

Temperature Controlled High Throughput Diode Tester A Case Study in Using LVOOP to Save Development Time

by
Benjamin Rayner
Senior Architect, LabVIEW Champion
Data Science Automation, Inc.
USA

Category:
Advanced Control Systems

Products Used:
LabVIEW 2009
NI USB-GPIB
LVOOP

The Challenge:
Testing of IR diodes manually was taking too much time. The market demand for IR diodes exceeds the capacity of manual testing. Testing over the entire rated temperature range could not be implemented without exposing the person conducting the test to the same conditions.

The Solution:
A test station was developed using LabVIEW (Figure 1) that is capable of controlling a temperature chamber, a pair of SMU (Source Measurement Unit) and a set of custom built multiplexers controlled via SPI interfaces.

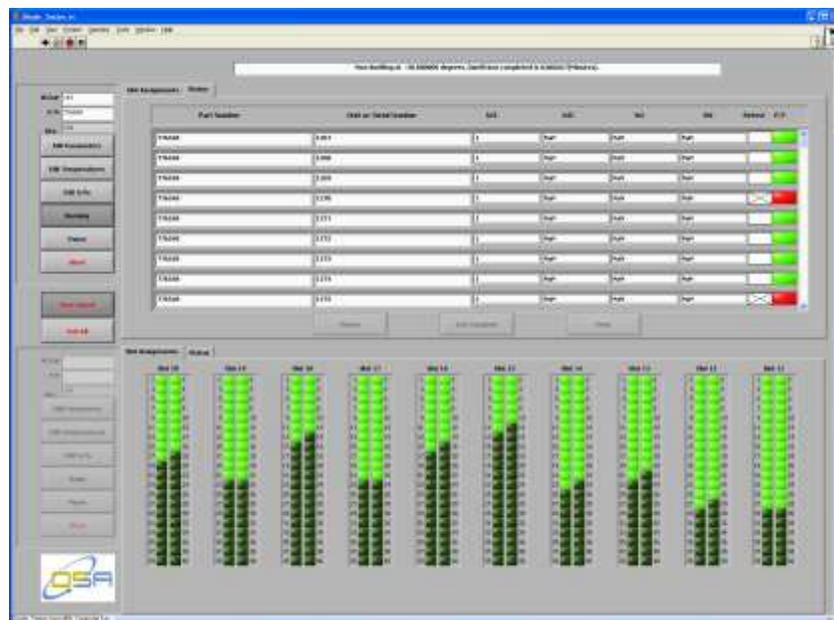


Figure 1. Diode Tester Allows Monitoring of Running Tests While Setting Up for Another.

Abstract:

A fully automated test stand was developed using NI LabVIEW to control, monitor and log all measurements of all device at all required temperatures. Groups of 400 diodes (up to 800 total) can be tested during a single test cycle. If using two groups, one set can be tested while another set is being loaded. Report files record all readings from each device and are grouped by diode part number.

Background:

A manufacturer of infrared diodes had a need to increase the production rate of IR diodes. The production rate was limited by the amount of time required by an operator to manually perform the required checks and record the measurements. The desire to test the devices at extreme temperatures was not possible manually due to material handling limitations at the extreme temperatures.

Details:

A full test of an IR diode requires measuring various attributes (Figure 2) such as dark current and bias currents at a number of temperatures (Figure 3) while keeping the diode itself in the dark. A temperature chamber was used to provide the environmental control. For each test condition, the chamber was first ramped to the target temperature and allowed to stabilize before measurements were taken. Since the temperature stabilization was slow process, the test stand was designed to allow multiple diodes to be loaded and ramped to the appropriate temperature as a batch.

