# Basics of File I/O

A typical file I/O operation involves the following process.

1. You create or open a file. After the file opens, a unique identifier called a refnum represents the file.
2. The File I/O VI or function reads from or writes to the file.
3. You close the file.

File I/O VIs and some File I/O functions, such as the Read from Text File and Write to Text File functions, can perform all three steps for typical file I/O operations. The VIs and functions designed for multiple operations might not be as efficient as the functions that perform individual operations.

# Choosing a File I/O Format

The VIs on the File I/O palette you use depend on the format of the files. You can read data from or write data to files in three formats—text, binary, and datalog. The format you use depends on the data you acquire or create and the applications that will access that data.

Use the following basic guidelines to determine which format to use:

- If you want to make your data available to other applications, such as Microsoft Excel, use text files because they are the most common and the most portable.
- If you need to perform random access file reads or writes or if speed and compact disk space are crucial, use binary files because they are more efficient than text files in disk space and in speed.
- If you want to manipulate complex records of data or different data types in LabVIEW, use datalog files because they are the best way to store data if you intend to access the data only from LabVIEW and you need to store complex data structures.

## *When to Use Text Files*

Use text format files for your data to make it available to other users or applications, if disk space and file I/O speed are not crucial, if you do not need to perform random access reads or writes, and if numeric precision is not important.

Text files are the easiest format to use and to share. Almost any computer can read from or write to a text file. A variety of text-based programs can read text-based files. Most instrument control applications use text strings.

Store data in text files when you want to access it from another application, such as a word processing or spreadsheet application. To store data in text format, use the String functions to convert all data to text strings. Text files can contain information of different data types.

Text files typically take up more memory than binary and datalog files if the data is not originally in text form, such as graph or chart data, because the ASCII representation of data usually is larger than the data itself. For example, you can store the number –123.4567 in 4 bytes as a single-precision floating-point number. However, its ASCII representation takes 9 bytes, one for each character.

In addition, it is difficult to randomly access numeric data in text files. Although each character in a string takes up exactly 1 byte of space, the space required to express a number as text typically is not fixed. To find the ninth number in a text file, LabVIEW must first read and convert the preceding eight numbers.

You might lose precision if you store numeric data in text files. Computers store numeric data as binary data, and typically you write numeric data to a text file in decimal notation. A loss of precision might occur when you write the data to the text file. Loss of precision is not an issue with binary files.

Use the File I/O VIs and functions to read from or write to text files and to read from or write to spreadsheet files.

## *When to Use Binary Files*

Storing binary data, such as an integer, uses a fixed number of bytes on disk. For example, storing any number from 0 to 4 billion in binary format, such as 1, 1,000, or 1,000,000, takes up 4 bytes for each number.

Use binary files to save numeric data and to access specific numbers from a file or randomly access numbers from a file. Binary files are machine readable only, unlike text files, which are human readable. Binary files are the most compact and fastest format for storing data. You can use multiple data types in binary files, but it is uncommon.

Binary files are more efficient because they use less disk space and because you do not need to convert data to and from a text representation when you store and retrieve data. A binary file can represent 256 values in 1 byte of disk space. Often, binary files contain a byte-for-byte image of the data as it was stored in memory, except for cases like extended and complex numeric values. When the file contains a byte-for-byte image of the data as it was stored in memory, reading the file is faster because conversion is not necessary.

**Note:** Text and binary files are both known as byte stream files, which mean they store data as a sequence of characters or bytes.

Use the File I/O VIs and functions to read from and write to binary files. Consider using the binary file functions if you want to read numeric data from or write numeric data to a file or if you want to create text files for use on multiple operating systems.

## *When to Use Datalog Files*

Use datalog files to access and manipulate data only in LabVIEW and to store complex data structures quickly and easily.

A datalog file stores data as a sequence of identically structured records, similar to a spreadsheet, where each row represents a record. Each record in a datalog file must have the same data types associated with it. LabVIEW writes each record to the file as a cluster containing the data to store. However, the components of a datalog record can be any data type, which you determine when you create the file.

For example, you can create a datalog whose record data type is a cluster of a string and a number. Then, each record of the datalog is a cluster of a string and a number. However, the first record could be (“abc”,1), while the second record could be (“xyz”,7).

Using datalog files requires little manipulation, which makes writing and reading much faster. It also simplifies data retrieval because you can read the original blocks of data back as a record without having to read all records that precede it in the file. Random access is fast and easy with datalog files because all you need to access the record is the record number. LabVIEW sequentially assigns the record number to each record when it creates the datalog file.

You can access datalog files from the front panel and from the block diagram.

LabVIEW writes a record to a datalog file each time the associated VI runs. You cannot overwrite a record after LabVIEW writes it to a datalog file. When you read a datalog file, you can read one or more records at a time.

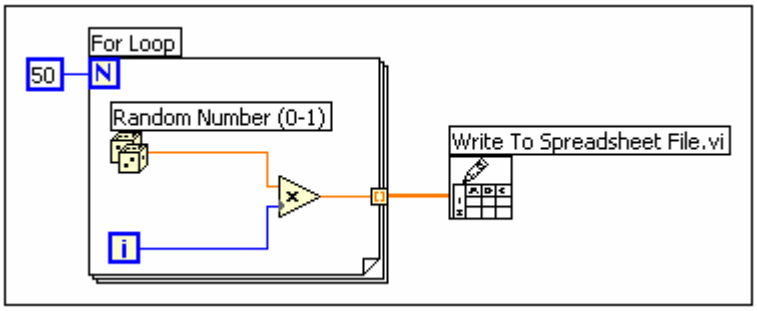
# Using VIs and Functions for Common File I/O Operations

The File I/O palette includes VIs and functions designed for common file I/O operations, such as writing to or reading from the following types of data:

- Numeric values to or from spreadsheet text files
- Characters to or from text files
- Lines from text files
- Data to or from binary files

You also can configure read and write functions, such as the Read from Text File and Write to Text File functions, to perform common file I/O operations. These VIs and functions for common operations can open the file or prompt you to do so in a dialog box, perform the read or write operation, then close the file to save you time and programming effort. Avoid placing the File I/O VIs and functions configured to perform multiple operations in loops because the VIs and functions perform open and close operations each time they run. Configure the functions to keep files open while you perform multiple operations.

