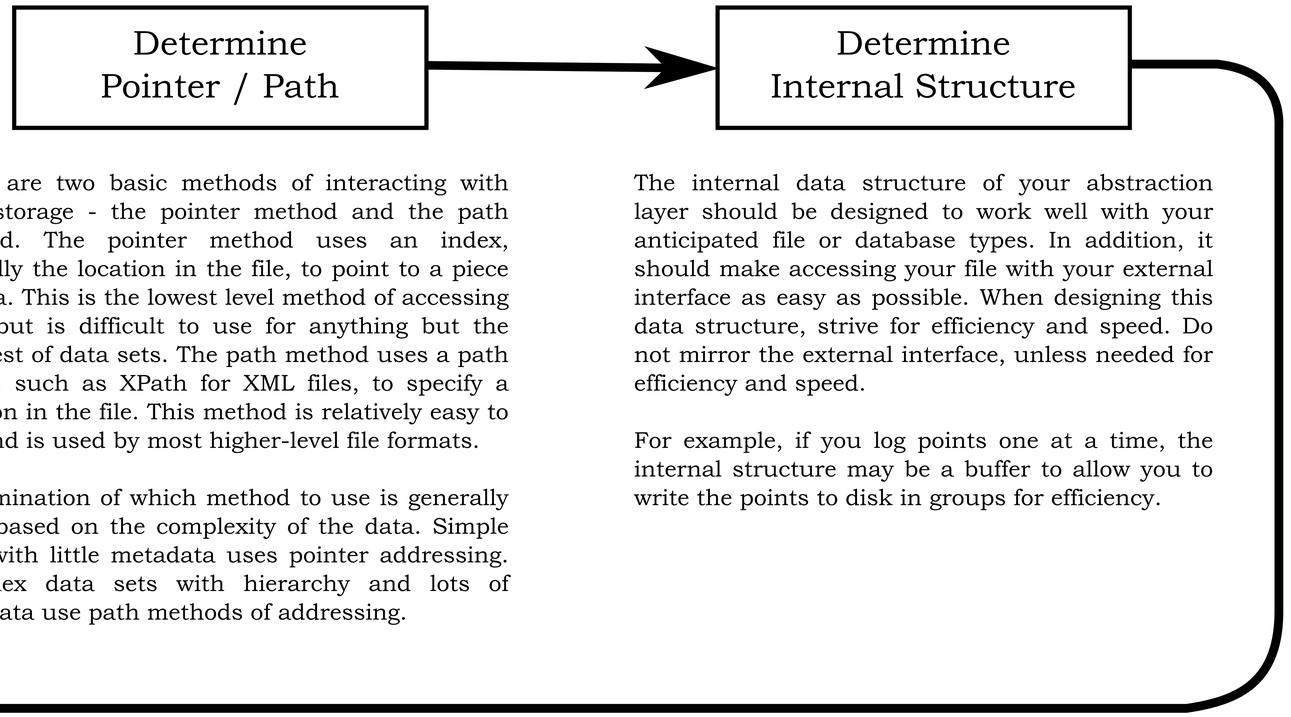
# File Abstraction Layers for Data Storage

Damien F. Gray, PhD - Senior Software Engineer - National Instruments

## Overview

Files come in many, many different formats. It is impossible to create an abstraction method that will cover all file types. However, you can create an abstraction for many types that will work for your application. This will allow you to write your top level logging and access code once to use many different types of files (database, XML, etc.). Start with determining how you want to access your file, using pointer or path format.




There are two basic methods of interacting with data storage - the pointer method and the path method. The pointer method uses an index, typically the location in the file, to point to a piece of data. This is the lowest level method of accessing files, but is difficult to use for anything but the simplest of data sets. The path method uses a path string, such as XPath for XML files, to specify a location in the file. This method is relatively easy to use and is used by most higher-level file formats.

Determination of which method to use is generally done based on the complexity of the data. Simple data with little metadata uses pointer addressing. Complex data sets with hierarchy and lots of metadata use path methods of addressing.

The internal data structure of your abstraction layer should be designed to work well with your anticipated file or database types. In addition, it should make accessing your file with your external interface as easy as possible. When designing this data structure, strive for efficiency and speed. Do not mirror the external interface, unless needed for efficiency and speed.

For example, if you log points one at a time, the internal structure may be a buffer to allow you to write the points to disk in groups for efficiency.

## What is an abstraction layer?

An abstraction layer is an API which is the same for several different implementations. Object oriented computer languages make them easier to construct, but they can be done with older languages, such as C. For example:

saveWfm (char* WfmName, double* Wfm);

could save data to a text file, a binary file, an XML file, or an HDF5 file. It is a simple abstraction layer.

The external interface is the programming layer you use to write your applications. It should be designed to make writing those applications as simple as possible. For example, if you store time/value pairs of data, you probably want a function to return a value, given a time.

Implement your external interface in your language of choice. Object oriented languages, such as Java, C#, or LabVIEW, make this much easier. This should be an abstract class or interface which only creates the methods for interacting with your files. The actual working code will be written using this code as a template.

With your interface/abstract classes written, write the actual working code for all the different file types you would like to use, one set per file type. For example, you pick XML for your first file type, since you have lots of metadata and a complex file structure. To get better performance, you pick HDF5 as your second file type. Your calling code uses the abstract classes, so does not change.

## File Type Hierarchy