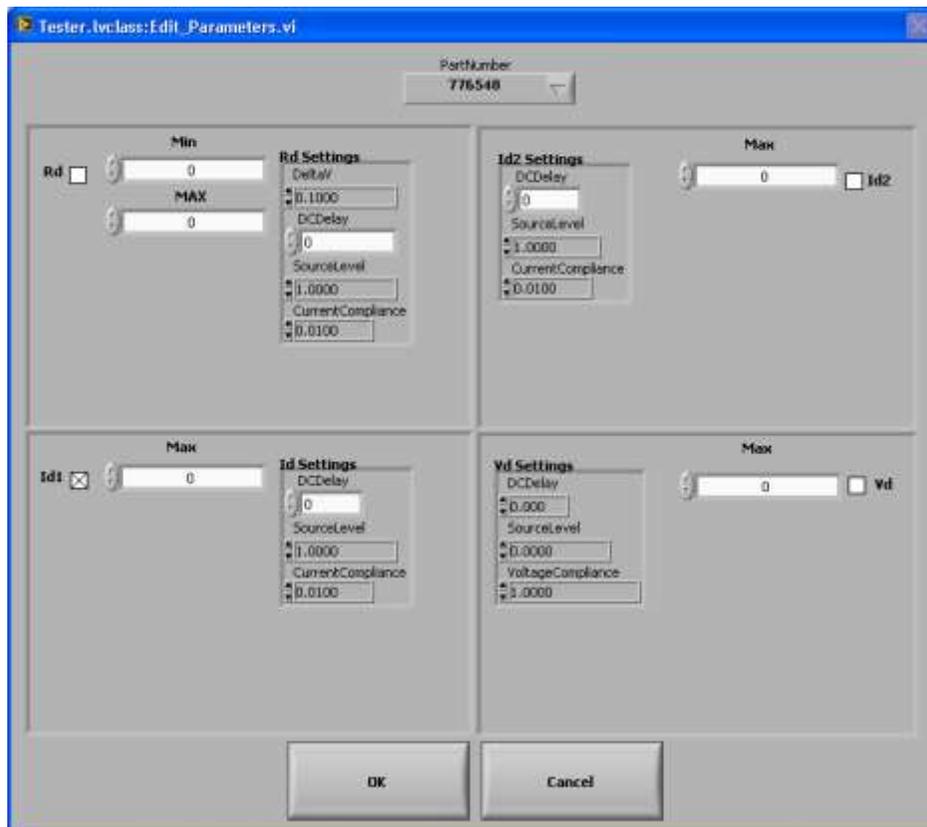


Figure 2. Acceptable Ranges of Readings as Well as the Test Criteria.

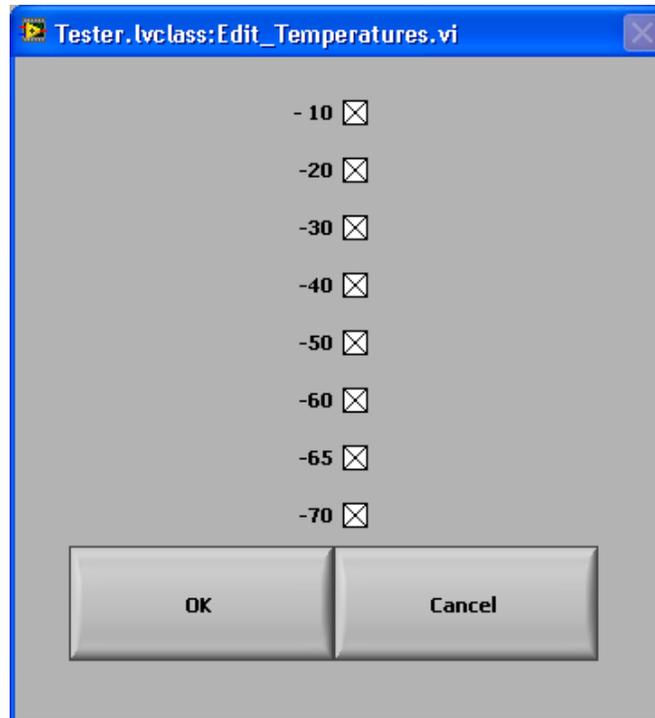


Figure 3. User Can Select the Temperatures Used for Testing.

The measurements performed on the diodes were completed using a pair of Keithley 236 Source Measure Units (SMU). Two SMUs permitted two modes of operation that resulted in higher throughput. In the normal mode of operation, each SMU measured all of the diodes in a group of 400 devices. In this mode, 400 diodes could be under-going evaluation while another batch of diodes was being loaded in preparation for testing. When a batch was loaded, it could be started independently of any other testing that may be taking place at the same time (Figure 4).

