PHY 406 - Microprocessor Interfacing Techniques
LabVIEW Tutorial - Part X
File Output and Input

Introduction

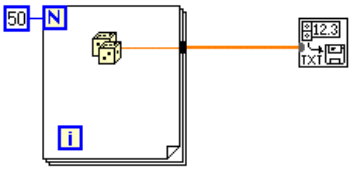
File I/O tends to be complex - simply because there are a myriad of things that you *might* want to do and the software has to be able to let you do (most of) them. LabVIEW can read and write four type of files:

- *Spreadheet*: These files consist of ASCII (normal) text with **tabs** in suitable places for importing into a spreadsheet. The VIs which control this form write one array to the file at a time.
- *Character:* These are similar to spreadsheet files in that they consist of ASCII text, but the format is freer.
- *Binary:* Consist of the data in computer format instead of ASCII. That means that the computer can read them, but you can't! They are more compact than character files and do not suffer from round-off errors. They be trasnlated into ASCII by other programs. Since the data do not have to be translated from binary to ASCII, they are also faster to write. Binary files can be files of 16-bit integers or single-precision reals.
- *Datalog:* These are similar to binary files except that the data can be more complex types (eg clusters). They must be read in the same format as they are writeen which constrains them (more or less) to be read by a similar LabVIEW program to the one which wrote them.

This tutorial is going to be concerned with *spreadsheet* files only. If you understand these - then you will be able to understand the LabVIEW elements of the others very quickly.

Writing Arrays

The unique thing about spreadsheet files is that they take either 1-D or 2-D tables of data. This is a simple VI to write a 1-D array of 50 random numbers to a file.



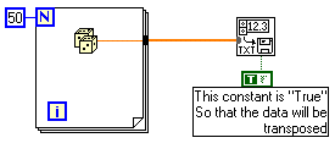
This is a representation of the output you get from this VI:

```
0.123     0.456     0.789     0.012     0.345     0.678.......
```

In other words a long line of numbers. This is probably not quite what you were thinking of - I suspect that if you had thought about it at all, then you would have expected a column of numbers, one to a line. However in LabVIEW terms *Rows* come before *Columns* and so we gat a

row of data.

Here is a second VI to make the point clear. It writes a 2-D array of nmbers with the second row of the array being just 10x the first row.



The result is:

```
0.123     0.456     0.789     0.012     0.345     0.678.......
1.234     4.567     7.890     0.123     3.456     6.789.......
```

Fortunately the LabVIEW people have thought of us and you can *transpose* the array, either 1-D or 2-D so that columns become rows and vice versa. This is accomplished with the *transpose* input set to "true" (These complex functions cry out for you to use the **Ctrl-h** help capability to sort out the terminals) as shown in the diagram.

The output from the second example now becomes:

```
0.123     1.234
0.456     4.567
0.789     7.890
0.012     0.123
.....
```

Writing to Files

Logically there are a large number of things to do with selecting the filename which need to be sorted out.

Do we want to select the filename at run-time or use a fixed filename?

Do we want to select the filename arbitrarily or have a pre-selected one we can over-ride?

If the file exists do we want to

i) give up
ii) append the data to the file
iii) Ask if we should over-write it.
iv) always over-write the file with the new data

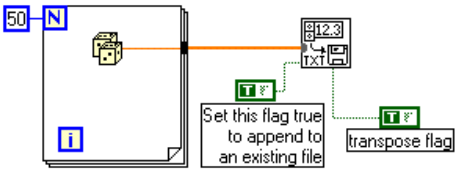
The examples above do absolutely nothing about any of these questions so they use the LabVIEW default. If you construct this VI using **function>>File I/O>>Write to Spreadsheet File** and run it, you will find that it comes up with a graphical selector allowing you to navigate the disk and directory structure of the machine and then **double-click** on a file or type in a new filename. Notice that if the file exists, you are queried as to whether you want to over-write it or not.

Filenames in Window95 are quite special. Having got rid of the "8+3" naming convention of DOS, you would thing that everything was free-form. Well, it is but... There is a very useful property of Windows95 that it can associate a file with a program and then when you **double-click** on the filename it will launch that program with the file in it. The specific example we are going to use here is that we want text files to be launched into the text editor where we can read the contents. The marvelous thing about this is that it is operated by the three-letter extension of the filename. Thus "something.txt" will be recognised as a text file.

