Using Fluke MET/CAL® to Implement a Flexible Measurement Driver Model with Expanded Measurement Uncertainties, Error Checking, and Standard Flexibility

Presented by

Michael Schwartz

Cal Lab Solutions/Cal Lab Magazine

Introduction

Software Is An Investment

- If you develop it in house, it costs man hours.
- If you purchase it, it costs dollars.

Software Has a Life Span

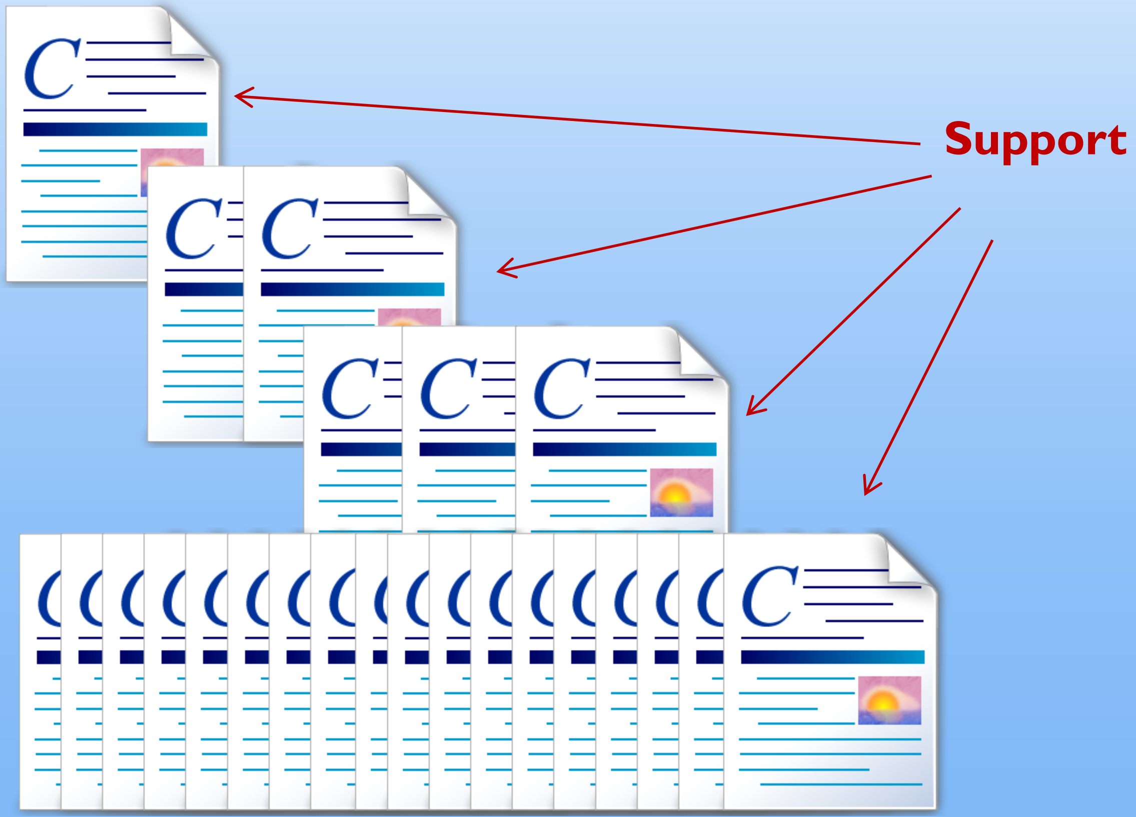
- At some point, it will have to be re-written (re-factored).
- Update \ Change Standards
- Comply with New Regulatory Requirements
- Add New Features

Programming Fundamentals

- Every line of code you write is a line of code you have to support & debug.
- So, we need to do more with less (Code).
- “Better, Cheaper, Faster”

The Problem

- First we write ONE procedure.
- Then we copy that procedure & create TWO.
- Then we copy one of them & create THREE.
- Pretty soon we have ONE HUNDRED or more.
- Now we need to change one thing in more than 100 procedures.
- And we realize we are spending more time supporting our procedures than writing them.

