9 Accessing Files in LabVIEW

In this lesson you will learn to describe the basic concept of file I/O and apply the appropriate File I/O functions to a given scenario.

Topics

+ Accessing Files from LabVIEW
+ High-Level and Low-Level File I/O Functions
+ Comparing File Formats

Exercises

Exercise 9-1 Exploring High-Level File I/O

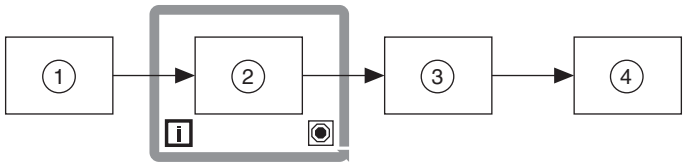
Exercise 9-2 Temperature Monitor VI—Logging Data

A. Accessing Files from LabVIEW

Objective: Identify the steps for writing and reading files from a LabVIEW application.

Typical File I/O Operations

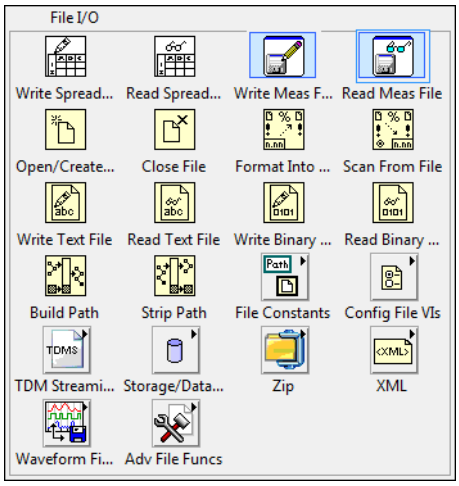
File I/O operations pass data to and from files.



1	Open File	3	Close File
2	Read/Write File	4	Check for Errors

File I/O Palette for File Operation Functions

The File I/O palette includes functions to create or open a file, read data from or write data to the file, and close the file. You can also use the functions to create directories; move, copy, or delete files; list directory contents; change file characteristics; or manipulate paths.



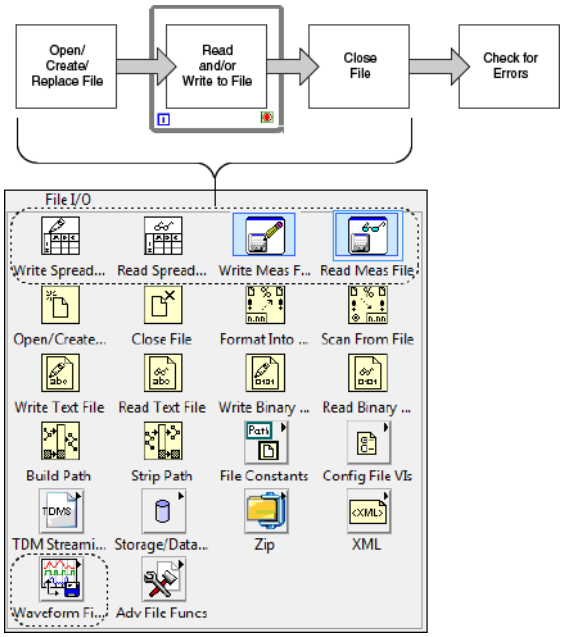
B. High-Level and Low-Level File I/O Functions

Objective: Identify when to use high-level and low-level File I/O functions.

High-Level File I/O Functions	Low-Level File I/O Functions
Perform three steps of the file I/O process. (open/write/close)	Perform one step of the file I/O process.
Simplify block diagram.	Provide finer control of file access.
Create unnecessary resource overhead when used in loops.	Save memory resources when used in loops.
Are good to use when writing to a file in a single operation.	Are good to use when streaming data to disk.

High-Level File I/O

High-level File I/O functions combine three steps (open, read/write, close) for common file I/O operations. High-level file I/O functions simplify the block diagram but might not provide the control or configuration you need for your application.



The following table compares the high-level file I/O functions.

