SAND2011-3127P

Flexible Tester Framework:
An Object Oriented Application Framework

Technical Overview

1 June 2011

Mark E. Smith
John M. Linebarger
Dept. 02661

NNSA National Nuclear Security Administration

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

- The Flexible Tester Framework (FTF) is an Object Oriented Application Framework written in National Instruments LabVIEW. "A framework is a reusable, ``semi-complete" application that can be specialized to produce custom applications " (http://www.cse.wustl.edu/~schmidt/CACM-frameworks.html). This framework is directed specifically to automated test and measurement and contains immutable base classes (designed for inheritance) with both method and interface definitions.
- The framework includes base classes for measurement instruments and stimulus sources, test configuration, data persistence, test monitoring, evaluation of results, report creation, and user interfaces. An error class is included that automates error handling and logging for classes that inherit from the framework.
- The capability of the framework is further extended by providing a class that integrates text-based scripting using the LuaVIEW framework (http://www.citengineering.com/LuaVIEW/index.html), an integration of LabVIEW and the Lua scripting language (http://www.lua.org/).
- Applications deployed with scripting enabled can be programmed by the user to customize the deployed application to specific test requirements.

Sandia National Laboratories

What?

- An Object Oriented Application Framework
  - "A framework is a reusable, ``semi-complete'' application that can be specialized to produce custom applications “
    - http://www.cse.wustl.edu/~schmidt/CACM-frameworks.html
  - The application domain is automated test and measurement
  - Web application frameworks are very common
    - Google Web Toolkit
    - Struts
    - Ruby on Rails
  - Frameworks address code re-use from the “Top Down”
    - Existing low-level re-use code (class libraries, llb’s (LabVIEW), toolkits) are still useful for framework development

Why?

- We need automated test for many of the products that 2660 delivers and supports
  - Manual testing is too time-intensive, expensive, and prone to error due to
    - Complicated interface requirements
    - Many repeated tests under many environments for qualification and acceptance
- Developing automated testers can be time-intensive, expensive, and prone to error, as well
  - Poorly defined requirements or constantly changing requirements
  - “Re-inventing the wheel”
    - Currently three and possibly a fourth automated test projects underway – don’t have time or resources to create each independently
  - Lack of software testing and configuration control

How?

- Using an object oriented application specific framework
  - Contains all the common automated test software components (prevents omission of the component even if the requirements have omitted it)
  - Common code gets directly re-used (no “re-inventing the wheel” for each new project)
  - The framework is (or will be)
    - Documented (no need to create new documentation for the common components)
    - Configuration controlled (SVN and IMS)
    - Tested (unit tests and documented results for framework components)
    - Immutable – specific implementations use the framework but do not modify it in any way

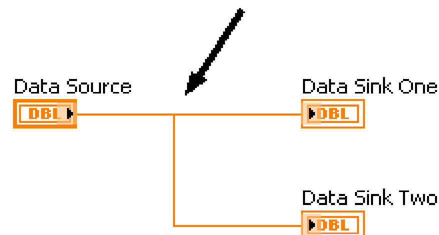
- Frameworks can be Black Box or White Box
  - White box requires inserting user code into the framework and requires extensive knowledge of the framework details
  - Black Box abstracts the framework details – users implement applications without modifying the framework in any way
  - The Flexible Tester Framework is a Black Box framework

Using an OO Black Box Framework

- Black Box OO Frameworks contain base, abstract, and concrete classes
  - Instantiation of an application is by
    - Composition – using framework classes to compose new, specialized classes
    - Inheritance (subclassing) – using framework classes as superclasses (parents) of specialized subclasses (children)
      - Subclasses can override methods of the parent class to provide specialized behavior

LabVIEW Object Oriented Programming

- LabVIEW is a graphical computer language by National Instruments
  - Dataflow paradigm
    - Data Sources (controls)
    - Data Sinks (Indicators)
    - No variables in the conventional sense – the data is always "on the wire"
    - Data is copied to multiple sinks (pass by value) rather than passing a reference (or handle or pointer) to the data



Sandia National Laboratories

LabVIEW Object Oriented Programming (2)

- LVOOP is also pass by value – the object is on the wire
  - Works great in a single thread or multiple threads in a single diagram
  - Not so good for multiple threads that need to each access and operate on the object
  - Doesn't work for OO scripting, where there are no wires
- So, we need to pass objects by reference
  - Use a LabVIEW "Action Engine" (an intelligent LV2 style global) – a container that allows us the create, access, modify, and release objects anywhere in the application instance
    - This may move to a queue-based approach if performance (access times) proves inadequate with the Action Engine – so far, it has not
    - http://forums.ni.com/ni/board/message?board.id=170&message.id=264865&query.id=72398#M264865

