PHYC 500: Introduction to LabView

M.P. Hasselbeck, University of New Mexico

Exercise 10 (v 1.2)

This exercise demonstrates resource management in LabView. The goal is to: i) read a data file on the Internet, ii) convert the data file to a 1-D array of floating point numbers, iii) write this 1-D array to a file on the local disk, iv) open the file and use its data in a summing For Loop, and v) write the new data to disk. This is implemented in two independent VIs.

The file of interest is located at this URL:

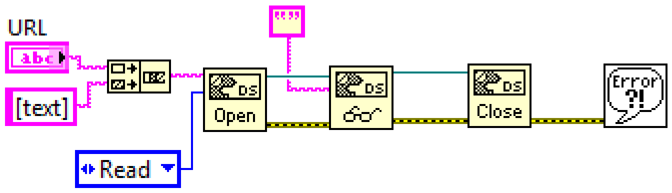
http://www.unm.edu/~mph/500/Ex10

The file can be accessed in LabView with a Data Socket, which should be open, read, and then closed.

Open a blank VI. Create a string control on the front panel and copy the entire URL above into it. The characters [text] must be appended to the URL string. This is performed with the Concatenate Strings function, which splices two strings together to make a single string of characters. Here it will be:

http://www.unm.edu/~mph/500/Ex10[text]

Wire this string to the **URL** input of Data Socket Open (locate it with Quick Drop) and configure its **mode** terminal for Read. Right-click on the terminal and create a constant; then select Read. Pass the **connection id** to Data Socket Read. This function must be instructed as to what type of data it is dealing with; wiring any string to the **type (Variant)** terminal defines the data type. In the diagram shown below, a blank string constant is used, but any string will work because the content is ignored. Close the resource when the read completes.



Error Cluster

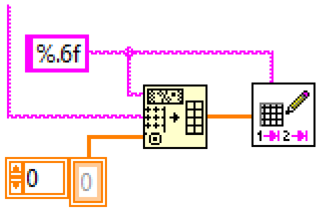
Error handling is important for reliable VI operation and is essential when making connections to external devices. An error cluster wire (dark yellow/white) is shown in the Block Diagram. LabView provides error information on many of its built-in functions; this information appears on an error cluster. The error cluster terminals are usually located on the lower left (error in) and lower right (error out) corners of the icons.

There are three components of an error cluster: i) a T/F Boolean indicates whether an error occurred, ii) an integer code for the error, and iii) a string with the location of the error. When an error occurs, this information propagates down the cluster wire. Any operation that encounters the error condition will be skipped.

For example, in the above Block Diagram, failure to open the URL will result in an error that appears on its error out terminal. The error propagates through the Read and Close operations, which are are skipped. The error is then read by the Simple Error Handler, which provides an informative pop-up dialog to the user.

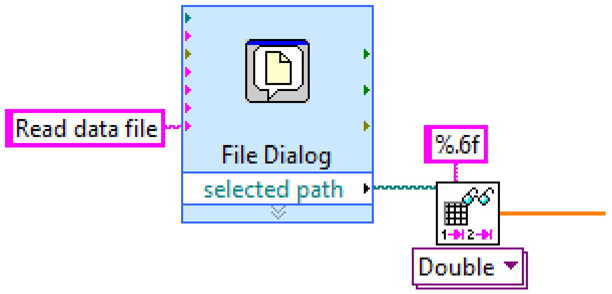
With a working Internet connect, test the VI.

The data string is converted to a 1-D array using the Spreadsheet String to Array function. In addition to the **spreadsheet string** from the Data Socket Read, this function requires formatting and configuration for a 1-D array (2-D is default). LabView has a powerful numeric formatting capabilities. Use the format syntax %.6f on the **format string** terminal. This specifies a floating point number with 6 digits of precision after the decimal point. Next, place an Array Constant on the Block Diagram. Create a DBL Numeric Constant and drop it into the Array Constant to define a 1-D array. Connect it to the **array type** terminal. This forces the function to produce a 1-D array of DBL floating point numbers.



A spreadsheet file can be created and written in a single, high-level operation using Write Delimited Spreadsheet File.vi found on the File I/O palette. Wire the 1-D array data to the appropriate input and format as %.6f as before. When the VI is run, the user will be prompted to save a file to disk. Also save the working VI.

Open a blank VI and place Read Delimited Spreadsheet File.vi on the Block Diagram. This high-level VI will be used to access the 1-D array file written above. It will deliver the data on the output terminal labeled **first row**. Set the format syntax to %.6f and wire the output into a For Loop. To setup a user prompt for opening the file, the Express VI File Dialog can be used:



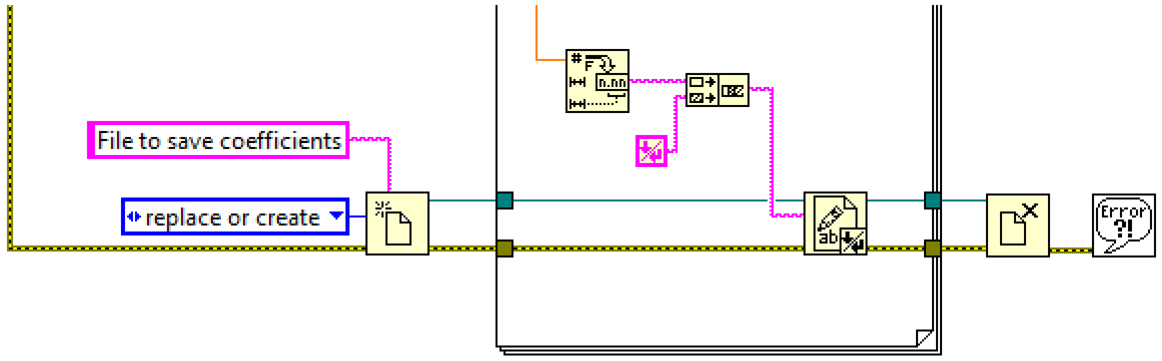
Using LabView's factorial function, divide each term in the data file by i! where i is the iteration count of the For Loop. Sum the result of each iteration using a shift-register (don't forget to initialize it to 0). Mathematically, the following finite series is being constructed:

$$\sum_{i=0} \frac{x_i}{i!}$$

If performed correctly, the series should sum to a value very close to 0.05.

Each term in the series should be written to a separate text file. This can be done when the For Loop exits, but it is often desirable to record data as it is being generated. This would be useful in a lengthy experiment, allowing some data to be recovered if a crash occurred before the experiment completed.

From the File I/O palette, select the Open/Create/Replace File VI and place it on the left outside of the For Loop. Configure it to **replace or create** and add a custom user prompt. The **refnum** is wired to a Write to Text VI inside the For Loop. Each series term is a floating point number that must be converted to a text string using Number to Fractional String VI. The Concatenate Strings functions adds a Line Feed character to the end of each data point so the data is displayed in a single column; otherwise, a single continuous line of text would be written. When the For Loop exits, the file is closed as shown:



You will need to disable indexing at both of these output tunnels by selecting Last Value in the Tunnel Mode.

This VI is dealing with two files: one is being read and a second is written. To force the read operation followed by write, connect the **error out** cluster from the File Dialog Express VI to the **error in** terminal of the Open/Replace/Create File VI. Terminate the error cluster in both VIs with a Simple Error Handler. This illustrates a second powerful feature of the error cluster. Data flow programming requires that *every* input terminal *must* be populated with data before a function or node can execute. The error cluster allows the LabView programmer to determine exact order of execution inside a VI.