STS Global Test Data Management

NI STS Test Program Development Best Practice

Matthew Roberts

NI Senior Field Applications Engineer

Glossary & Resources

Abbreviated Terminology

- TS – TestStand
- TSM – TestStand Semiconductor Module
- STS – Semiconductor Test System
- LV – LabVIEW
- Lot – The queue of devices under test that are ran sequentially or as a batch across multiple test sites during a production run.
- ProcessSetup – Callback in TestStand used to define steps that execute when a Lot starts

Resources/Assumed-knowledge

- Test Program Development with STS and LabVIEW Course - https://www.ni.com/en-gb/shop/services/education-services/customer-education-courses/test-program-development-with-sts-and-labview.html
- LabVIEW Object Oriented - https://www.ni.com/docs/en-US/bundle/labview/page/lvconcepts/front_oolv.html
- LabVIEW Property Nodes - https://www.ni.com/docs/en-US/bundle/labview/page/glang/property_node.html
- TSM Data Management - https://www.ni.com/docs/en-US/bundle/teststand-2020-semiconductor-module/page/tssemiconductor/tsmdatamanagement.html
- TSM Sharing Data Between Code Modules - https://www.ni.com/docs/en-US/bundle/teststand-2020-semiconductor-module/page/tssemiconductor/tsm_sharingdatabetweencodemodules.html

Overview

Problem Statement

Status Quo

- When developing an NI STS Test Program certain data will typically be stored in one code module and retrieved later in another code module.
- Such data can and should be stored in the SemiconductorModuleContext (TSM Context) using an ID string and can either be stored as a single data value available globally to all sites or as a data value per site.
- When developing with LabVIEW (LV), this is done using the Set Global Data VI and the Get Global Data VI when storing and retrieving data shared by all sites. Or using the Set Site Data VI and the Get Site Data VI to storing and retrieving site specific data.
- Refer to the following NI STS Help Documentation: https://www.ni.com/docs/en-US/bundle/teststand-2020-semiconductor-module/page/tssemiconductor/tsm_sharingdatabetweencodemodules.html

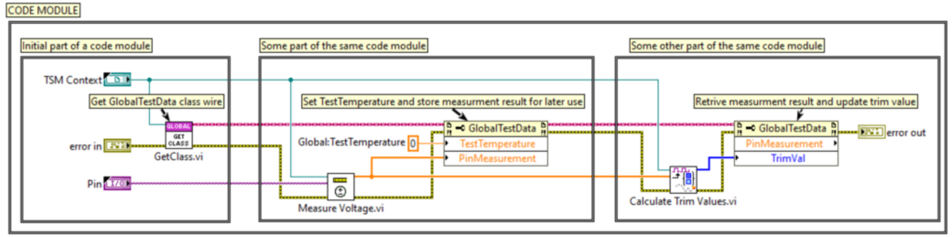
Problem Statement

- The LV API functions for storing and retrieving data within the TSM Context…
  - rely on ID strings that test developers can easily mistype and must ensure are consistent across code modules. In practice, this can be hard to manage and debug.
  - are type agnostic and require the test developer to micromanage when retrieving the data. In practice, it can be difficult to keep track of what data type is associated with the string ID being retrieved, and this casting from variant data to the desired data type adds clutter to the block diagram, especially in the per-site use case.

Important Note

- Sharing data between modules → While there are other, LabVIEW native mechanicians for storing the same type of data, none provide the convivence and robustness of using the TSM Context when it comes to managing site data. All other mechanisms require the test developer to micromanage the order of the site data, and whether certain sites are inactive (site goes down mid lot) or not being called from within the code module's context (independent code module execution per site or group of sites).

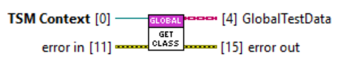
Overview of the TSM Data Access Concept


CODE MODULE
Initial part of a code module
Some part of the same code module
Some other part of the same code module
TSM Context
Get GlobalTestData class wire
GLOBAL
GET CLASS
GetClass.vi
error in
Pin
Set TestTemperature and store measurment result for later use
Global:TestTemperature
0
GlobalTestData
TestTemperature
PinMeasurement
Measure Voltage.vi
Retrive measurment result and update trim value
GlobalTestData
PinMeasurement
TrimVal
error out
Calculate Trim Values.vi