Function	Type of Data	File Formats
Write To Spreadsheet File Read From Spreadsheet File	1D or 2D arrays	Text
Write To Measurement File Read From Measurement File	Signals (dynamic data types)	.lvm (text) .tdms (binary)
Write Waveforms to File Read Waveforms from File	Waveform data	Text

Exercise 9-1 Exploring High-Level File I/O

Goal

Observe how a high-level file I/O VI writes to a spreadsheet-readable file.

Scenario

The Spreadsheet Example VI does the following:

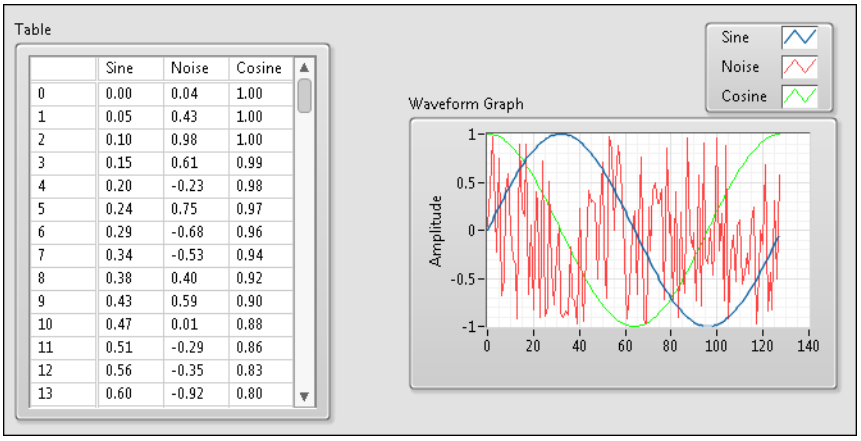
- Generates sine, noise, and cosine data for 128 points
- Stores this data in a 2D array that is 128 rows × 3 columns.
- Displays the 2D array in a **Table** indicator with three columns (Sine, Noise, and Cosine) for the first 14 rows of the array.
- Plots each column in a **Waveform Graph** indicator.
- Uses the Write To Spreadsheet File VI to save the numeric 2D array in a text file so a spreadsheet application can access the file.

Implementation

Complete the following steps to examine how the Spreadsheet Example VI performs the tasks described in the *Scenario* section.

1. Open `Spreadsheet Example.lvproj` in the `<Exercises>\LabVIEW Core 1\Spreadsheet Example` directory.
2. Open **Spreadsheet Example.vi** from the **Project Explorer** window.

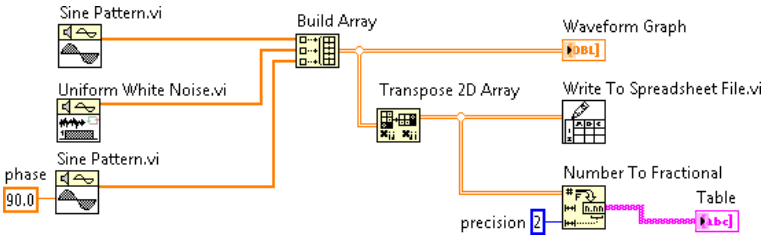
Figure 9-1. Spreadsheet Example VI Front Panel



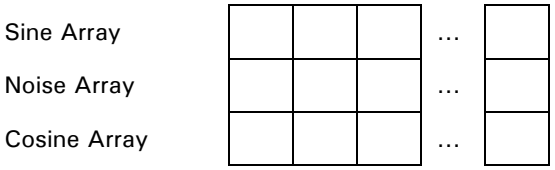
	Sine	Noise	Cosine
0	0.00	0.04	1.00
1	0.05	0.43	1.00
2	0.10	0.98	1.00
3	0.15	0.61	0.99
4	0.20	-0.23	0.98
5	0.24	0.75	0.97
6	0.29	-0.68	0.96
7	0.34	-0.53	0.94
8	0.38	0.40	0.92
9	0.43	0.59	0.90
10	0.47	0.01	0.88
11	0.51	-0.29	0.86
12	0.56	-0.35	0.83
13	0.60	-0.92	0.80

3. Run the VI.
4. Save the file, when prompted, as `wave.txt` in the `<Exercises>\LabVIEW Core 1\Spreadsheet Example` directory and click the **OK** button. You examine this file later.
5. Display and examine the block diagram for this VI.