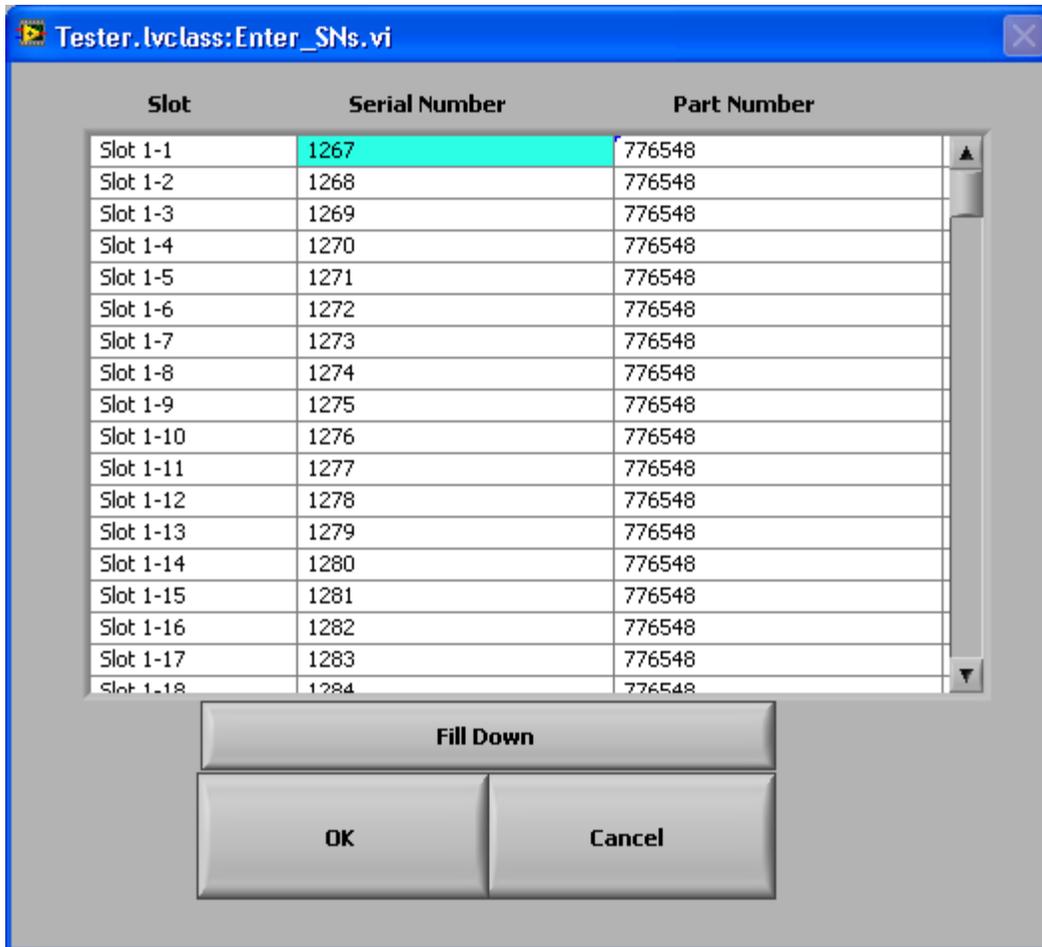


Figure 4. Arbitrary Slots Assignments, Mixed Device Types and Non-contiguous S/N's Entered While Another Test is Running.

In Linked mode, all 800 diodes are tested as a single batch with each SMU measuring half of the diodes. This mode allows for 800 diodes to be tested in the same time required to test 400.

The connectivity between the SMUs and the diodes was implemented using a set of custom built multiplexers (Mux). Two mux subsystems were implemented, one for each SMU. The muxes themselves utilized an array of relay contacts to make the physical connection between the diodes and the SMUs. The computer control of the mux sub-systems was implemented using Cheetah USB-SPI adapters. A proprietary protocol was designed and implemented using the SPI bus to the mux control logic.

The design of the mux sub-system underwent some changes during development to allow for easier use by the operator and increase throughput. Stand-alone testers were developed early in the development cycle to verify the operation of each sub-system individually. The testers for the mux revealed issues that were not recognized at initial design time. The issues were able to be resolved well before the various components were integrated due to the software testing features that were developed.