TSM Data Access Concept

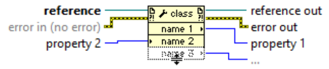
- Utilizes LabVIEW Class Behavior for Convenient Test Program Data Storage, Access, & Management
- Compares of flat LabVIEW class with a pre-defined set of properties
  - Read & Write accessor functions created for each individual data item being stored
    - Read & Write accessor strictly defines the data item's data type (i.e. i32 vs String)
    - The data item can be any LabVIEW native data type, including enums and custom type-definitions*
  - Uses Semiconductor Module Context & corresponding API as the means for memory storage
    - Inherently Global & Per-Site Capable
  - Extracts data types & sites management away from the users
- Only two LabVIEW nodes used during test program development
  - Get Class.vi 🡐 Encapsulates the TSM context into an LV class wire that can be accessed via property node
  - Property Nodes 🡐 Used to invoke pre-defined read/write accessor functions for a given data item defined in the class
- Scalable to add as many data items as needed throughout the development process

GlobalTestData.lvclass:GetClass.vi



This function should be called at the start of test code module. It will construct the GlobalTestData class wire by referencing the approprate instance of the Semiconductor Module Context.

Property Node



Gets (reads) and/or sets (writes) properties of a reference. Use the property node to get or set properties and methods on local or remote application instances, VIs, and objects. You also can use the Property Node to access the private data of a LabVIEW class.

Important Notes

1. Where is the actual data being stored?
   - Data is stored by reference in TSM's memory space (dynamic memory within TSM's .NET assembly), it is not actually being stored in LabVIEW's memory space.
   - The class wire is essentially a wrapped reference to the TSM context, when you operate on a data item via property node the property node will call in the TSM context's memory space to access the data item.
   - Although you are passing the data by value on the LabVIEW block diagram, the accessing of the data happens by reference… that is the reference to the TSM context.
2. Property Node elements are executed on sequentially, which means whichever order you read or write from, top-->down, would be the order in which the data would be manipulated.
   - This inherently prevents any race conditions within a certain VI if there is only ever one Property Node accessing the class at a given time.
   - Since the wire is passing around and operating on the data by reference behind the scenes, you should still incorporate data flow where possible to ensure two property nodes are not being used at the same time, where one can just as easily do the job. Avoid using branching the same Class wire to multiple Property Nodes unless you can ensure those Property Nodes are accessing unique data items. Otherwise, access all necessary data items for a particular part of your code module using one Property node at a time.
3. Lifetime of Get Class wire reference: Class wire inherently same context as the TSM context, and therefore this VI must be called in every test code module that intends to access a data item.
   - The intended use case for a given VI is to only ever need to call the "GetClass.vi" function once, to get the class wire and then use the class wire as a reference to access the data via property node and connect the wire to the next instance of property node which access the data.
4. Data Types: Data items can be of any LabVIEW type, including Enums & Type Definitions.
   - *If the data type you want to store is a type def, the .ctl file for the type def should exist within the ...\Code Module\Common\<TSM Context Data Access Class Name>\Public\Typedefs directory.

Pros & Cons of the TSM Data Access Technique

Pros

- Only two LabVIEW nodes are needed during test code module development to store & retrieve data
- Declutters & simplifies the block diagram, especially when storing/retrieving multiple data items
- Abstracts management of site-base data away from the test developer
- Eliminates the need to manage data types during test code module development
  - Properties are strictly typed and do not require the test developer to cast data type when retrieving a data item
- Ensures the storage and retrieval of data is consistent across code modules
  - ID strings are abstracted from the test developer, eliminating the possibility that they can be mistyped.

Cons

- Separate files (read & write) on disk for each data property defined for the test program.
- Test developer must create a data property as needed and update the data type upon its creation.
  - LabVIEW's VI scripting feature helps with this burden.
- Storing/retrieving multiple data items is not optimized when called from same property node, due to how property node items sequentially execute. Test developers must use multiple property nodes to gain parallel execution.
- Because data is being stored in TSM context, there is some amount of overhead when calling into TSM to store and retrieve the data.