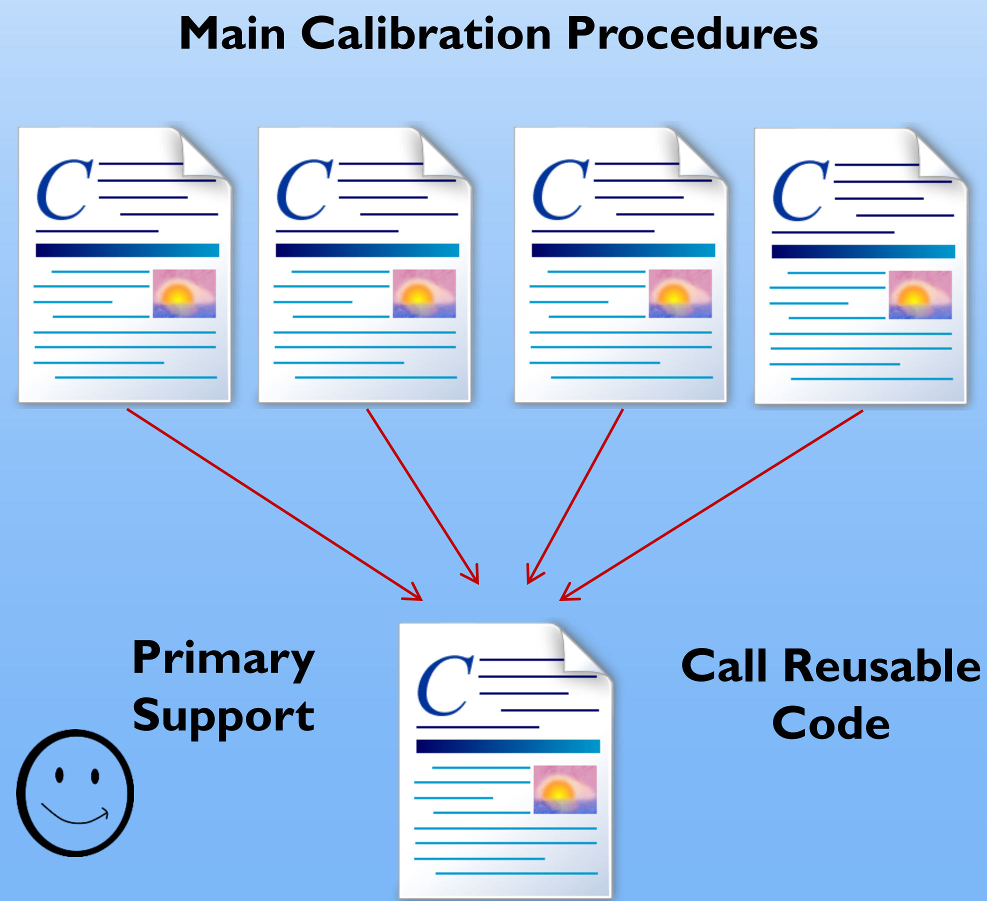


The Solution --- Reusable Code

To work "Better, Cheaper, Faster," we need less code.

To do that, we need reuse of code.

To do that, we need a modular design.



In This MET/CAL Sample

What is Object Orientated Programming?

Abstract Class Pattern – Provides an interface for creating families of related or dependent objects without specifying their concrete class or implementation*.

