An Introduction to LabVIEW for 4th year projects

Stephan Eisenhardt, University of Edinburgh

S.Eisenhardt@ed.ac.uk

LabVIEW 2009

09.2010

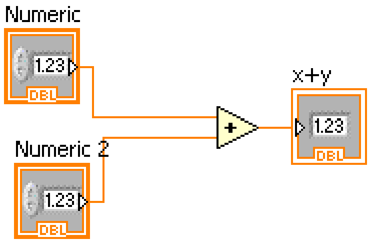
Objectives

- To acquire familiarity with the LabVIEW Programming language
- To be able to write LabVIEW programmes incorporating pre-written and new code
- To be able to understand, adopt and modify third-party code
- To learn how to investigate about unknown functionality
- To acquire the skills needed to complete the 4th year project

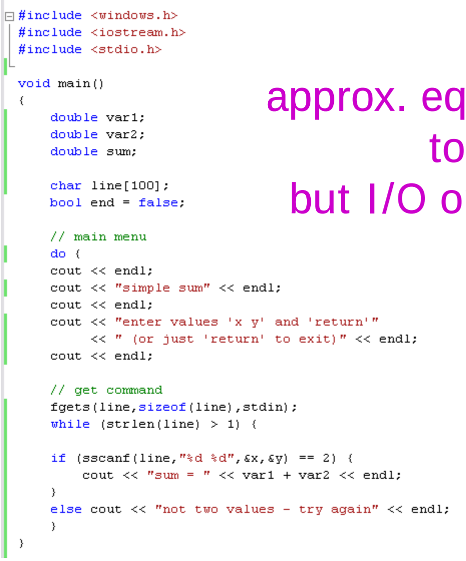
Stage 0 : Concepts & Principles

The LabVIEW Concept I

- LabVIEW (in short ‘LV’) is a graphical programming language developed and marketed by National Instruments:
  - The look and feel is very different from textual programming languages



complete LabVIEW programme
to calculate a sum,
it includes the graphical I/O



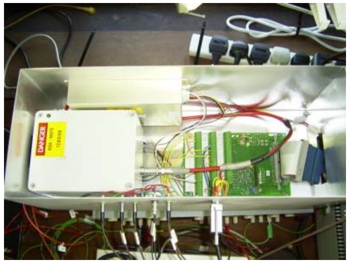
```
#include <windows.h>
#include <iostream.h>
#include <stdio.h>

void main()
{
    double var1;
    double var2;
    double sum;

    char line[100];
    bool end = false;

    // main menu
    do {
    cout << endl;
    cout << "simple sum" << endl;
    cout << endl;
    cout << "enter values 'x y' and 'return'"
         << " (or just 'return' to exit)" << endl;
    cout << endl;

    // get command
    fgets(line,sizeof(line),stdin);
    while (strlen(line) > 1) {

    if (sscanf(line,"%d %d",&x,&y) == 2) {
        cout << "sum = " << var1 + var2 << endl;
    }
    else cout << "not two values - try again" << endl;
    }
}
```

approx. equivalent C++ programme
to calculate a sum,
but I/O only works on a text shell

  - All the usual programming concepts are available
  - But what is the benefit of reinventing the wheel, causing overhead, another learning curve and paying substantial licence fees?

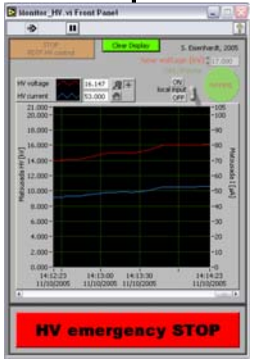
The LabVIEW Concept II

- LabVIEW is designed to build graphical user interfaces (GUIs) for laboratory instrumentation – also called ‘Virtual Instruments’ (VIs):
  - A VIs can read out to a single sensor, emulate the front-face of commercial devices or control large integrated systems
  - The idea is to provide a user-friendly interface, tailored to the needs of the application, to an otherwise possibly obscure piece of hardware



monitoring

control



  - Like with a dashboard of a car, the user interface is what the operator experiences for steering and monitoring of the hardware, while the details of the implementation and the interfaces to the devices are hidden under the bonnet
  - Programming LabVIEW is like building a car, running a VI like driving one

Principle of ‘Data Flow’

- Programme execution follows the principle of ‘data flow’:
  - Each instruction comes with an interface of input and output parameters, called ‘terminals’ (e.g. the ‘+’-operation has ‘x’ and ‘y’ as input and the ‘sum’ as output terminal)
  - An instruction is only executed once *all* its input terminals hold valid data (i.e. the instructions it depends on, all have produced valid data)
  - If there are several instructions which could be executed in parallel, LV has reasonable defaults to chose; but if the order matters, e.g. due to ‘race conditions’, there are ways to control this
  - Each VI has it own set of terminals and can be called as an instruction in another VI (e.g. also function calls provided by LV are VIs themselves)

Implemented Design Principles

- Encapsulation: (main design feature)
  - Each VI has its user-definable interface, and can be operated as ‘black box’ element within other VIs once it is working reliably
- Type checking: (main design feature)
  - Checked at time of coding, VI cannot run if output and input between two nodes do not match in type
- Templates: (now fully integrated since v8.0 and promoted as default)
  - Called ‘Express VIs’: configurable, pre-written code for standard tasks
- Scope of parameters: (supported)
  - VIs run with their own set of local variables, and one can manage different sets of global variables...
- Recursive functions: (supported)
  - VIs can be configured to run as parallel instances and can call themselves
- Inheritance: (not in the way you know it from C++, only while coding)
  - Configurations of existing elements may be ‘inherited’ by new elements you place
- Function overloading/Polymorphism: (implemented for library functions)
  - Provided for many standard functions in the library, but hard to code by one-self

Stage 1 : The Way around LabVIEW

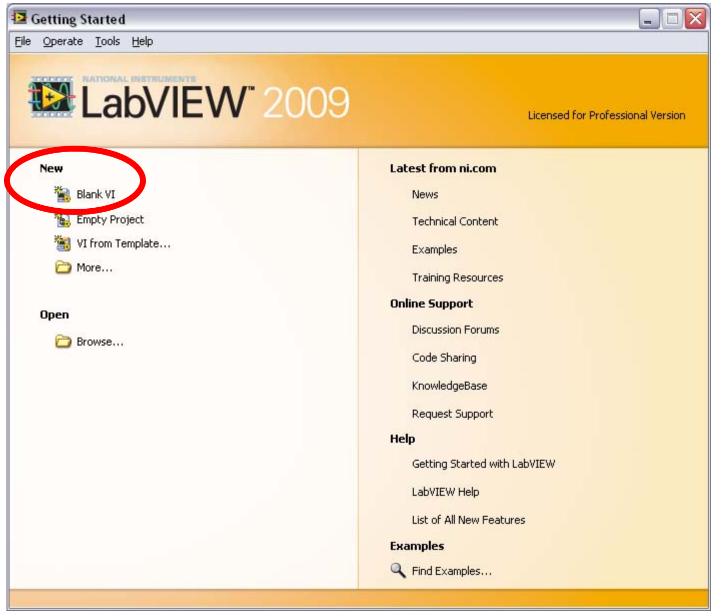
How to use this Guide

- Layout:
  - Proposed instructions are green
  - Items to note are orange

- While reading the following pages:
  - Run LabVIEW in parallel and try all the discussed actions for yourself
  - Feel free to branch out from the guide to browse some of the many other possibilities and return to the text at your own pace
  - At this stage you are deliberately *not* given any example.vi – the first stage is about learning to find your way around and ways to help yourself
  - A second stage will deal with the functionality you will need to employ during your project

Getting Started

- Start LabVIEW: (used here: version 9.0.1)
  - Windows: double-click LV icon or Start menu → All Programs
  - Linux/SunOS: type ‘labview &’
- Startup screen: select ‘New: Blank VI’
  - We start from scratch and keep it simple
  - A simple data member and a sum will give the opportunity to learn much about the way around LV



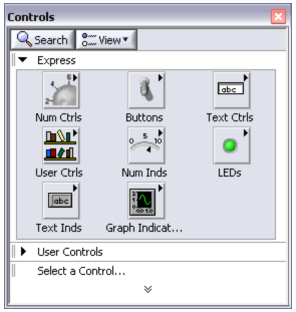
  - That should give you the tools and techniques to better find the needed information later on

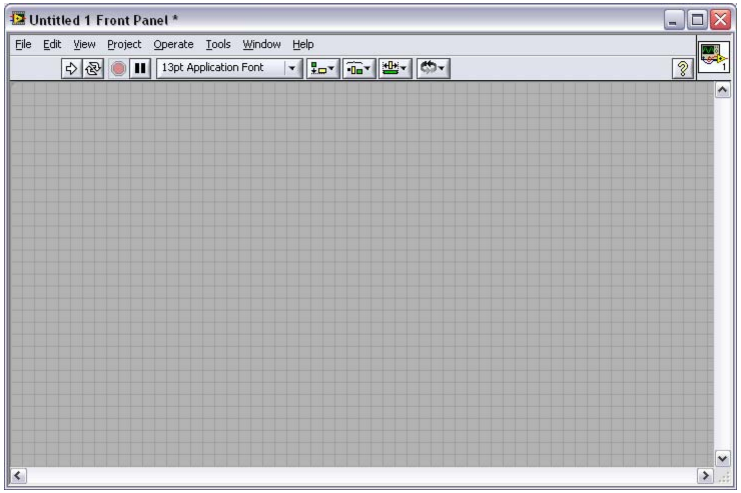
Basics

- A LabVIEW programme comprises of two types of screens:
  - The ‘front panel’ (FP):

    this acts as the user interface, with controls and displays

    associated with it is a ‘Controls’ window (Cnt) to drag & drop GUI interface elements

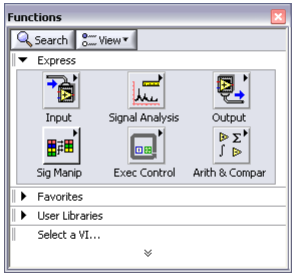


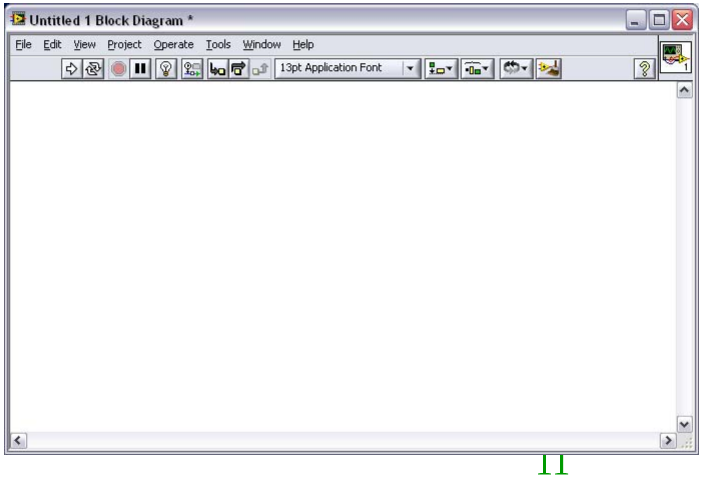


  - The ‘Block Diagram’ (BD): (invoke with Ctrl-e if needed)

    this contains the computational code which handles the data

    associated with it is a ‘Functions’ window (Ftn) to drag & drop code elements





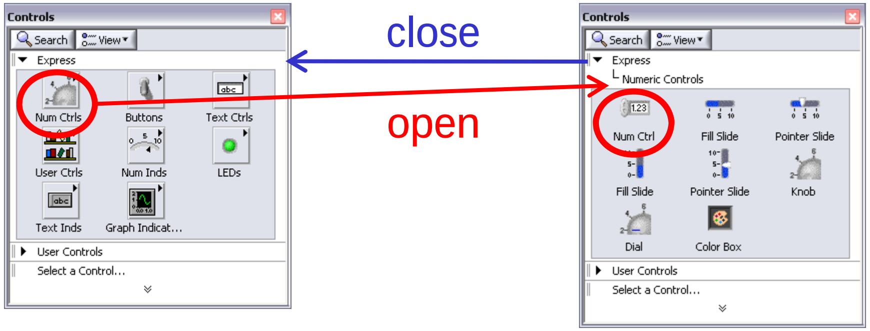
  - it is stored in a file <name>.vi

Mouse Tips

- Basic manipulation techniques:
  - hovering over items selects the focus
  - left-click selects an item
  - double left-click selects to edit an item
  - right-click opens a menu for configuration and digging further