Implementation

All of the code used was developed using LabVIEW Object Oriented Programming (LVOOP). Utilizing LVOOP allowed the development time to be reduced by taking advantage of the Dynamic Dispatch features available in LVOOP. The reduction in development time manifested itself by being able to model all of the hardware interfaces as children of a single "Hardware"

class (see Figure 5). Operations that were common for all of the hardware sub-systems were implemented in the parent “Hardware” class allowing all of the other classes to inherit those functions without having to write code for each. Examples of the common functionality implemented in the Hardware class are “Define configuration paths”, reading configuration files and creating unique instances for each hardware sub-system.

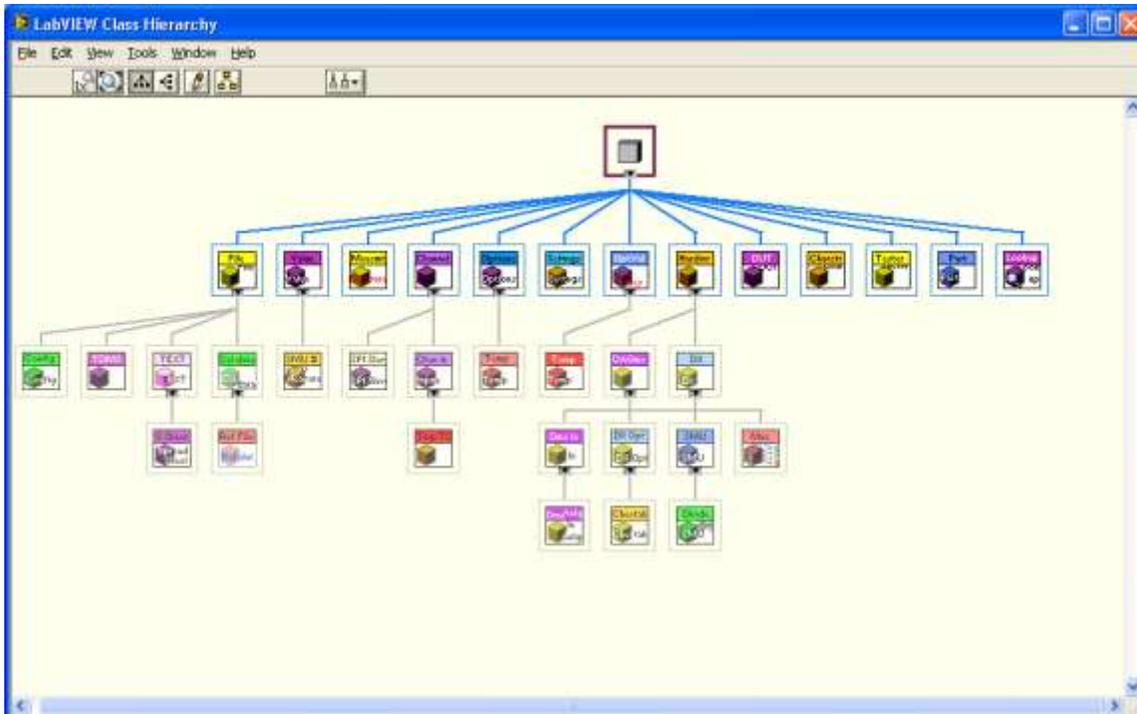


Figure 5. LVOOP Object Class Hierarchy.

Figure 6 shows the File Class hierarchy in the first application where the LVOOP library was born. Development started with an intent to support three types of files: Configuration, TDMS and MVAS (a proprietary form of TDMS file, not discussed in this document). More on each of these classes follows.

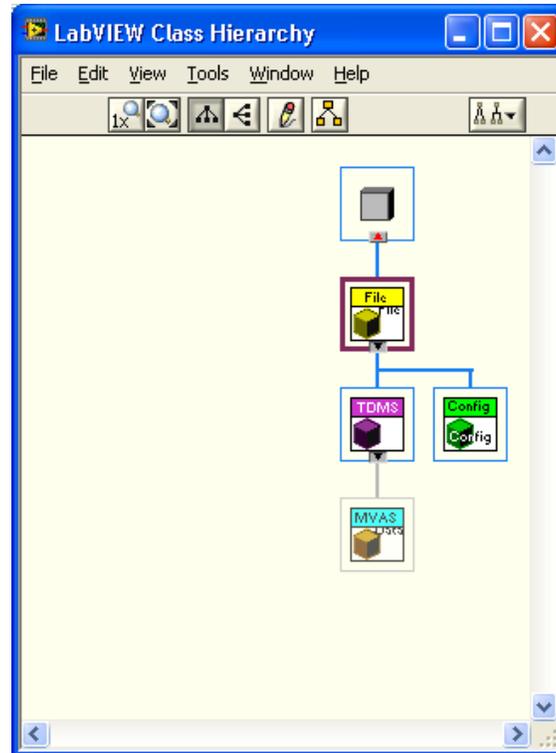


Figure 6. LVOOP Class Library Stage 1.