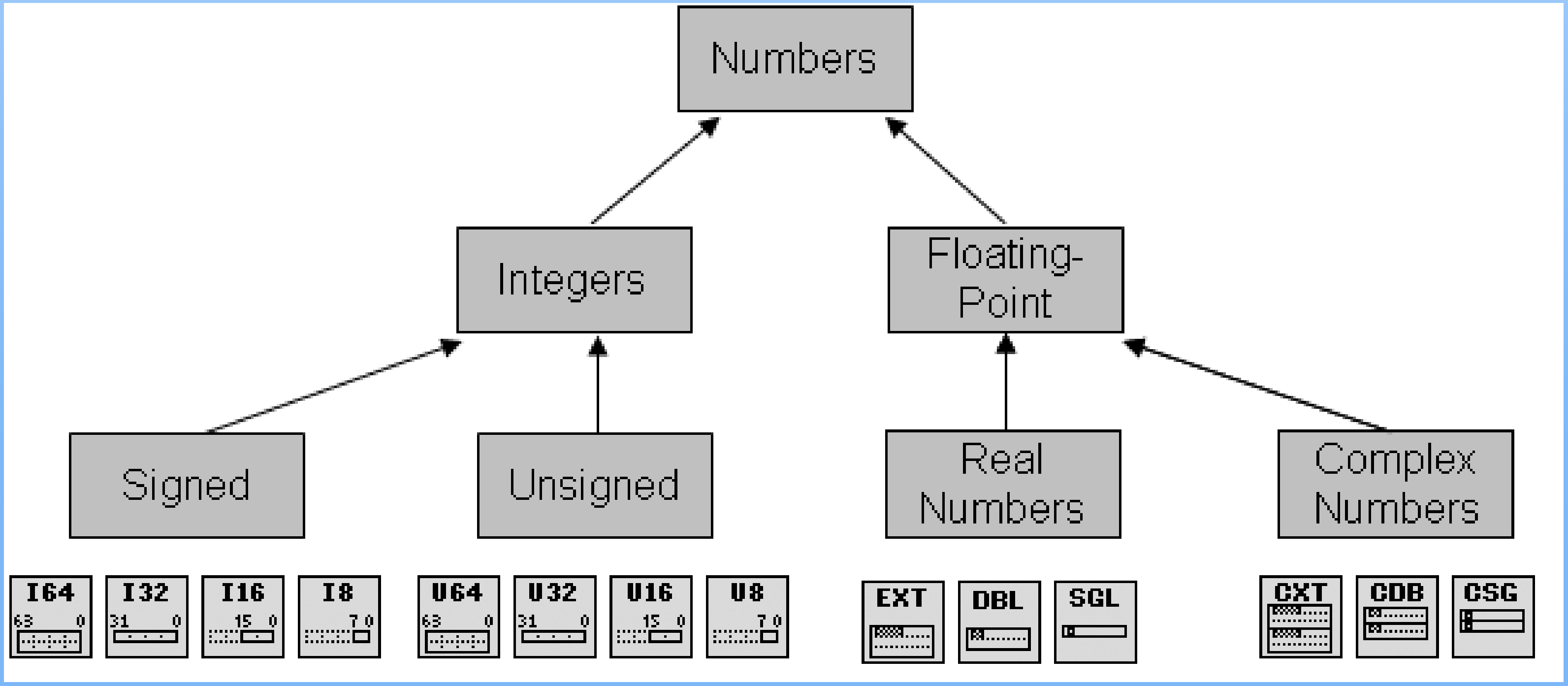
How the Abstract Class Pattern solves the standard flexibility problems

How We Implement a Hybrid Abstract Factory Pattern in MET/CAL® (a non-OOP language).

* E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns, p. 87, 1995.

The Object Oriented Programming Paradigm

OOP allows the programmer to create a simple abstract top level layer of code that interfaces with the lower levels (i.e. objects) which become more specific handling the exact details.



Example MET/CAL® Procedure

In this example, we demonstrate how to implement an Abstract Class Factory pattern using MET/CAL.

```
#=====  Sample Test Point 1 ============
  4.001  5520         1.0000V                                    S   2W
  4.002  TARGET       -m
  4.003  IEEE         Read?[i]
  4.004  MEMCX        V              0.0001U
```

Example 1. Sample Test Point 1.