Windows95 will, when it recognises a file extension make the file have an appropriate icon (in this case a little notepad) and will display the filename *without* the extension.

The point of this box therefore is that it is probably a good idea when creating spreadsheet files to give them a ".txt" extension and then you can open them easily. But remember that the extension will not show up in the graphical representation of the directory. It will show up in a command-line (MS-DOS) directory listing.

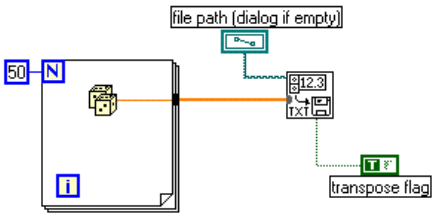
To view your files you should **double-click** on the "My Computer" icon and then follow down the disk and directory structure with **double-click** until you can see your file. **Double-click** on the file will bring it up in the editor so that you can read it.



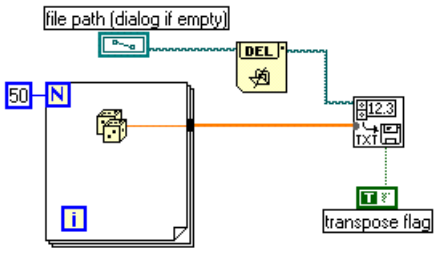
Assuming that giving up is not the option that you want, the next viable one is to append the data to the file. This can be accomplished by the "append" input which defaults to "false" but can be set to "true". I've added it to the "transpose" input here to clarify the presence of both flags.

If the file does not exist then the file is created.

If we want the option of over-writing, we have only to omit the "append" flag or set it false and we will always be queried about existing files when we **double-click** on them.

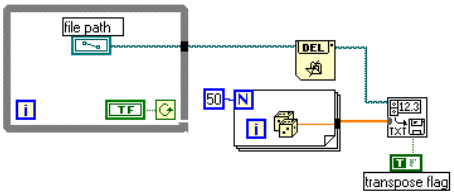


The more complex option is to pre-select a filename. This is simple enough to do by creating a constant or control of the "file path" type (easiest way is to use the wiring tool to **pop-up** on the terminal and create a constant or control). We will select a control. If the input selection is blank, then we will have exactly the same situation as we had before because the file path input defaults to an empty path and therefore a dialog box. However if we now enter a filename we will use that filename.



It isn't exactly right because if you play around with it a bit, you will find that it will create a file nicely, but it defaults to the "query before over-write" option whereas we might be more interested in the "over-write whatever" option. The only way of doing that is to delete the file before it starts with the "advanced" **function>>File I/O>>Advanced File Functions>>Delete**. Putting this in-between the control and the write function will cause the file to be deleted if it already exists. Notice the use of "dataflow" programming here as the path is taken as an output from the "delete" function to ensure the execution order is correct.

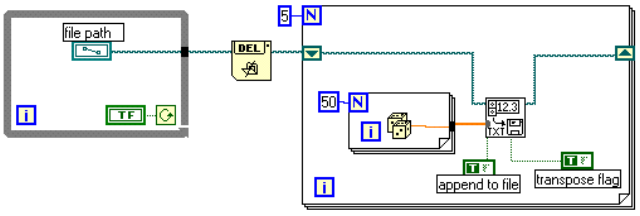
You might want to ponder (or better still - try out) what this does:



As a small clue - here is the corresponding front panel:

You can obviously get even more sophisticated with selecting a filename, but this walk-through should cover a lot of situations.

The next issue to deal with is the problem of writing several arrays to one file. This can be accomplished by using the fact that the **write to spreadsheet file** function has a path output as well as a path input and therefore you can use the concept of "dataflow" to establish a filename and then proceed to write to it. Here is a concept for writing a series of arrays to a file.



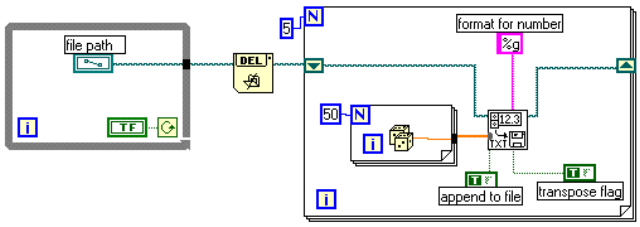
Notice that we have used a shift register to transfer the filename from one iteration of the loop to the nextbut we have also *initialised* the shift register from outside the loop to give it the correct initial filename.