Learn from a Simple Example

- A simple example will show the basic functionality: a data member
  - In the ‘Controls’ window: under the ‘Express’ tab open the ‘Numerical Controls’ sub-tab and select the plain ‘Numeric Control’ GUI



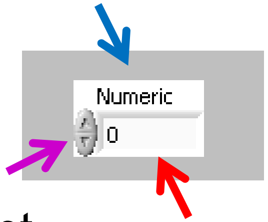
close

open

  - drag & drop it to the FP
  - notice what happened:

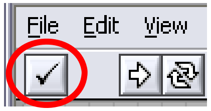
1. on the FP you got an interface

   with a data display, increment handle and name label

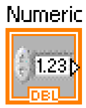


   → play around with it: edit the value, the label, use the increment

   → notice the tick button whenever a value is still transient



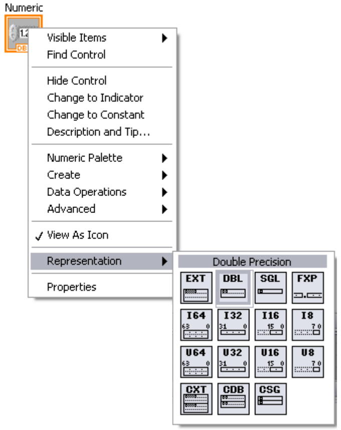
2. on the BD you got an input ‘terminal’ with the same label



   → it represents the memory allocation and displays the data type

Element Manipulation I

- Lets dig a bit deeper into what can be done to the data members:
  - right-click the input terminal and select ‘Representation’:



you see the numerical data types LV can handle by rows, in the different precisions:

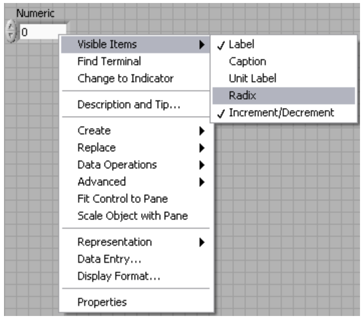
- floating point
- signed integer
- unsigned integer
- complex

- by default LV chooses ‘double precision’ = 32-bit floating point

→ note how colour and icon of the terminal changes for the various types

→ try it out, e.g. to change to ‘complex extended precision’ and see how the data display on the FP changes

Element Manipulation II



- And on the front panel:
  - right-click the data display and select 'Visible Items':

    you see now also the other visual items available:

    → try out the radix, select the 'U8' data type and value '10':

    → click the radix and change to 'Hex', 'Octal' and 'Binary'

    → note that you have to chose an integer data type for these settings to be available

    → try to activate the 'Unit Label', you cannot...

    → you have to chose a floating data type first to use a 'Unit Label', use e.g. 'V' as label

    → now you cannot change the radix to 'Hex', 'Octal' or 'Binary' anymore, nor can you change to integer data types...

    → you must clear the 'Unit Label' again by editing (double-click), to enable integer types once more
  - that is all due to the data type checking already performed at the coding stage

Element Manipulation III

- Dizzy already? There still is more worth to know at this stage:

  - for the terminal or data display find the 'Data Operations' menu and 'Reinitialize to the Default Value': the data display should show '0' again

    → enter a value<>0 and 'Make Current Value Default'

    ⇒ provided you save now the .vi, next time you call it you will start with the value you have chosen

  - This works fine: but for transparency I strongly recommend to explicitly initialise all terminals to sensible defaults at the start of running, it makes life (debugging) so much easier... – we will come to that...

Element Manipulation IV

- And one more on ‘states’ of elements:
  - for the data display find the ‘Advanced’ menu and use ‘Hide Control’

    ... oops... – you want it back? → terminal: ‘Show Control’, easy
  - another game on the ‘state’ of our element:

    data display → Advanced → Enabled State → select Disabled & Grayed

    ... hmm... you still can do everything you have done so far...

    yes, because you are in ‘editing mode’, ‘disabling’ means to lock the access at run-time. i.e. for the operator
  - Now: all operations on ‘states’ of elements, e.g. data operations, colouring, animation, formatting, visibility of components, etc., are available at run-time as well; so called ‘Property Nodes’ allow for their test and manipulation during program execution... – imagine ...
  - you find a comprehensive collection of states of an element in the right-click menu under ‘Properties’ → go and explore...

Quo Vadis?

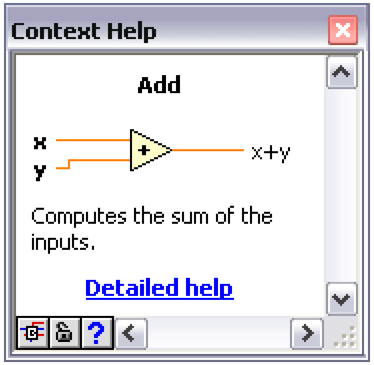
- You are worried because you 'haven't done anything' yet?
  - So wrong! You have achieved so much more than in the usual approach of putting 'some lines of code together and run it'...
  - You have gasped a taste of the scope of LabVIEW!
  - You have gained a set of tools to query and understand the elements of your programme!

  - we continue, and you should expect much more from LabVIEW: as each new principle opens a new dimension in the spectrum of possibilities
  - don't shy away from the mightiness of LV, curiously explore

  - BTW: Vee is the competitor language developed by Agilent, one of the main competitors of National Instruments
    - when you learn LV, you quickly can pick up Vee as well as they are similar
    - but the languages are not compatible...: imagine how the C-compilers of Microsoft and Borland would look like if there wouldn't be the ANSI consortium to standardise the C-language...

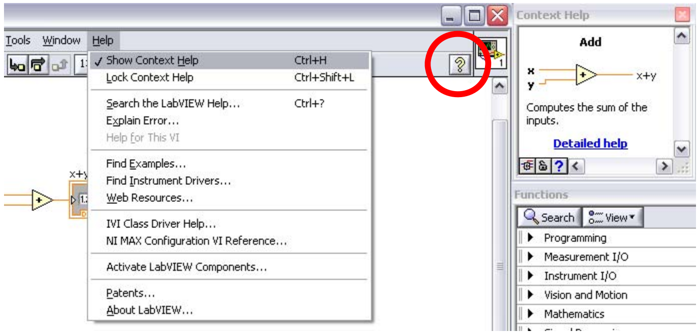
Help Yourself

- Here is what you can do to get more information:
  - the ‘Context Help’ window is extremely useful:
    - it gives you the synopsis of the element you mouse-over
    - describes all I/O ports of an element
    - gives a link to the full reference (for all library functions)

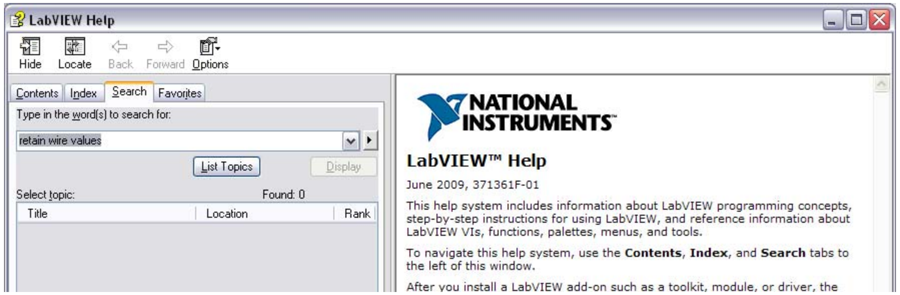


Computes the sum of the inputs.

  - to invoke do either:
    - Help → Show Context Help
    - push the ‘?’ button
    - Ctrl-h



  - for free textual search go to LabVIEW help:
    - Help → Search the LabVIEW Help...
    - Ctrl-?



LabVIEW™ Help

June 2009, 371361F-01

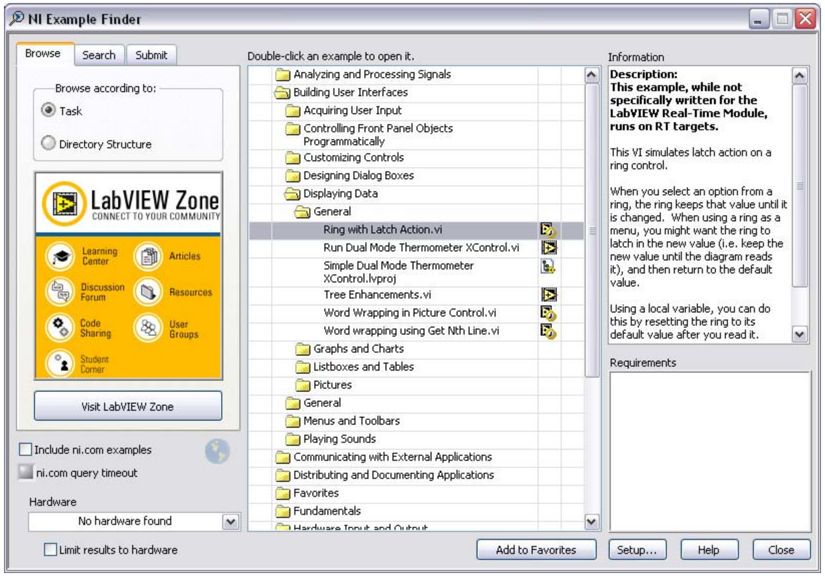
This help system includes information about LabVIEW programming concepts, step-by-step instructions for using LabVIEW, and reference information about LabVIEW VIs, functions, palettes, menus, and tools.

To navigate this help system, use the **Contents**, **Index**, and **Search** tabs to the left of this window.

After you install a LabVIEW add-on such as a toolkit, module, or driver, the

Reading Third-Party Code

- Learning from examples is a great way to progress:
  - menu: Help → Find examples...
    - explore the possibilities to:
      - narrow search to specific hardware
      - maintain your favourites
      - online information in the LabVIEW Zone
  - read existing third-party code
  - some examples are simple, some complex
  - to make the most of it: use the following strategies:
    - follow the flow of data
      understand the I/O of elements and the used functionality, get ideas
    - follow the variables
      use right-click → Find xxx tools to see where data is used/re-used in the code
    - use the debugging tools
      to study the data flow and sequence of events



Browse according to:
● Task
○ Directory Structure

□ Include ni.com examples
ni.com query timeout

Hardware
No hardware found

□ Limit results to hardware

Double-click an example to open it.

- Analyzing and Processing Signals
- Building User Interfaces
  - Acquiring User Input
  - Controlling Front Panel Objects Programmatically
  - Customizing Controls
  - Designing Dialog Boxes
  - Displaying Data
    - General
      - Ring with Latch Action.vi
      - Run Dual Mode Thermometer XControl.vi
      - Simple Dual Mode Thermometer XControl.lvproj
      - Tree Enhancements.vi
      - Word Wrapping in Picture Control.vi
      - Word wrapping using Get Nth Line.vi
    - Graphs and Charts
    - Listboxes and Tables
    - Pictures
  - General
  - Menus and Toolbars
  - Playing Sounds
- Communicating with External Applications
- Distributing and Documenting Applications
- Favorites
- Fundamentals
- Hardware Input and Output

Information

Description:
This example, while not specifically written for the LabVIEW Real-Time Module, runs on RT targets.

This VI simulates latch action on a ring control.

When you select an option from a ring, the ring keeps that value until it is changed. When using a ring as a menu, you might want the ring to latch in the new value (i.e. keep the new value until the diagram reads it), and then return to the default value.

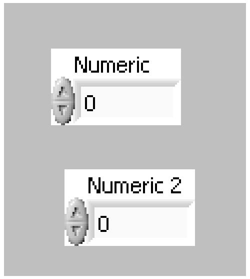
Using a local variable, you can do this by resetting the ring to its default value after you read it.

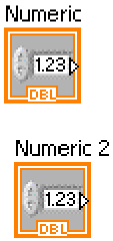
Requirements

Ways to Build: the second element

- Now relax as this is easy – but there are several ways to Rome:
  - revert back to the default settings of the first Numeric Control and add a second Numeric Control
  - which way have you done it?
    - via the Controls window again? OK
    - you also could have done via Copy & Paste (Ctrl-c & Ctrl-v) on either

      the FB or the BD → try both, and delete (Ctrl-x) or undo (Ctrl-z)





the advantage and risk with Copy&Paste is that you also copy all configurations you have done to the source (did you really reset everything?)