LVOOP is not “all or nothing”. An application can utilize LVOOP for some of its functionality and use standard LabVIEW methodology for the rest of the application.

File - parent class for all file related classes. Implements all functionality common across all types of files e.g. Open, Close, Set Path, etc. An example of the robust implementation used in the File class is shown in Figure 6.

Configuration - Child of File and exposes methods to access configuration files. The methods exposed closely mimic those available for configuration files but also implement methods to support Rings which are not supported by the standard configuration file functions.

TDMS - Child of File and exposes methods to allow accessing TDMS files.

MVAS - Application specific class that provides special functionality required by the original application including the ability to save and restore jpgs as part of the test set-up.

Going the Extra Mile

Figure 7 shows an example of the robustness built into the File Class. When invoked, the "Set_Path" method performs a check of the path specified by its caller to ensure a valid path was specified. If it is not valid an error is declared (not shown) and the operation is skipped. Provided the path is valid, an accessor method is used to cache the path information and a flag is cleared indicating the file is not open. The path validity checks will allow for easier debugging since it will declare the provided path was not valid when this method is invoked rather than accepting an invalid path that would result in error when attempting to open the file at a later time and possibly in another call chain. Declaring errors as early as possible in program flow makes for easier to maintain code.

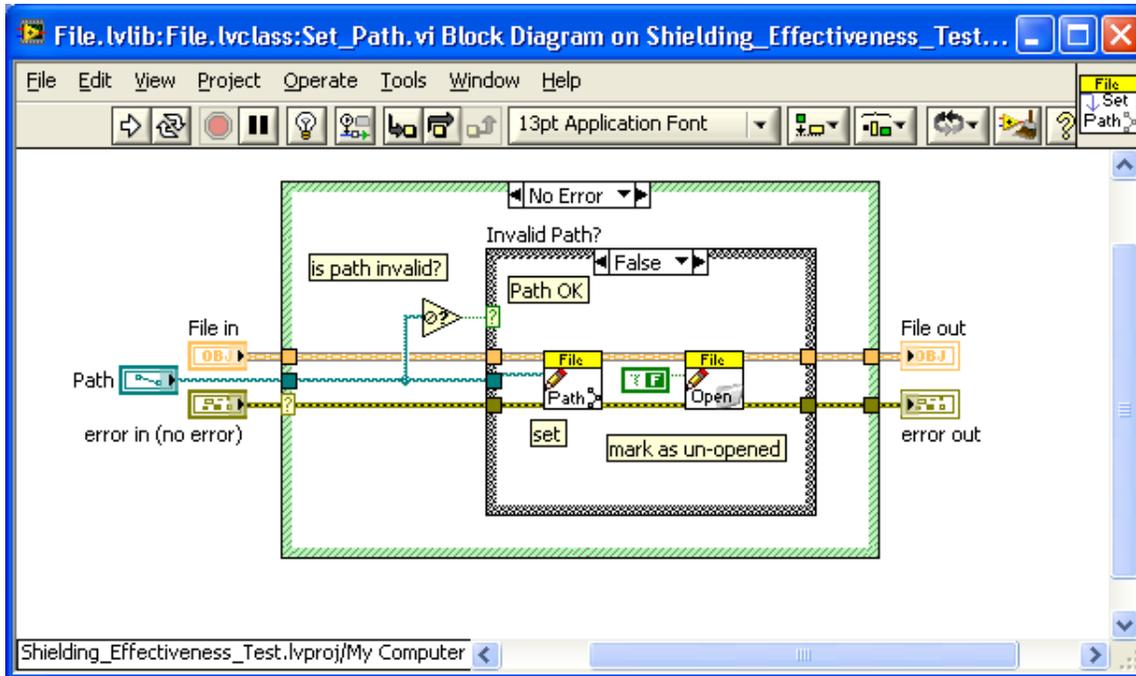


Figure 7. "Going the Extra Mile."