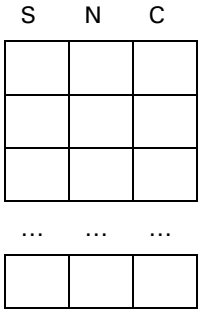
Figure 9-2. Spreadsheet Example VI Block Diagram



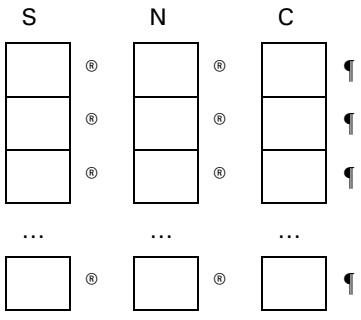
- Sine Pattern VI—Returns a numeric array of 128 elements containing a sine pattern. The constant `90.0`, in the second instance of the Sine Pattern VI, specifies the phase of the sine pattern which generates the cosine pattern.
- Uniform White Noise VI—Returns a numeric array of 128 elements containing a noise pattern.
- Build Array function—Builds the following 2D array from the sine array, noise array, and cosine array.



- Transpose 2D Array function—Rearranges the elements of the 2D array so element `[i,j]` becomes element `[j,i]`, as follows.



- Write To Spreadsheet File VI—Formats the 2D array into a spreadsheet string and writes the string to a file. The string has the following format, where an arrow (→) indicates a tab, and a paragraph symbol (¶) indicates an end of line character.



- Number To Fractional String function—Converts an array of numeric values to an array of strings that the table displays.

6. Close the VI. Do not save changes.

Note This example stores only three arrays in the file. To include more arrays, increase the number of inputs to the Build Array function.

7. Open the `wave.txt` file using a word processor, spreadsheet application, or text editor and view its contents.

   ☐ Open a word processor, spreadsheet application, or text editor, such as Notepad or WordPad.

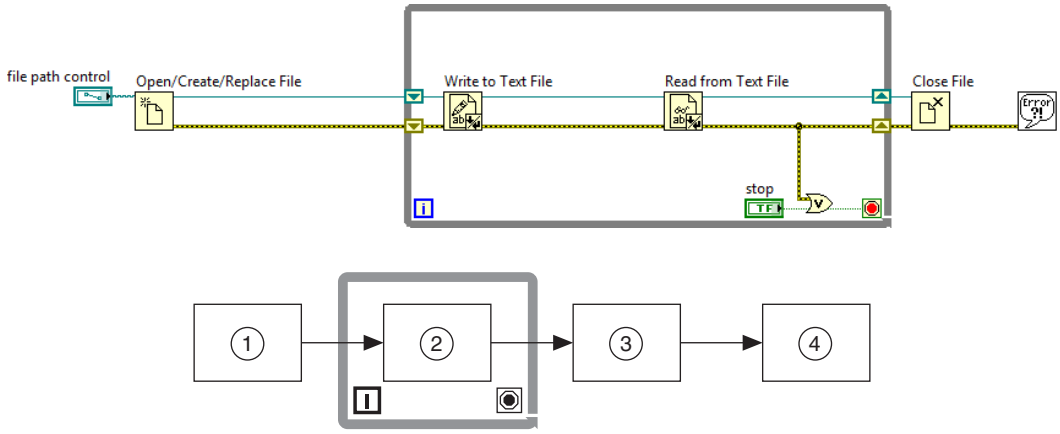
   ☐ Open `wave.txt`. The sine waveform data appear in the first column, the random (noise) waveform data appear in the second column, and the cosine waveform data appear in the third column.