LabVIEW Object Oriented Programming (3)

- Supports a simple inheritance model
  - No multiple inheritance
  - No interfaces
  - No enforced abstract classes
  - All overridden methods must share the same connector pane as the superclass method that is overridden (so no overloading, all function prototypes must be identical)
    - This is due to LV override methods being a specialization of LV polymorphic VIs
- Strict encapsulation
  - All class data members are private - no public or protected fields
  - All class data must use accessors (“get” and “set” methods, to borrow from C# terminology)

Framework Architecture

- Common automated test components are represented as base classes (intended for use as superclasses)
  - Instrument
    - Stimulus
    - Measurement
  - Configuration
  - Data Persistence
  - Evaluation
  - Monitor
  - Reporting
  - User Interface
- Concrete Classes – intended to be used without modification or inheritance
  - Error Logging
  - Scripting

Sandia National Laboratories

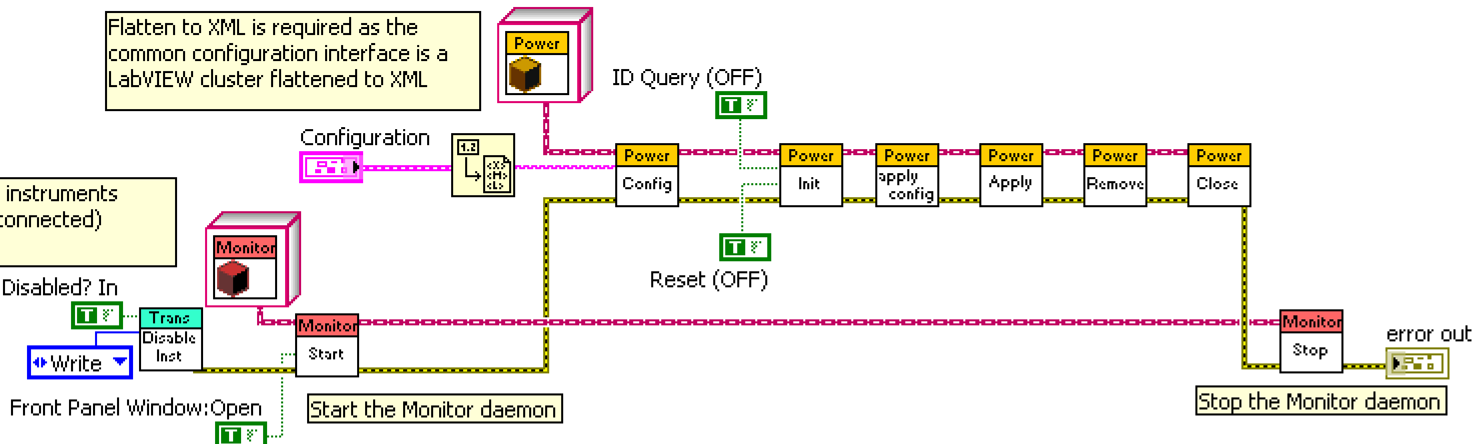
Instrument Class

- The Instrument class is the superclass of the Stimulus and Measurement classes
  - Instrument has methods common to all instrument control that are interface methods (no code implemented in the class)
    - Initialize
    - Configure
    - ApplyConfiguration
    - Close
  - In addition, there are methods that are implemented and designed to be called directly or overridden in the subclasses
    - toXML (similar to a “toString” method)
    - SendStatus (sends a status message to the monitor daemon)
  - Lastly, there are
    - Templates to use for creating override classes
    - The InstanceEngine method – the Action Engine for “By Reference” instantiation

- The Stimulus class specializes the Instrument class by adding interface definitions (methods) for
  - Apply
  - Remove
- Common Calling sequence for a stimulus class object is
  - New (ActionEngine) for By Ref or Class constant for By Value
  - Configure
  - Initialize
  - ApplyConfiguration
  - Apply
  - Remove
  - Close
- What does this look like in LV Code?