Global Test Data Template Files

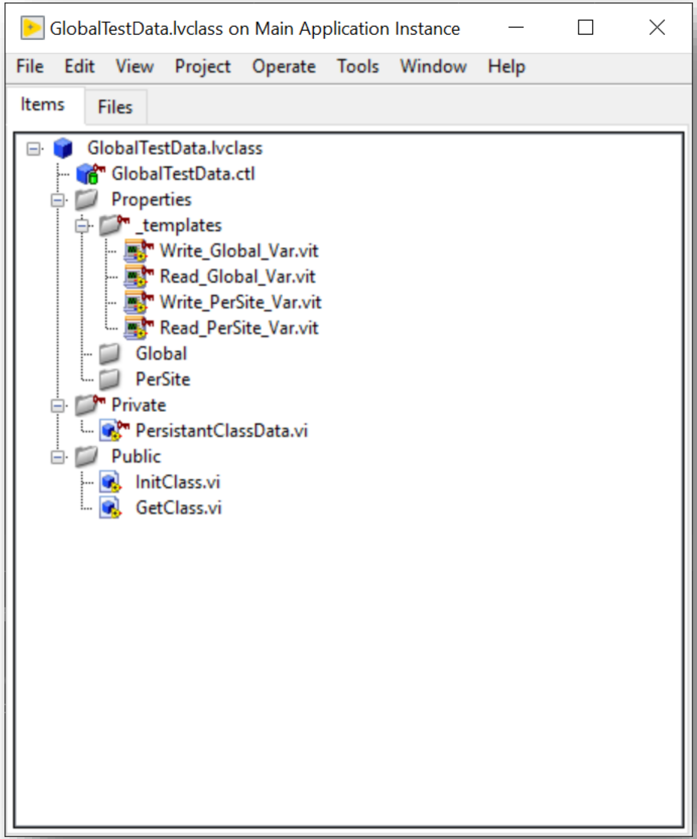
Template files available on GitHub:

https://github.com/Mattjet27/STSGlobalTestDataTemplate

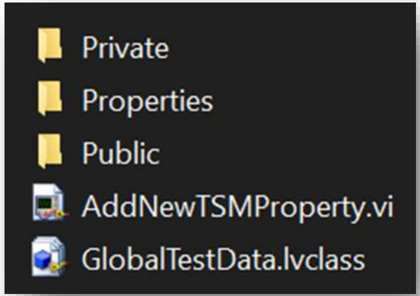
Includes:

- GlobalTestData LabVIEW class
- Property VI Templates - Global & PerSite
- AddNewTSMPropery.vi - scripts new properties and adds them to the GlobalTestData class automatically based on the name and scope specified by front panel controls.

Note: these templates files can but should not be modified.


GlobalTestData.lvclass on Main Application Instance
File Edit View Project Operate Tools Window Help
Items Files

```
GlobalTestData.lvclass
  GlobalTestData.ctl
  Properties
    _templates
      Write_Global_Var.vit
      Read_Global_Var.vit
      Write_PerSite_Var.vit
      Read_PerSite_Var.vit
    Global
    PerSite
  Private
    PersistantClassData.vi
  Public
    InitClass.vi
    GetClass.vi
```



```
Private
Properties
Public
AddNewTSMProperty.vi
GlobalTestData.lvclass
```

Workflow – Adding Global Test Data to Existing Project

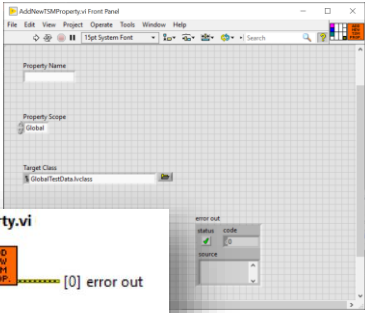
1. Download the Global Test Data template files from GitHub
2. Copy or move the **GlobalTestData** directory into the **Code Modules/Common** directory of your test program's source
3. Ensure **Code Modules/Common/GlobalTestData** has been added to the LabVIEW project
4. Define & add new application specific properties to the Global Test Data class
   a) Create both a Read & Write VI for each new property.
      - Refer to proceeding slides labeled: "Workflow – Creating A New Property slides"

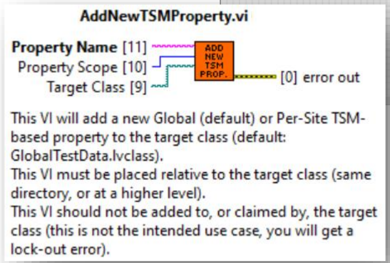
   b) Repeat as needed to scale during test program development

Workflow – Creating A New Property (Scripted Method)

1. Open the AddNewTSMProperty.vi
2. Input the desired property name into the Property Name control
3. Select either Global or PerSite from the Property Scope control to set the property's scope.
   - Global scope – single value at runtime for all sites (site agnostic).
   - PerSite scope – separate values at runtime for each site defined by pin map (site unique)
4. Run the AddNewTSMProperty VI
5. After the VI finishes executing, locate the newly created properties from within the LabVIEW Project Explore window
6. Open both the Write_<scope>_<name>.vi & Read_<scope>_<name>.vi modify the following:
   a) Right click the variant control/indicator (make sure to hover over just the element if scope is PerSite) and select replace, then choose the appropriate data type for the property.