8. Exit the word processor or spreadsheet application and return to LabVIEW.

End of Exercise 9-1

Low-Level File I/O

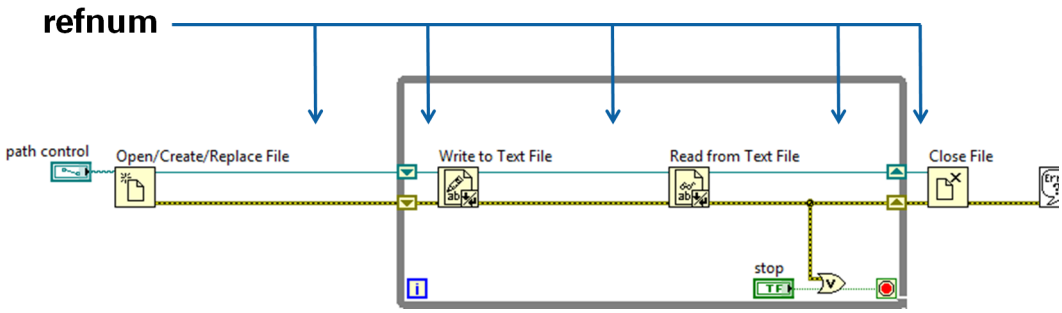
Low-level File I/O functions provide individual functions for each step in file I/O operations. For example, one function opens an ASCII file, one function reads an ASCII file, and one function closes an ASCII file.



1 Open, Initialize or create file resource—LabVIEW creates a reference number (refnum) as a unique identifier for the resource.
2 Read/Write to file
3 Close file resource—The refnum becomes obsolete
4 Check for errors—Display any errors from the file resource.

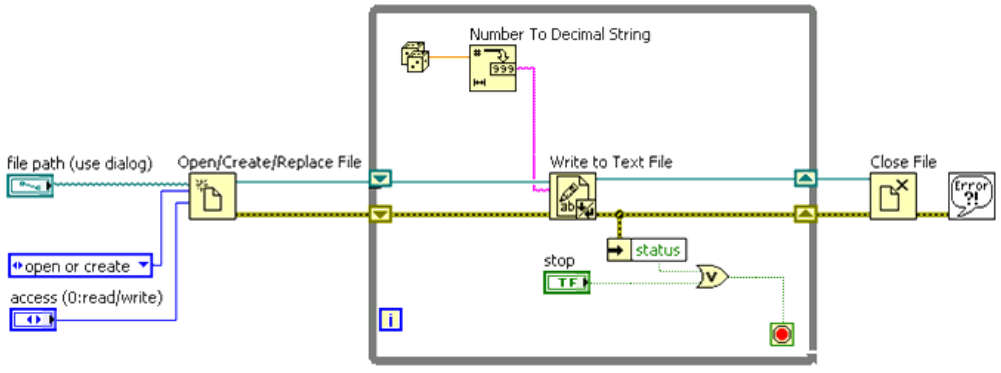
File Refnums

File refnums identify unique file I/O sessions.



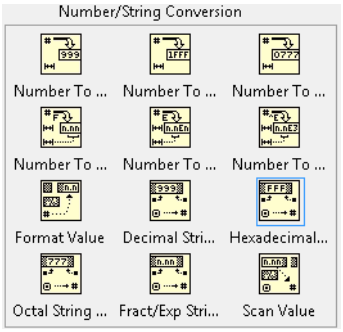
Streaming Data to Disk

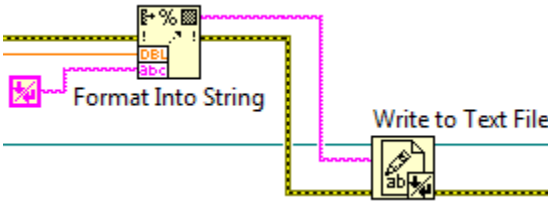
Disk streaming is a technique for keeping files open while you perform multiple write operations. Use low-level functions when file I/O is occurring within a loop to save memory resources.

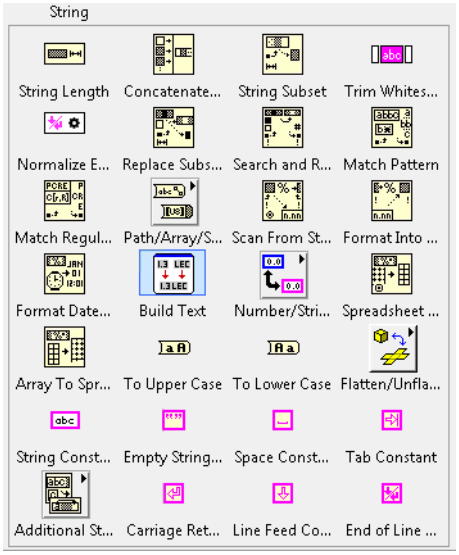


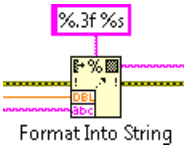
String Functions

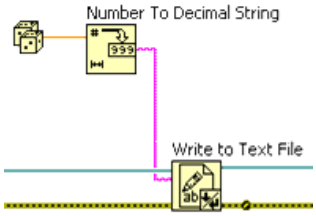
To write data to a text file, you must first convert the data into a string data type. Use the items from the String palette to convert numerics and other data types to text.