“Going the extra mile” and checking the validity of parameters may seem like over-kill but proves to be worth the effort in producing easy to use code that not only works but also is capable of identifying when it is being abused.

The flag that is cleared after the path is cached is used to prevent an error when attempting to write or read from the file that is not open. If an attempt is made to write or read from a file that is not open, a clear message is returned (not shown above) that indicates the attempt to do so was aborted due to the file not being open.

This code shows checks that are often skipped when code is developed on a project by project basis where the philosophy "Get it done" overrides "Do it right". LVOOP lets us easily develop new classes that inherit the robust implementation of their parents.

Building on Solid Ground

Figure 8 shows the class hierarchy for the second project in which the File class was used and enhanced with new child classes. Careful comparison of the File class hierarchies shown in Figure 6 and Figure 8 shows that four new child classes have been added and the application specific class "MVAS" is not used in this application.

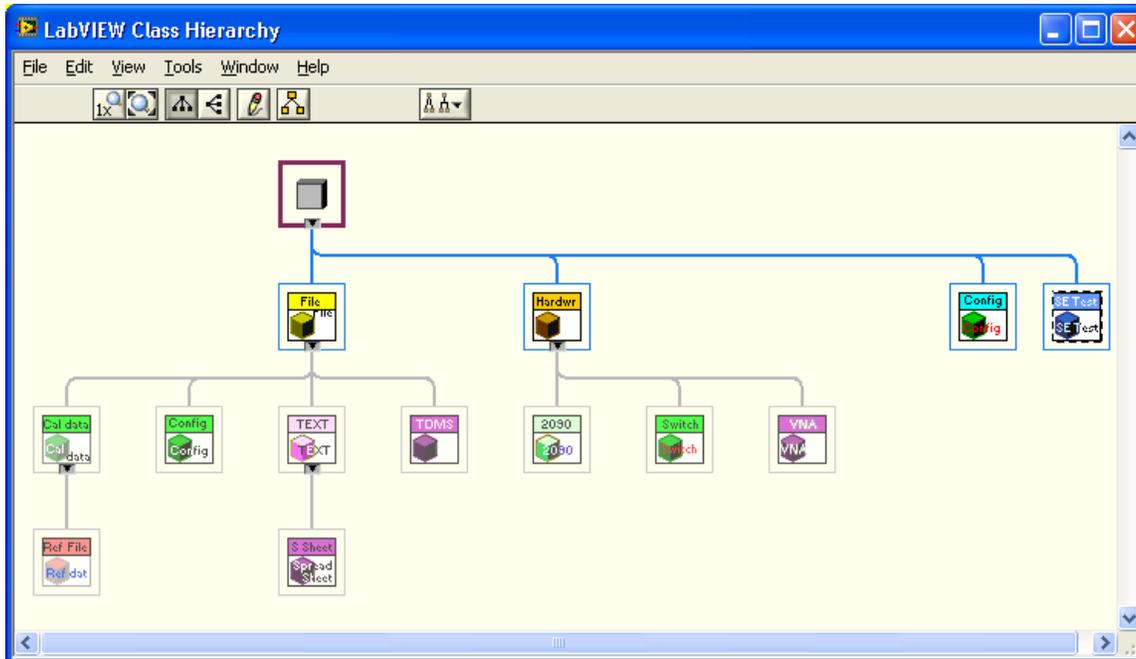


Figure 8. LVOOP Class Library Stage 2 Building on a Solid Foundation.

The class "Cal Data" and "Ref file" are application specific classes that utilize the functions in File but implement methods to provide functionality required in the application to interact with proprietary file structures. Since both of the file structures used the same formatting, the "ref File" class only required a minimal set of methods and relies on parent methods to do common tasks.

The two new classes "Text" and "Spreadsheet" were added to implement the ability to write to a tab delimited spreadsheet file. Since text files are common in many applications, the decision was made to implement the spreadsheet functions by first developing a Text class to allow for other text based files to be added easily at a latter date.