We do not want to limit this procedure to just a Fluke 5520A. We want to be able to use any DC Voltage source--anyone that supports **Source.Volts.DC.**

```
# #===== Sample Test Point 2 ============
  4.001  MATH          S[30]="Source.Volts.DC Volts= 1"
  4.002  CALL          My Config Sub
  4.003  MATH          L[1]=Fld(S[31],2,"VoltsUnc=")
  4.004  MATH          MEM=Fld(S[31],2,"Volts=")
  4.005  TSET          UUT_Res= 0.0001
  4.006  ACC           V        L1U
  4.007  TARGET        -m
  4.008  IEEE          Read?[i]
  4.009  MEMCX         V              0.0001U
```

Example 2. Sample Test Point 2.

- In our objective model we remove the 5520 FSC and replace it with an "iSource.Volts.DC" with a Set Parameter value of "Volts= 1" (Line 4.001).
- Then we call the "My Config Sub" (our Abstract Class Pattern);

  This will select the specific Standard that will generate 1V DC.
- Lines 4.003 – 4.006 perform additional steps the 5520 FCS did for us:

  1) 4.003 Get the Measurement Uncertainty

  2) 4.004 Get the Set Value for 1Volt DC

  3) 4.005 Set the Resolution of the Test

  4) 4.006 Set up the ACC in place of the 5520 FSC

```
#====== My Config Sub Sample Code ==========================================
  4.001  LABEL        VoltsDC
  4.002  JMPL         VoltsDC_Conn      Find(S[30],"Connect",1)>0
  4.003  JMPL         VoltsDC_Source    Find(S[30],"Source.Volts.DC",1)>0
  4.004  DISP         Error Calling Sub
  4.005  END
#========================================
  4.006  LABEL        VoltsDC_Conn
  4.007  DISP         Connect the Fluke 5520 to the UUT as Follows;
  4.007  DISP         [32]  NORMAL HI <-----> V
  4.007  DISP         [32]  NORMAL LO <-----> COM
  4.008  END
#========================================
  4.009  LABEL        VoltsDC_Source
  4.010  CALL         CLSD-Source.Volts.DC                     (5520A Normal)
  4.011  END
```

Example 3. My Config Sub Sample Code.

In the "My Config Sub"

4.001 This section handles all calls to the Volt.DC

Both Connection and Driver Calls.

4.006 The main does not know the Specific Standard

So we handle the Specific Connection Message.

4.009 Now we can call any Source.volts.DC driver

In this case, we are still using a Fluke 5520 Normal Output.

Our CLSD-Driver Model

You must have a well-documented programming standard!