Sandia National Laboratories

Stimulus Example



Scripting

- The framework has been designed to support scripting using the LuaVIEW toolkit
  - LuaVIEW is a product of CIT Engineering
    - http://www.citengineering.com/pagesEN/products/LuaVIEW.aspx
  - Uses the Lua scripting language, a “powerful, fast, lightweight, embeddable scripting language “
    - http://www.lua.org/about.html
  - Makes any VI callable from a text script
  - Allows distribution of user-programmable applications (no LabVIEW development environment required)

Stimulus Example (as Lua script)

C:\MA231 Tester\Software\trunk\Scripting\Scripts\BrownBag Example One.lua - SciTE

File Edit Search View Tools Options Language Buffers Debug Help

```
---#description "Test Script"
register.private_dir([[C:\MA231 Tester\Software\trunk\Scripting\Scriptable VIs]])

print([[
start of script
]])

-- disable instruments for testing
disable_instruments(true,"write")

-- start the monitor daemon
Monitor.start()

-- create the PowerSupply object
ps1=PowerSupply.new("ps1")

print([[
configuring power supply..............
]])

-- configure the PowerSupply object
ps1:config([[
<Cluster>
  <Name>PowerSupply</Name>
  <NumElts>3</NumElts>
  <String>
    <Name>Name</Name>
    <Val>My PS_Test</Val>
  </String>
  <Refnum>
    <Name>VISA resource name</Name>
    <RefKind>VISA</RefKind>
    <Val>COM22</Val>
  </Refnum>
  <DBL>
    <Name>Set Voltage (V)</Name>
    <Val>12.34000000000000</Val>
  </DBL>
</Cluster>
]])
```

Ln: 39 Col: 11 Sel: 0 | Saved: 6/23/2009 8:37:34 AM | [INS] [CR+LF] |

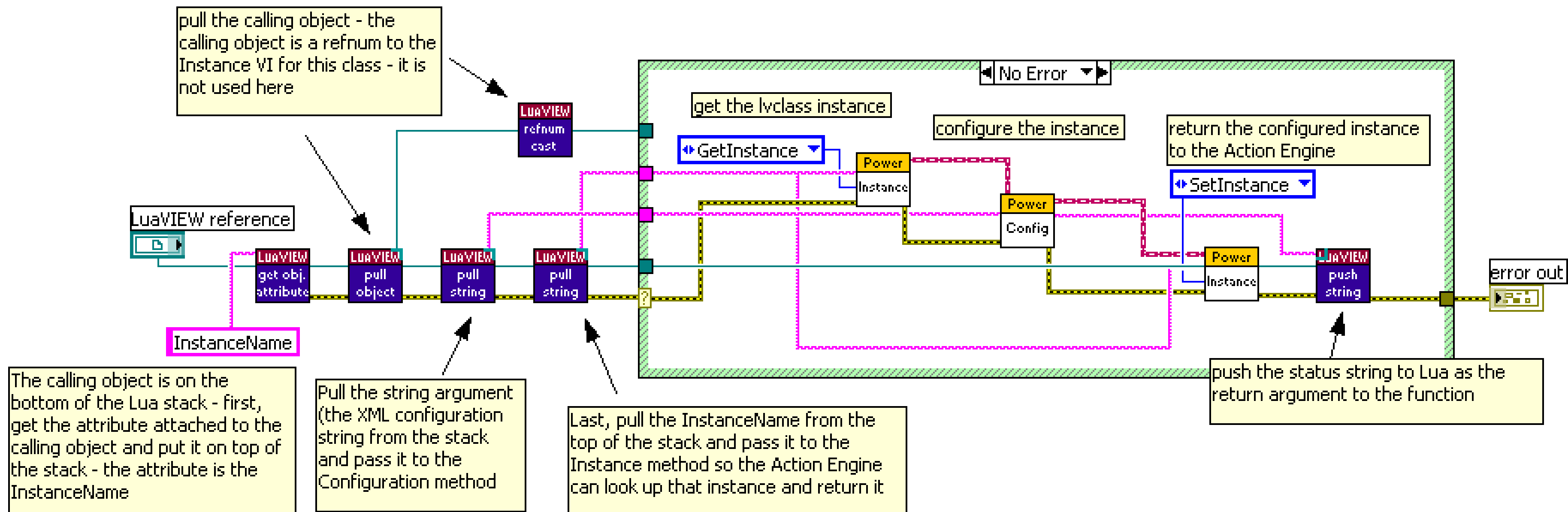
Sandia National Laboratories

Object Oriented Scripting Syntax

- Lua supports an OO Scripting syntax that maps to the LabVIEW OO Framework
  - Not true OO, since Lua doesn't support inheritance
- Syntax is "object:method(args….)
- For the PowerSupply, we first create the object ps1 and then configure it
  - ps1=PowerSupply.new("ps1") – note the "dot" notation here since PowerSupply is an identifier and not an object
  - ps1:config([[configuration xml string]]) – here we use the colon to indicate an operation on the calling object