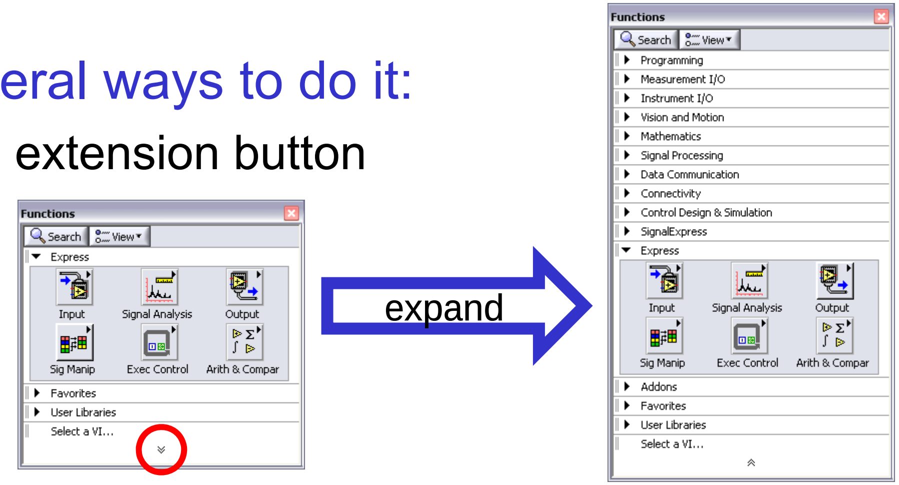
  - note the different behaviour if you perform the actions on the FP or BD:
    - if you place the new element on the FP: the new element on the BD will be placed in (approximately) the same relative position to the first element
    - if you place the new element on the BD: the new element on the FP will be placed as near as possible (without overlap) to the top-left corner of the FP

Ways to build: chose an operation

- As always in LV, again you have several ways to do it:
  - in the Functions window: activate the extension button

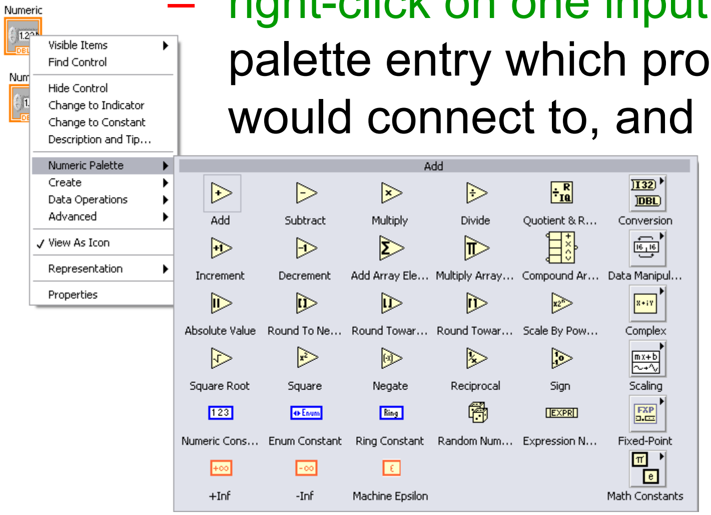
    follow: → Programming → Numeric

    drag the ‘Add’ operation to the BD
  - note the tree structure of available functionality
  - go & find in the ‘Programming’ branch: While Loop, Local Variable, Array Subset, Transpose Matrix, Bundle Cluster, π constant, True constant, type conversion Boolean to (0,1), Carriage Return constant, Max&Min test, Wait Until Next ms Multiple, Three Button Dialog, Scan From File, Property Node, Acquire Semaphore, Play Sound File
  - note that many of these occur redundantly again in other parts of the tree
  - others do not: so go & find: Gauss Peak Fit, Numeric Integration, Sine Wave Generator, Random Noise Generator, Convolution, Normalisation, 1/f Filter, Fast Fourier Transformation, Initialize Mouse



Ways to build: use the context menu

- No, that was only one way so far, here comes the second:
  - right-click on one input terminal: depending on its data type you will get a palette entry which provides the most common elements this terminal would connect to, and links to further related sub-palettes

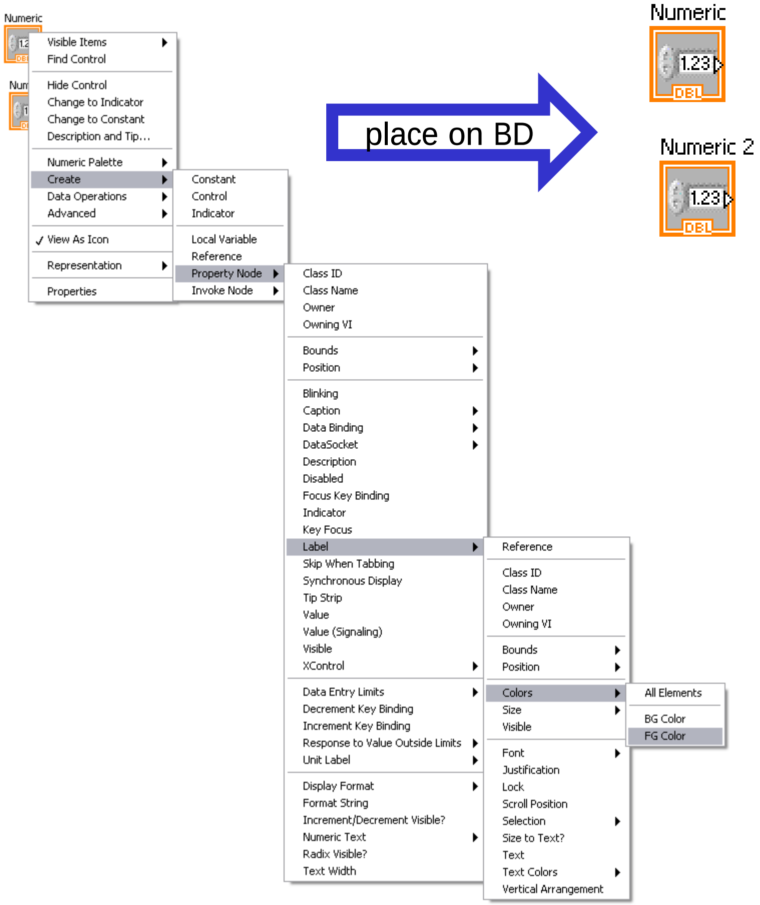


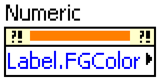
this context-menu provides a tailored selection of options fitting to the element it is used for,
in general it is much faster to use this one –
and it seems pretty comprehensive

  - a look ahead: in the same way under the ‘Create’ entry you find an exhaustive list of related elements to create
  - the clou with this is: the new element is already configured and linked to the ‘parent’ element, it inherits its properties
  - try to create a ‘Property Node’ to programmatically control the foreground colour of the label...

Ways to build: use inheritance

- This way of coding, by using context menus, is powerful and efficient:
  - now verify the inherited properties:



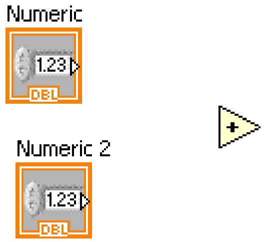


- right-click the coloured part of the property node → Link To → Pane:

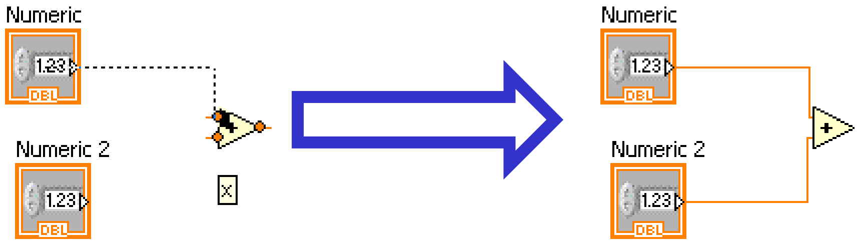
  see to which terminal it is linked
- note that the property Label.FGColor is already configured
- right-click the property itself: note the additional entries in the menu, enabling manipulation, e.g. to add an element and Change to Write
- when you left-click on the property you can chose a different one
- now you can get rid of the Property Node again

Wiring Basics

- Now we connect the elements:
  - remember that ‘Data Flow’ regulates the flow of the programme
  - the data travels between elements through ‘wires’
  - your Block Diagram should look like this again:



  - mouse-over the terminal and ‘+’ operation:
  - note the changes to the mouse and the icons: the mouse looks like a little spindle when you hover over the I/O ports of the elements – which are indicated by orange circles and black background for the active one
  - by pointing & dragging: string a wire from one input terminal to the x-input of the ‘+’ operation
  - and add the wire for the second

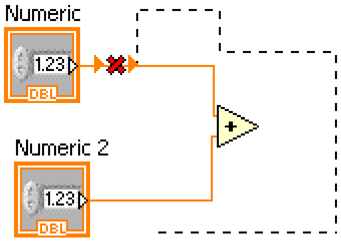


  - note: output goes to the right

    input comes from the left

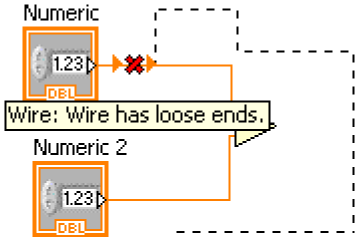
More Wire Skills

- You will use this a lot...:
  - mouse over a wire: actually slightly next to it, so that the mouse pointer is a spindle
  - click and start to draw an extension to the wire
  - click again to create fixed support points while you drag
  - right-click to abandon your wiring
  - try again and now double-click to end the wire mid-air



  - note that the new wire, by dashing, is indicated as non-functional
  - also the original wire bears a cross to say that it doesn't work anymore
  - mouse-over the cross and see what is wrong:

    fair enough, LV doesn't like loose ends

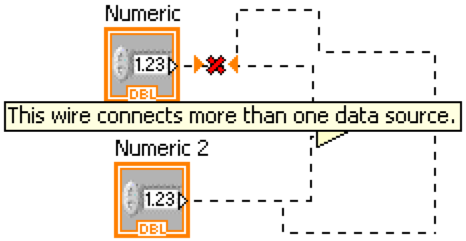


Wire: Wire has loose ends.

  - so pick up the open-ended wire and connect it to the sane one:

    bugger! now everything is messed up...
  - LV comprehensively checks while you code

    and lets you immediately know if something is wrong



This wire connects more than one data source.

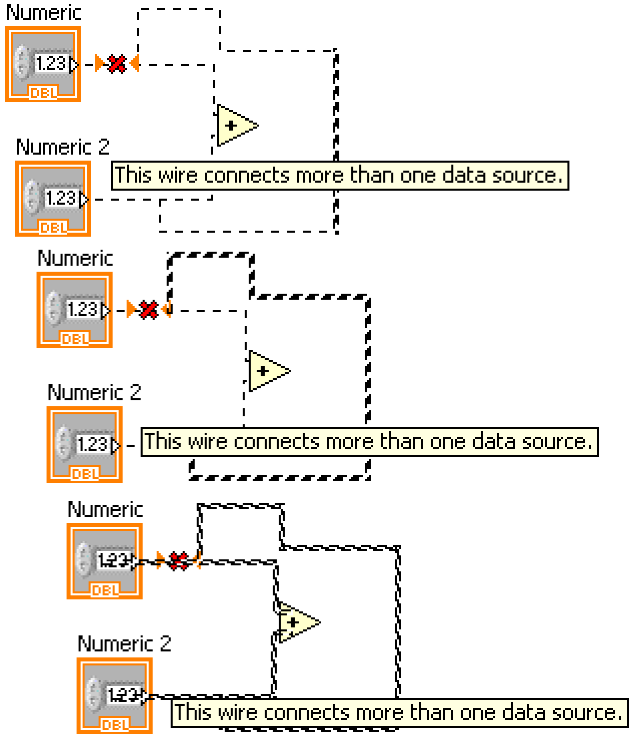
And More Wire Skills

- Of course, we know what is wrong:
  - mouse-over a segment of the offending wire
  - this time precisely, for the pointer being an arrow

    click once to select a segment

    double-click to select up to the next branches

    triple-click to select everything up to the I/O ports
  - get rid of only the offending part, in either way:
    - Ctrl-x
    - right-click: Delete Wire Branch



This wire connects more than one data source.

This wire connects more than one data source.

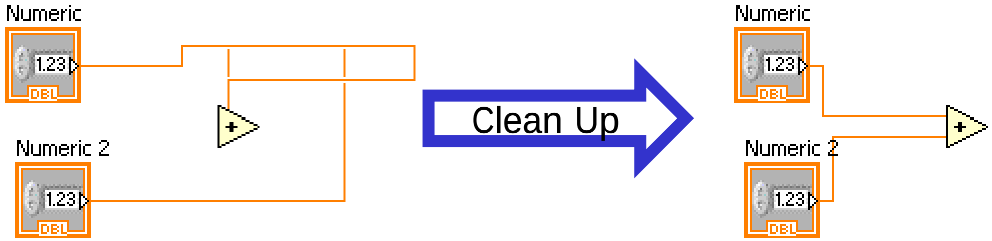
This wire connects more than one data source.