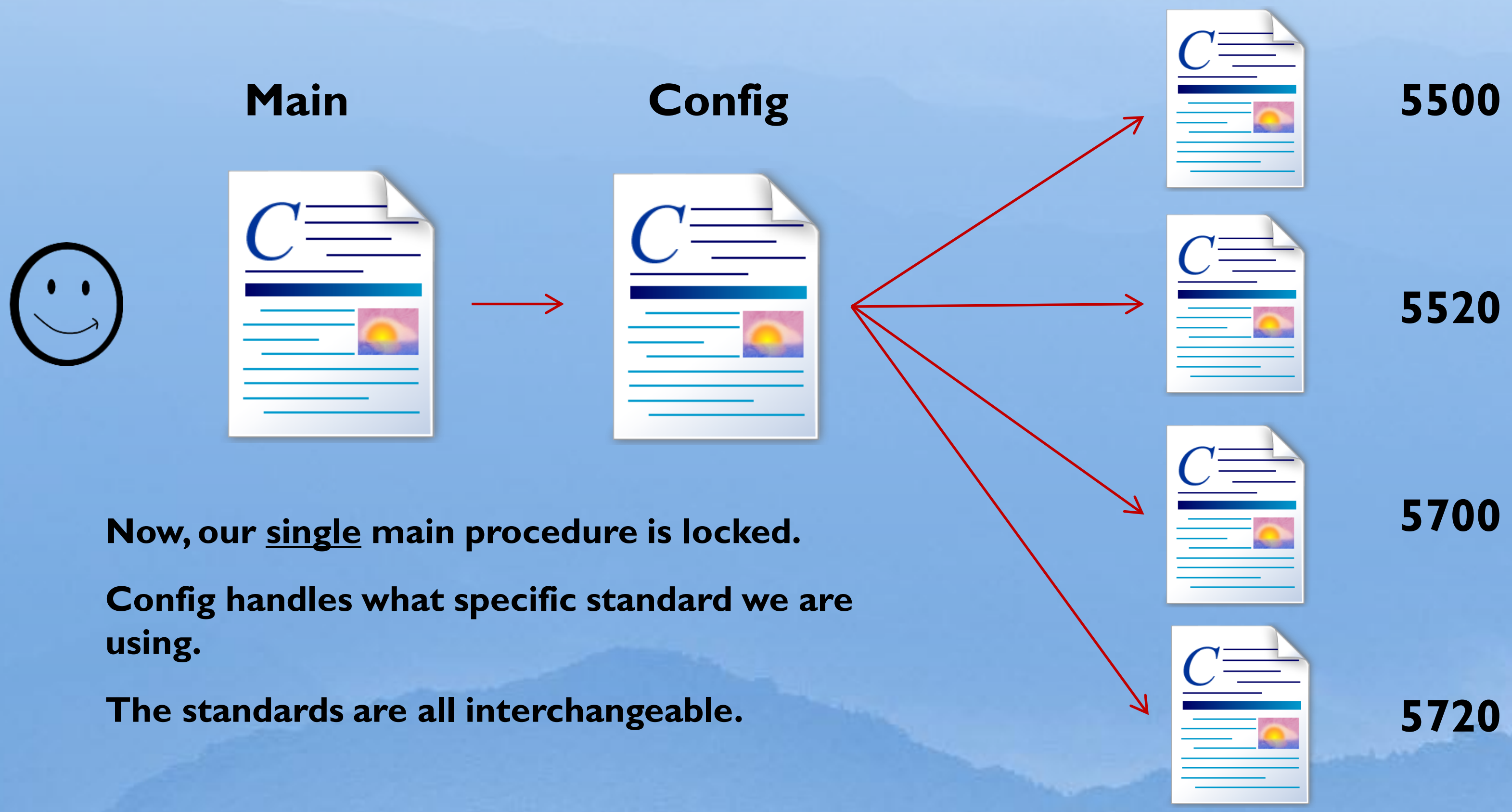
Every driver must support the following commands:

- **Name** – Returns the Name of the STD and Connection Point
- **Setup** – Performs any required Setup/Configuration Tasks
- **Reset** – Resets the Standard(s)
- **OutputOff** – Turns the Output Off (Implemented on Sources Only)
- **<Metrology Method>** – Source.Volts.DC in this example

Now We Make a Series of Interchangeable Drivers

Driver	Standard
CLSD-Source.Volts.DC (5500A Normal)	Fluke 5500 Volts Connection Post
CLSD-Source.Volts.DC (5570A Normal)	Fluke 5520 Volts Connection Post
CLSD-Source.Volts.DC (5700A Normal)	Fluke 5700 Volts Connection Post
CLSD-Source.Volts.DC (5720A Normal)	Fluke 5720 Volts Connection Post

Table 1. Examples of interchangeable drivers with this paper.



Now, our <u>single</u> main procedure is locked.

Config handles what specific standard we are using.

The standards are all interchangeable.