In addition to the new members of the File Hierarchy, Figure 8 shows another hierarchy, derived from the class "Hardware". This class and its children (2090, Switch, and VNA - Vector Network Analyzer) provide the functionality required to access a set of GPIB instruments. A similar approach was used to develop the Hardware hierarchy with an intent to develop powerful easy to use code that can be used again in other projects with little or no changes. The Hardware class utilizes the methods exposed by the Config class to implement methods that allow setting a config file path and file section as well as Init which will read the configuration information for an instance of the Hardware class to perform any initialization tasks associated with that instance. This gives the Hardware class and its descendents the ability to configure themselves and in so doing provides the ability to configure any Hardware class (or its children) instance with little or no involvement by the code utilizing those classes. This loose-coupling allows for greater reuse potential. Classes in Figure 8 that are not discussed herein are application specific.

Hardware is Hardware

Figure 9 shows the third stage of library development. The same File classes are used as well as the Hardware class. This project added classes to the Hardware hierarchy as well as two new hierarchies, Channel and Options. The Options class allows accessing optional settings e.g. Channel settings such as Differential, Single-ended, etc. The "Temp" child of Options is used for options associated with temperature readings. Examples of temperature related options are thermocouple type, CJC source, and scale (F vs. C). The Options and its children are used by the Channel hierarchy.

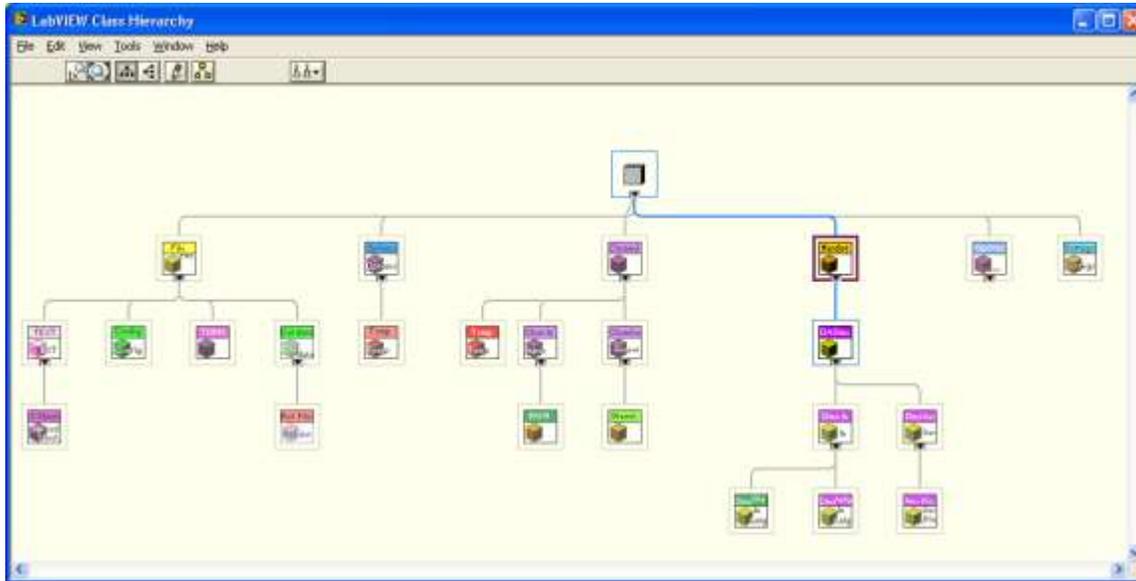


Figure 9. LVOOP Class Library Stage 3 NI Hardware Support Added.

The Channel hierarchy again uses a robust implementation on which its children can build. It allows for channel specific settings to be associated with a channel of a data acquisition device and is fully capable of configuring a DAQ channel. The various channel types shown in the hierarchy support Digital input and out as well as temperature channels. The Channel classes are implemented using dynamic dispatching in many of the children of the DAQmx class and can be enhanced at any later time simply by adding the new class to the folders where the application is located giving a user of the illustrated Class hierarchies the ability add functionality to existing exe's without the need to recompile and rebuild the exe. The Channels class, like the Hardware Class is capable of retrieving its own configuration information idealizing the methods exposed by the Config class.

The Hardware hierarchy shown derives from that same parent as was used in the previous project but the unused hardware classes were omitted and classes capable of interacting with NI DAQmx devices were added. The hierarchy shown illustrates that this version of the hardware hierarchy is capable of accessing Analog input channels as well as Digital input and output.