Tip Refer to the *LabVIEW Help* for details about formatting a string using the Format Into String function.

Exercise 9-2 Temperature Monitor VI—Logging Data

Goal

Modify a VI to create an ASCII file using disk streaming.

Scenario

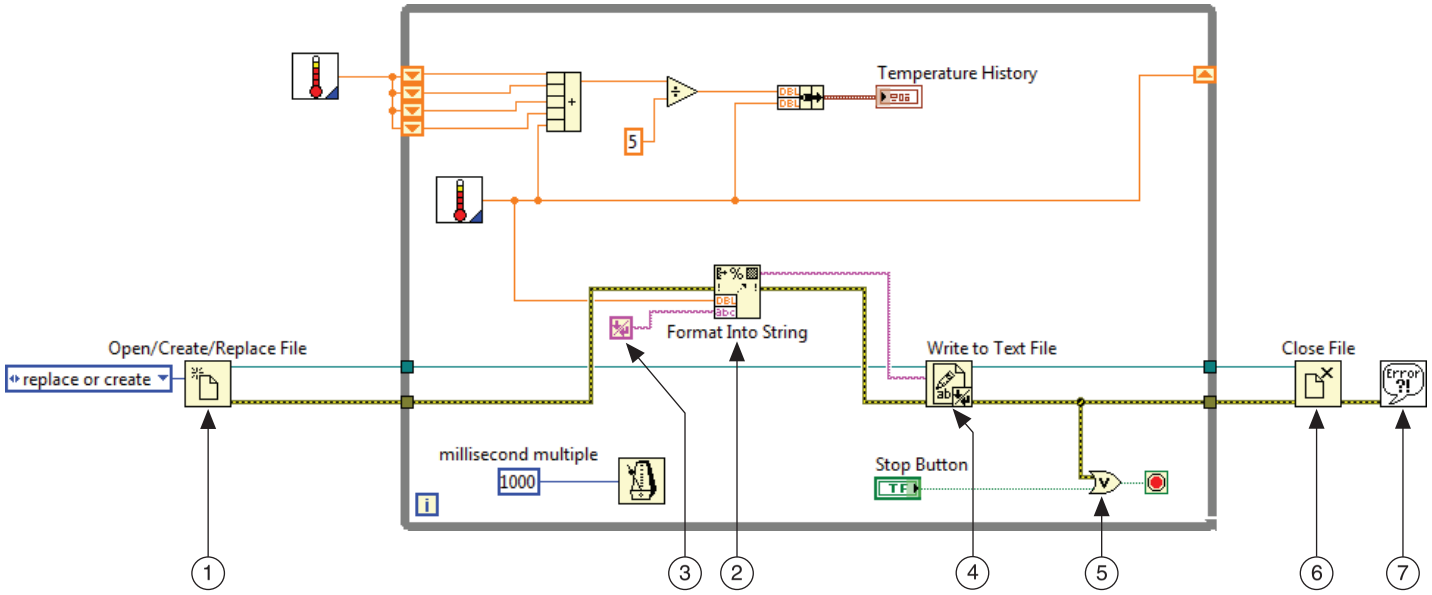
You have been given a VI that plots the current temperature and the average of the last three temperatures. Modify the VI to log the current temperature to an ASCII file.

Implementation

1. Open `Temperature Monitor.lvproj` in the `<Exercises>\LabVIEW Core 1\Temperature Monitor` directory.
2. Open **Temperature Monitor.vi** from the **Project Explorer** window.