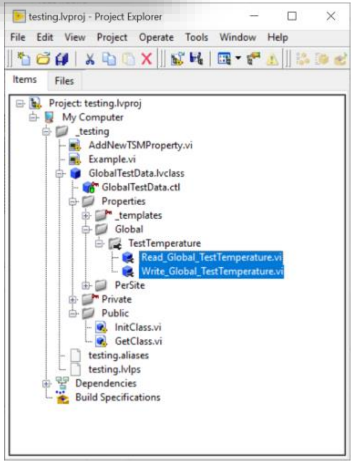
      Note that the data type must match between both VIs or the run arrow for both VIs will be broken.
   b) Update the VI Icon of both functions to match the property name. For example, if the property name is TestTemperature see example below
   c) Edit the VI Properties to add documentation to each VI as well.



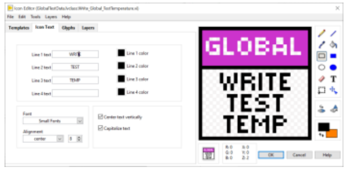

AddNewTSMProperty.vi

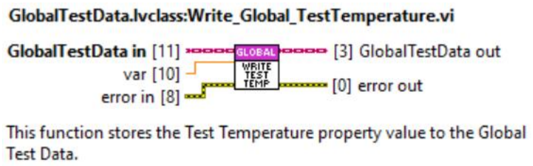
Property Name [11]
Property Scope [10]
Target Class [9]
[0] error out

This VI will add a new Global (default) or Per-Site TSM-based property to the target class (default: GlobalTestData.lvclass).
This VI must be placed relative to the target class (same directory, or at a higher level).
This VI should not be added to, or claimed by, the target class (this is not the intended use case, you will get a lock-out error).


testing.lvproj - Project Explorer

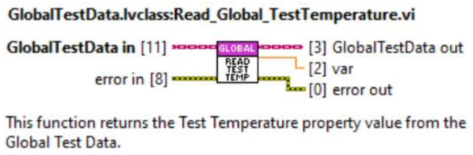
- Project: testing.lvproj
  - My Computer
    - _testing
      - AddNewTSMProperty.vi
      - Example.vi
      - GlobalTestData.lvclass
        - GlobalTestData.ctl
        - Properties
          - _templates
          - Global
            - TestTemperature
              - Read_Global_TestTemperature.vi
              - Write_Global_TestTemperature.vi
          - PerSite
        - Private
        - Public
          - InitClass.vi
          - GetClass.vi
      - testing.aliases
      - testing.lvlps
    - Dependencies
    - Build Specifications




GlobalTestData.lvclass:Write_Global_TestTemperature.vi

GlobalTestData in [11]
var [10]
error in [8]
[3] GlobalTestData out
[0] error out

This function stores the Test Temperature property value to the Global Test Data.


GlobalTestData.lvclass:Read_Global_TestTemperature.vi

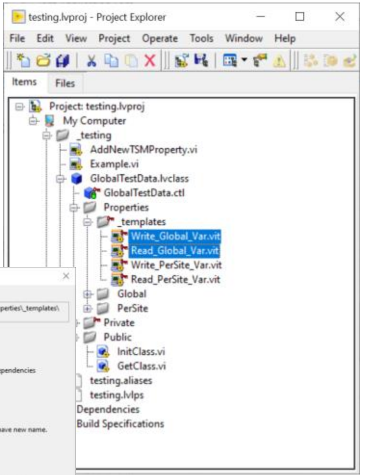
GlobalTestData in [11]
error in [8]
[3] GlobalTestData out
[2] var
[0] error out

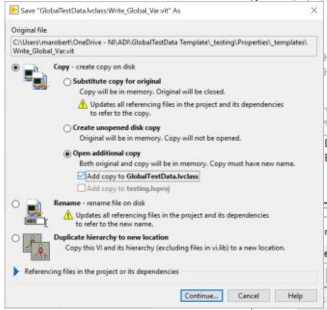
This function returns the Test Temperature property value from the Global Test Data.