  - now select one of the remaining segments and drag it around
  - try different positions and different segments and observe:
    - how does it look if two wires cross each other? look closely...
    - what happens if you try to produce a loop in one wire?
  - note: the actual connection to the elements is in the centre of the I/O ports, i.e. under the icons

Wiring needs Order

- Is your wiring well messed up – making the code incomprehensible?
  - right-click: Clean Up Wire

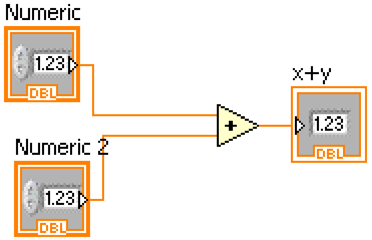
    ...aaah... now I get it

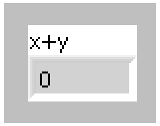


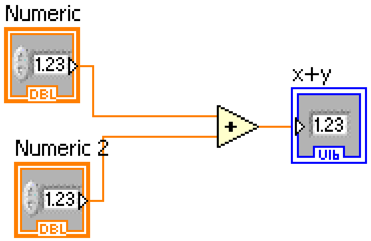
  - only: the LV algorithms to order wires are not perfect/ergonomic, at times annoying, and in a few cases do bad choices (this may or may not have improved now since v7.1), expect that:
    - at wiring the layout may differ depending on which direction you go between the ports of two elements
    - ‘Clean Up Wire’ nearly always will come up with a different layout than the wire placement
    - labels are not cared for: wires run straight through them
    - in some circumstances (usually with bigger structures) the wires may leave a port in the opposite direction than one would expect, and emerge under the other side of the object (especially hard to decipher in third-party code...)
  - plea: use manual re-ordering where needed, improve readability of code

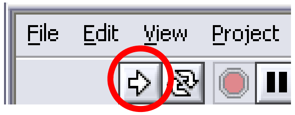
Finalise the Sum: the Indicator

- You want to display the calculated sum. What is the best way - think:
  - I suggest: right-click the output port of the sum and create an Indicator
  - this way a new variable is allocated, already wired and it inherits the necessary properties (e.g. data type, name label of the output port)
  - and where did the data display end up?
  - here is a short-cut: right-click the new element → Find Indicator
  - and the way back: right-click the new element → Find Terminal
  - note the different features of the Indicator:
    - greyed data display, no controls
    - thin-lined terminal, with data entering from the left
  - though you still can edit everything as for the terminal, try:
    - radix, Unit Label, data value, make some data default (even if it doesn't make sense at this stage...)
    - now change the data representation to U16 and enter (x,y)=(2.5, 3.1)
    - run the VI: , see it is rounding, now try (2.5, -3.1), then I16, ...



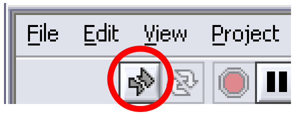




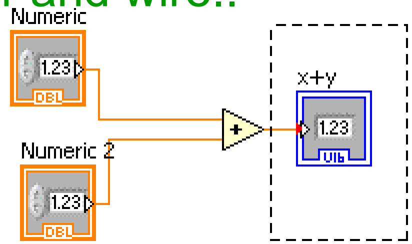


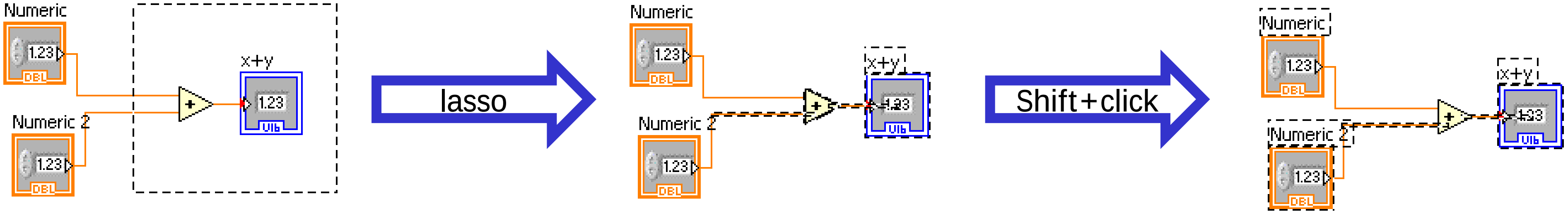
Selecting

- You have messed up the configuration? OK, start again:
  - on the FP click the indicator and remove it (Ctrl-x)
  - notice that the 'Run' button breaks, ngrrr... :

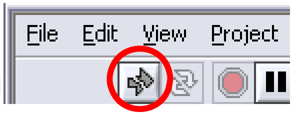


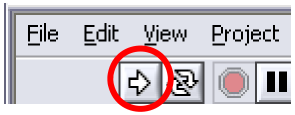
  - go to the BD and look what is wrong: aha, a loose wire was left over
  - right-click the wire stub → Remove Loose Ends: OK
  - could we have done that easier? yes!
  - 2x Ctrl-z to revert to the problem...
  - on the BD now point to the background and lasso the indicator and wire..
  - selecting this way needs a bit of practise, as:
    - all elements (or labels) touched will be selected
    - wire segments only will be selected if >50% covered
  - hold Shift, point & click to select and deselect other items





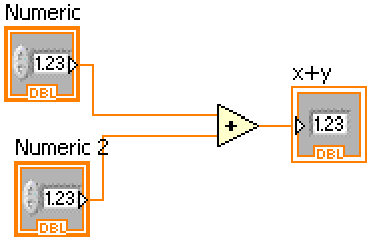
Programme Execution

- Have you noted?
  - while you code the ‘Run’ button changes between 

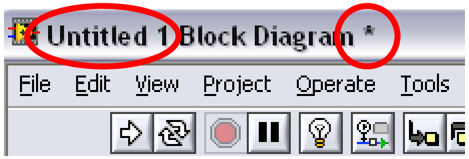
and 

, i.e. enabling and disabling the execution of the VI
  - the LV interpreter is constantly checking whether your VI complies with all rules
  - some basic reasons for broken VIs you know already:
    - at least one input port of one element does not get data
    - unused wire stubs are present, i.e. data is not properly delivered (though it is OK to leave output ports unwired, i.e. unused)
    - mismatch in data type between output and input of two connected elements (though the input port of an element may provide limited data coercing with default casting rules – it is not recommendable to rely on this)
  - a working ‘Run’ button means that your program compiles and executes, this does not state anything about the programme logic, i.e. whether it makes sense... that remains your job...

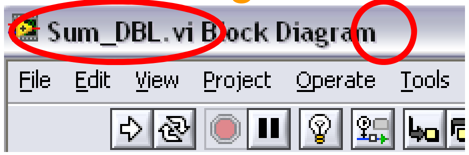
Saving



- Revert back to the good, standard case:
  - it’s now a good time to save your VI,
    you may have to return to it
  - Ctrl-s to select path and name of the VI (or to overwrite if existing)
  - note the change in the window title, from



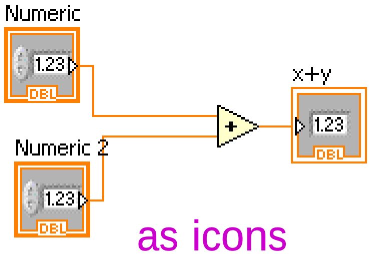
to



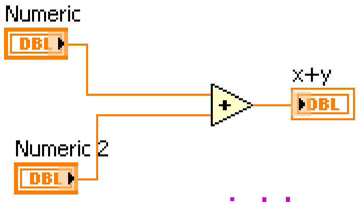
    - the ‘*’ always indicates unsaved changes, even if only of cosmetic nature
    - and you have now named the VI
  - if you exit a VI with unsaved changes, LV will ask you whether they should be saved
    - if there are any sub-VIs with unsaved changes which this VI depends on, LV will also ask you whether you want to have them saved as well
  - use Ctrl-Shift-s if you want to save your current VI and all it depends on
  - use menu File → Save As for more options

Various Manifestations of Elements

- Elements on the BD and FP can have various manifestations:
  - on the BD: terminals and indicators can be shown as
    - icon
    - variable
  - with exactly the same functionality
  - to toggle: right-click → View As Icon

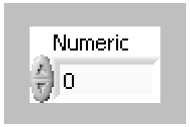


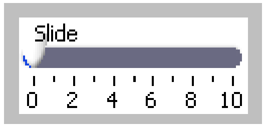
as icons

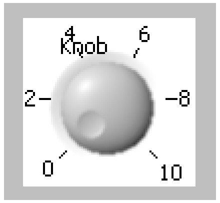


as variables

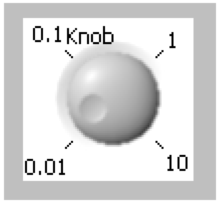
  - on the FP: controls and indicators have a wide range of representations
  - to select style: right-click → Replace → Num Ctrls → <Style>
    - some examples of inputs for double variables:





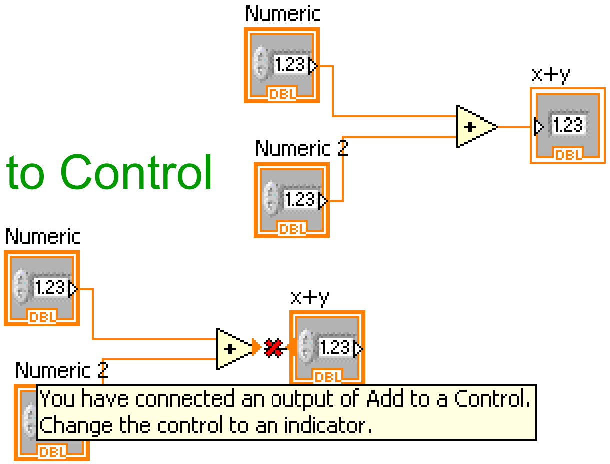


  - functionality can be configured to suit the task:
    - change the input range of the slide: double-click the axis labels to edit
      - what happens if you edit a centre axis label?
    - operate the knob as input for a double, now change the representation to U8
      - how does the operation of the knob change?
    - right-click a slide/knob → Mapping → Logarithmic
      - note: you cannot enter “0” anymore in DBL representation, but in U8

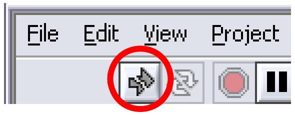


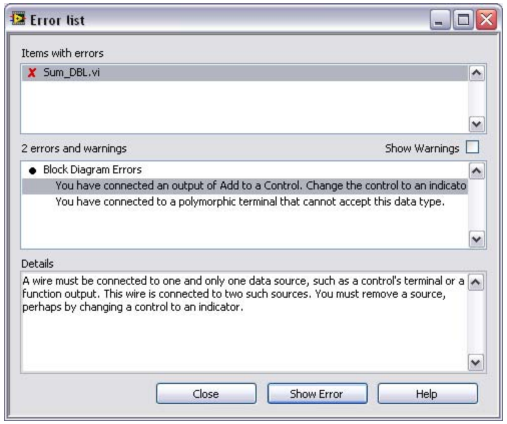
Debugging: Coding Errors

- Now we deliberately break the VI to learn how to investigate problems:
  - convert the indicator into a terminal:
    on the FP or BD right-click the indicator → Change to Control
  - explore the ways to find out about the error:
    - in the BD mouse-over the red cross/broken wire
      to get a quick explanation/suggestion
    - hit the broken run button
      to get an Error List
      with a more verbose explanation for each error/warning
      Show Error will focus back to the location on the BD
      Help will open LV Help with an exhaustive discussion



You have connected an output of Add to a Control.
Change the control to an indicator.





Items with errors

Sum_DBL.vi

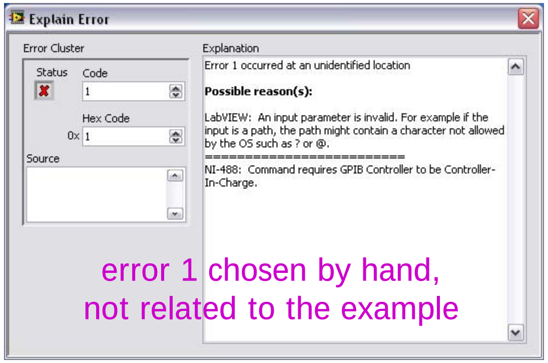
2 errors and warnings

Show Warnings □

- Block Diagram Errors
  You have connected an output of Add to a Control. Change the control to an indicato
  You have connected to a polymorphic terminal that cannot accept this data type.