3. Modify the block diagram as shown in Figure 9-3.

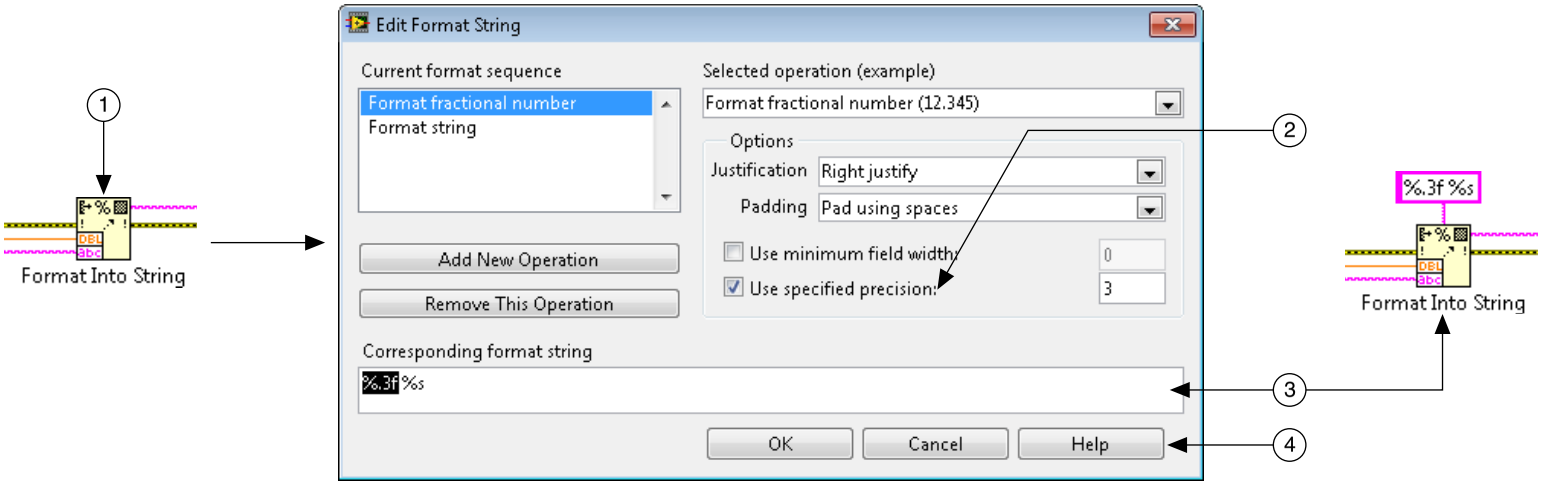
Figure 9-3. Temperature Monitor VI with Logging Block Diagram



1 **Open/Create/Replace File**—Creates or replaces an existing file for the data log. Right-click the **operation** input and select **Create»Constant**. Set the constant to **replace or create**.
2 **Format Into String**—Formats temperature data into a string. Expand the node to accept two inputs.
3 **End of Line Constant**—Adds an end-of-line constant after each piece of data so that data values are separated by line breaks.
4 **Write to Text File**—Writes the data to a file.
5 **Or**—Stops the VI when an error occurs or when the **Stop Button** is clicked.
6 **Close File**—Closes the data log file created or replaced when the VI started running.
7 **Simple Error Handler**—Indicates whether an error occurred. If an error occurred, this VI returns a description of the error and optionally displays a dialog box.

4. Configure the Format Into String function as shown in Figure 9-4.

Figure 9-4. Configuring the Format Into String Function



1 Format Into String—Double-click the Format Into String function to open the **Edit Format String** dialog box.
2 **Use specified precision**—Place a checkmark in this checkbox and enter 3 in the text box to specify that data have a floating point precision of three digits.
3 **Corresponding format string**—This text box automatically updates based on the configuration you specify. After you click the **OK** button in the dialog box, the block diagram updates to display the format string.
4 **Help** button—Click the **Help** button for more information about format specifier elements, such as %3f, and configuration options for the Format Into String function.

5. Test the VI.

   ☐ Run the VI.

   ☐ Give the text file a name and a location.

   ☐ Click the **Stop** button after the VI has been running for a few samples.

   ☐ Navigate to the text file created and explore it.

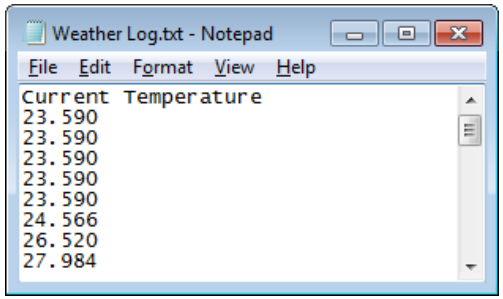
6. Close the VI and text file when you have finished.