Error Checking in the Drivers

```
#====== Source.Volts.DC 5520 Normal ====================================
 1.024  LABEL        Source.Volts.DC
# Get the Voltage
  1.025  MATH         L[1]=Fld(S[30],2,"Volts=")

# Error Check the Values
  1.026  IF           Abs(L[1]>1000)
  1.027  DISP         Error [L1] Volts is Out of the Fluke 5500's Range
  1.028  END
  1.029  ENDIF

# Setup The Standard
  1.030  MATH         MEM=L[1]
  1.031  5500         V                                                  S  2W
```

Example 4. Source.Volts.DC 5520 Normal.

Measurement Uncertainties & Additional Contributors

```
#====== Source.Volts.DC 5520 Normal ======================================
# Calculate the Uncertainties
  1.028  MATH          L[11]=ACCV("Fluke 5520A","Volts", MEM)
  1.029  MATH          S[31]=       " Value= " & MEM
  1.030  MATH          S[31]=S[31]& " Unc= " & L[11]
  1.031  MATH          S[31]=S[31]& " Volts= " & MEM
  1.032  MATH          S[31]=S[31]& " VoltsUnc= " &L[11]
# Standard Resolution
  1.033  IF            L[1]<330e-3
  1.034  MATH          L[31] = .1e-6
  1.035  ELSEIF        L[1]<3.30
  1.036  MATH          L[31] = 1e-6
  1.037  ELSEIF        L[1]<33.0
  1.038  MATH          L[31] = 10e-6
  1.039  ELSEIF        L[1]<330
  1.040  MATH          L[31] = 100e-6
  1.041  ELSE
  1.042  MATH          L[31] = 1000e-6
  1.043  ENDIF
  1.044  MATH          L[31]=L[31]/2/Sqrt(3)
  1.045  TSET          U7 = [L31]
# Standard Traceability (Assuming 4 to 1 or Better)
  1.046  MATH          L[31]=L[11]*.25
  1.047  TSET          U7 = [L31]
```

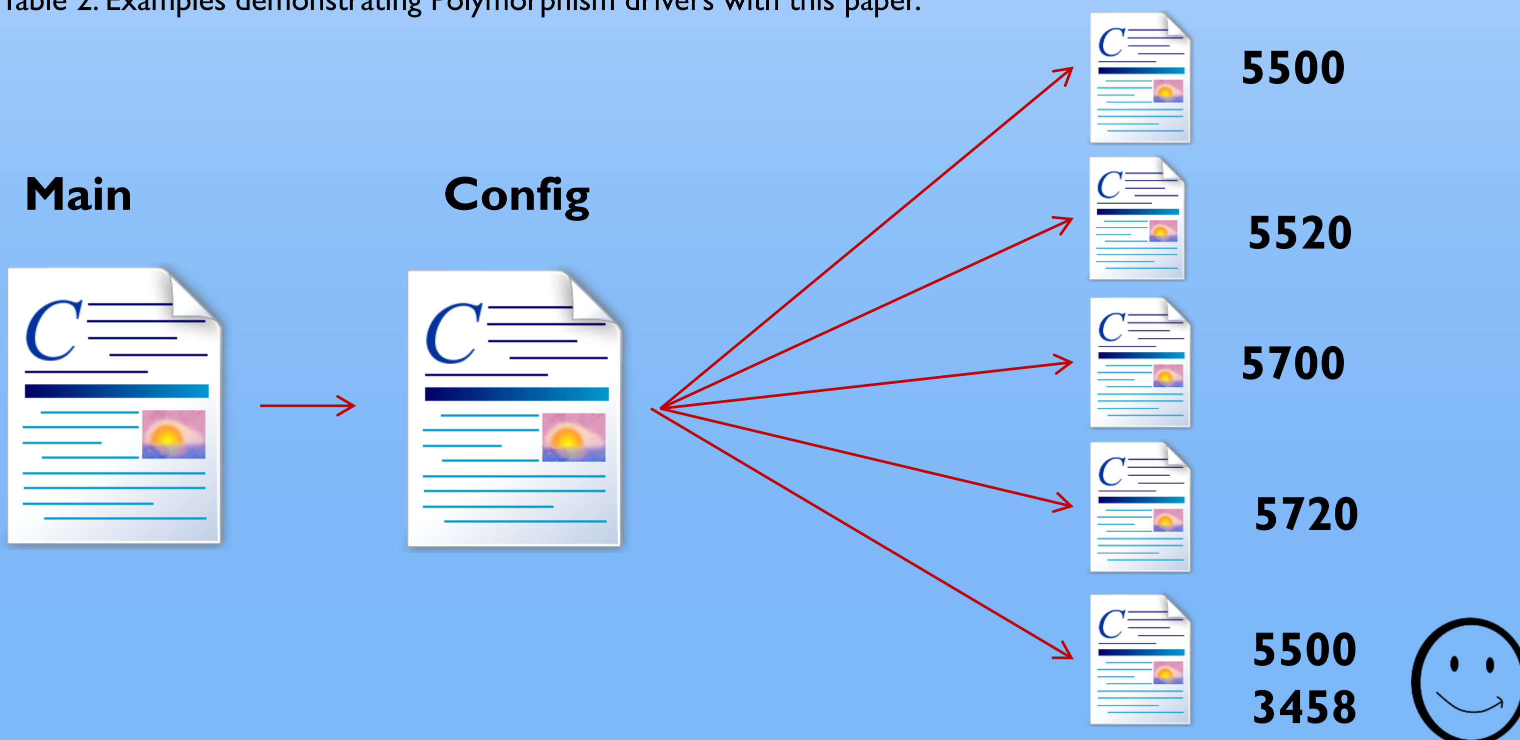
Example 4. Source.Volts.DC 5520 Normal.