Details

A wire must be connected to one and only one data source, such as a control's terminal or a function output. This wire is connected to two such sources. You must remove a source, perhaps by changing a control to an indicator.



Error Cluster

Status

Code: 1

Hex Code: 0x 1

Source

Explanation

Error 1 occurred at an unidentified location

Possible reason(s):

LabVIEW: An input parameter is invalid. For example if the input is a path, the path might contain a character not allowed by the OS such as ? or @.

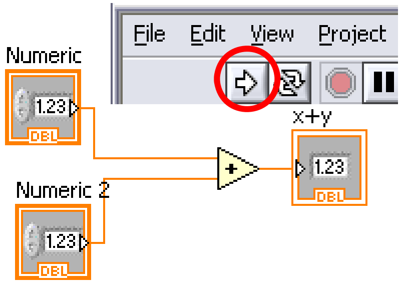
```
==============================
```

NI-488: Command requires GPIB Controller to be Controller-In-Charge.

error 1 chosen by hand,
not related to the example

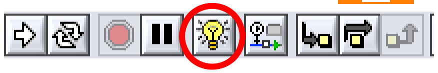
- menu Help → Explain Error...
  still after resolving problems like the above there may be run-time errors, they usually come with error codes, here you can get them better explained

Debugging: Programming Logic



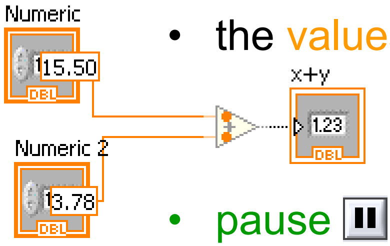
- When LV finds all rules obeyed the run button is unbroken, but this does not check for flaws in the programming logic:
  - these you can sift for in the debugging running mode:

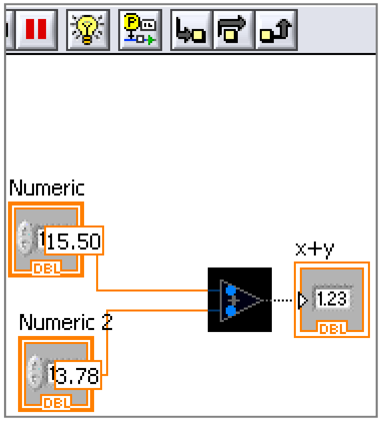
    toggle the running mode with the lamp button



    single ⇨ or continuous ⟳ operation then will execute in slow-motion

    - you can enter arbitrary values at the input terminals before execution
    - on the BD you can watch the data packets moving between the elements
    - the values of the data packets are shown at the I/O ports

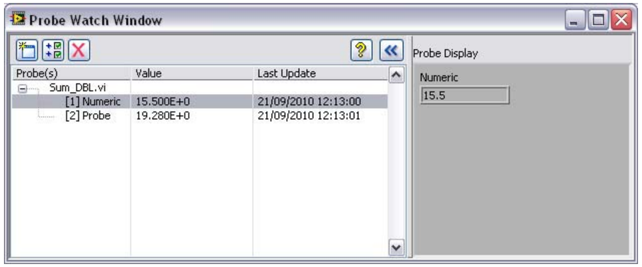




    - pause ⏸ or stop ● operation at any point
    - when paused, the element just being executed will blink
    - mouse-over the wires and I/O ports to inspect the values
    - step-buttons allow to execute the code element by element:
      - branch into sub-VI and stop at its first element
      - step over this element
      - finish this VI and stop after its call in the parent VI

Debugging: using Probes

- Probes are useful to watch values from various locations:
  - probes are updated as the VI executes
  - right-click a wire → Probe
  - this will add the probe to the Probe Watch Window
    - double-clicking a probe will focus back to the BD

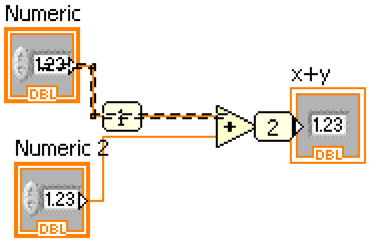


Probe Watch Window

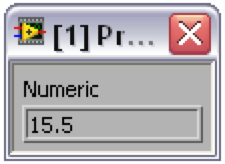
Probe Display

Probe(s)	Value	Last Update
Sum_DBL.vi		
[1] Numeric	15.500E+0	21/09/2010 12:13:00
[2] Probe	19.280E+0	21/09/2010 12:13:01

Numeric: 15.5

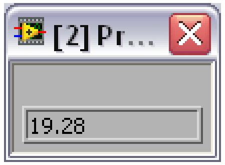


    - right-click a probe to access further options, e.g. copy data
    - buttons provide further options to manage your probes:
      - opens a new window for the selected probe
        use it next to the BD or FP during execution
      - selects all probes in the Probe Watch Window
        use Shift-click and Ctrl-click to alter the selection
      - remove a probe from the list
      - collapse/expand the probe display on the right



[1] Pr...

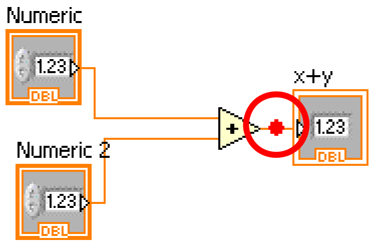
Numeric: 15.5

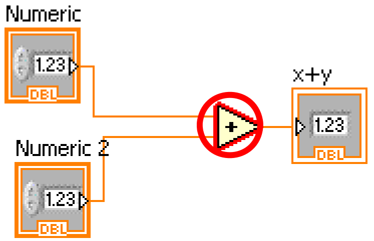


[2] Pr...

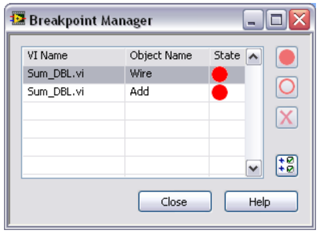
19.28

Debugging: using Breakpoints





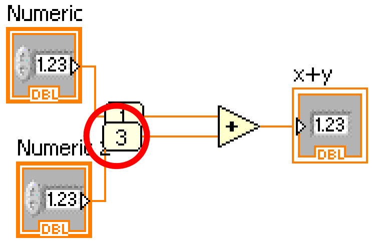
- Program execution will pause at breakpoints:
  - right-click a wire or element → Breakpoint → Set Breakpoint
    the execution will pause when a data packet passes the point
    and you can take it step-by-step from there
  - to manage the points:
    - right-click a breakpoint → Breakpoint → Clear Breakpoint
      to remove the point
    - right-click a breakpoint → Breakpoint → Disable Breakpoint
      to keep but ignore the point
  - multiple points can be easily handled with the Breakpoint Manager:
    - right-click a breakpoint → Breakpoint → Breakpoint Manager
    - similar to the Probe Watch Window us can use:
      - ● to enable a selected point
      - ○ to disable a selected point
      - ✗ to remove a selected point
      - ☑ selects all points
        use Shift-click and Ctrl-click to alter the selection

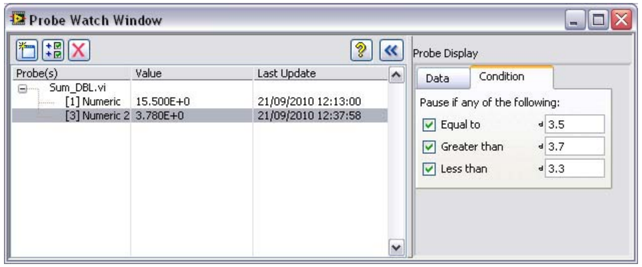


VI Name	Object Name	State
Sum_DBL.vi	Wire	●
Sum_DBL.vi	Add	●

Debugging: using Conditional Probes

- An even more powerful tool is the Conditional Probe:
  - right-click a wire → Custom Probe → Conditional xxx Probe
  - depending on the data type you can set conditions on which this probe acts as a breakpoint





Probe(s)	Value	Last Update
Sum_DBL.vi		
[1] Numeric	15.500E+0	21/09/2010 12:13:00
[3] Numeric 2	3.780E+0	21/09/2010 12:37:58

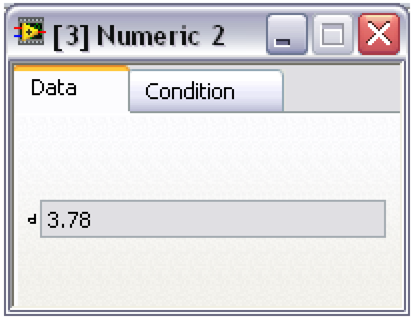
Pause if any of the following:

☑ Equal to 3.5

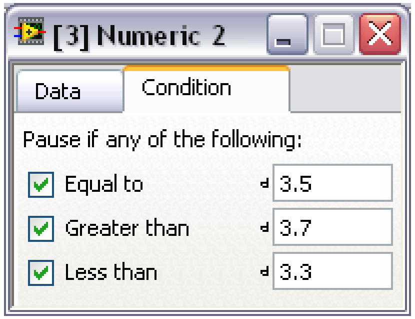
☑ Greater than 3.7

☑ Less than 3.3

  - you can switch between the data view and the conditions view in the Probe Watch Window as well as in the new window to permanently watch the probe:



3.78



Pause if any of the following:

☑ Equal to 3.5

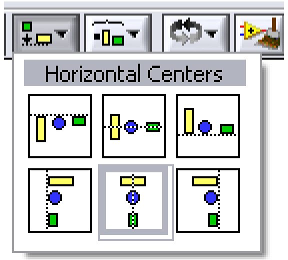
☑ Greater than 3.7

☑ Less than 3.3

  - otherwise it behaves like ordinary probes

Keeping Order

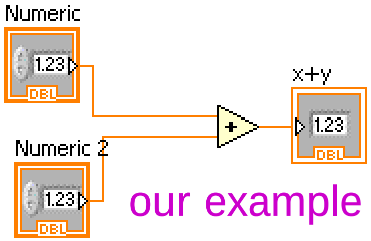
- If VIs become more complex keeping order is essential to maintain readability (for others and yourself...):
  - even though LV provides some support for that, coding discipline is essential as the algorithms will leave you wanting
  - lasso/select some elements on the DB/FP and play with the tools to see:



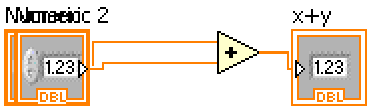
- Align Objects: align object boundaries or centres of selected elements
  - object boundaries are always the square around the full visible items, i.e. depending on the manifestation and including labels, but only if they are shown...
  - when aligning to top/bottom/left/right always the xxx-most will be the reference
  - when aligning to centres alignment will be done to the centre of the selection
- Distribute Objects: arrange objects apart
  - object boundaries as for alignment
  - distribution takes place within boundaries of selected objects
  - compression makes objects touch
- Clean Up Diagram: fully automatic rearrangement
- Reorder:
  - group or ungroup selections
  - select stacking order if elements overlap (I advise strongly to avoid overlaps!)

Keeping Order: Examples

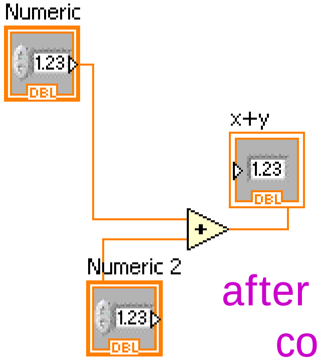
- Here is some taste of what the automatic tools can do:



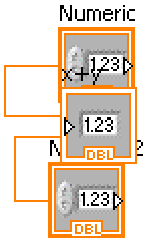
our example



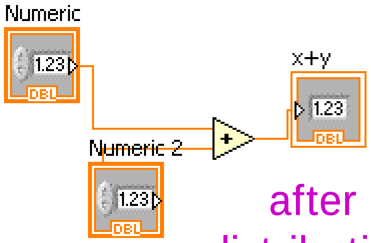
after vertical alignment of centres



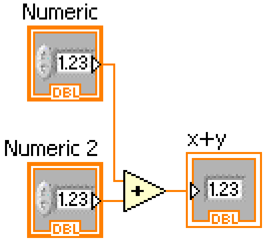
after hor.&vert. compress



after horizontal alignment of centres



after hor.&vert. distribution of centres



after auto-align
note the change
in the label position

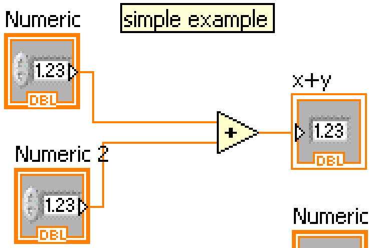
– so: use the tools with care and don’t shy away from manual improvement
– and there is always Ctrl-z ...