Formatting the Numbers

You will have realised by now that default way of showing the numbers in the spreadsheet file is as a floating point number with three places after the decimal point. This can be changed by specifying a format with a string. The syntax used is the C-language syntax and for those who don't know it - here are a few useful examples:

%d	-	show as an integer number (for I16)
%g	-	show as a :"general" number
%fa.b	-	show as a floating point number with "a" places for the entire field and "b" places after the decimal point.

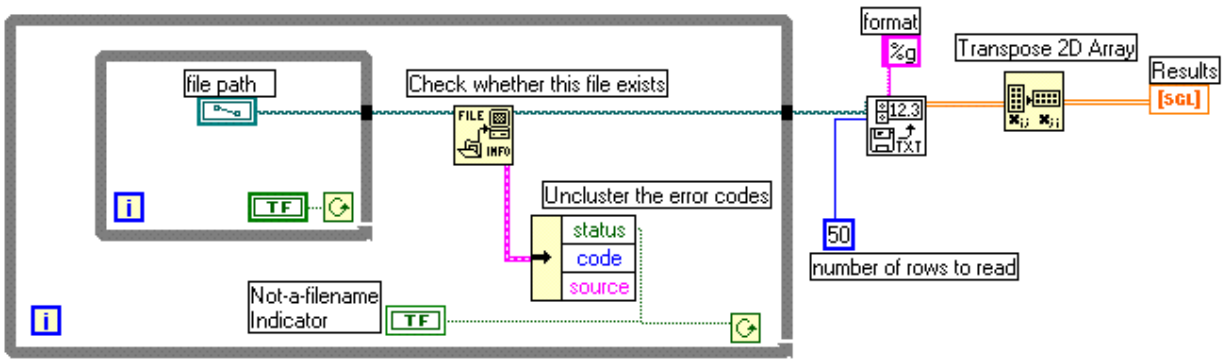
Here is a case where the syntax is used specifically.



The "%g" syntax in this case picks a suitable format for the numbers - in this case it picked "0.123456" as the format instead of the "%.3f" which is the default if you don't supply anything.

Reading Files

Reading files is probably a bit less important in LabVIEW because it is primarily designed for getting data from other places than the filestore. However everything that can be written in LabVIEW can be read. Here is a VI that will read a file of 50 numbers, one to a line, and plot them:



There are several things to note about this VI.

- Since LabVIEW considers rows before columns. Asking it to read 50 numbers, one to a line, is like asking it to read a 2-D array one wide and 50 deep. To get to a 1-D array, we need to transpose the matrix - 1x50 becomes 50x1.
- The 50 on the read VI corresponds to the number of rows read, irrespective of the number of numbers along a row.
- The %g refers to the format to read numbers with. %g will read just about any format - useful if you don't know what's coming!
- The double loop permits you to do two things: First confirm that the right filename is in the file path control and second, repeat the selection if the file is invalid.
- The checking of the file validity is done by a call to **function>>File I/O>>Advanced File Functions>>File/Directory Info**.
- The error cluster is unbundled and the error status (true/false) is used to terminate or repeat the loop and drive the "not a filename" indicator which is a modified "button" on the front panel
- Dataflow is used to ensure that the file operations are done in the right order.

Summary

- Spreadsheet file contain numbers separated by **Tabs**
- Spreadsheet files can map arrays in two different ways - the default is Rows before Columns.
- Getting the correct filename and getting the correct action for a potential over-write situation requires some care and thought
- Writing multiple arrays to a file requires using dataflow to move the filename through the various writes
- Formatting the numbers is simple - follow the C conventions.
- Reading is as simple as writing, but the file must exist.

Exercise

Write a program to write the a file of 30 lines each line consisting of the integer line number and the factorial (n!) Of the number thus:

```
1    1
2    2
3    6
4    24
5    120
....
```

Why won't I let you go any higher than 30?

Write a program te read a file of numbers at the rate of two a second and display the numbers in strip chart form.