Sandia National Laboratories

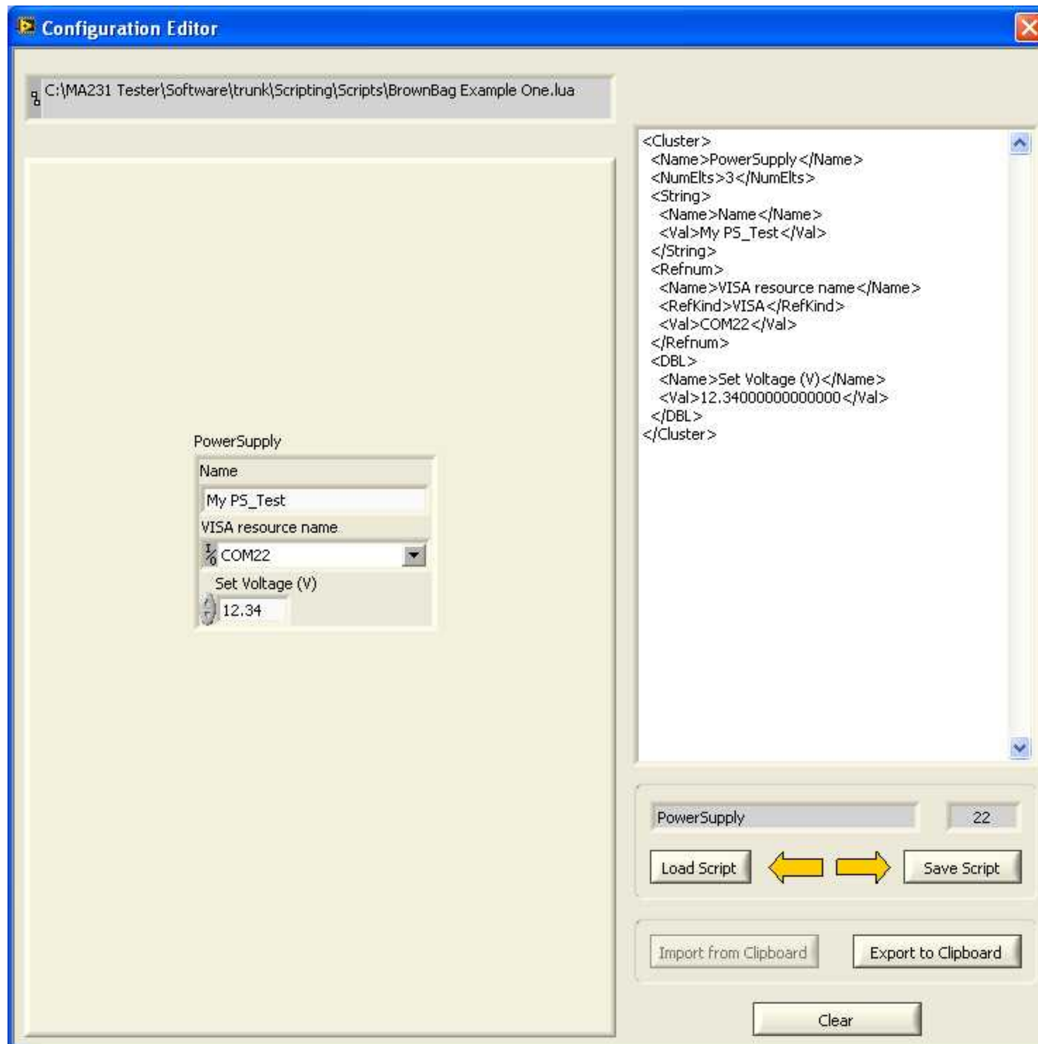
Lua Callable OO LabVIEW Function



Sandia National Laboratories

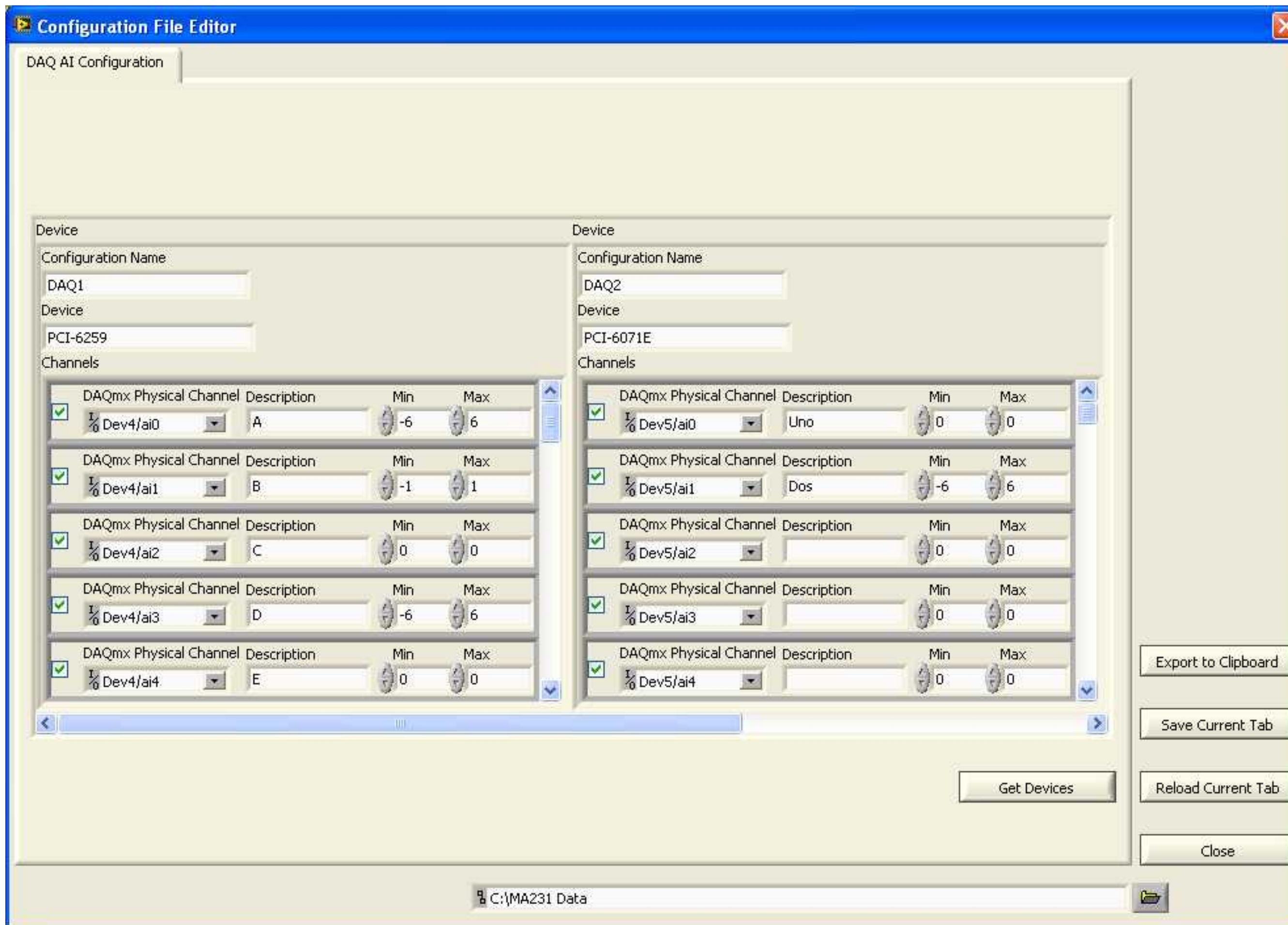
- All classes in the framework require configuration input as strings for two reasons
  - 1) Common interface – the overridden configuration methods require a common interface (argument list)
  - 2) Strings are easily embedded and read (sort of) in text based scripts – this allows a script to be a complete definition of a given test, including all instrument and tester configurations
    - Many tasks can be sub-scripted, as the Lua script can call other Lua scripts
- To support configurations as XML strings, the framework has a Configuration class with two template methods
  - File Editor – supports XML configurations saved in config files and called by “file path/configuration name” at run time
  - Script Editor – supports editing Lua scripts with embedded XML configuration strings

Script Editor



- **Script Editor parses embedded XML Configurations for editing**
- **Exports/Imports configurations to/from clipboard**

File Editor



- **Create and edit named configurations that can be called from file**

Sandia National Laboratories

- Frameworks need to be
  - Flexible enough to meet the developer’s needs
    - Make it infinitely flexible and you’ve just redistributed the programming environment – doesn’t help much
  - Rigid enough to enforce common architecture
    - The first time the developer finds “it just won’t support that” is the time the developer stops using it
  - This framework tries to address that problem by having a rigid structure (to enforce commonality) that is infinitely extensible (to allow any capability)
- Framework deployments are large!
  - This framework will deploy all of the classes referenced in their entirety
  - Automated build scripts and installers make building and installing deployed apps relatively easy, but the executable will include many support files