LVOOP Powers Shine

Figure 10 shows the LVOOP objects used to implement the Diode Test system described previously. As was the case in previous stages, many of the previously developed classes are serving as they did in previous applications and as we observed earlier are enhanced with new children to meet the needs of the application under development. A discussion of some of the new classes follows. Application specific classes are not shown in Figure 10.

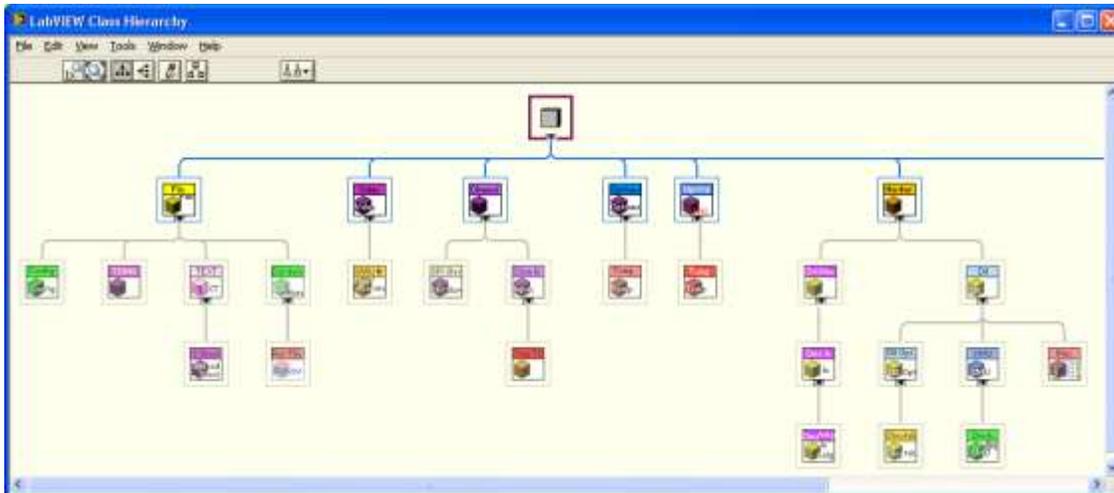


Figure 10. LVOOP Class Library Stage 4 The power of LVOOP Reveals Itself.

The SMU Measurement (“SMU #” in image) child of the parent class “Value” (actually Measurement Value) enhances the class hierarchy by adding the additional parameters associated with a measurement value with those required for a SMU measurement such as “Source”, “Delay Time” and others.

The SMU class builds on the Dll class to expose methods appropriate for controlling and monitoring a Keithley SMU and uses the SMU Measurement class as part of its private data. The SMU class can be used for any application using a compatible SMU. The Diode child class of the SMU class exposes methods suitable for performing diode measurements by utilizing the method exposed by the SMU class. An example of a Diode method that builds on the SMU functions is the “Measure Dark Resistance” (Rd). Although only a single value is associated with Rd the measurement is derived by performing two measurements and using those to derive the Rd value.

The Cheetah class inherits the ability to have optional settings from the Dll_Options class which it uses to set the many hardware settings required to implement SPI protocol. The Cheetah class was implemented to operate a Cheetah SPI interface (made by Total Phase) capable of operating two SPI ports. The SPI port’s unique nature (variable word length) prompted the addition of the SPI Slave class (under Channel) to function correctly.

The Mux class implements the custom built multiplexer used by the application to route the connections for each of the diodes inside the temperature chamber to one of the available SMUs prior to a measurement. The custom hardware was controlled via either one or two SPI interfaces. Methods were implemented to control the multiplexers for example “Select Device”, “Verify_Selection” as well as standard functions found in the Hardware class like “Init”.

The Init method of the Mux class is of particular interest because it illustrates some of the power of LVOOP reentrancy and dynamic dispatching. The Mux private data contains one or more instances of the Class Cheetah. When the Mux “Init” method is invoked it calls the Dll Init method and then invokes the Init method for the Cheetah which in turn invokes the Init method for the DLL_Option class and it in turn invokes the Dll Init method. So in the process of initializing the Mux class the Dll Init method is invoked multiple times to initialize two different classes.

Summary

The progressive growth and enhancement of several base classes was implemented over several distinct projects. By using LVOOP, class hierarchies that are initially inadequate for a new project can be enhanced incrementally without negatively impacting prior projects while positively impacting current and future projects.