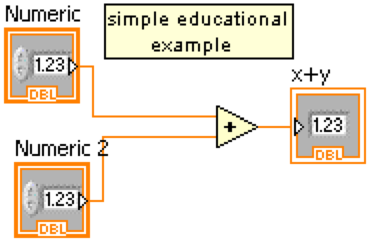
Extra Labels & Decorations

- Be kind to others and yourself – add explanatory labels:
  - you will value them when you revisit your code after some time
  - double-click on the background of the BD/FP to start editing a label
  - double-click the label to re-edit
  - select to move or change the font

13pt Application Font

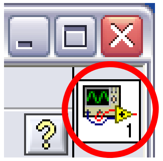
  - right-click the label for further options





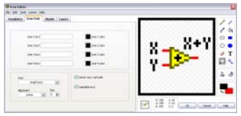
  - decorations on the FP or BD like frames, bars or arrows may significantly improve the overview:
    - FP: Controls Palette → System → Decorations
    - BD: Functions Palette → Programming → Structures → Decorations
  - colouring them is possible via Property Nodes and Colour Boxes (but that is too complex to be convenient for a 'quick tint' of some elements)

The Interface

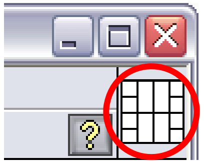


- Encapsulation is mediated via the VI interface:
  - lets customise the interface of our example (currently default):

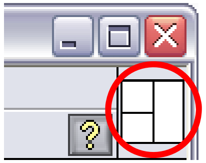
    - double-click icon to open the icon editor:
      - this 32x32 pixels image is helpful if you design modules which you call later
      - you can play with it if you have time, e.g. you can alter it to:



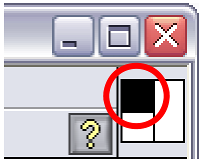
  - more importantly, manage the hidden connector layer:



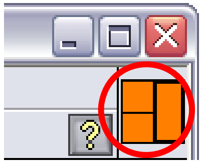
    - on the FP right-click the icon → Show Connector
      - in the default layout you see 6 input ports on the left and 6 output ports on the right
    - right-click connector → Pattern, to select the layout suitable for this VI
      - or right-click connector → Add/Remove Terminal, to adjust left or right side



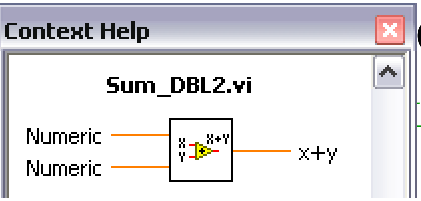
    - click a terminal in the connector to select – note the wire tool appearing – and next select an input or output element on the FP to make a connection



      - break such a connection by right-click connector → Disconnect This Terminal
      - note the colour of connected terminals representing the data type
      - note that (input: left, output: right) is only a convention, you can cross-connect
      - configure the connection with right-click connector → This Connection Is → XXX



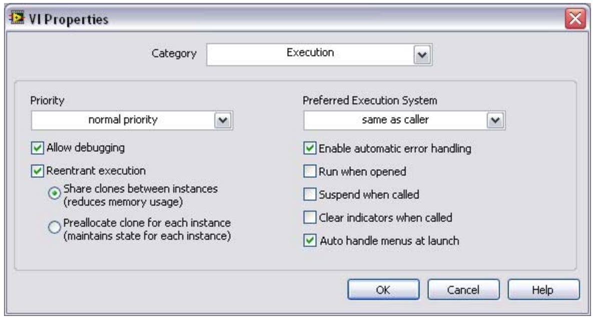
    - mouse over the connector or icon: note the automatic documentation in the Context Help – now your VI is a new building block, ready to be called



Configuring VI Behaviour

- To individually configure the behaviour of your VI during edit and running:
  - Ctrl-I or menu → File → VI Properties, explore categories, e.g.:
    - General: Current revision counter increments every time you save the VI
    - Editor Options: you can alter the grid sizes for FP and BD
      - the defaults are sensible...
    - Window Appearance: customise features available at run-time
      - look up the options in Custom → Customize
    - Execution: Allow debugging: switch off if performance matters
    - Execution: Reentrant execution:
      - several instances of VI can run in parallel(second can start before first has finished)
      - use ‘Preallocate clone for each instance’ if they run independently
      - use ‘Share clones between instances’ for recursive operation

        configure the example this way for later use in a recursive setting (OK), and save the VI



Category: Execution

Priority: normal priority

Preferred Execution System: same as caller

☑ Allow debugging

☑ Reentrant execution

● Share clones between instances (reduces memory usage)

○ Preallocate clone for each instance (maintains state for each instance)

☑ Enable automatic error handling

□ Run when opened

□ Suspend when called

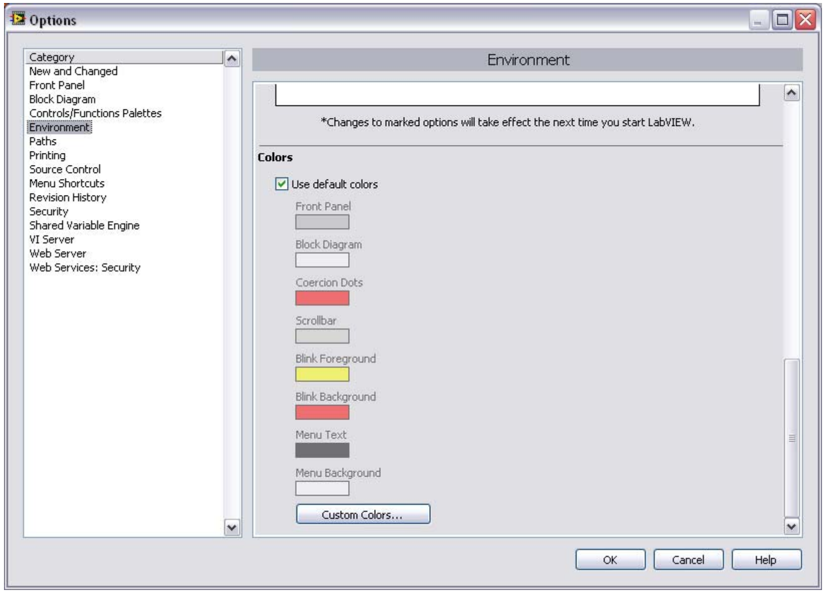
□ Clear indicators when called

☑ Auto handle menus at launch

- Execution: Run when opened: e.g. when the user shall not edit...

Configuring LabVIEW Behaviour

- To configure the general behaviour of LV, across the VIs:
  - menu: Tools → Options, explore categories, e.g.:
    - Front Panel: Front Panel Grid: Enable Grid Alignment
      - toggle the snapping to the grid if objects are moved
    - Block Diagram: Block Panel Grid:
      - Enable Grid Alignment
      - useful for structured coding
    - Environment: Maximum undo steps per VI:
      - set to 99 (max value, default = 30)
    - Environment: note the default colour scheme



*Changes to marked options will take effect the next time you start LabVIEW.

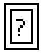
Colors

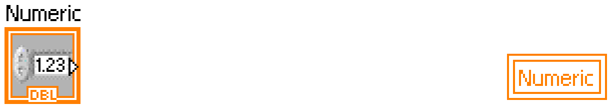
☑ Use default colors

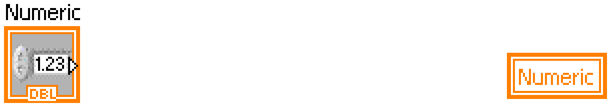
- Menu Shortcuts: manage you fancy key sequences
- VI Server: to configure conditions under which a remote application can call a local VI

Local Variables

- Traditionally LV knows Local and Global variables:
  - the scope of a Local Variable is the VI it resides in:
    - it can be bound ('linked') to any I/O terminal of the VI
    - writing/reading to the Local Variable is as if writing/reading the terminal itself
    - this way one also can write to a control or read from an indicator
  - just to play with it:
    - to add one, right-click BD → Programming → Structures → Local Variable
    - to link it: right-click → Select Item → xxx
    - change I/O direction: right-click → Change to Read
    - and for now get rid of it again...





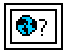


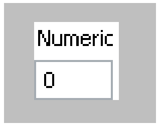
Global Variables

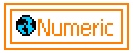
- Global variables can be powerful and dangerous:
  - the scope of a Global Variable is, surprise, global:
    - it resides in a special VI, that only consists of a FP to contain global variables
    - any I/O terminal in any VI can be linked to it
    - several distinct VIs with global variables can be maintained
    - matching names of two global variables from different VIs are possible...

      ... and bear a high risk of confusion (hard to debug!!)
    - they are useful for general configurations
    - they may be used for communication between independently running VIs
  - just to play with it:
    - to add one, right-click BD → Programming → Structures → Global Variable
    - to create it: double-click → “Global X Front Panel” opens → place a Numeric
    - to link it: right-click → Select Item → xxx
    - note: only the 🌐 symbol distinguishes the global variable now from the terminal

      advice: use clear, distinctive naming conventions!
    - to change the Global VI to select from: right-click → Replace → All Palettes → Select a VI







Shared Variables

- Shared Variables are a relatively new concept in LV:
  - they are designed for fail-safe inter-VI communications:
    - e.g. they can be buffered and protected against multiple write accesses
    - but they are only available on Windows and Real-Time platforms
    - inter-communications are supported for:
      - different parts of a BD where wiring is difficult or impossible
      - between independently running VIs on one host
      - between independently running VIs on different hosts on the same network

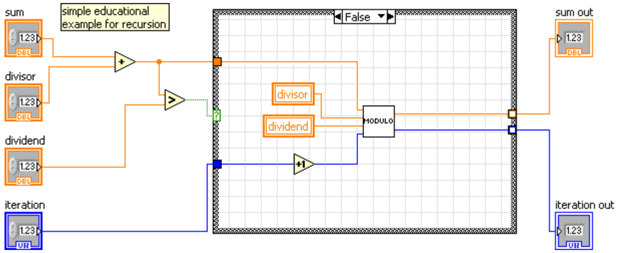
        e.g. between a master VI on a control PC and an autonomously running VI on a remote Real-Time data acquisition system
  - to get a flavour of Shared Variables you may feel adventurous and try:
    - embed the VIs into a project (menu: Project → New → Add VIs)
    - in the Project Explorer window: select an item and use New to create a shared variable, then configure to need (to complex to discuss here...)

    - ... we don't need all that now: to get rid of all quit LV without saving...

A Recursive Example I

- We start from our simple example again and build a recursive VI:
  - we need to rebuild:
    - open your VI and save it e.g. as Modulo.vi, update the label text
    - rename the controls to ‘sum’ and ‘divisor’
    - remove the ‘x+y’ indicator
    - add: DBL control ‘dividend’, U32 control ‘iteration’
    - add: DBL indicator ‘sum out’, U32 indicator ‘iteration out’
    - ensure for all controls/indicators that ‘0’ is the default value
    - edit the VI connector to 4 input and 2 output ports and wire up the terminals
    - edit the VI icon to bear the name ‘modulo’
    - add a ‘x>y’ comparison & link the output of the ‘+’ to x and the ‘dividend’ to y
    - add a Case Structure and link the ‘?’ to the output of the ‘>’ operation
    - on the BD right-click the ‘divisor’ control, create a local variable and place it in the ‘false’ case and change it to Read; repeat for the ‘dividend’
    - add a ‘+1’ increment in the false case
    - right-click BD → Select VI... → place Modulo.vi in false case
    - wire up the VI inputs/outputs: the sum from the ‘+’, the iteration via the ‘+1’

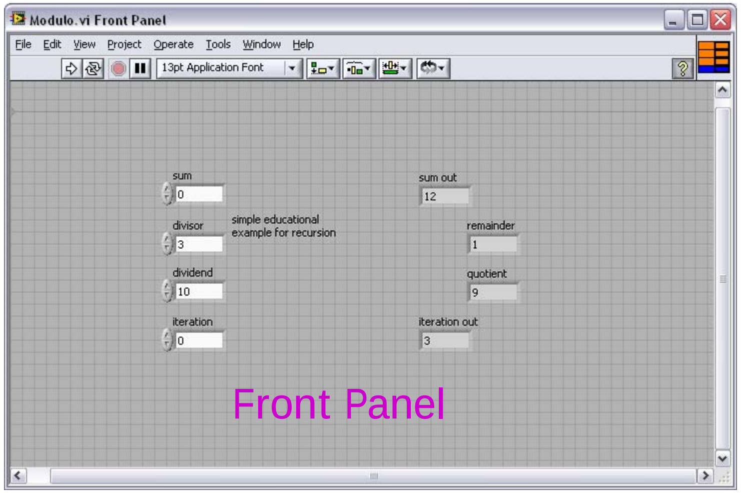
A Recursive Example II



- □ How does your VI look like? Maybe like this:
  - – note the 4 tunnels (= feed-through terminals):
    - to hand over data: filled on the left → OK
    - broken on the right: because there ins no input in the ‘true’ case yet...
  - – finish the VI:
    - in the ‘true’ case: feed through the ‘sum’ and ‘iteration’
    - multiply the ‘iteration’ with the ‘divisor’
    - from the product create an indicator ‘quotient’
    - move the ‘quotient’ outside the case structure, and wire back to the product
    - in the ‘true’ case: subtract the ‘quotient’ from the ‘dividend’
    - from the difference create an indicator ‘remainder’
    - move the ‘remainder’ outside the case structure, and wire back ...
    - upgrade the connector to 4 outputs and connect the ‘quotient’ & ‘remainder’
    - fix the link to Modulo.vi: right-click → Relink To SubVI
    - fix the ‘false’ case by wiring the ‘quotient’ & ‘remainder’
  - – try out the VI with values>0 for dividend & divisor: it should work...

