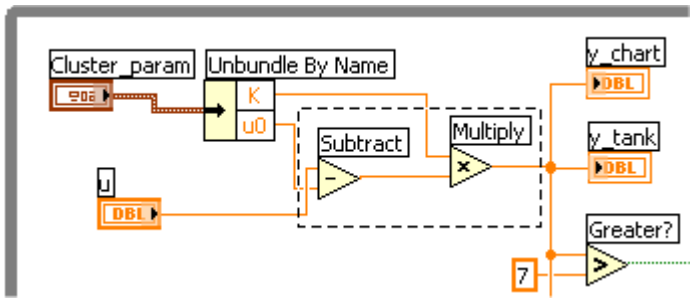
In the above section we added an *existing* SubVI into a Block diagram. But how do you *create* a SubVI? It can be done in two ways:

- **Manually**: You start with a blank VI and develop it further as a SubVI.
- **Automatically**: You start with some existing Block diagram code that you want to put into a SubVI, and tell LabVIEW to create a SubVI based on that code.

In the following only the automatic method will be described in detail.

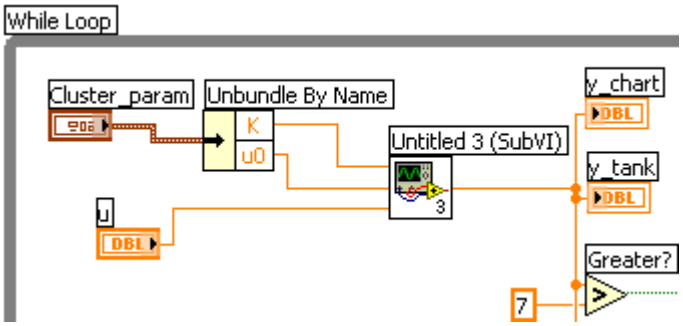
Let us try to create a SubVI using the automatic method:

- Download level_measurement.vi to any folder you prefer, and rename it as **my_main_vi.vi**.
- Open the block diagram of **my_main_vi.vi**.
- Mark the code that will be put into the SubVI, see the figure below.



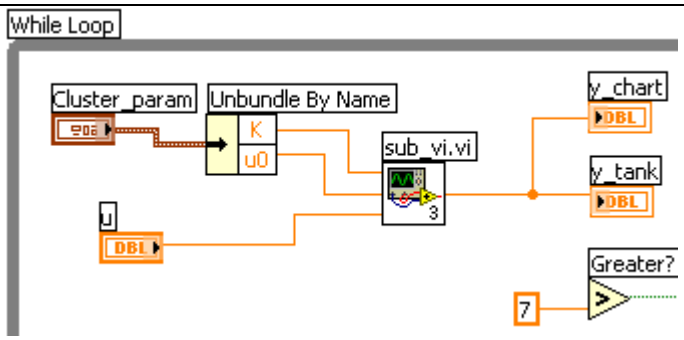
Marking the code that will be put into a SubVI

- Let LabVIEW create the SubVI automatically with the following menu selection: **Edit / Create SubVI**. The figure below shows the resulting Block diagram.



The Block diagram of the main VI with the SubVI with the preliminary name Untitled 3

- The SubVI automatically gets a name as Untitled 3 (or some other number). Open the SubVI by double-clicking on it, and then save it with the name e.g. sub_vi.vi. The name of the SubVI in the Block diagram is then updated accordingly, see the figure below.

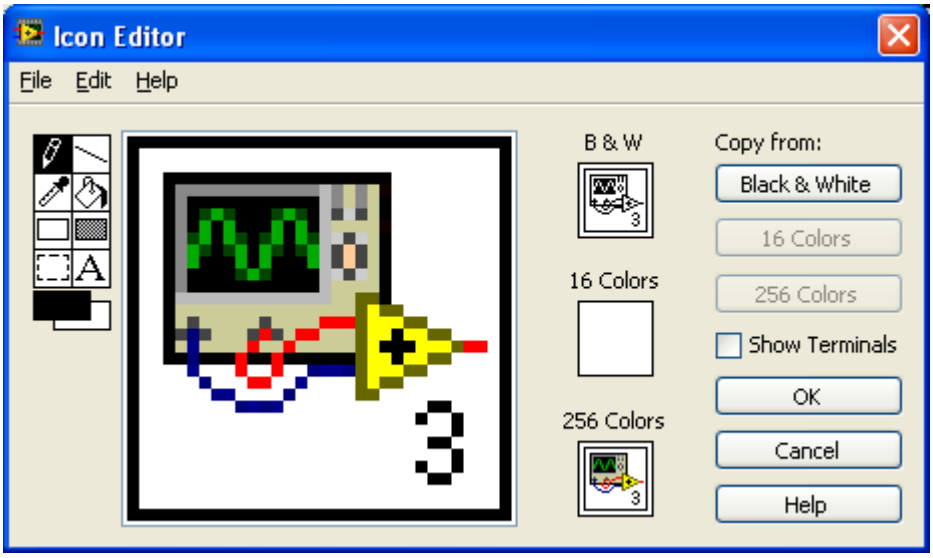


Block diagram containing the SubVI named sub_vi.vi

- Save the VI. Open the front panel of the VI. Run the VI. It works, hopefully.

Here are a few additional comments to SubVIs:

- **File saving**:
  - The ordinary **File / Save** menu just saves the main VI.
  - To save also the SubVI(s), select the menu **File / Save All**.
  - To save all the involved files (here: the main VI and the SubVIs) into one *LabVIEW Project file* (however, LabVIEW project is not described here): **File /Save As**, thereby opening a **Save As** dialog window. In this dialog window, select the **Duplicate hierarchy to new location** option, etc.
- **Editing the SubVI icon**: Open the Icon Editor as follows: Right-click the SubVI icon which is the icon at the upper right corner of the Front panel window, and then select **Edit icon** in the context menu, see the figure below. Now you can edit the icon using the graphical tools at the left part of the editor.



The Icon Editor

- **Setting a SubVI to become reentrant**: If several copies of the same SubVI will be used in one, or in several Block diagrams (of other VIs) it is important that each of these copies are independent of each other. This is implemented by making the

original SubVI *reentrant*. To define a SubVI as reentrant: **Right-click on the SubVI icon in the upper right corner of the SubVI Front panel / Select VI Properties / Select Category: Execution / Select Reentrant Execution**.

- **Editing the SubVI**: You can edit a SubVI as you would edit any VI, e.g. you can add code to the Block diagram. You can also add or remove Front panel controls and indicators corresponding to new SubVI inputs or outputs, respectively. To edit the connectors (representing the SubVI inputs and outputs): **Right-click on the SubVI icon in the upper right corner of the SubVI Front panel / Select Show Connector**, therby opening a context menu containing several options for editing the Connector pane, cf. this figure. For example, you can add and remove inputs/outputs, can change the pattern of the connectors, disconnect a connector from a Front panel element, etc.

[Table of contents]

8.5 Logfile writing and reading

Introduction

LabVIEW has several functions for writing data to files - both continuous writing (online, while the VI runs) and batch (offline) writing. LabVIEW has also functions for reading data in such logfiles. These functions are available on the **File I/O** palette.

File formats

The file formats that is supported by the above two File I/O functions are:

- **TDMS** (Technical Data Management - Streaming) which is are binary files in an internal LabVIEW file format. The TDMS format gives more effective and accurate data storage than the LVM format. TDMS files may be opened in LabVIEW, of course, and in NI DIAdem which is a software tool for for managing, analyzing, and reporting data in logfiles. TDMS files can save data in an organized way using a number of Groups and a number of Channels within a given Group. (The older TDM file format is still supported.)
- **LVM** (LabVIEW Measurement) which are text files that contain data that can be read by a human being because numbers are represented as text. For example, the number 0.231 is stored as the text (or string) "0.231". A large benfit of storing numerical data in the text format is that the file can be opened and displayed in any tool that supports text files, e.g. MS Word, Notepad, Excel, Matlab, Web browsers. (It may be necessary to change the file name extension from lvm to e.g. txt or dat before opening the file in such tools). Thus, text files provides great portability. However, the text files will be larger than if the data were written in a binary format, and the data are stored with less accuracy, but these issues are not important on ordinary PC's, but may be important on dedicated computers with limited storage.

As a general rule I suggest that you use the binary file format (TDMS) unless you really

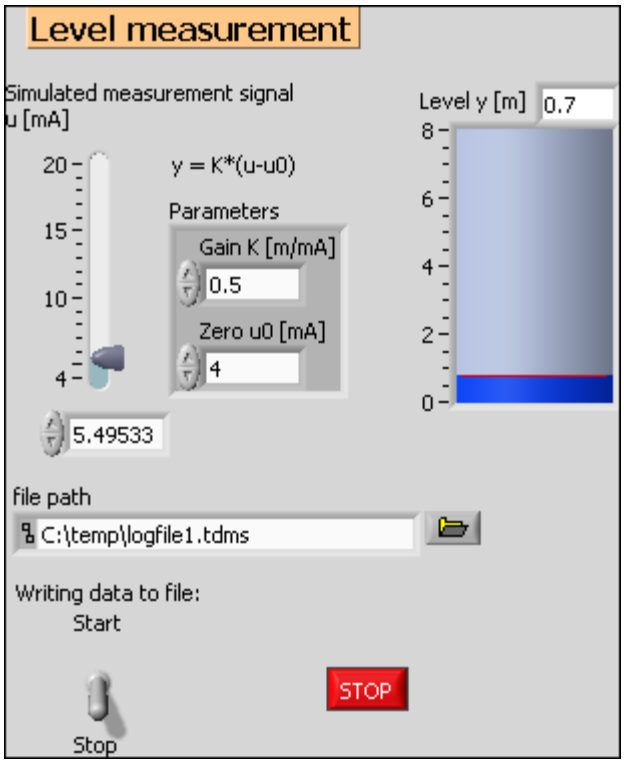
need the text format (LVM).

Both TDMS (binary) and LVM (text) files writing and reading are described in the subsequent sections.