Hello Polymorphism

The Problem: Fluke 5500 is not accurate enough!

Driver	Standard
CLSD-Source.Volts.DC (5500 Normal & 3458A)	Fluke 5500 Volts Connection Post monitored and corrected with an HP/Agilent 3458A.

Table 2. Examples demonstrating Polymorphism drivers with this paper.



```
# Check the Input Terminals
#========================================
  1.031  LABEL        SetInput
  1.032  IEEE         [@3458][Term LF]TERM?[I]
  1.033  IF           MEM!=1
  1.034  DISP         Set the 3458A Front\Rear Input to Front
  1.035  JMPL         SetInput
  1.036  ENDIF
# Setup The Standard
  1.037  MATH         MEM=L[1]
  1.038  5500         V                                                  S  2W

# Settle the Reading
  1.039  IEEE         [@3458]FUNC DCV
  1.040  IEEE         [@3458]NDIG 8
  1.041  IEEE         [@3458]NPLC 200
  1.042  IEEE         [@3458][Term LF][T0][i]
  1.043  IEEE         [@3458][Term LF][T0][i]
  1.044  IEEE         [Term OFF]

# Calculate the Uncertainties
#==================================
  1.045  MATH         L[11]=ACCV("HP 3458A","Volts E", MEM)
  1.046  MATH         S[31]=       " Value= " & MEM
  1.047  MATH         S[31]=S[31]& " Unc= " & L[11]
  1.048  MATH         S[31]=S[31]& " Volts= " & L[1]
  1.049  MATH         S[31]=S[31]& " VoltsUnc= " & (L[11]+(MEM-L[1]))
```

Example 5. Source.Volts.DC 3458A 5520 Normal.

Conclusion

Though MET/CAL® is not by design an Object Oriented Programming Language like Microsoft .Net and Java, we can still take full advantage of the architectural principles, design patterns and other fundamentals of OOP to write more robust, innovative and fault tolerant procedures.

A packet of samples can be found online at http://www.callabsolutions.com.

Questions?

Contact Information

Michael Schwartz
Owner/Publisher
Cal Lab Solutions/Cal Lab Magazine
E: Mschwartz@callabsolutions.com
T: (303) 317-6670