Workflow – Creating A New Property (Manual Method)

1. Locate the _**templates** private virtual folder of the GlobalTestData.lvclass from within the LabVIEW Project Explore window
2. Open the appropriate Write template file for the desired scope (Global or PerSite) and then Save As with the Open additional copy: Add copy to GlobalTestData.class option selected.
   - Global scope – single value at runtime for all sites (site agnostic).
   - PerSite scope – separate values at runtime for each site defined by pin map (site unique)
3. Save the new Write VI with the following file naming convention and save it to the GlobalTestData/Properties directory on disk: Write_<scope>_<name>.vi
4. Repeat steps 2 & 3 with the Read template file, naming the copy Read_<scope>_<name>.vi
5. From within the LabVIEW Project Explore window, locate and expand the Properties virtual folder of the GlobalTestData.lvclass and right click the appropriate virtual folder for the scope selected (Global or PerSite) and choose New→Property Definition Folder
6. Rename the newly created Property Definition Folder to the name you have chosen for the new property.
7. From within the LabVIEW Project Explore window, move the both the Write_<scope>_<name>.vi & Read_<scope>_<name>.vi files you created in steps 2-4 into the newly created Property Definition Folder.
8. Open both the Write_<scope>_<name>.vi & Read_<scope>_<name>.vi modify the following:
   a) Right click the variant control/indicator (make sure to hover over just the element if scope is PerSite) and select replace, then choose the appropriate data type for the property.

      Note that the data type must match between both VIs or the run arrow for both VIs will be broken.
   b) Update the VI Icon of both functions to match the property name. For example, if the property name is TestTemperature see example below
   c) Edit the VI Properties to add documentation to each VI as well.




Save "GlobalTestData.lvclass:Write_Global_Var.vit" As

Original file

C:\Users\marobert\OneDrive - NI\ADI\GlobalTestData Template_testing\Properties_templates\Write_Global_Var.vit

[✓ **Copy** - create copy on disk]

Substitute copy for original
Copy will be in memory. Original will be closed.
Updates all referencing files in the project and its dependencies to refer to the copy.

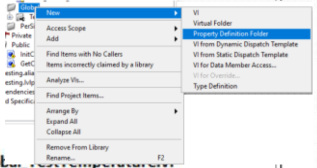
Create unopened disk copy
Original will be in memory. Copy will not be opened.

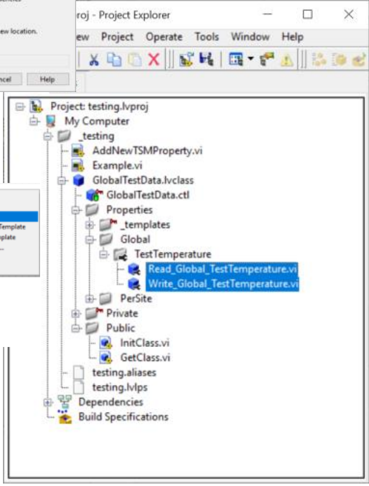
[✓ **Open additional copy**]
Both original and copy will be in memory. Copy must have new name.
[✓ Add copy to **GlobalTestData.lvclass**]
Add copy to testing.lvproj

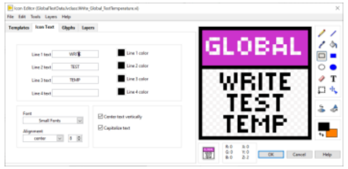
Rename - rename file on disk
Updates all referencing files in the project and its dependencies to refer to the new name.

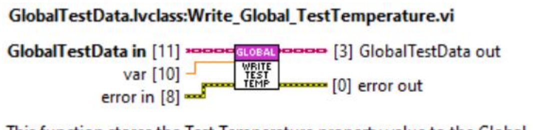
Duplicate hierarchy to new location
Copy this VI and its hierarchy (excluding files in vi.lib) to a new location.

Referencing files in the project or its dependencies

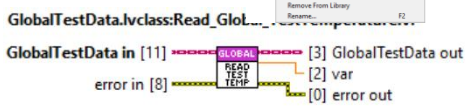








This function stores the Test Temperature property value to the Global Test Data.



This function returns the Test Temperature property value from the Global Test Data.

Workflow – Using Global Test Data Properties in Code Modules

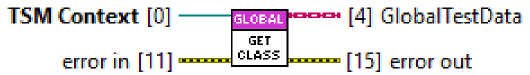
Accessing Data During Code Module Development

1. **Drop down the GlobalTestData.lvclass:Get Class.vi** onto the block diagram of your code module and wire the Semiconductor Module context to its top input terminal (TSM Context).
2. **Drop down a LabVIEW Property Node** (any type) and wire the GlobalTestData class wire from the Get Class from TSM Context.vi's output terminal to the Property Node's input terminal.