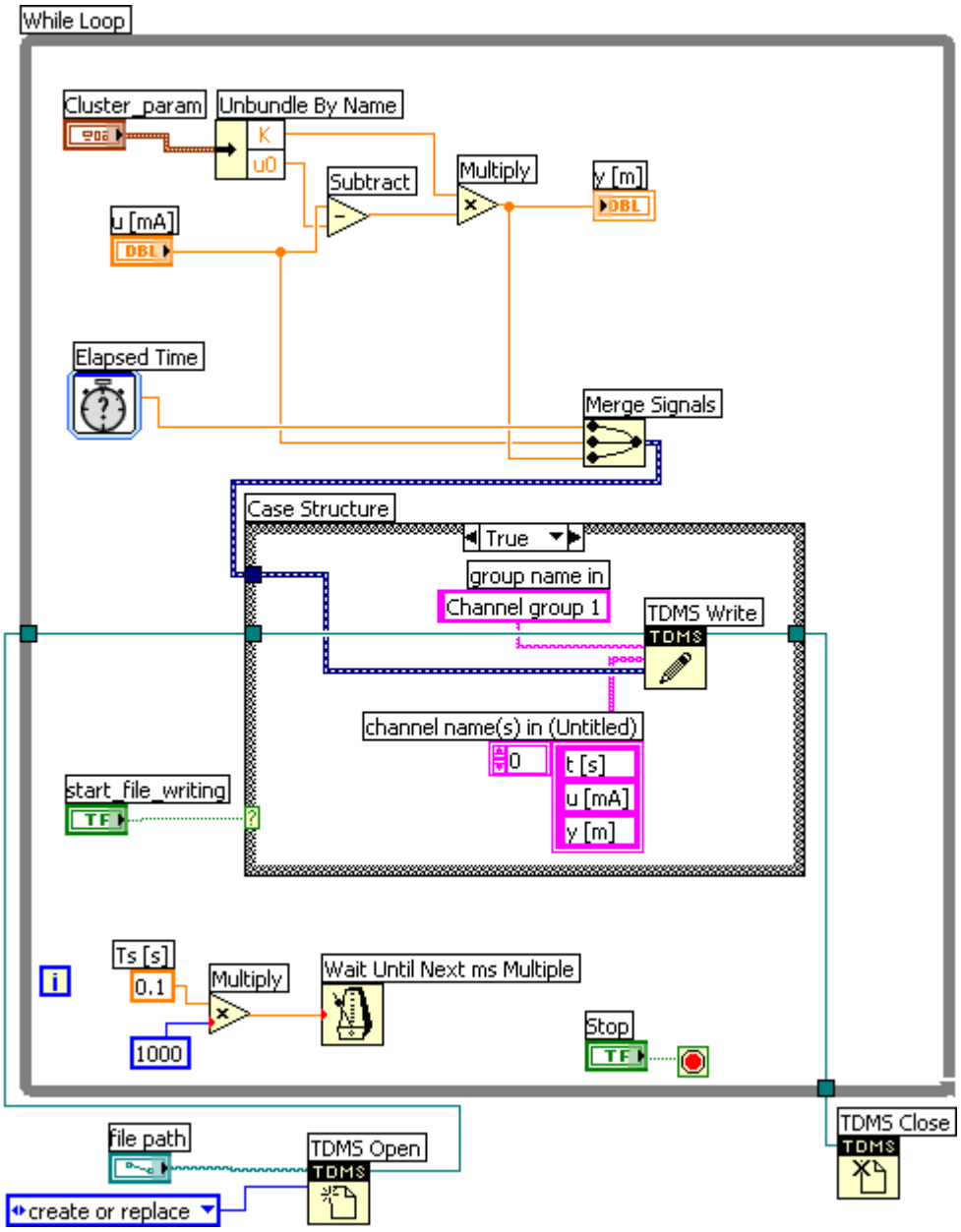
Writing data to and reading data from a TDMS file

Writing data to a TDMS file

The figures below show the Front panel and the Block diagram of write_to_tdms_file.vi.



Front panel of write_to_tdms_file.vi



Block diagram of write_to_tdms_file.vi

Comments:

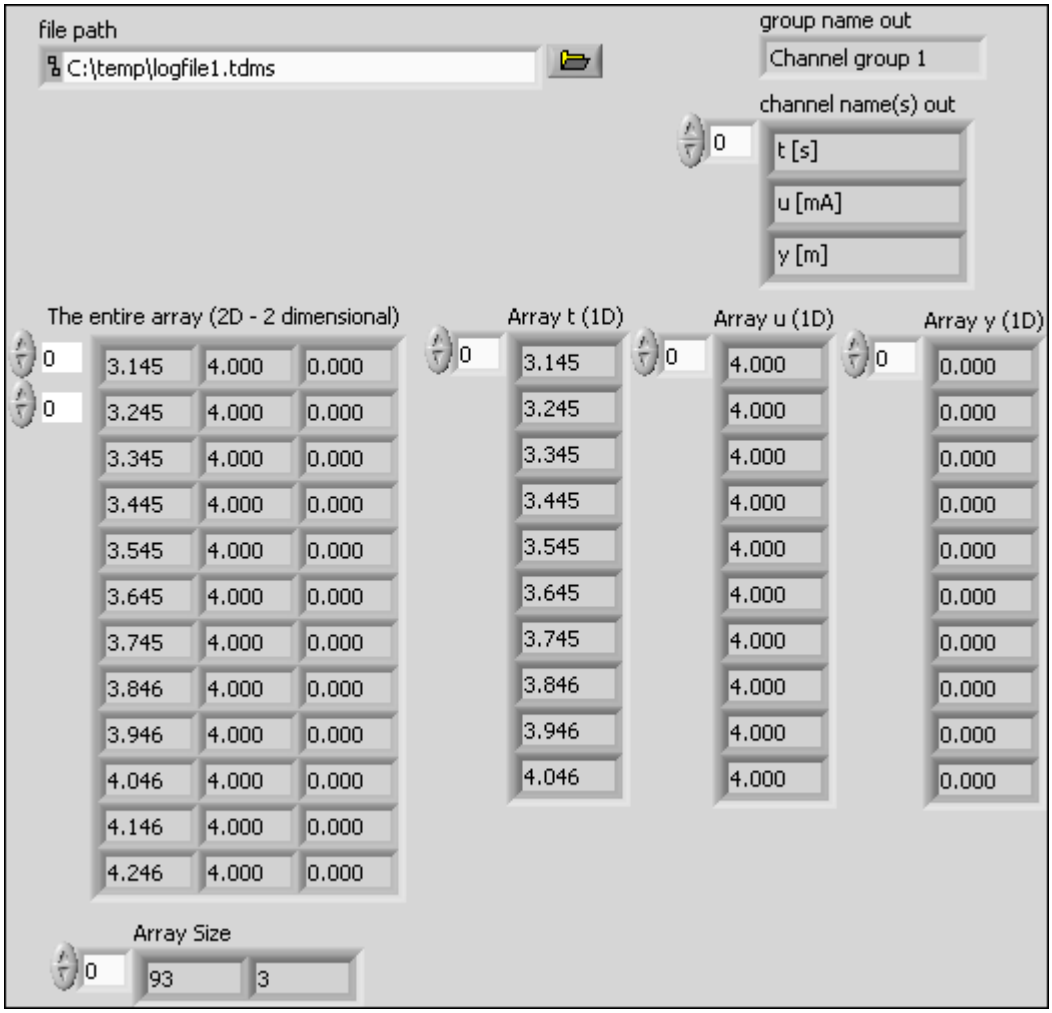
- The **TDSM Write** function writes the data continuously to the file. (It contains a data buffer so that the While loop is not unnecessarily delayed by the file writing.)
- The **Case** structure is used to start, and stop, the file writing. (In the figure the True case is shown. The False case does not contain any code except the file path wire from the input tunnel to the output tunnel.)
- Three signals - t, u, y - are collected with a **Merge Signals** function (which is on the **Express / Signal Manipulation** palette) before they are fed into the the **TDMS Write** function.
- The **channel group name** and the **channel names** are written to the logfile together with the data. This is convenient for later use of the data, e.g. when

opening the data in the DIAdem tool.

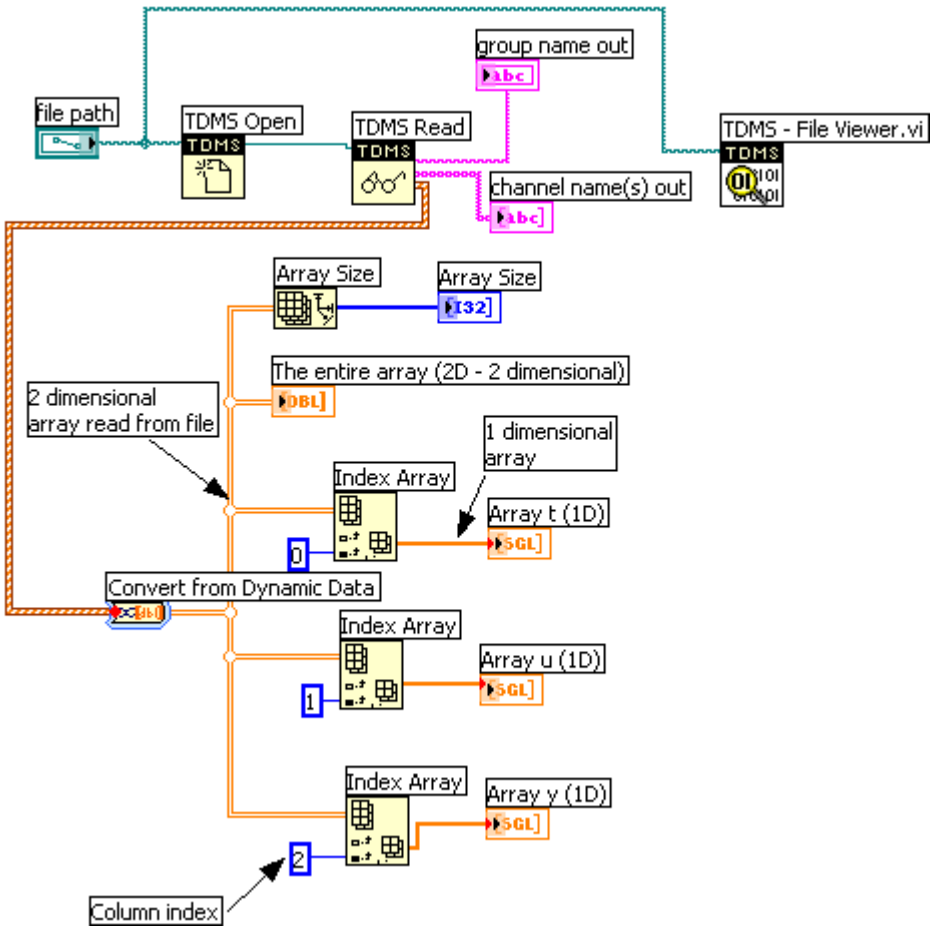
- In this example the points of time generated by the **Elapsed Time** function are saved as data together with the values of u and y. (If the data to be saved are Waveform data the time information is contained in the data, and then it is not necessary to define the time signal explicitely as in the example above.)

Reading data from a TDMS file

The figures below show the Front panel and the Block diagram of read_from_tdms_file.vi.