A Recursive Example III

- The basic functionality is OK now:
  - here is how my Modulo.vi looks like:



Modulo.vi Front Panel

sum: 0

divisor: 3

dividend: 10

iteration: 0

simple educational example for recursion

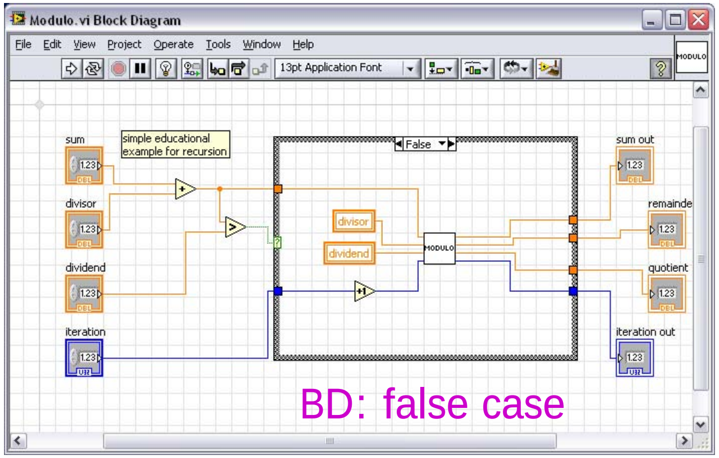
sum out: 12

remainder: 1

quotient: 9

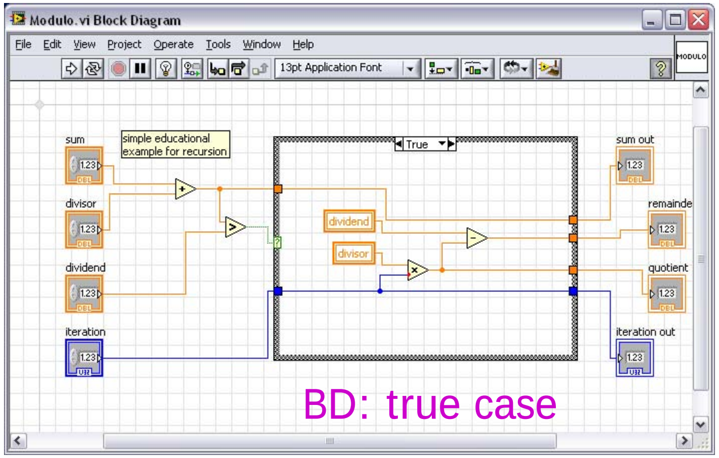
iteration out: 3

Front Panel



Modulo.vi Block Diagram

BD: false case



Modulo.vi Block Diagram

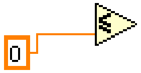
BD: true case

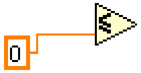
  - but I am not happy:

    there are four major flaws

    1) the FP shows to much information
    2) the default values for dividend and divisor (0) cause an infinite recursion
    3) the VI is not safe against ill chosen input values
    4) the VI immediately starts execution: a user dialog for input is desirable

A Recursive Example IV

- I also get a bit ambitious and want the VI to have a pop-up GUI:
  - solving issue 1) is easy:
    - for the sum/iteration control/indicators → Advanced → Hide control/indicator
  - issues 2) & 3) actually can be solved in one go: with a test against 

  - but the GUI needs infrastructure and it is sensible to write a wrapper VI:
    - create a Modulo_GUI.vi – menu: File → New VI (Ctrl-n)
    - add a ‘Prompt User’ dialog box to the BD:
      - right-click BD: Programming → Dialog & User interface → Prompt User
    - configure the dialog box to ask for dividend & divisor
    - wire collected data to the tests 

    - the output of the tests should ‘enable’ two Display Messages informing about the numerical requirements for the inputs
    - place a While Loop around everything:
      - change the Loop Condition to Continue If True
    - add an OR between the two test results and the Loop Condition
    - right-click loop edge to add two Shift Registers ,wire user values to them
    - viola: you have an input dialog and safe-guard against unsupported inputs

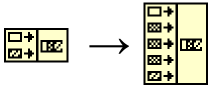
Express VIs

- The User Prompt and Display Message structure you just have used are Express VIs:
  - their main purpose is to make the life easy for the programmer
  - they provide quickly configurable solutions for standard tasks
  - note: corresponding fundamental functions may offer more functionality and control than Express VIs, but are significantly less convenient to use
  - Express VIs are most powerful in the I/O towards:
    - user
    - files
    - (virtual or real) instruments (Signal Express)

A Recursive Example V

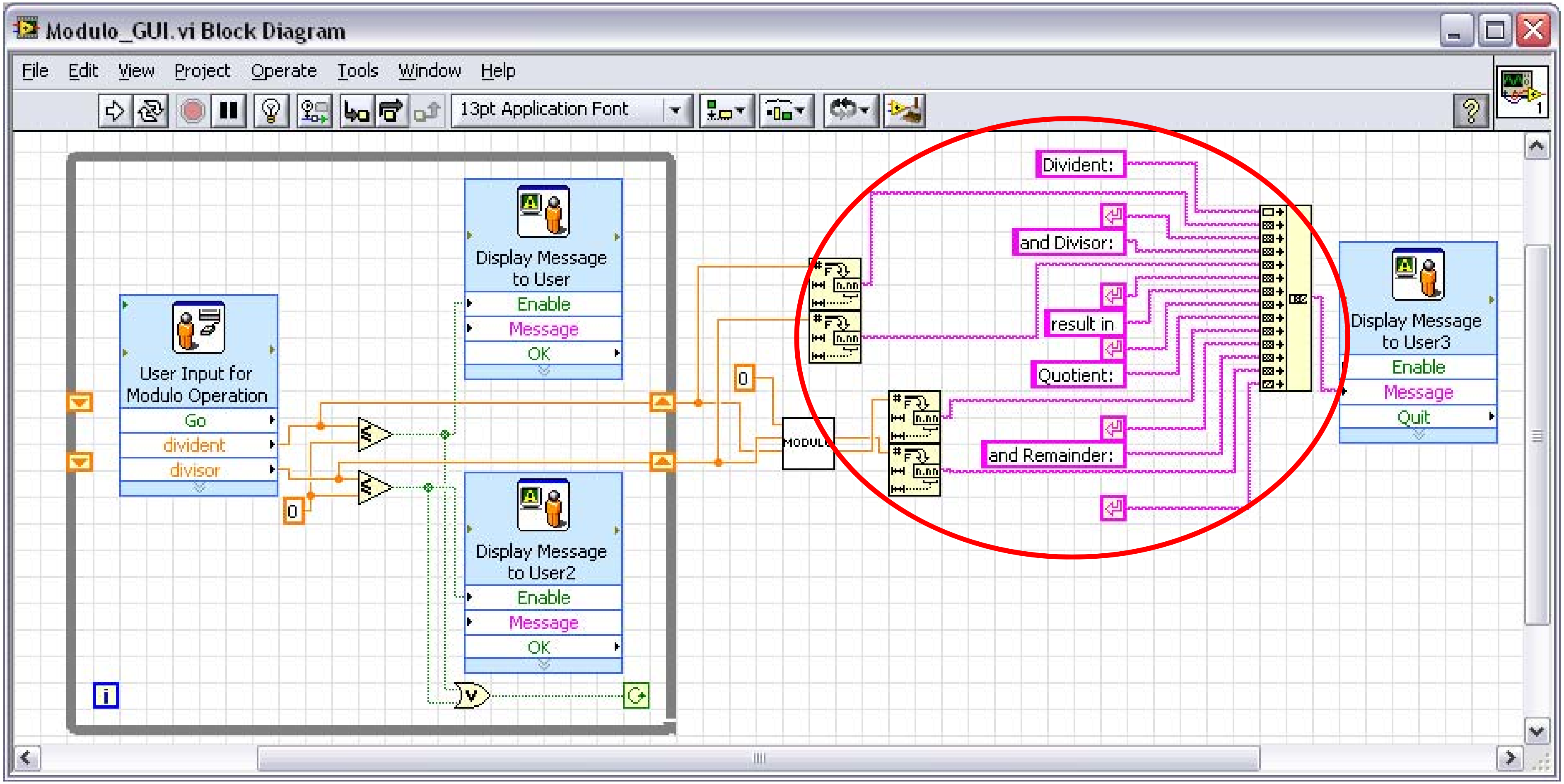
- Continue towards the output GUI:
  - first we do it ‘per pedes’, i.e. with fundamental functions:
    - place the Modulo.vi on the BD
    - wire the user variables and an initial ‘0’ to the ‘sum’ terminal of Modulo.vi
    - convert the 4 DBL variables to strings:

      Programming → Sting → String/Number Conversion → Number To Fractional String
    - add text constants (Divident:)and Carriage Return Constants ↵ and use the expandable Concatenate Strings function to build a suitable output text

      
    - add another Display Massage to present the calculated result to the user
    - for the wiring to look reasonable: use an iterative combination of
      - Clean Up Wire
      - manual corrections

A Recursive Example VI

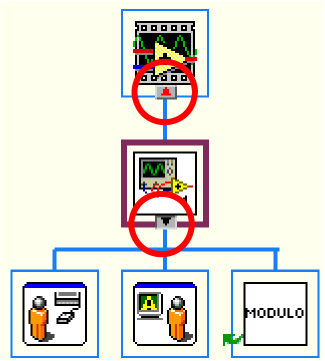
- How does your VI look like? Here is mine...



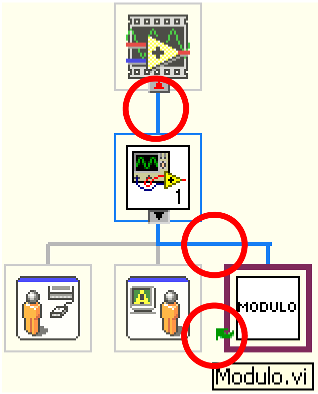
- – now replace the string configuration with a Build Text Express VI...

Hierarchies

- We have a little hierarchy of VIs now: let's have a look at that...
  - Modulo_GUI.vi: menu: View → VI Hierarchy
  - note the difference shown if done in Modulo.vi
  - explore the possibilities, e.g.:
    - switch on/off branches
    - manage the display
    - include/exclude VI Lib / Global Variables



- menu: Tools → Find VIs on Disk...
- right-click VIs:
  - Show/Hide All SubVIs
  - Show All Callers
  - Find All instances
- double-click: to open VIs



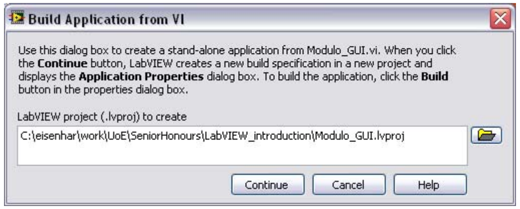
VIs as Executables

- Finally: VIs ready for use can be compiled to stand-alone executables:
  - this has benefits in the performance
  - the code ins protected against (accidental) alteration
  - menu: Tools → Build Application (exe) from VI...
    - from: Modulo_GUI.vi
    - this generates a Project:
      - Modulo.lvproj
      - Modulo.aliases
    - and opens the VI Properties dialog:
      - select target name and location
      - Build
    - bingo:
      - it runs...