Challenge

Objective: Create Log File with Single Header

To improve the usability of the log file, you are asked to include a header at the top of the log file as shown in Figure 9-5.

Figure 9-5. Temperature Monitor VI Log File with Header



Modify the Temperature Monitor VI to include the header `Current Temperature`.

Hints:

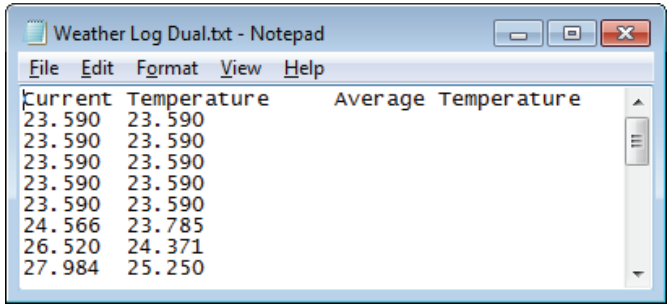
- Because you write the header to the text file only once, you should write to the header outside the While Loop.
- Use the functions on the **Strings** palette to manipulate and format a string for use in a word processing or spreadsheet application.

Challenge

Objective: Create Log File with Two Columns and Headers

Modify the VI to write both the current temperature and the average temperature to the log file. Separate the columns of data with a tab character and place a header at the top of each column as shown in Figure 9-6.

Figure 9-6. Temperature Monitor VI Log File with Two Columns and Headers



Current Temperature	Average Temperature
23.59	23.59
23.59	23.59
23.59	23.59
23.59	23.59
23.59	23.59
24.566	23.785
26.52	24.371
27.984	25.25

1 Tabbed columns in a text editor.

2 Tabbed columns in a spreadsheet application.

Hint: Use the Format Into String function and expand it to convert and format the data into a string.

End of Exercise 9-2

C. Comparing File Formats

Objective: Recognize different options for logging data to disk.

Common Log File Formats

LabVIEW can use or create the following file formats: Binary, ASCII, LVM, and TDMS.

- **ASCII**—An ASCII file is a specific type of binary file that is a standard used by most programs. It consists of a series of ASCII codes. ASCII files are also called text files.
- **LVM**—The LabVIEW measurement data file (`.lvm`) is a tab-delimited text file you can open with a spreadsheet application or a text-editing application. The `.lvm` file includes information about the data, such as the date and time the data was generated. This file format is a specific type of ASCII file created for LabVIEW.
- **Binary**—Binary files are the underlying file format of all other file formats.
- **TDMS**—This file format is a specific type of binary file created for National Instruments products. It actually consists of two separate files—a binary file that contains data and stores properties about the data, and a binary index file that provides consolidated information on all the attributes and pointers in the binary file.

Comparing File Formats

Use text files when you want to access the file from another application, if disk space and file I/O speed are not crucial, if you do not need to perform random access read or writes, and if numeric precision is not important.

	ASCII	Binary	TDMS	LVM
Easily Exchangeable	X		X	X
Small disk footprint		X	X	
Random read/write access		X	X	X
Inherent attributes		X	X	
High-speed streaming			X	

Additional Resources

Learn More About	LabVIEW Help Topic
File I/O	*File I/O*
Streaming data to disk	*Saving Memory using Disk Streaming*
	Streaming External Data to a TDMS File (Windows)
File Formats for file I/O	*Determining Which File Format to Use*
	Converting Numbers into Strings

Activity 9-1: Lesson Review

1. After opening a file, which output does the Open/Create/Replace File I/O function return?
   a. File path
   b. File name
   c. Refnum out
   d. Task out

2. Which file format is best in an application that requires high-speed streaming?
   a. ASCII
   b. Binary
   c. TDMS
   d. LVM

Activity 9-1: Lesson Review - Answers

1. After opening a file, which output does the Open/Create/Replace File I/O function return?
   a. File path
   b. File name
   c. Refnum out
   d. Task out

2. Which file format is best in an application that requires high-speed streaming?
   a. ASCII
   b. Binary
   c. TDMS
   d. LVM