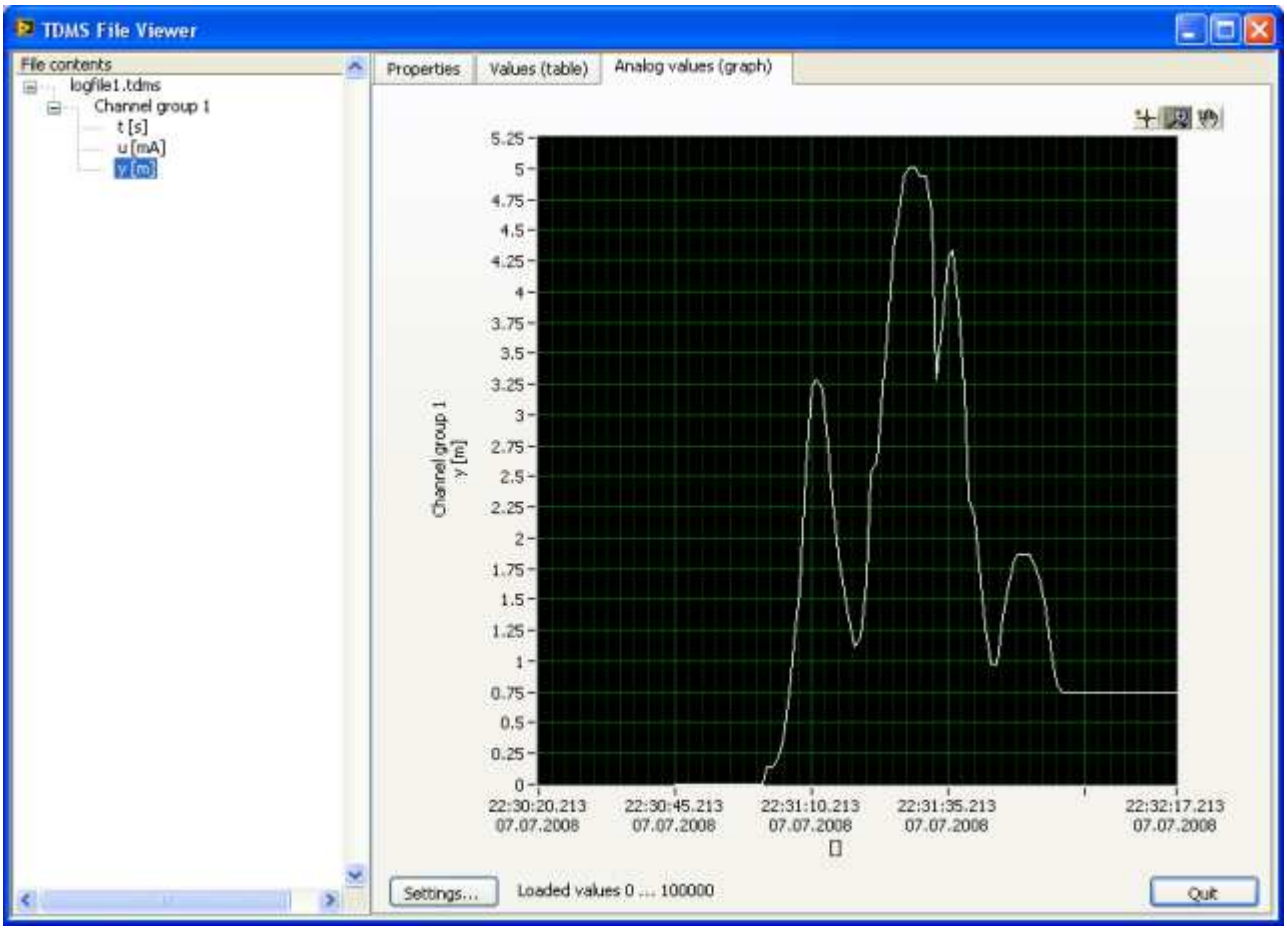


Front panel of read_from_tdms_file.vi



Block diagram of read_from_tdms_file.vi

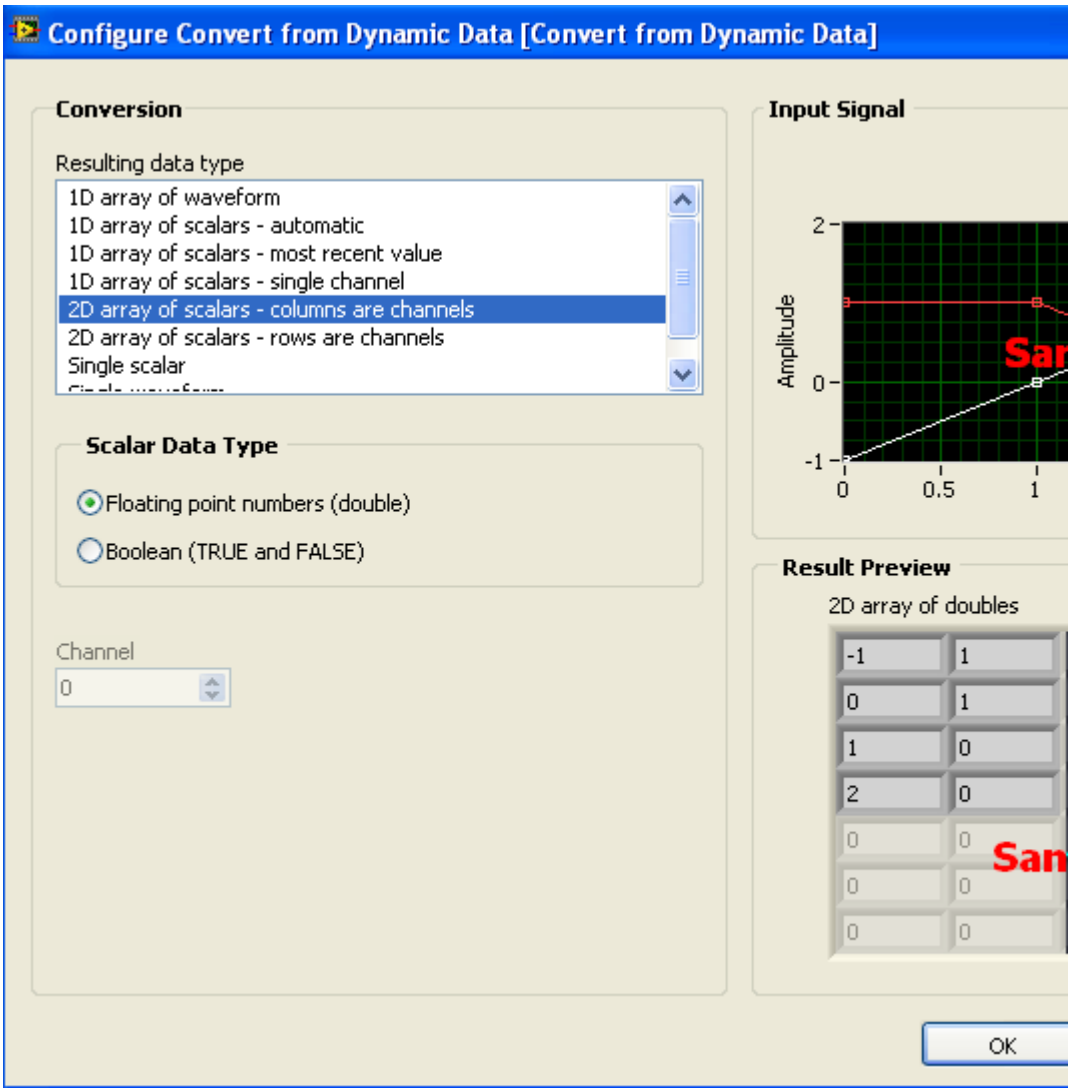
If you run the read_from_tdms_file.vi the TDMS Viewer is also opened, in addition to the Front panel, see the figure below. The **TDMS Viewer** is a graph tool for quickly plotting the channels in the logfile. Note that you have to click the Quit button in this window to quit the TDMS Viewer, and to stop the VI.



TDMS Viewer

Here are comments to read_from_tdms_file.vi:

- The VI does not contain any While loop since the read from file operation is done once. (This is the typical situation.)
- The **TDMS Read** function (on the **File I/O** palette) reads data from the logfile. Note that this function has outputs containing the group name and the channel(s) name(s).
- The data that are read from the file are converted to a ordnary 2-dimensional array containing the three columns of data (t, u, y). The convertion is made by the **Convert from Dynamic Data** function which is on the **Express / Signal Manipulation** palette. This conversion function can be configured (i.e. selecting the correct data type) by double-clicking it, cf. the dialog window shown in the figure below.
- The three **Index Array** functions are used to extract each of the columns from the data and create 1 dimensional arrays, labeled Array t (1D), Array u (1D) and Array y (1D).
- The Block diagram also contains code for displaying the array in an array indicator and to display the size of the array on the Front panel.
- The **TDMS Viewer** function is a graph tool for quickly plotting the channels in the logfile.



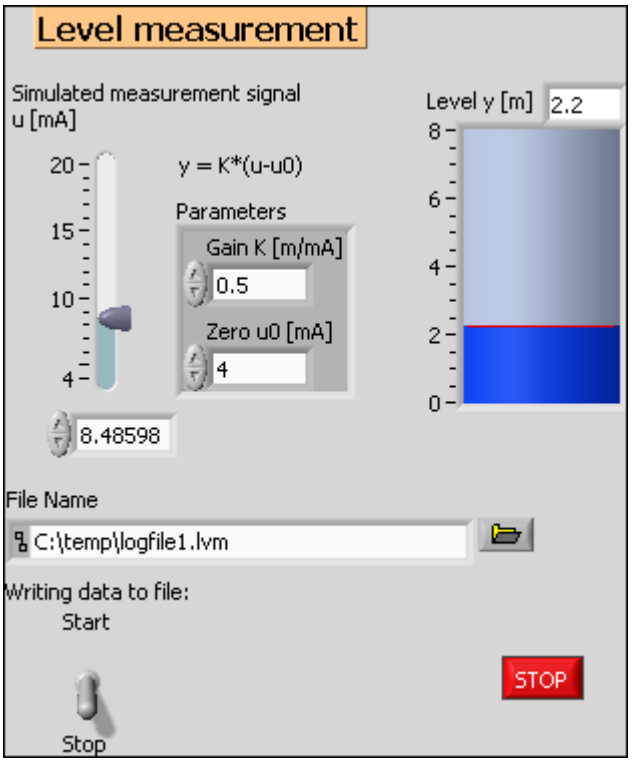
Dialog window of the Convert from Dynamic Data function

[Table of contents]

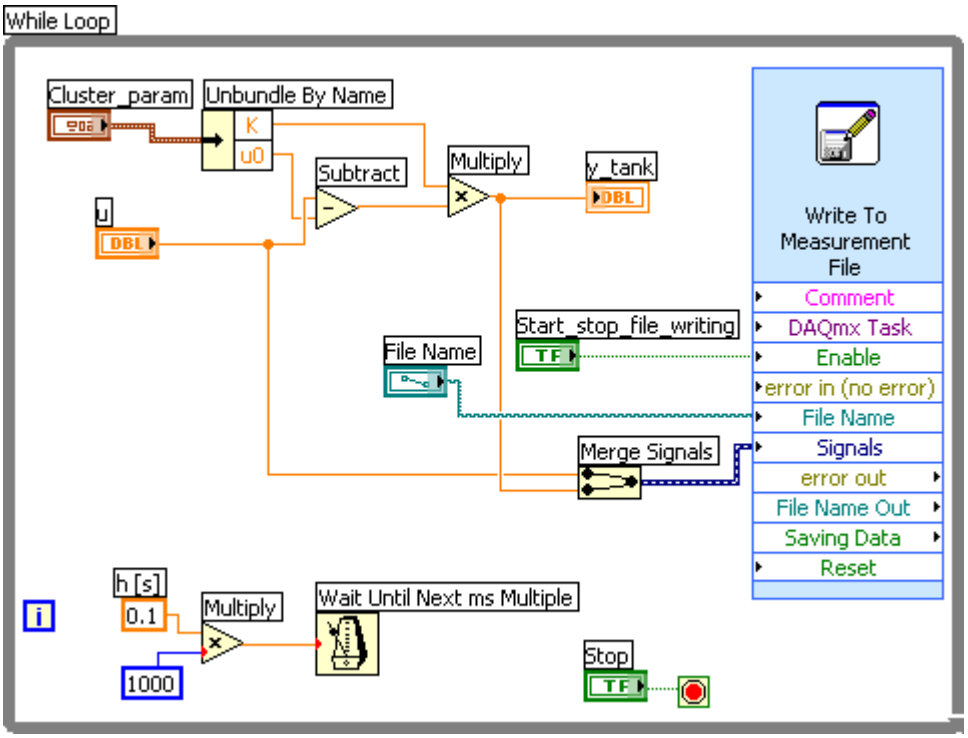
Writing data to and reading data from a LVM file

Writing data to a LVM file

The figures below show the Front panel and the Block diagram of write_to_lvm_file.vi.



Front panel of write_to_lvm_file.vi



Block diagram of write_to_lvm_file.vi

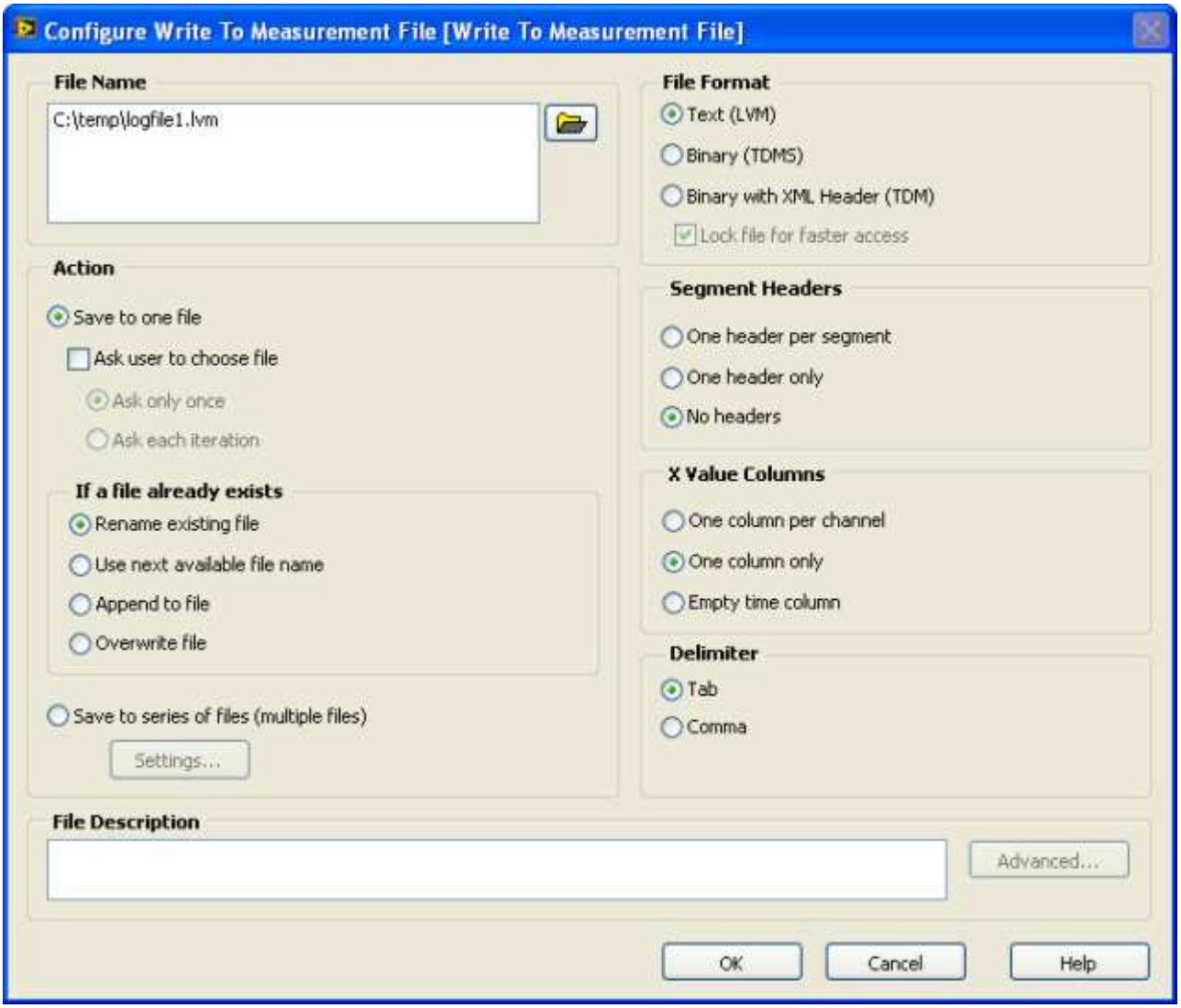
Comments:

- In this example the data are writting to a file with the **Write To Measurement File** function, which is an Express VI. (The **Write to Spreadsheet File** function could

also have been used.)

- The boolean (switch) **Start_stop_file_writing** is used to start (enable) and stop the continuous file writing. This boolean terminal (control) is connected to the **Enable** input to the **Write To Measurement File** function.
- Two signals, u and y, are collected with a **Merge Signals** function (which is on the **Express / Signal Manipulation** palette) before they are fed into the the **Write To Measurement File** function. Regarding writing the time signal, cf. the comments about the **X Value Columns** in the list of comments below.

By double-clicking the **Write To Measurement File** function in the block diagram a dialog window opens, see the figure below.



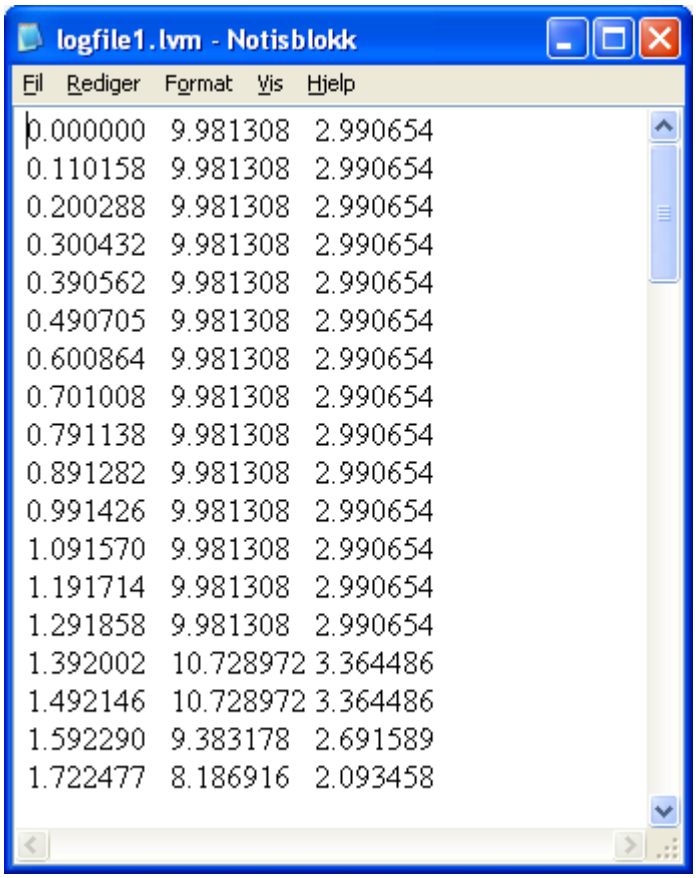
Dialog window of the Write To Measurement File function in the block diagram of write_to_lvm_file.vi

Comments to the dialog window shown in the figure above:

- **File Name** box: Here you can select the file name, but if you have wired a file name control (or constant) to the File Name input to this Express VI, the wired name is used instead (this is the case in the example above).

- **File Format** radio buttons: Here you select the data format: LVM or TDMS. (The TDM file format also available is a simpler and not so flexible data format. TDMS is a newer format.) It seems it is not possible to change the LVM file name extension into e.g. TXT (but you can of course do this change manually using Windows Explorer).
- **Segment Headers**: By selecting **No headers** the files will contain just plain data, which is convenient if you plan to import your data into tools as Excel, Matlab, etc. If you choose to have header(s), you will have to remove the header (perhaps manually) from the data file before you can load the file into tools mentioned above.
- **X Value Columns**: The X values are actually the time stamps. By selecting **One column only** the file will contain one left column containing the time stamps. These time stamps are recorded and written to the file automatically, so you do not have to create any time signal yourself.
- **Delimiter**: The delimiter is the character that separates columns.
- **File Description**: The text you write here will appear in the header of the file, unless you have selected the **No headers** options mentioned above.
- Note: There is no way to define the group name or the channels names for the data in the **Write to Measurement File** Express VI.
- You may wonder why I did not use the **Write To Measurement File** function for writing the data in TDMS format (cf. the previous section). In stead I used the **TDMS Write** function. The reason is that the channel group name and the channel names can not be defined in the **Write To Measurement File** function, but if this is not an issue, you may of course use **the Write To Measurement File** function for writing TDMS data.

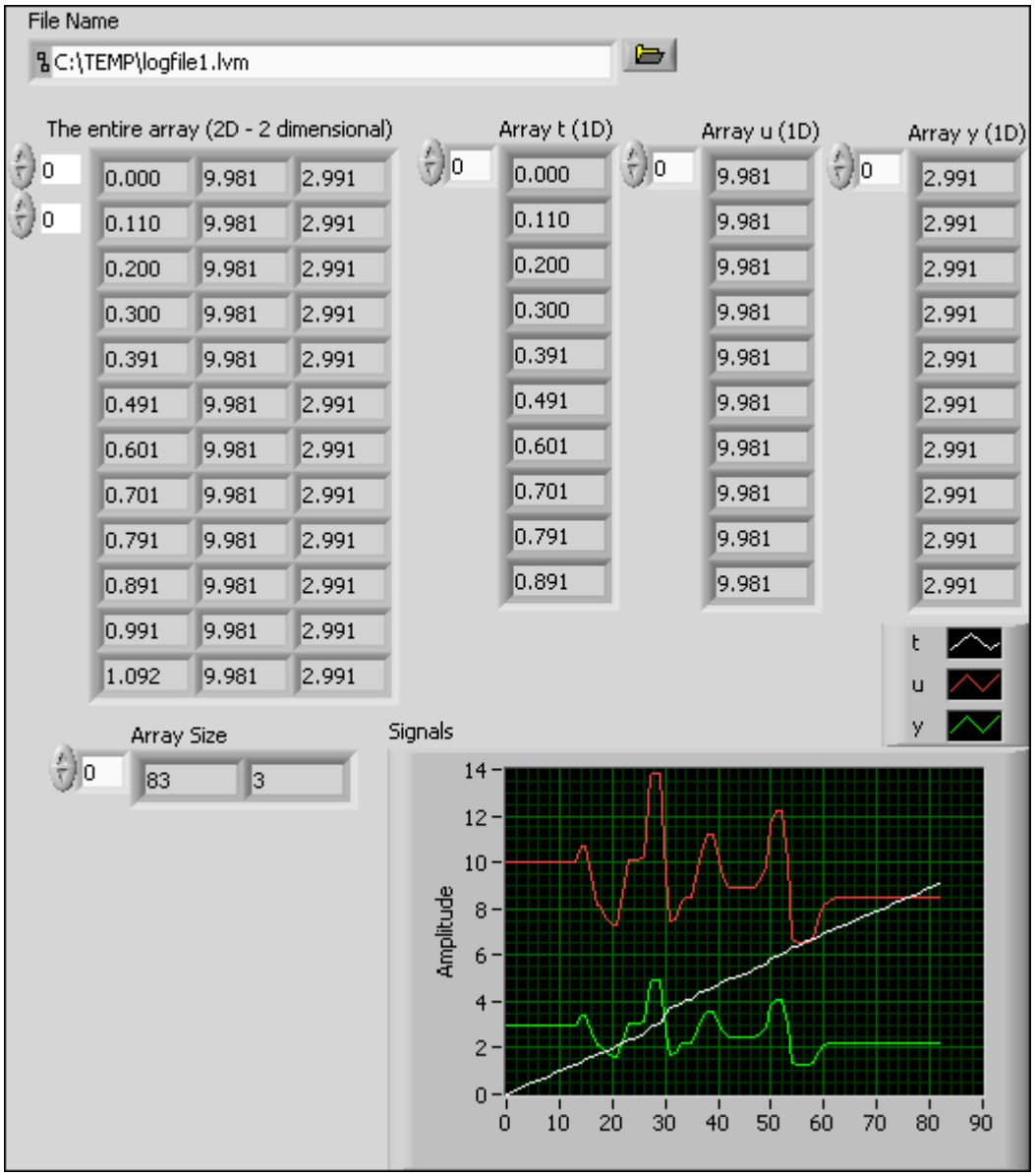
In our example, the result of running write_to_lvm_file.vi is a text file named **logfile1.lvm**. The figure below shows the file opened in Notepad from one experiment. (The cycle time of the While loop was set to 0.1 s, cf.the block diagram shown above. As you see, the actual cycle time varies a little, but it is very close to the specified cycle time of 0.1 s.)



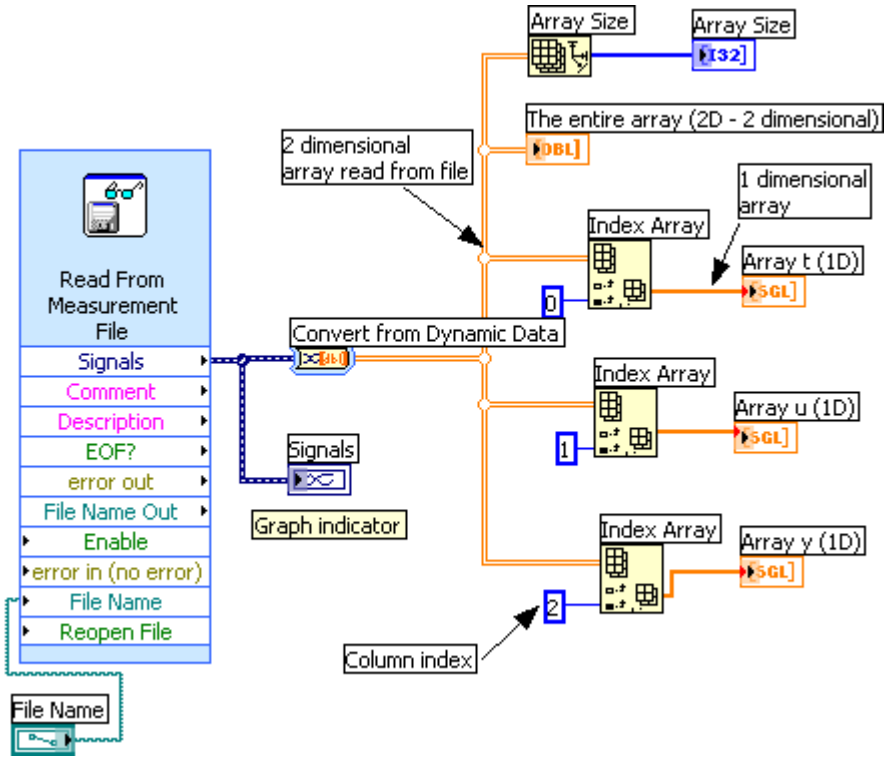
logfile1.lvm file opened in Notepad

Reading data from a LVM file

The figures below show the Front panel and the Block diagram of read_from_lvm_file.vi.



Front panel of read_from_lvm_file.vi



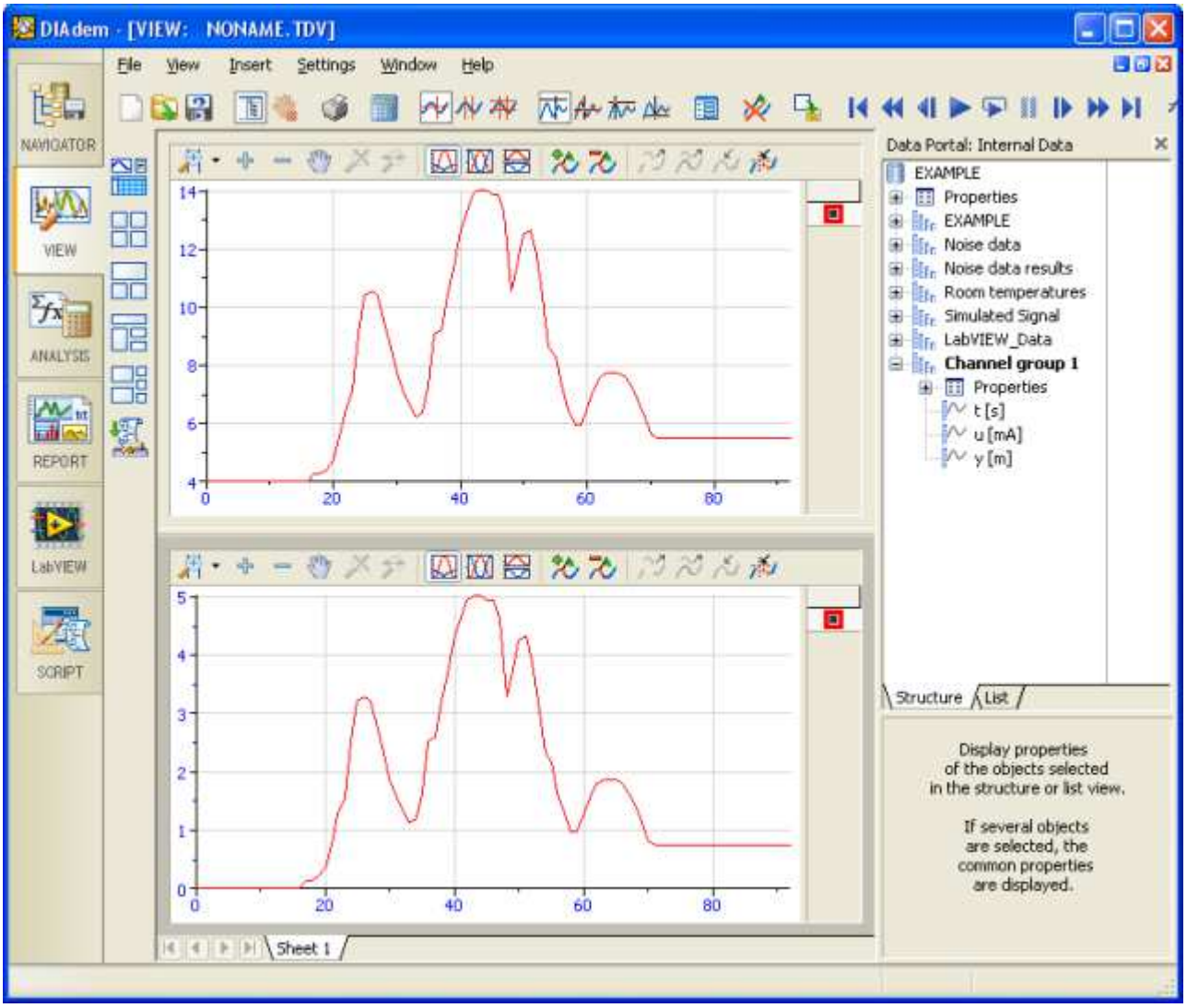
Block diagram of read_from_lvm_file.vi

Comments to read_from_lvm_file.vi:

- The VI does not contain any While loop since the read from file operation is done once. (This is the typical situation.)
- The **Read From Measurement File** function reads data from the logfile. (By double-clicking the function a dialog window is opened. The parameters in this window should be self-explanatory.
- The data that are read from the file are converted to a ordnary 2-dimensional array containing the three columns of data, cf. the logfile1.lvm shown above. The conversion is made by the **Convert from Dynamic Data** function which is on the **Express / Signal Manipulation** palette. Note that you can configure the **Convert from Dynamic Data** function (i.e. selecting the correct data type) by double-clicking it.
- The three **Index Array** functions are used to extract each of the columns from the data and create 1 dimensional arrays, labeled Array t (1D), Array u (1D) and Array y (1D).
- The **Signal** indicator is a **Graph** indicator which is created by right-clicking on the signal line out from the Expresss VI. The x-axis shows the time-index (integer). You can configure the Graph via its Property dialog window so that the x-axis shows time in secons (or minutes etc.).
- The Block diagram also contains code for displaying the array in an array indicator and to display the size of the array on the Front panel.

8.6 Displaying and analyzing data in DIAdem

DIAdem is a powerful tool for displaying and analyzing data offline. (It is a National Instruments tool, but it is used independent of LabVIEW.) Both TDMS and LVM data files can be imported into DIAdem. The figure below shoes the DIAdem dialog window. DIAdem is quite intuitive to use. It is beyond the scope of this LabVIEW tutorial to describe DIAdem in further details here.



The DIAdem tool for displaying ans analyzing data

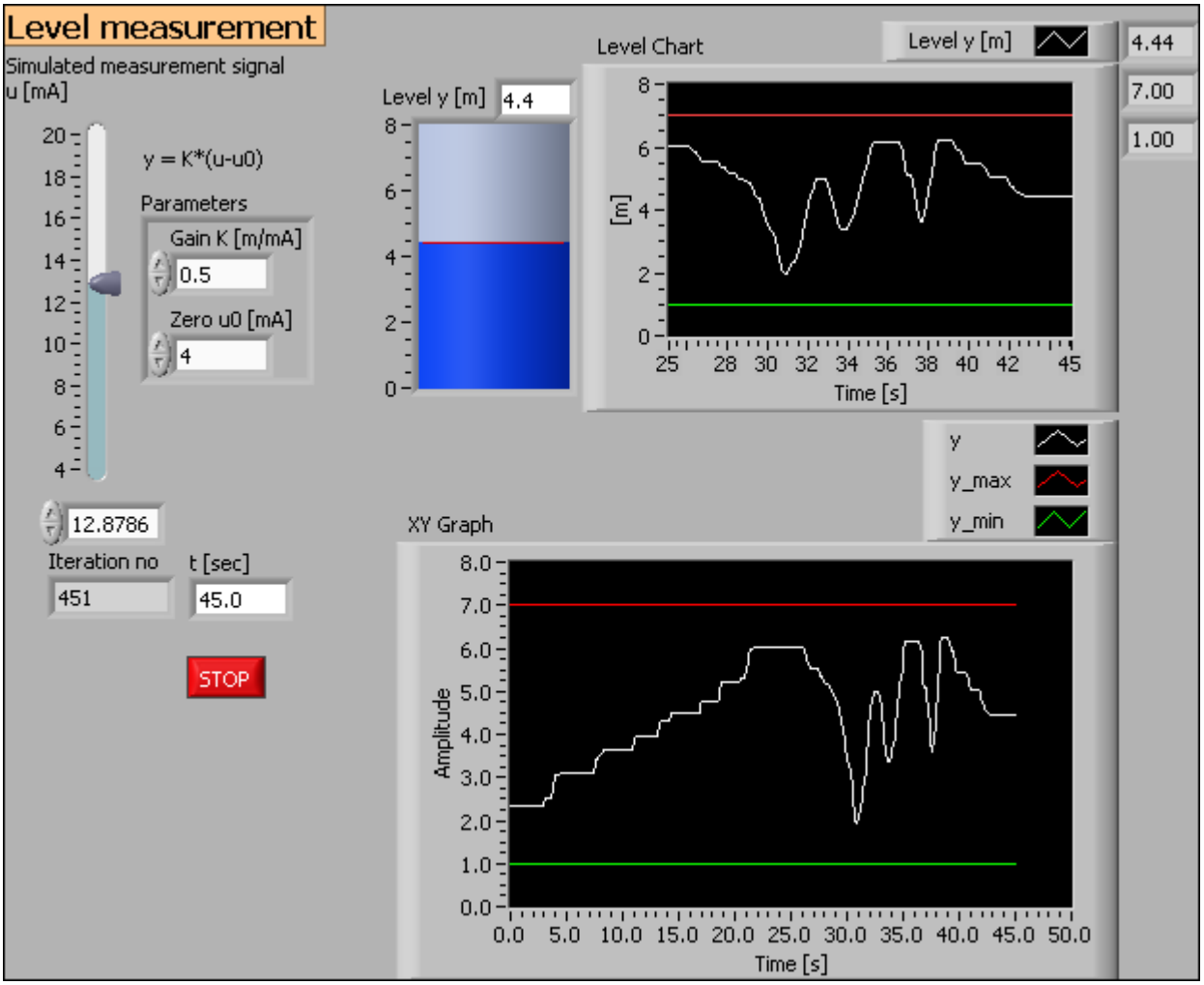
[Table of contents]

8.7 Plotting in graphs

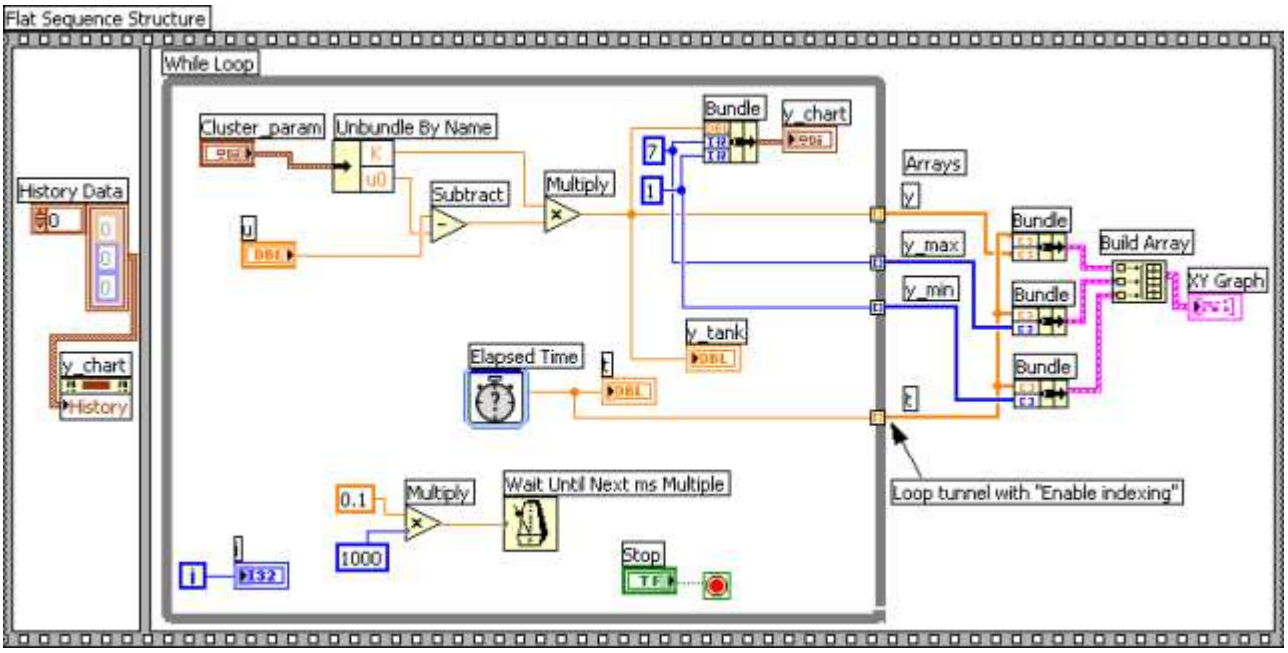
You have already learnt to plot signals in *Charts*. A Chart is continuously updated by the most recent signal values, and the x axis is typically the time axis. *A graph* is another type of plots. Graphs plots any array versus any other array in an XY (and the increment in the X array does not have to be constant). There are several Graphs on the Graph palette on the Functions palette, the most general being the *XY Graph*.

Here is an example based on a somewhat simplified verson of level_meas. The three

arrays consisting of the historical values of the y, y_max, and y_min are plotted against one time array containing the discrete times covering the elapsed time as the VI was run. The Chart is updated once the While loop has stopped. (The VI also contains a Chart, which is continuously updated as the VI runs.) Below are the Front panel and the Block diagram of graph.vi.



Front panel of graph.vi



Block diagram of graph.vi

Comments to this VI:

- The XY Chart indicator is available on the **Controls palette / Graph subpalette**.
- In the Block diagram of graph.vi the XY Graph terminal is placed outside the While loop. This is necessary since the Chart is to be updated once the While loop has stopped. (The Chart terminal is however placed inside the While loop since it is to be updates as the While loop runs.)
- The arrays of y, y_max, y_min, and t are generated in the respective tunnels on the right border of the While loop. These tunnels are set to "Enable Indexing" (this is set by right-clicking on the tunnel and selecting Enable Indexing in the context menu).
- **Bundle** functions are used to bundle the t array with the y array, the t array with the y_max array, and the t array with the y_min array. The output of a **Bundle** function is a *cluster*. Then, these three clusters are collected using a **Build Array** function. The resulting array of clusters of arrays (!) is then wired to the XY Graph terminal.

[Table of contents]

8.8 Structuring VIs using parallel While loops

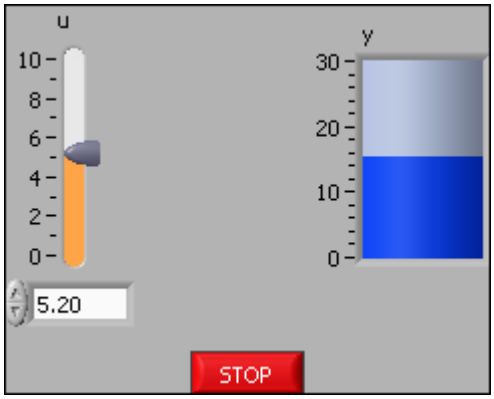
Often your application implements a number of different tasks to be executed simultaneously, e.g.

- Reading measurement signals from input devices
- Signal processing, e.f. lowpass filtering
- Simulation
- Calculation of a control signal using a feedback controller, e.g. a PID controller
- Writing control signals to output devices

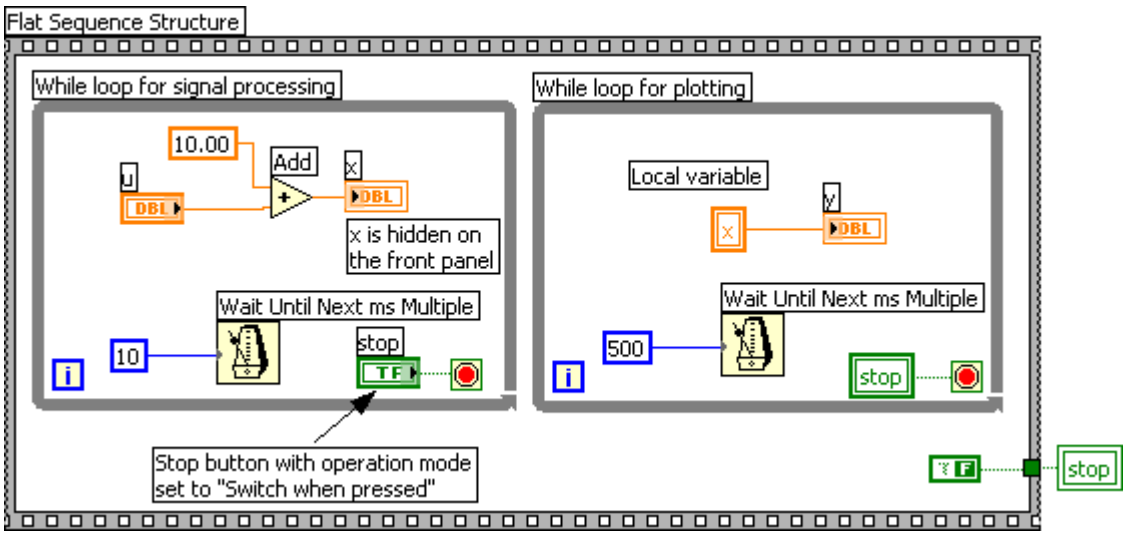
- Plotting data in charts on the Front panel
- Saving data to a file

One conventient way to organize such parallel tasks is to put the tasks into individual While loops that run in parallel (simulateneously). These While loops may run with different cycle times. LabVIEW will allocate resources to each While loop (parallel tasks) so that each get a cycle time equal to or close to the cycle time set by the Metronome function (Wait Until Next ms Multiple).

Here is an example: Below are the Front panel and the Block diagram of parallel.vi.



The Front panel of parallel.vi

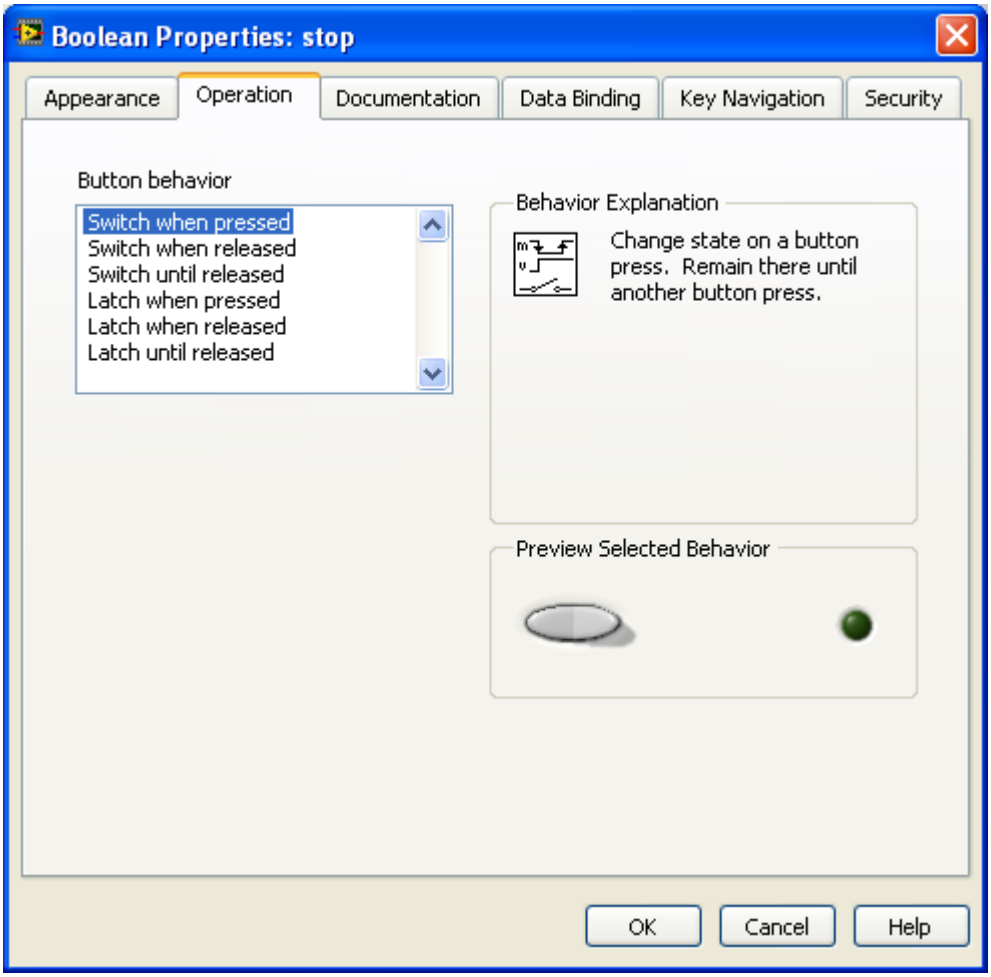


The Block diagram of parallel.vi

Comments to parallel.vi:

- The VI adds 10 to a user adjusted value of u (this is denoted Signal processing in the Block diagram). The resulting value is then plotted in a Chart (denoted Plotting).
- The VI contains two parallel While loops inside a Sequence structure. The two While loops implements signal processing and plotting, respectively.
- The While loops runs with different cycle times, 10 and 500 ms, respectively.

- Data is transferred from the Signal processing loop to the Plotting lopp using a *local variable* of x.
- The x indicator is hidden on the Front panel (**Right-click on the element / Hide indicator**).
- One Stop button is used to stop both loops. A Local variable of the button is used to stop the Plotting loop. Note: It is necessary that the button remains in the down position until both loops have stopped. Therefore the mechanical operation mode of the button is set to *Switch when pressed*. This is set in the Properties window of the Stop button, see the figure below.



The button behaviour is set to *Switch when pressed* in the Property window of the button

When both loops have stopped, the Stop button must of cousre pup out again. This is implemented with the Sequence structure embracing both While loops, and with the False constant being written to a Local variable of the Stop button outside the Sequence structure.

[Table of contents]

8.9 Text-based (C-) programming using Formula node

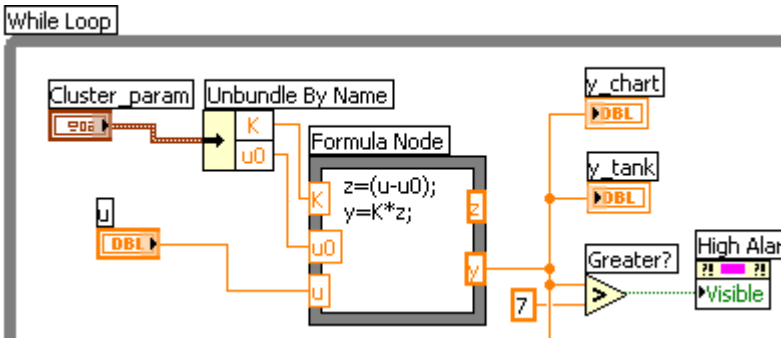
The Formula node is a window in the block diagram where you can write textual program code following the C-syntax. Using a Formula node for mathematical expressions is often more convenient than building the expression using elementary blocks in the ordinary graphical way in LabVIEW since it is easier to write and maintain textual mathematical expressions than drawing equivalent block diagram code.

Here is the interesting part of the block diagram of the VI formula_node.vi where the **Subtract** and **Multiply** functions have been replaced by a Formula node containing the textual code

z = u-u0;

y = K*z;

The VI is otherwise the same as level_meas.vi.



Block diagram of the VI formula_node.vi where the Subtract and Multiply functions has been replaced by a Formula node containing the textual code

Add a Formula node to the block diagram as follows:

- Save level_measurement.vi as my_formula_node.vi.
- Open the block diagram of my_formula_node.vi.
- Remove the Subtract function and the Multiply function.
- Insert a Formula node from the **Mathetmatics / Scripts & Formulas** palette in the same area where these functions were, cf. the figure above.
- Take a brief look at the Help information about the Formula node: **Right-click on the border of the Formula node / Help**. After reading close the Help window.
- Add an input to the Formula node: **Right-click on the left border of the node / Add input**. Give the input the name K by typing "K" at the input tunnel.
- Add two more input named u0 and u, respectively.
- Add two outputs on the right border of the Formula node with names z and y, respectively.
- Type the textual code inside the Formula node, cf. the figure above. Remember to end each expression by semicolon.
- Wire the proper signals to the three inputs, ans wire the output y to the

proper elements, cf. the figure above.
- Save the VI. Run the VI, and adjust the elements on the Front panel. The VI should behave as before.

You may wonder why you entered the code

```
z = u-u0;

y = K*z;
```

in stead of the somewhat simpler

```
y = K*(u-u0);
```

Actually, the latter would work perfect. However, often you will want to split an expression into smaller parts each consisting of expressions (which must be ended with a semicolon), and then having one final expression based on the intermediate results from these expressions. In such cases, remember to define one output for each of the left-hand variables used in the expressions. What actually happens is that each variable is *defined*. It is not necessary to wire the outputs to some terminal of functions outside the Formula node. As an alternative to defining variables using such "dummy" outputs, you can use ordinary variable declaring expressions as in the C language. In our case we could have used the following code:

```
float z;

z = u-u0;

y = K*z;
```

[Table of contents]

8.10 Text-based (Matlab-like) mathematics using MathScript

MathScript is a LabVIEW tool for executing *textual mathematical commands or expressions equal to Matlab*. Please see the Introduction to MathScript.