The following block diagram shows how to use the Write To Spreadsheet File VI to send numbers to a tab-delimited spreadsheet file. When you run this VI, LabVIEW prompts you to write the data to an existing file or to create a new file.




The open, read, and write functions expect a file path input. If you do not wire a file path, a dialog box appears for you to specify a file to read from or write to.

# Using Functions for Advanced File I/O Operations

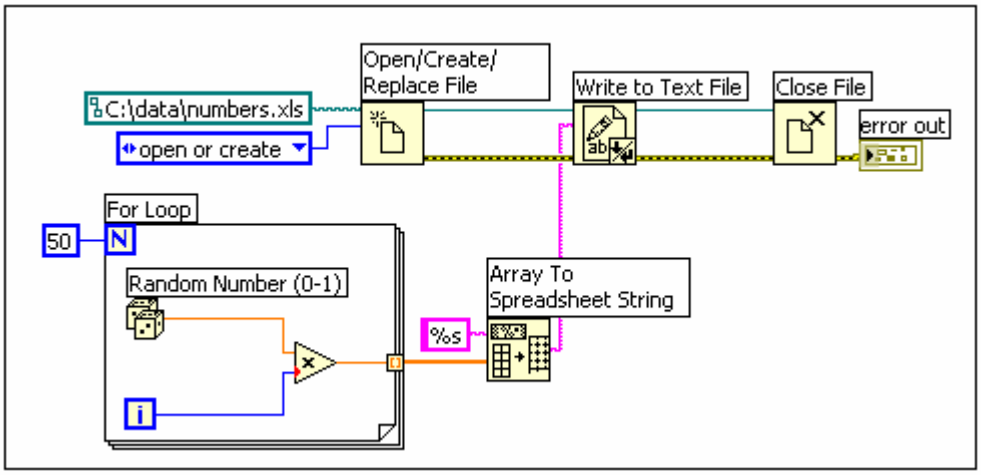
The File I/O palette includes functions to control each file I/O operation individually. Use these functions to create or open a file, read data from or write data to the file, and close the file. You also can use them to perform the following tasks:

- Create directories.
- Move, copy, or delete files.
- List directory contents.
- Change file characteristics.
- Manipulate paths.

A path, shown as follows, is a LabVIEW data type that identifies the location of a file on disk.

The path describes the volume that contains the file, the directories between the top-level of the file system and the file, and the name of the file. Enter or display a path using the standard syntax for a given platform with the path control or indicator.

The following block diagram shows how to use File I/O functions to send numeric data to a tab-delimited spreadsheet file. When you run this VI, the Open/Create/Replace File function opens the numbers.xls file. The Write to Text File function writes the string of numbers to the file. The Close File function closes the file. If you do not close the file, the file stays in memory and is not accessible from other applications or to other users.




Compare the previous block diagram to the Write To Spreadsheet File VI, which completes the same task. The previous block diagram uses individual functions for each file operation, including using the Array To Spreadsheet String function to format the array of numbers as a string. The Write to Spreadsheet File VI completes multiple file operations, including opening the file, converting the array of numbers to a string, and closing the file.

## *Using Path Controls and Refnums*

In the previous example, the Open/Create/Replace File function returns a refnum, which is an identifier LabVIEW uses as a reference to a file. File I/O functions that accept path or refnum data types as inputs include an input with the phrase "(use dialog)", such as the **path (use dialog)** input. If you do not wire a control or constant to this input, a dialog box appears prompting users to select a file or directory when the function executes. Each of these functions include a **cancelled** output. If users click the **Cancel** button in the dialog box, **cancelled** returns TRUE and the function returns an error. If you wire a path data type to a File I/O function, the function opens with the minimum permissions needed to perform the function and eliminates the need for explicitly opening or closing the file.

For example, if you wire a path data type to the Read from Text File function, the function opens the file as read-only. LabVIEW closes the file if you do not wire the **refnum out** output to another function. If you wire a path to the Write to Text File function, the function opens the file as read/write and appends any text you write to the file after the last line of the file. In both cases, the functions close the file if you do not wire the **refnum out** output to another function. If you wire the **refnum out** output to another function, the file remains in memory until you explicitly close the file with the Close File function, such as in disk-streaming, or until another function or VI later in the data flow closes the reference.

## *Disk Streaming*

You also can use File I/O functions for disk streaming operations, which save memory resources by reducing the number of times the function interacts with the operating system to open and close the file. Disk streaming is a technique for keeping files open while you perform multiple write operations, for example, within a loop. Wiring a path control or a constant to the Write to Text File function, the Write to Binary File function, or the Write

To Spreadsheet File VI adds the overhead of opening and closing the file each time the function or VI executes. VIs can be more efficient if you avoid opening and closing the same files frequently.

To create a typical disk-streaming operation, place the Open/Create/Replace File function before a loop, the read or write function in the loop, and the Close File function after the loop so continuous writing to a file can occur within the loop without the overhead associated with opening and closing the file in each iteration.

Disk streaming is ideal in lengthy data acquisition operations where speed is critical. You can write data continuously to a file while acquisition is still in progress. For best results, avoid running other VIs and functions, such as Analysis VIs and functions, until you complete the acquisition.