   Make sure to also wire up the error wires appropriately.
3. **Click to select the items to access within the property node and drag the bottom of the property node to expand out the number of items to be accessed**.

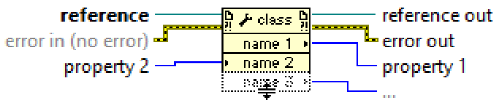
   Note: right clicking the item within the property node will allow you to change whether to read or right to that property, it will read by default unless changed. It will also give you an option to open the accessor function if it needs to be reviewed or modified.

GlobalTestData.lvclass:GetClass.vi

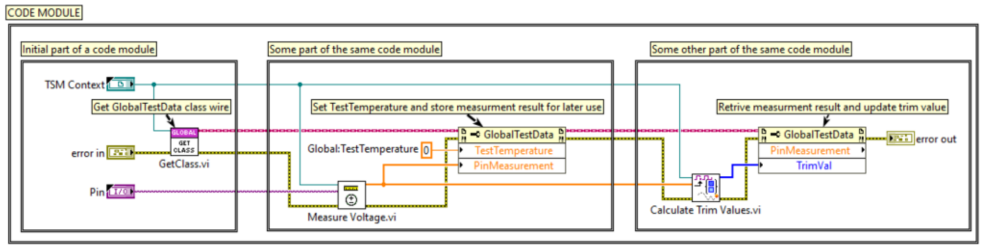


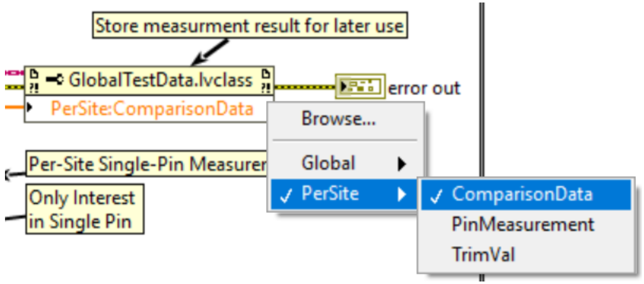
This function should be called at the start of test code module. It will construct the GlobalTestData class wire by referencing the approprate instance of the Semiconductor Module Context.

Property Node



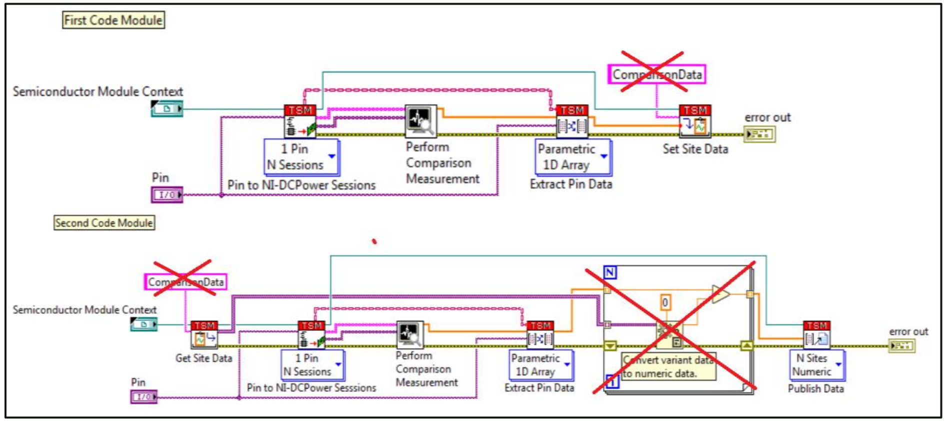
Gets (reads) and/or sets (writes) properties of a reference. Use the property node to get or set properties and methods on local or remote application instances, VIs, and objects. You also can use the Property Node to access the private data of a LabVIEW class.


CODE MODULE
Initial part of a code module
TSM Context
Get GlobalTestData class wire
error in
GetClass.vi
Pin
Some part of the same code module
Set TestTemperature and store measurment result for later use
Global:TestTemperature
GlobalTestData
TestTemperature
PinMeasurement
Measure Voltage.vi
Some other part of the same code module
Retrive measurment result and update trim value
GlobalTestData
PinMeasurement
TrimVal
error out
Calculate Trim Values.vi

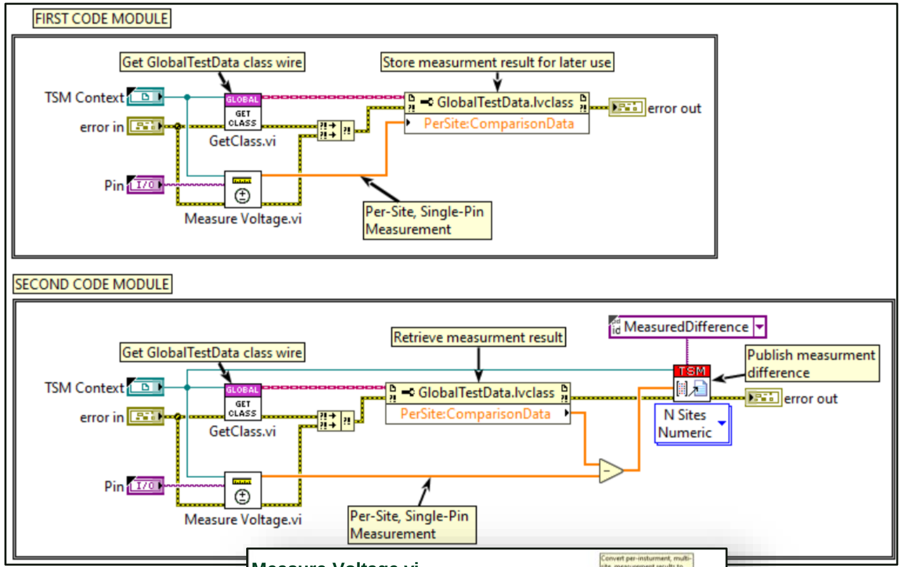


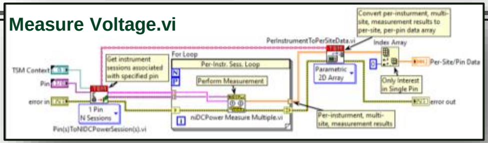
Example – Sharing Global Test Data Between Code Modules

With Native TSM Solution



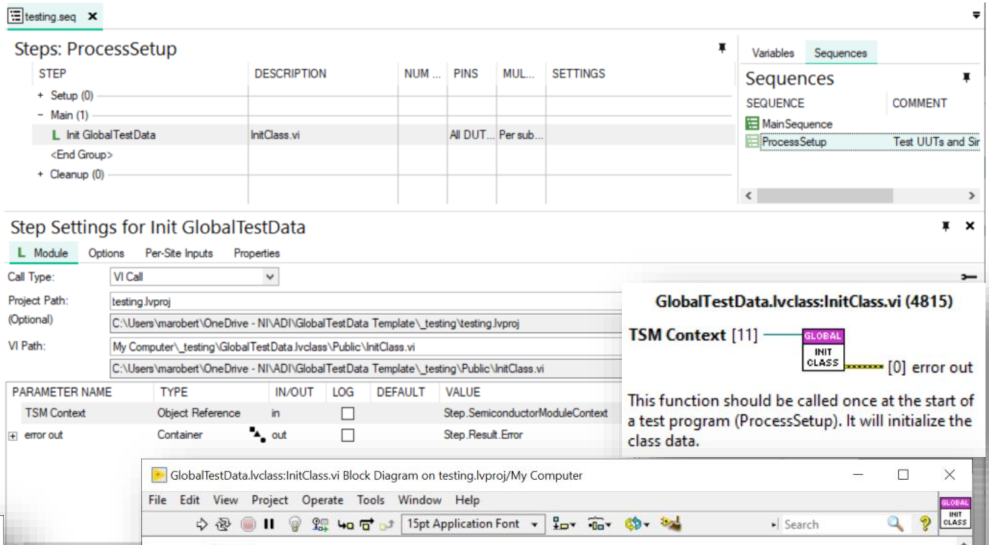
With TSM Context Data Access Concept




Measure Voltage.vi
Convert per-insturment, multi-site, measurement results to per-site, per-pin data array
PerInstrumentToPerSiteData.vi
Index Array
For Loop
Per-Instr. Sess. Loop
Get instrument sessions associated with specified pin
TSM Context
Pin
error in
Perform Measurement
Parametric 2D Array
Per-Site/Pin Data
Only Interest in Single Pin
error out
1 Pin
N Sessions
Pin(s)ToNIDCPowerSession(s).vi
niDCPower Measure Multiple.vi
Per-insturment, multi-site, measurement results

Workflow – Initializing Global Test Data in ProcessSetup

- The **InitClass.vi** in the GlobalTestData class **must be called from TestStand in ProcessStep within the Test Program's sequence file**.
- Inside this VI is where initial values of the Global Test Data properties should be defined and will be instantiated when a lot starts.



testing.seq

Steps: ProcessSetup

STEP	DESCRIPTION	NUM ...	PINS	MUL...	SETTINGS
+ Setup (0)					
− Main (1)					
Init GlobalTestData	InitClass.vi		All DUT...	Per sub...	
<End Group>					
+ Cleanup (0)					

Variables · Sequences

Sequences

SEQUENCE	COMMENT
MainSequence	
ProcessSetup	Test UUTs and Sir

Step Settings for Init GlobalTestData

Module · Options · Per-Site Inputs · Properties

Call Type: VI Call

Project Path: (Optional) testing.lvproj

C:\Users\marobert\OneDrive - NI\ADI\GlobalTestData Template_testing\testing.lvproj

VI Path: My Computer_testing\GlobalTestData.lvclass\Public\InitClass.vi

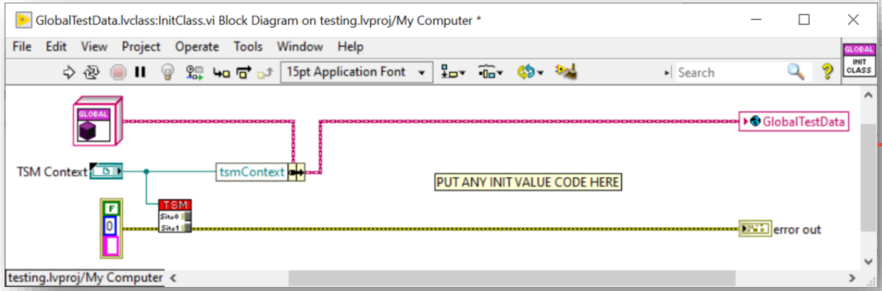
C:\Users\marobert\OneDrive - NI\ADI\GlobalTestData Template_testing\Public\InitClass.vi

PARAMETER NAME	TYPE	IN/OUT	LOG	DEFAULT	VALUE
TSM Context	Object Reference	in	☐		Step.SemiconductorModuleContext
error out	Container	out	☐		Step.Result.Error

GlobalTestData.lvclass:InitClass.vi (4815)

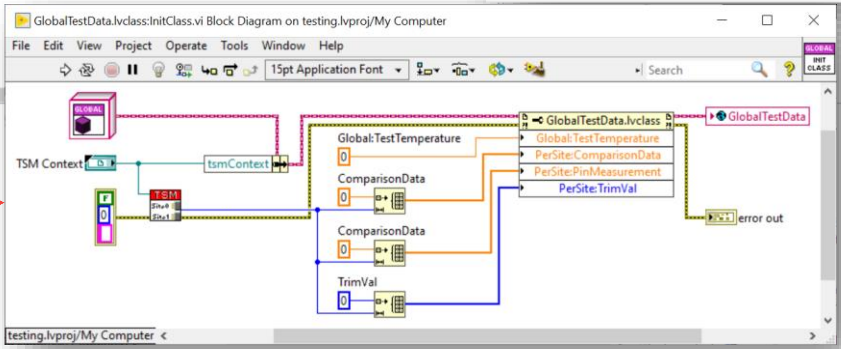
TSM Context [11] — [0] error out

This function should be called once at the start of a test program (ProcessSetup). It will initialize the class data.



GlobalTestData.lvclass:InitClass.vi Block Diagram on testing.lvproj/My Computer *

File Edit View Project Operate Tools Window Help
15pt Application Font
Search
TSM Context
tsmContext
GlobalTestData
PUT ANY INIT VALUE CODE HERE
error out
testing.lvproj/My Computer



GlobalTestData.lvclass:InitClass.vi Block Diagram on testing.lvproj/My Computer

Taking this concept to the next level!

- This concept of class-based data encapsulation in LabVIEW can be used for other application specific class types in your Test Program, not just Global Test Data.
  - You may want to explore creating your own class template for an application specific class type, such a Fuse data.
- Making Data Private & Adding Methods
  - Any property in the Class can be marked as private, meaning only a VI claimed by the class (i.e. a class method) can access on that data from a property node.
  - Data can further be encapsulated by making properties private and only accessing or acting on the data via the use of class methods.
  - While methods should not be added for the Global Test Data use case, the LabVIEW class can incorporate data specific methods to act on the data however you see fit.