[Table of contents]

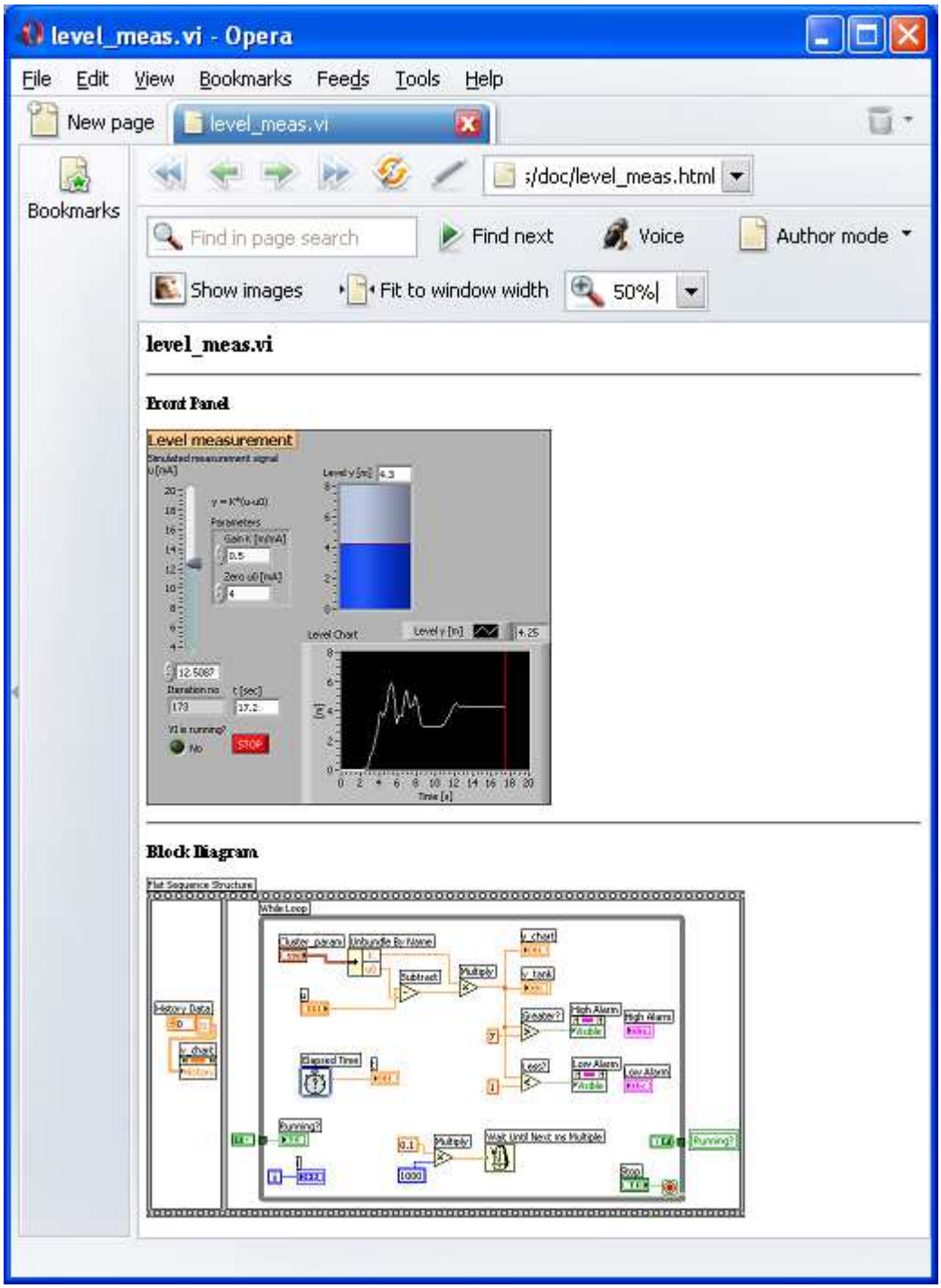
8.11 Generating documentation of your VI

You can create a documentation of your VI with various outputs:

- Paper print
- An HTML file which may be shown in any Web browser. Graphics show the Front panel and the Block diagram of the VI. You can edit the resulting HTML file in any HTML browser, e.g. Microsoft's FrontPage. The graphics files are ordinary graphic files.
- A RTF document which can be opened in various editing tools, e.g. Microsoft Word and Scientific Word.

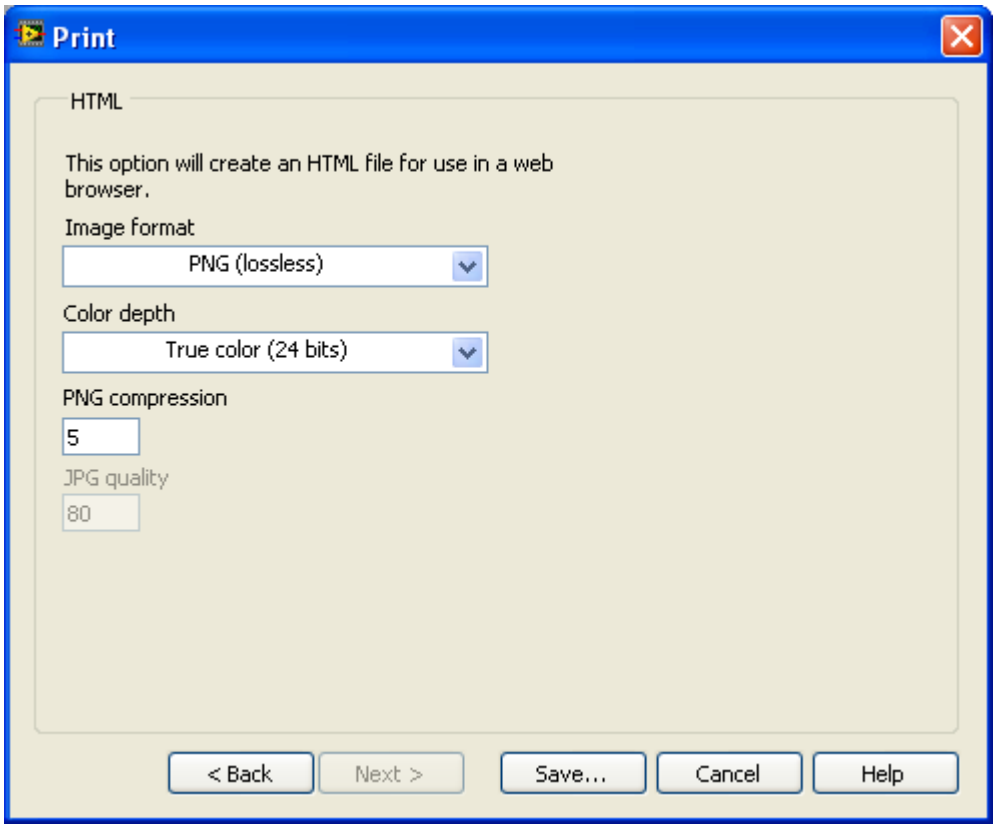
- A plain text document containing no graphics.

The figure below shows one such HTML file opened in a Web browser (Opera).



A documentation HTML file opened in a Web browser (Opera)

Creating the documentation is done via the **File / Print** menu. This opens a number of successive dialog windows which are not described in detail here. However, one tip for generating an HTML file is to select *True color* in the dialog window shown below, otherwise some colors on the Front panel may look strange in the resulting document.



The dialog window where you select Color depth. There you should select True color.

Let us try:

- Open the level_measurement.vi VI and save it in any folder.
- Run the VI for a while, and then stop it.
- Generate an HTML document showing the Front panel and the Block diagram. Select PNG as Image format (see the figure above) and True color as Color Depth. (There are many dialog windows where you have to select the proper options. I think the options are quite obvious.). Save the HTML document in any folder you want.
- Open the HTML document in our browser. It should appear as in this figure.

[Table of contents]

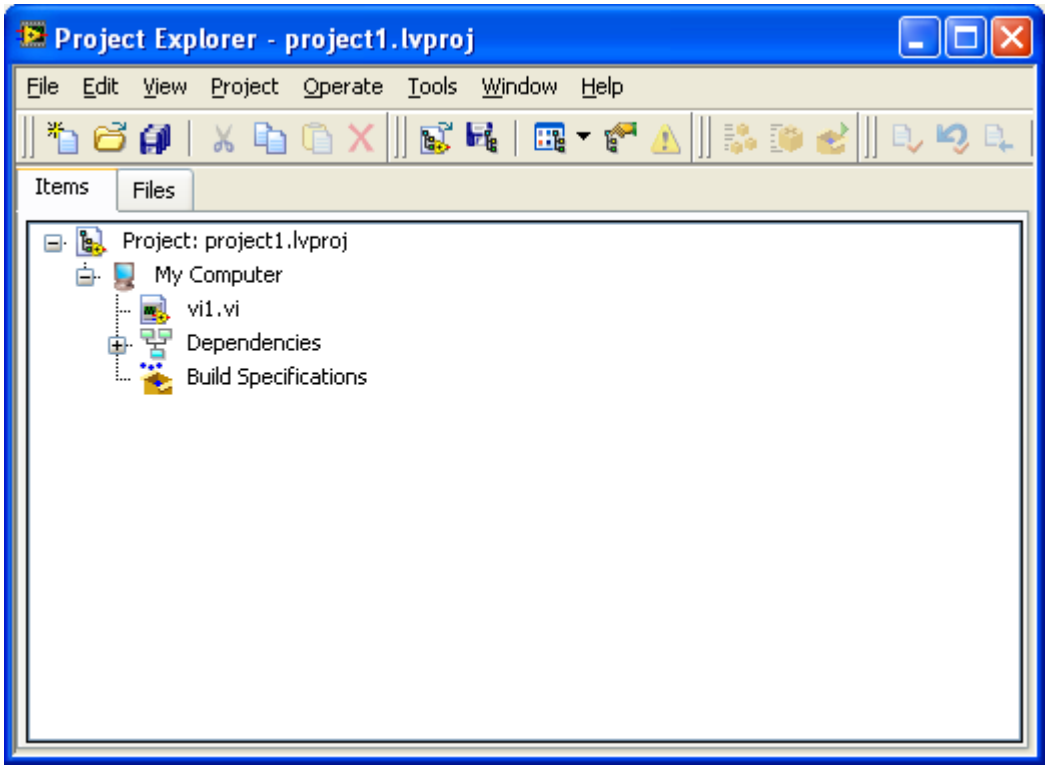
8.12 LabVIEW projects

LabVIEW projects defines a logical organization of various files used in an application. These files may be VIs, documentation files, support files, external code, data files and hardware configuration settings. LabVIEW projects makes it easier to manage and organize these files. The organization of the files in a LabVIEW project is independent of the physical location of the files on the computer.

It is actually not necessary to organize files into LabVIEW projects, although it makes the file management easier, but in one case it is required to use LabVIEW project, namely if

you are going to create a so-called *Build*, that is, if you are to create an executable, an installer, a DLL, a source distribution, or a zip file. (I use LabVIEW projects while developing SimView.)

To create a LabVIEW project, select **File / New Project** in any LabVIEW window. To include a file (e.g. a VI) or a folder in a project, select **My Computer** in the Project tree, and select the menu **Project / Add File** (or Add Folder). The figure below shows one example of a LabVIEW project.



LabVIEW project of name project1

To open a file that is in a project, just double-click the file in the project tree.

[Table of contents]

Finn's LabVIEW Page