        ... without LV open

        ... as a slim

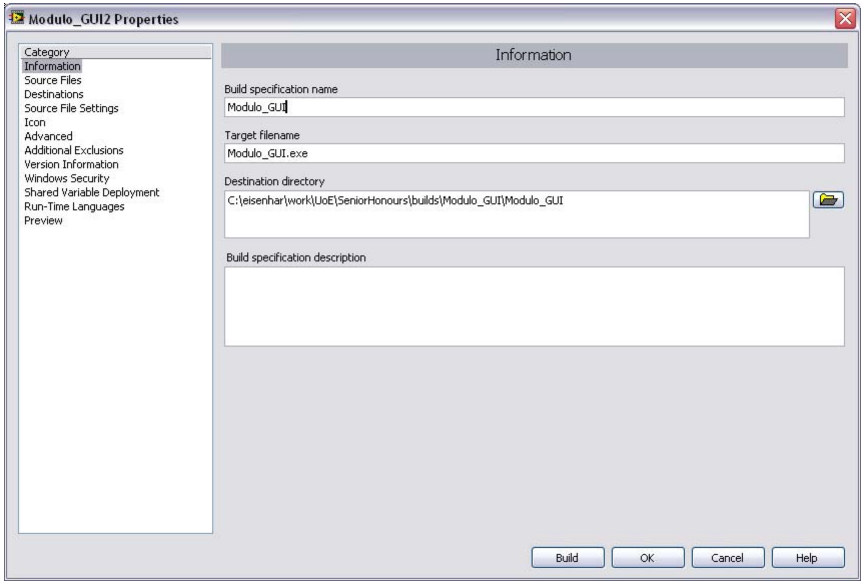
        executable



Use this dialog box to create a stand-alone application from Modulo_GUI.vi. When you click the **Continue** button, LabVIEW creates a new build specification in a new project and displays the **Application Properties** dialog box. To build the application, click the **Build** button in the properties dialog box.

LabVIEW project (.lvproj) to create

C:\eisenhar\work\UoE\SeniorHonours\LabVIEW_introduction\Modulo_GUI.lvproj

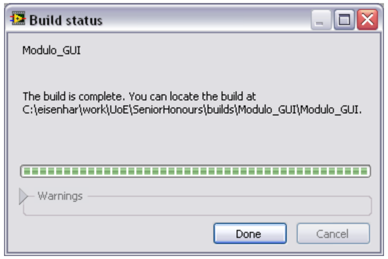


Build specification name: Modulo_GUI

Target filename: Modulo_GUI.exe

Destination directory: C:\eisenhar\work\UoE\SeniorHonours\builds\Modulo_GUI\Modulo_GUI

Build specification description



Modulo_GUI

The build is complete. You can locate the build at C:\eisenhar\work\UoE\SeniorHonours\builds\Modulo_GUI\Modulo_GUI.

No Coding Experience Yet?

- You are still worried that you have no coding experience yet?
  - you have!

  - just the set of functionality is limited yet...
  - but now you know a comprehensive set of tools and practices to find you own way around LabVIEW
  - it is time to make use of them

  - the planes of LabVIEW are vast and fruitful...
    - to sustain yourself you have been given bow, arrows and a knife
    - and you learnt how to use them
    - now go: hunt & explore

Best Coding Practices

- But it's a long way to write good, complex code...:
  - ... the following is not needed for small, quick hacks – but becomes beneficial for bigger projects
  - in LV Help: search for 'best practices', e.g. look at:
    - LabVIEW Style Checklist: revisit as you progress gaining experience
    - VI Memory Usage: dive deep...
      - get aware of optimisations
    - VI Execution Speed:
      - performance-orientated...
  - in LV Help: use Locate Locate ,

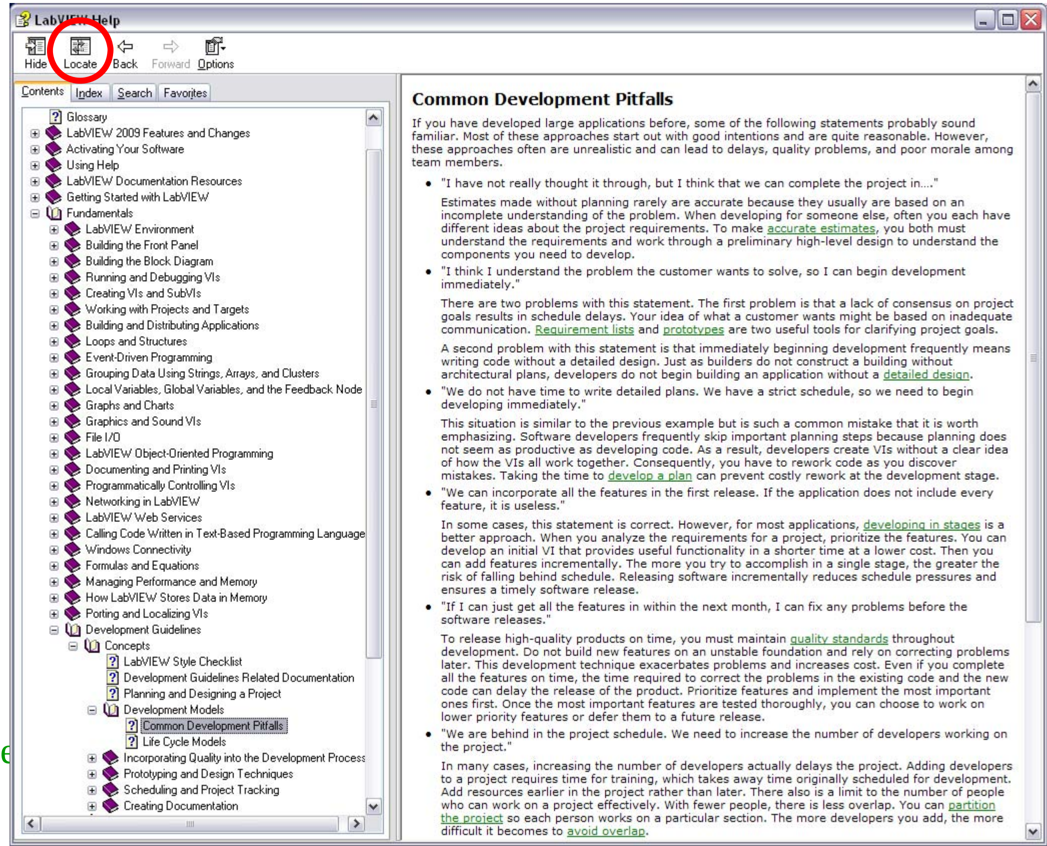
    e.g. look at:

    → Fundamentals

    → Development Guidelines

    → Concepts

    → Common Development Pitfalls


LabVIEW Help
Hide
Locate
Back
Forward
Options
Contents
Index
Search
Favorites
Glossary
LabVIEW 2009 Features and Changes
Activating Your Software
Using Help
LabVIEW Documentation Resources
Getting Started with LabVIEW
Fundamentals
LabVIEW Environment
Building the Front Panel
Building the Block Diagram
Running and Debugging VIs
Creating VIs and SubVIs
Working with Projects and Targets
Building and Distributing Applications
Loops and Structures
Event-Driven Programming
Grouping Data Using Strings, Arrays, and Clusters
Local Variables, Global Variables, and the Feedback Node
Graphs and Charts
Graphics and Sound VIs
File I/O
LabVIEW Object-Oriented Programming
Documenting and Printing VIs
Programmatically Controlling VIs
Networking in LabVIEW
LabVIEW Web Services
Calling Code Written in Text-Based Programming Language
Windows Connectivity
Formulas and Equations
Managing Performance and Memory
How LabVIEW Stores Data in Memory
Porting and Localizing VIs
Development Guidelines
Concepts
LabVIEW Style Checklist
Development Guidelines Related Documentation
Planning and Designing a Project
Development Models
Common Development Pitfalls
Life Cycle Models
Incorporating Quality into the Development Process
Prototyping and Design Techniques
Scheduling and Project Tracking
Creating Documentation

Common Development Pitfalls

If you have developed large applications before, some of the following statements probably sound familiar. Most of these approaches start out with good intentions and are quite reasonable. However, these approaches often are unrealistic and can lead to delays, quality problems, and poor morale among team members.

- "I have not really thought it through, but I think that we can complete the project in...."

  Estimates made without planning rarely are accurate because they usually are based on an incomplete understanding of the problem. When developing for someone else, often you each have different ideas about the project requirements. To make accurate estimates, you both must understand the requirements and work through a preliminary high-level design to understand the components you need to develop.
- "I think I understand the problem the customer wants to solve, so I can begin development immediately."

  There are two problems with this statement. The first problem is that a lack of consensus on project goals results in schedule delays. Your idea of what a customer wants might be based on inadequate communication. Requirement lists and prototypes are two useful tools for clarifying project goals.

  A second problem with this statement is that immediately beginning development frequently means writing code without a detailed design. Just as builders do not construct a building without architectural plans, developers do not begin building an application without a detailed design.
- "We do not have time to write detailed plans. We have a strict schedule, so we need to begin developing immediately."

  This situation is similar to the previous example but is such a common mistake that it is worth emphasizing. Software developers frequently skip important planning steps because planning does not seem as productive as developing code. As a result, developers create VIs without a clear idea of how the VIs all work together. Consequently, you have to rework code as you discover mistakes. Taking the time to develop a plan can prevent costly rework at the development stage.
- "We can incorporate all the features in the first release. If the application does not include every feature, it is useless."

  In some cases, this statement is correct. However, for most applications, developing in stages is a better approach. When you analyze the requirements for a project, prioritize the features. You can develop an initial VI that provides useful functionality in a shorter time at a lower cost. Then you can add features incrementally. The more you try to accomplish in a single stage, the greater the risk of falling behind schedule. Releasing software incrementally reduces schedule pressures and ensures a timely software release.
- "If I can just get all the features in within the next month, I can fix any problems before the software releases."

  To release high-quality products on time, you must maintain quality standards throughout development. Do not build new features on an unstable foundation and rely on correcting problems later. This development technique exacerbates problems and increases cost. Even if you complete all the features on time, the time required to correct the problems in the existing code and the new code can delay the release of the product. Prioritize features and implement the most important ones first. Once the most important features are tested thoroughly, you can choose to work on lower priority features or defer them to a future release.
- "We are behind in the project schedule. We need to increase the number of developers working on the project."

  In many cases, increasing the number of developers actually delays the project. Adding developers to a project requires time for training, which takes away time originally scheduled for development. Add resources earlier in the project rather than later. There also is a limit to the number of people who can work on a project effectively. With fewer people, there is less overlap. You can partition the project so each person works on a particular section. The more developers you add, the more difficult it becomes to avoid overlap.

Recommended Resources I

- Helpful resources from NI are:
  - LV menu: Help → Web Resources... (the official path)
    - → Technical Resources → Getting Started with LabVIEW: then e.g.

      → Learn LabVIEW Basics

      → On-Demand LabView Training (registration/login required)
  - NI: LabVIEW Introduction Course – Six Hours (the traditional approach)
    - http://zone.ni.com/devzone/cda/tut/p/id/5241
  - NI: Support Forum, Technical Support & Developer Zone (the Full Monty...)
    - http://forums.ni.com/ni/
    - http://www.ni.com/support/
    - http://zone.ni.com/dzhp/app/main
  - NI: LabVIEW Object-Oriented Programming (available only from LV 8.5)
    - http://zone.ni.com/devzone/cda/tut/p/id/3574
  - LabVIEW mailing list: read by experienced/helpful people, some from NI
    - http://www.info-labview.org/

Recommended Resources II

- Selected unofficial web and offline resources are:
  - LabVIEW Wiki: with LabVIEW Turorial
    - http://labviewwiki.org/Home & http://labviewwiki.org/LabVIEW_tutorial
  - Rensselaer Polytechnic Institute: Tutorials in G (for first practical steps)
    - http://www.cipce.rpi.edu/programs/remote_experiment/labview/
  - Connexions: online course on LabVIEW Graphical Programming (LV 7.1)
    - http://cnx.org/content/col10241/1.4/
  - LAVA: LabVIEW Advanced Virtual Architects (for advanced problems)
    - http://lavag.org/
  - Univ. of Utah, Dep. of Physics & Astro.: LabVIEW teaching materials
    - http://www.he-astro.physics.utah.edu/~jui/3620-6620/

      → Supplement Materials

      → Lectures and accompanying Materials
  - Stackoverflow: Q&A forum for programmers → search for LabVIEW
    - http://stackoverflow.com/questions/tagged/labview
  - Books: LabVIEW for Everyone & The LabVIEW Style Guide

Stage 2 : More Useful Functionality

Senior Honours Projects

- You have now a general overview of the functionalities of LV:
  - not all of the discussed you will need to complete your Senior Honours project
  - and there is more functionality you should familiarise yourself with before you start designing and coding for the project

- A:
  - T:

- A:
  - T:

Conclusions

- A:
  - T: