



connect

2024 AUSTIN



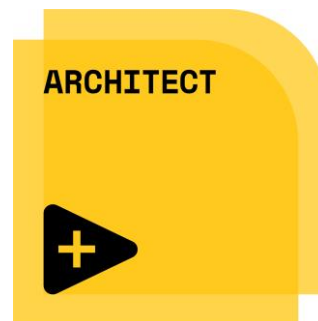
TestStandifier

Adapt your DQMH® modules to efficient usage in TestStand

Cyril GAMBINI, M. Eng, Neosoft Technologies

CLA, CTA, LabVIEW Champion

- 18 years working with LabVIEW (PC, RT, FPGA), TestStand, VeriStand, FlexLogger
- 17 years working for NI partners
- Currently Vice-President of Engineering at Neosoft Technologies (Canada)
- Involved in several open-source projects





NEOSOFT Technologies

Founded in 2000, NI partner since 2008

- **CLA/CLD/CLAD, CTA/CTD, CLED, CPI, LV Champion on staff**
- **DQMH® Trusted Advisor**
- System integration and retrofit – Software, mechanic and electric assemblies
- Automated test systems
- Acquisition and control systems
- Embedded RT and FPGA systems
- Hardware In the Loop systems (HIL)
- Automated inspection systems





Agenda

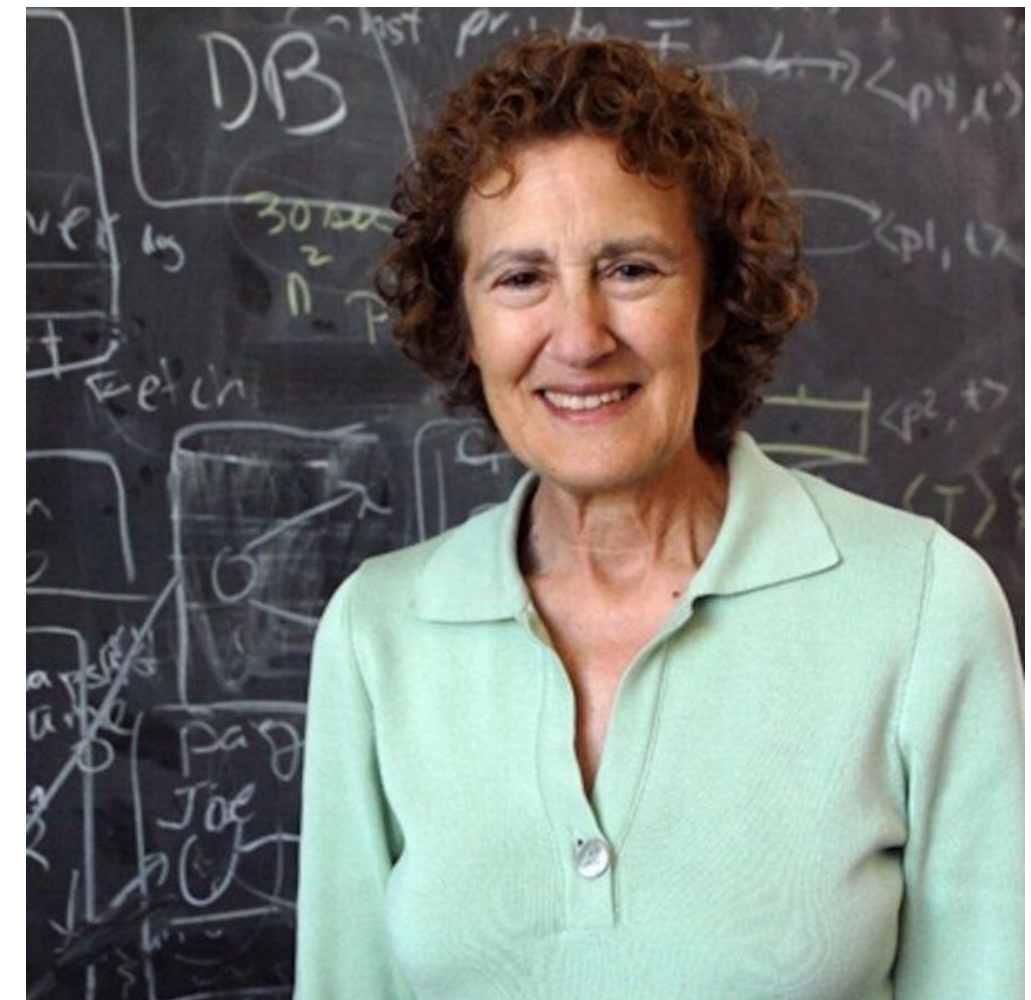
- DQMH® and TestStand
- TestStandifier
- Using the output in TestStand



#OurFemalesAreGiant – Barbara Liskov

Computer scientist

- Pioneering contributions to programming languages and distributed computing
- Introduced abstracted data types and principle of data abstraction
- Elaborated the principal of Liskov substitution (OOP)
- Second woman to receive the Turing award
- Professor at the MIT

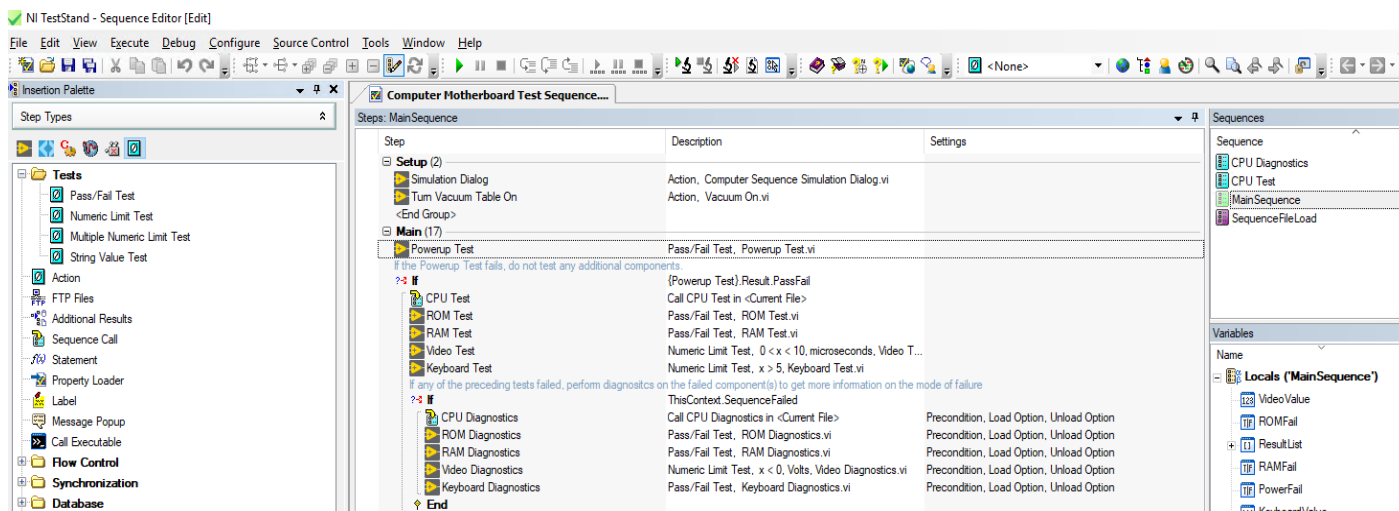




DQMH® and TestStand

What is TestStand ?

- TestStand is an ENGINE to run test sequences
- Calls adapters to execute code (LV, .NET, DLL, ...)
- UIs allow to create and visualize sequences



What is DQMH® ?

- DQMH® stands for Delacor Queued Message Handler
- Freely available reference design for LabVIEW (framework)
- Modules executing as long as needed
- Public API defining how to use the module
- Tester to manually trigger Public API
- Maintained by a consortium - <https://dqmh.org>





DQMH® and TestStand

TestStand

Purpose:

- Execute SCRIPTED sequences
- Give a status to a UUT
- Report results

Handles WHEN a code is executed:

- Using the process model

Handles 1 UUT or multiple UUT tests (batch or parallel)

- TS engine automatically duplicate sequences to execute

DQMH®

- Modules can be singletons or cloneables
- The tester is built for 'free' (and very useful)
- Enforces documentation
- Control when to start / stop a module
- Keep data context during its execution



DQMH® and TestStand

‘Traditional’ TestStand programming

Steps handle the logic

- All the code requested to be executed by TS is encapsulated within the step

Steps are short in time

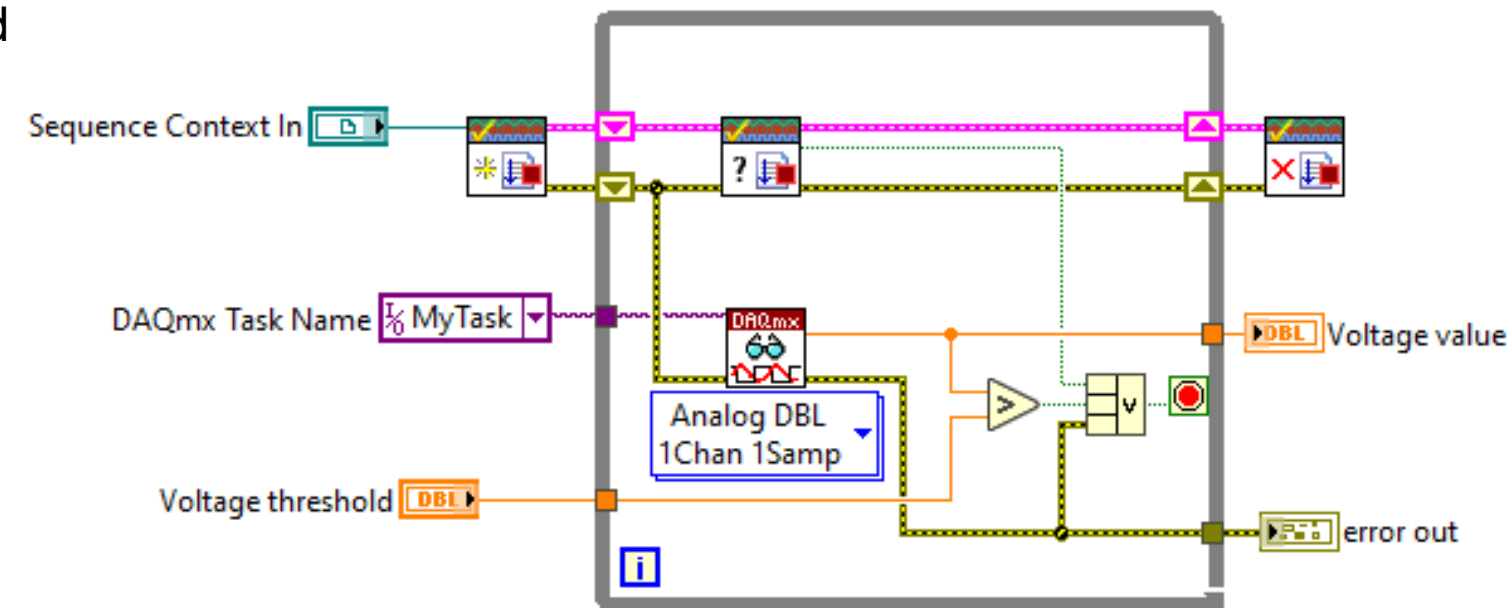
- When an adapter runs, TS waits !
- Handle termination within the step

Steps may produce errors

- TS can catch errors returned by a step and trigger a specific sequence to run (Error Callback)

Steps may produce data

- TS may collect the data for result processing / reporting





DQMH® and TestStand

Why using DQMH with TestStand ?

- Code can be done without TestStand in mind
- Access to the tester from LabVIEW and from TestStand
- Forces to create atomic steps by design (atomicity of the requests)
- Capability to keep a context outside of TS
- Easy to create helper loops (continuous monitoring in // of a sequence execution)
- Cloneables fit well with batch and parallel process models



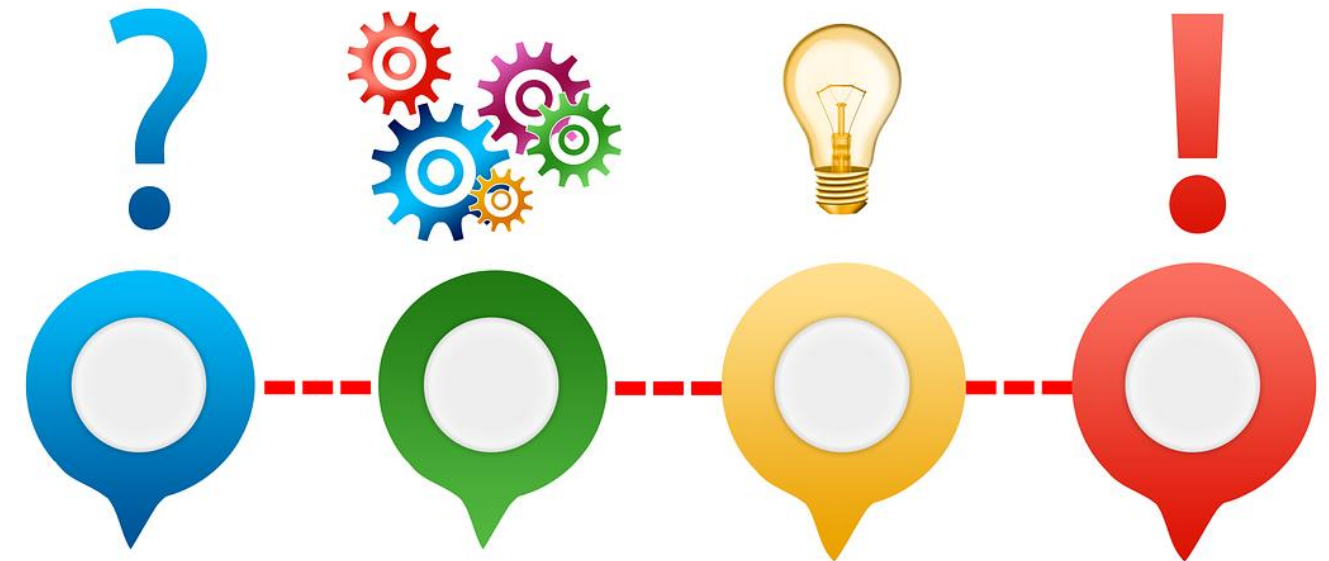


DQMH® and TestStand

As easy as it seems to be ?

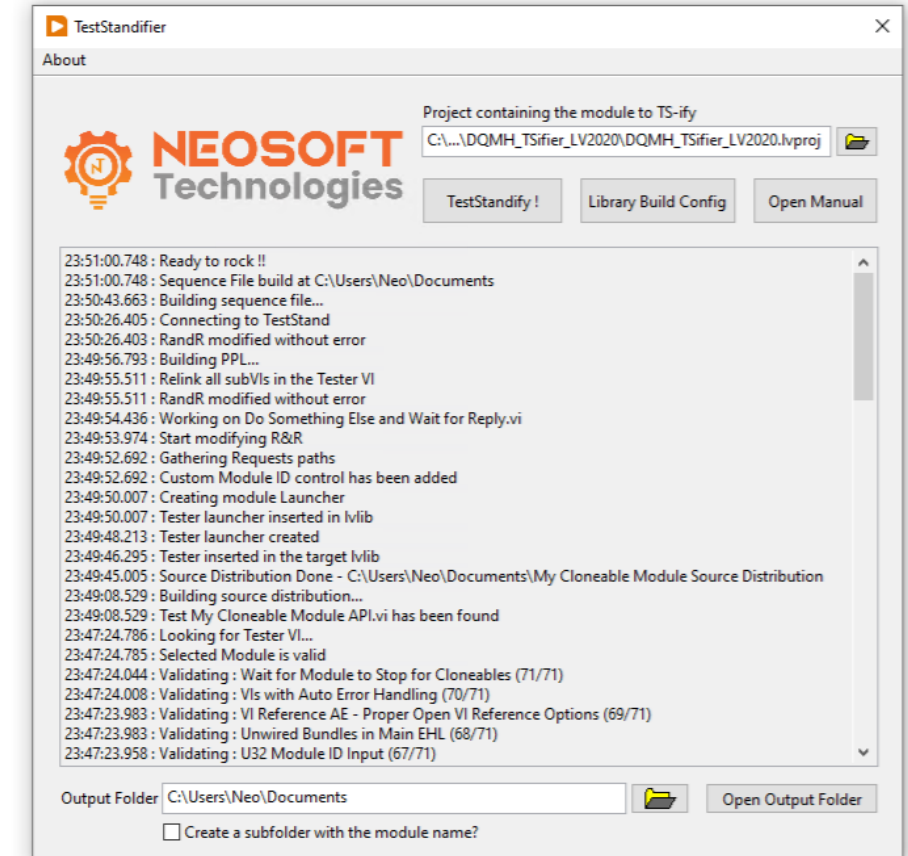
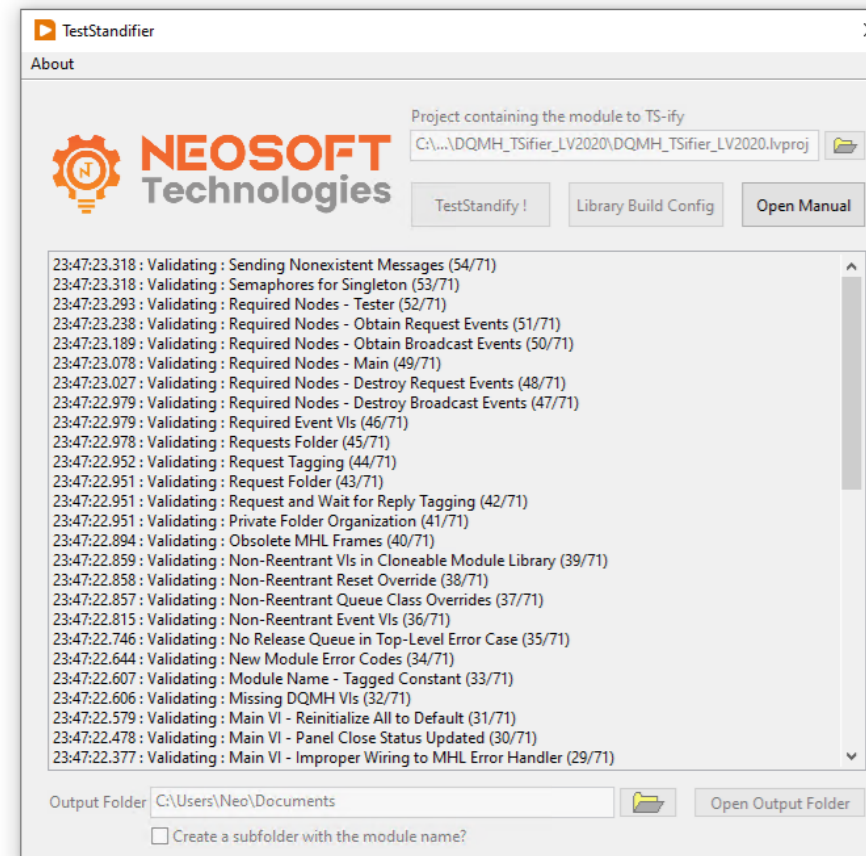
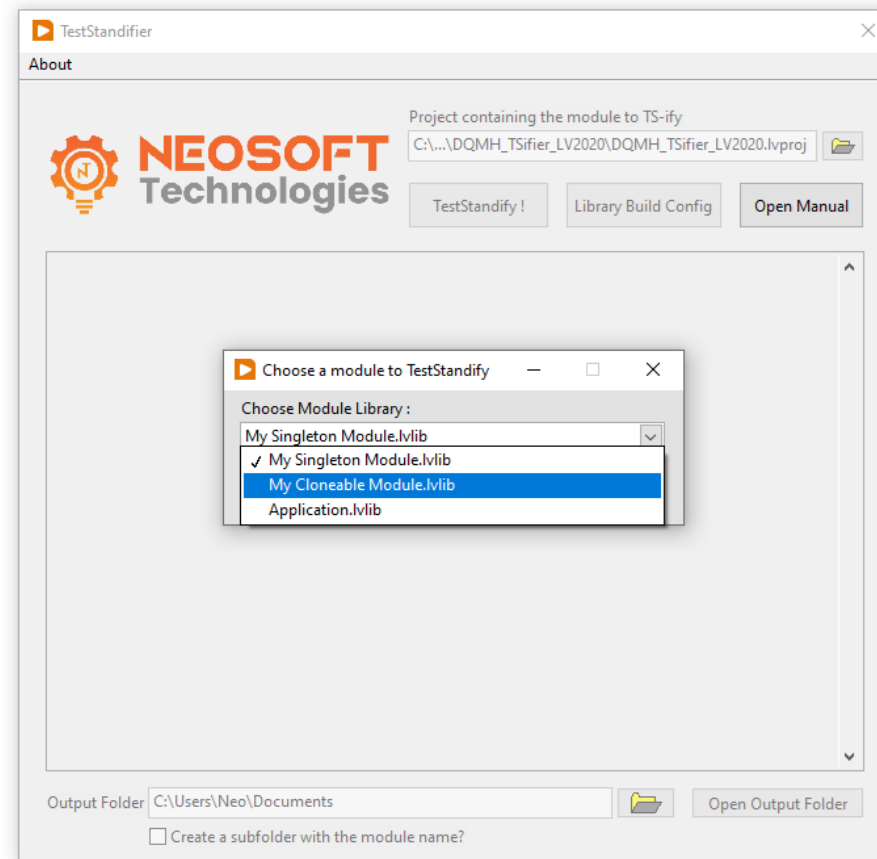
Few problems to overcome :

- Launching a module : several steps
- Rapidly stop your sequence execution on demand
- Where should be the error handling ?
- Tester : blocking execution !
- Synchronise ModuleID and TestSocketIndex





TestStandifier

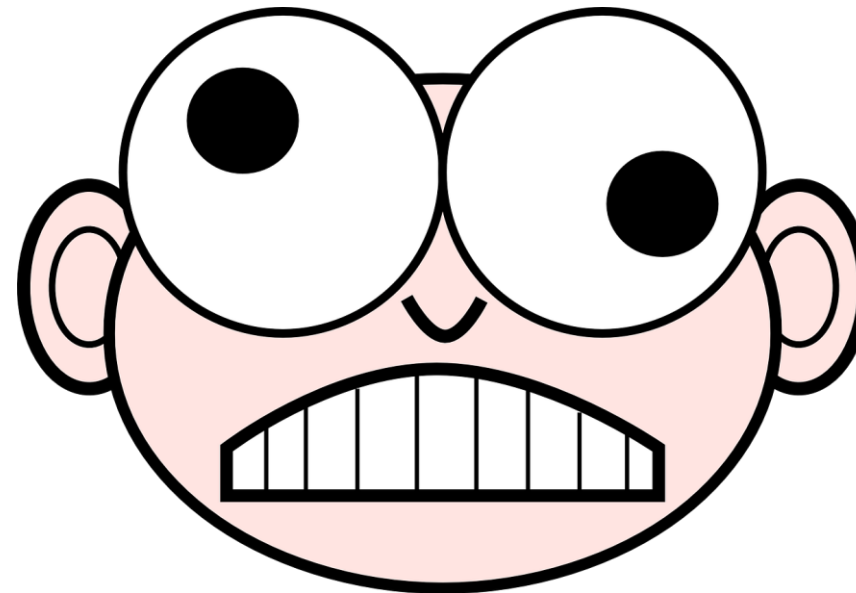




TestStandifier

Validating a module

- Relies on the DQMH® validator
- Continues only if the module is valid (according to the validator)
- A modified module MIGHT run well
- Can't imagine how many weird stuff we saw !



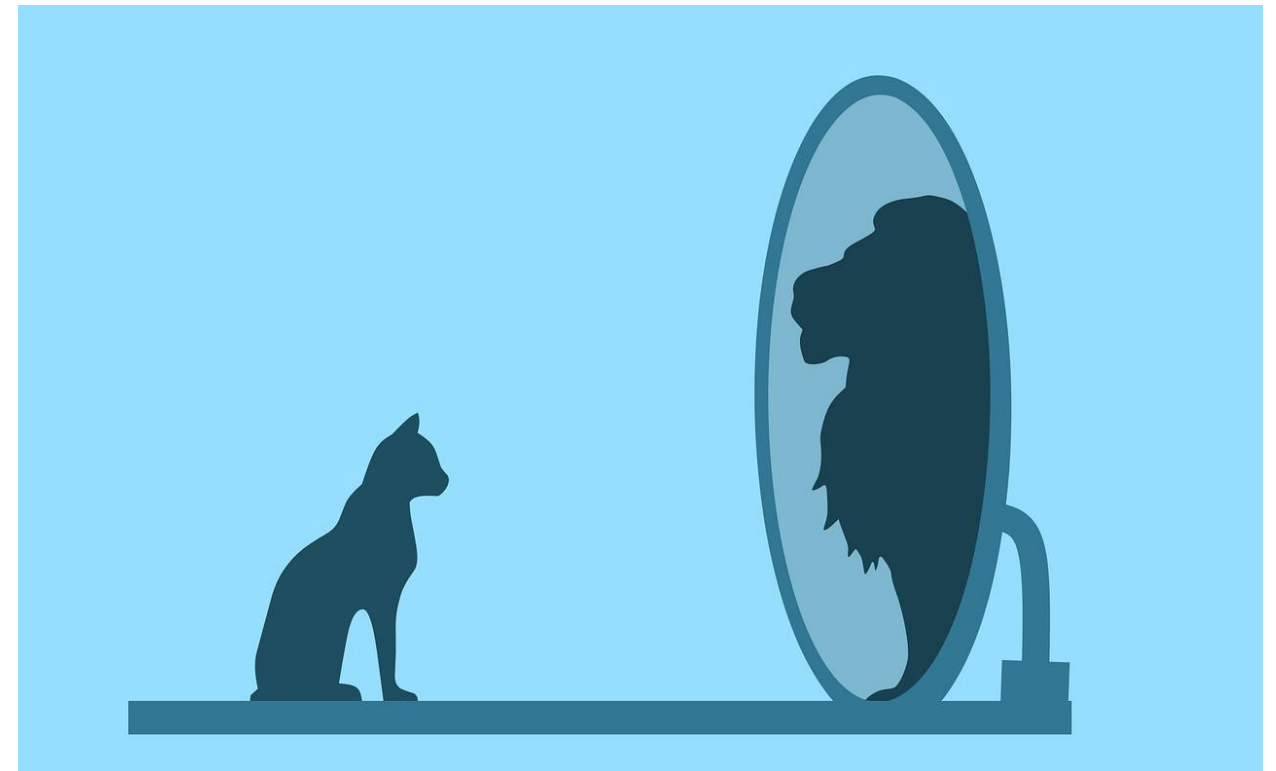


TestStandifier

Build a source distribution (and more)

- Duplicates the original module
- Relinks all subVIs to be flattened and not depend on vi.lib
- Inserts the tester in the lvlib and create wrappers
- Modifies Request and Replies
- Modifies ModuleID handling for Cloneables

Never touches the original module !

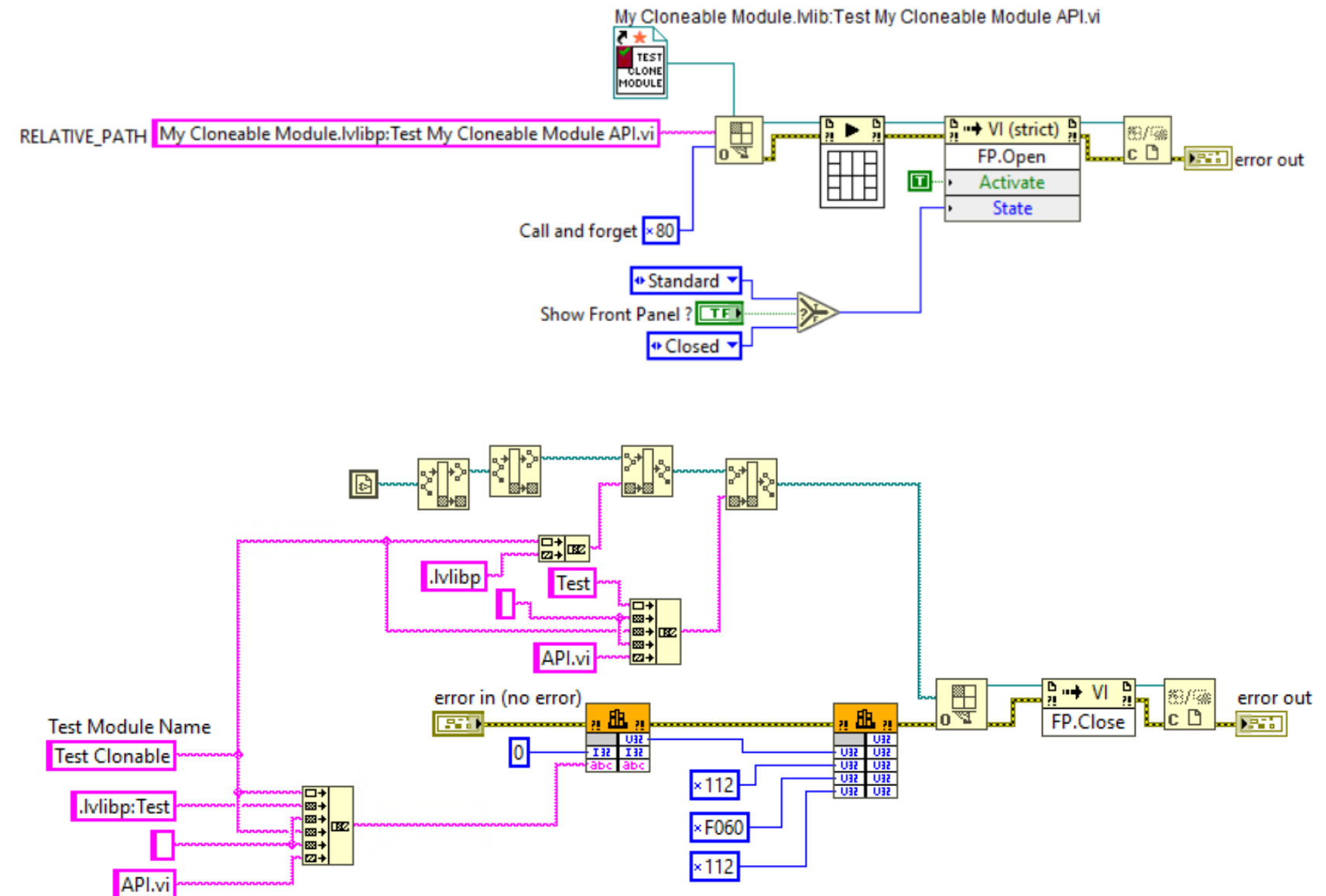




TestStandifier

Makes the DQMH® Tester run in parallel to your test sequence

- Standard DQMH® Tester:
 - Tester = While loop
 - TestStand will wait until the tester returns to continue executing the sequence
 - Can only be stopped if manually closed (Window closing)
- TestStandifier creates wrappers VIs to start / stop the tester and calls the wrappers in a sequence

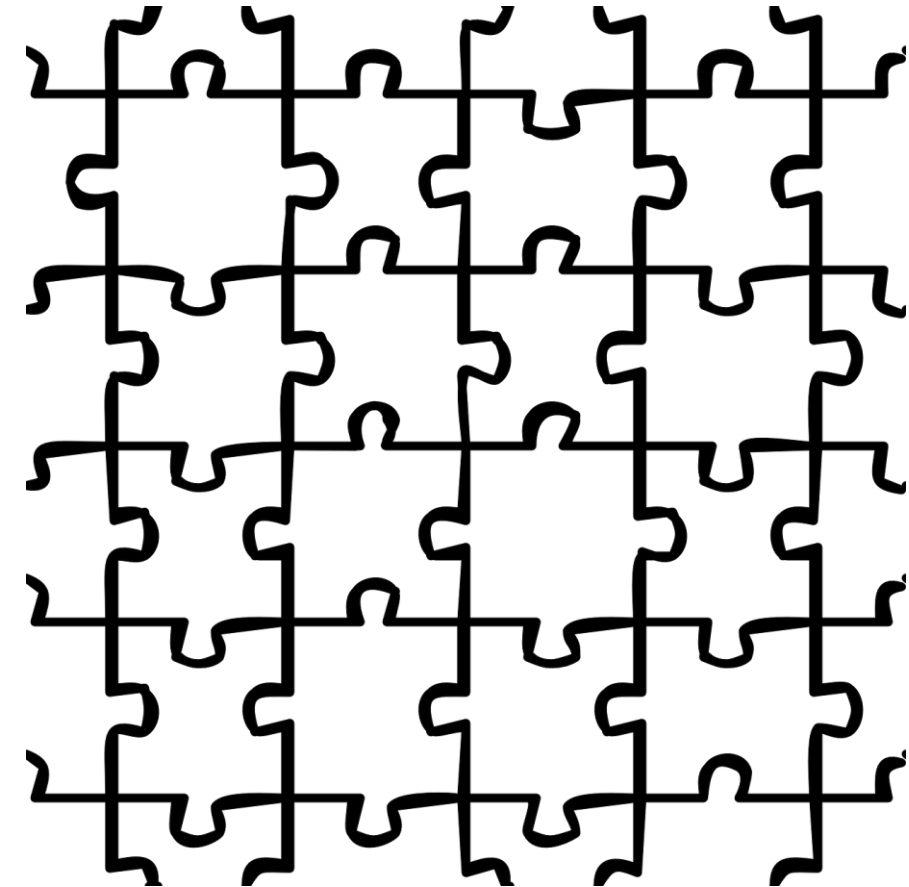




TestStandifier

Builds a PPL !

- Creates a temporary project to link the Ivlib
- Creates a PPL specification
- Enforces dependencies to be always included
- Only one file to handle !



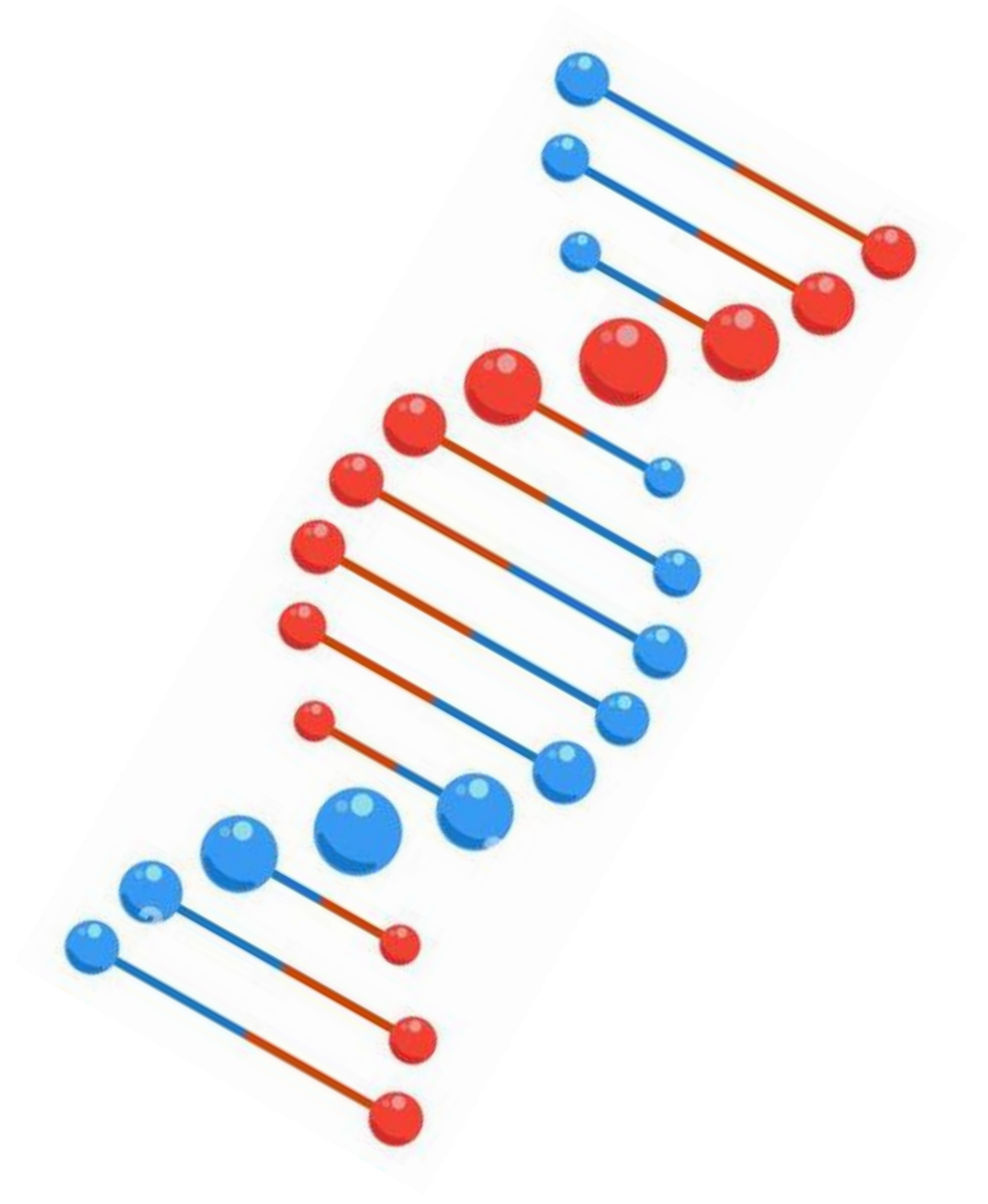
The TestStandified module is autonomous !



TestStandifier

Creates a sequence file !

- Creates sub-sequence to launch a module
- Creates a sub-sequence to launch the tester
- Creates a sub-sequence to stop the tester
- Creates a sub-sequence stop a module
- Creates a sub-sequence per Request & Reply
- Creates a PostStepRunTimeError callback within the sequence file
- Maps VI IOs to Sequence IOs





TestStandifier

Launching a module

Align the Start Module and Sync Module steps in a sequence

1. Place Start Module VI
2. Place Sync VI
3. Link them thanks to a local variable
4. Parameters published on the sequence call (by value / by reference)

My Cloneable Module.seq x

Steps: Start Module

STEP	DESCRIPTION	SETTINGS
+ Setup (0)		
- Main (2)		
L Start Module.vi	Action, Start Module.vi	
L Synchronize Module Events.vi	Action, Synchronize Module Events.vi	
<End Group>		
+ Cleanup (0)		

Variables Sequences

Sequences

SEQUENCE	COMMENT	REQUIREMEN
Start Module		
Tester Launcher		
Stop Tester		
Do Something Else and Wait for ...		
Do Something		
Do Something Else		
Show Panel		
Hide Panel		
Stop Module		
Show Diagram		
Launch Tester		
MainSequence		
SequenceFilePostStepRuntimeEr...		

Step Settings for Start Module.vi

Module Properties

Call Type: VI Call

Project Path: (No file specified)

VI Path: My Cloneable Module.lvlibp\Start Module.vi

C:\Users\Neo\Documents\My Cloneable Module.lvlibp\Start Module.vi

PARAMETER NAME	TYPE	IN/OUT	LOG	DEFAULT	VALUE	
Run as Singleton? (F)	Boolean	in	<input type="checkbox"/>	<input type="checkbox"/>	Parameters.Run_as_Singleton_F	fx ✓
Custom Module ID	Number (I32)	in	<input type="checkbox"/>	<input type="checkbox"/>	Parameters.Custom_Module_ID	fx ✓
Use Default DQMH Mod...	Boolean	in	<input type="checkbox"/>	<input type="checkbox"/>	Parameters.Use_Default_DQMH_Mod...	fx ✓
error in	Container	in	<input type="checkbox"/>	<input type="checkbox"/>		fx ✓
Show Main VI Diagram o...	Boolean	in	<input type="checkbox"/>	<input type="checkbox"/>	Parameters.Show_Main_VI_Diagram_o...	fx ✓
Module ID	Number (I32)	out	<input type="checkbox"/>		Parameters.Module_ID	fx ✓
My Cloneable Module Br...	Container	out	<input type="checkbox"/>			fx ✓
Wait for Event Sync?	Boolean	out	<input type="checkbox"/>		Parameters.Wait_for_Event_Sync	fx ✓
error out	Container	out	<input type="checkbox"/>		Step.Result.Error	fx ✓

My Cloneable Module.lvlibp:Start Module.vi

Run as Singleton? (F) Module ID
Custom Module ID My Cloneable Module I
Use Default DQMH Module ID Wait for Event Sync?
error in error out

Launches an instance of the module Main VI. After calling this VI, you c optionally register for broadcast events from the module by wiring the br events output of this VI to a Register For Events function.

After the optional Register For Events function call, you should always c Synchronize Module Events.vi for this module with the "Wait for Event S output of this VI to the corresponding input of the Synchronize Module Events.vi.

To see an example of the proper wiring pattern, see the "Run New Mod Value Change" event frame in the API Tester VI for this module.

Based on DQMH Project Template 7.0.1.1316.



TestStandifier

ModuleID synchronization

- TestStand sequences are launched in order but do not start in order
- ModuleID is incremented and stored in a feedback node
- Offer a way to override the ModuleID generation
- TestSocket Index = Module ID
(helps for Batch / Parallel Process Models)

Step Settings for Start Module.vi

Module Properties

Call Type: VI Call

Project Path: (No file specified)

VI Path: My Cloneable Module.lvlibp\Start Module.vi
C:\Users\Neo\Documents\My Cloneable Module.lvlibp\Start Module.vi

PARAMETER NAME	TYPE	IN/OUT	LOG	DEFAULT	VALUE	
Run as Singleton? (F)	Boolean	in	<input type="checkbox"/>	<input type="checkbox"/>	Parameters.Run_as_Singleton_F	fx ✓
Custom Module ID	Number (I32)	in	<input type="checkbox"/>	<input type="checkbox"/>	Parameters.Custom_Module_ID	fx ✓
Use Default DQMH Mod...	Boolean	in	<input type="checkbox"/>	<input type="checkbox"/>	Parameters.Use_Default_DQMH_Mod...	fx ✓
error in	Container	in	<input type="checkbox"/>	<input type="checkbox"/>		fx ✓
Show Main VI Diagram o...	Boolean	in	<input type="checkbox"/>	<input type="checkbox"/>	Parameters.Show_Main_VI_Diagram_o...	fx ✓
Module ID	Number (I32)	out	<input type="checkbox"/>		Parameters.Module_ID	fx ✓
My Cloneable Module Br...	Container	out	<input type="checkbox"/>			fx ✓
Wait for Event Sync?	Boolean	out	<input type="checkbox"/>		Parameters.Wait_for_Event_Sync	fx ✓
error out	Container	out	<input type="checkbox"/>		Step.Result.Error	fx ✓

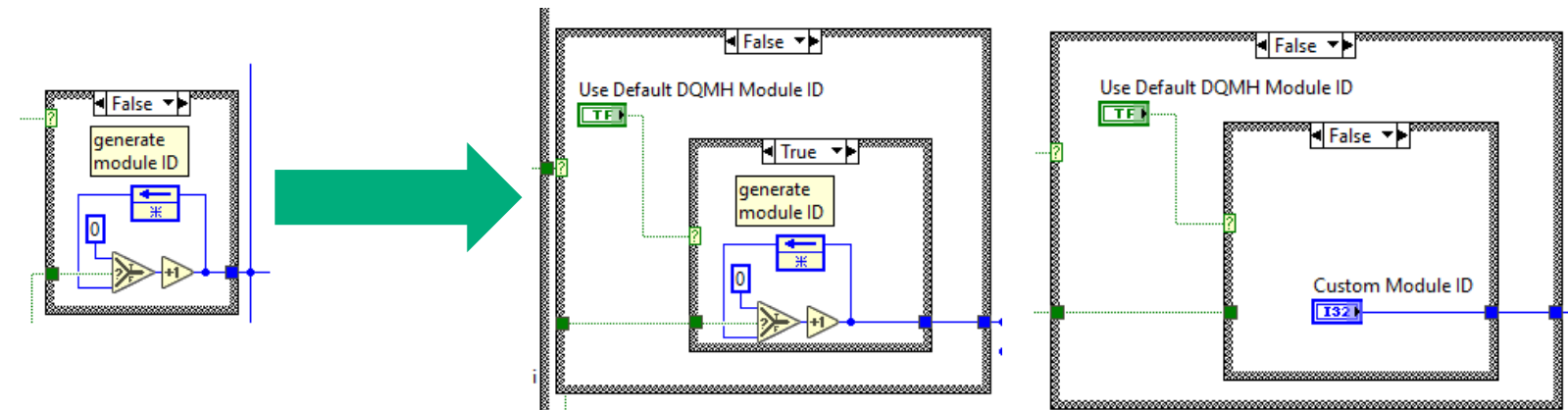
My Cloneable Module.lvlibp:Start Module.vi

Run as Singleton? (F) ... Module ID ... My Cloneable Module ... Wait for Event Sync? ... error out

Launches an instance of the module Main VI. After calling this VI, you c optionally register for broadcast events from the module by wiring the br events output of this VI to a Register For Events function.

After the optional Register For Events function call, you should always c Synchronize Module Events.vi for this module with the 'Wait for Event S output of this VI to the corresponding input of the Synchronize Module Events.vi.

To see an example of the proper wiring pattern, see the "Run New Mod Value Change" event frame in the API Tester VI for this module.

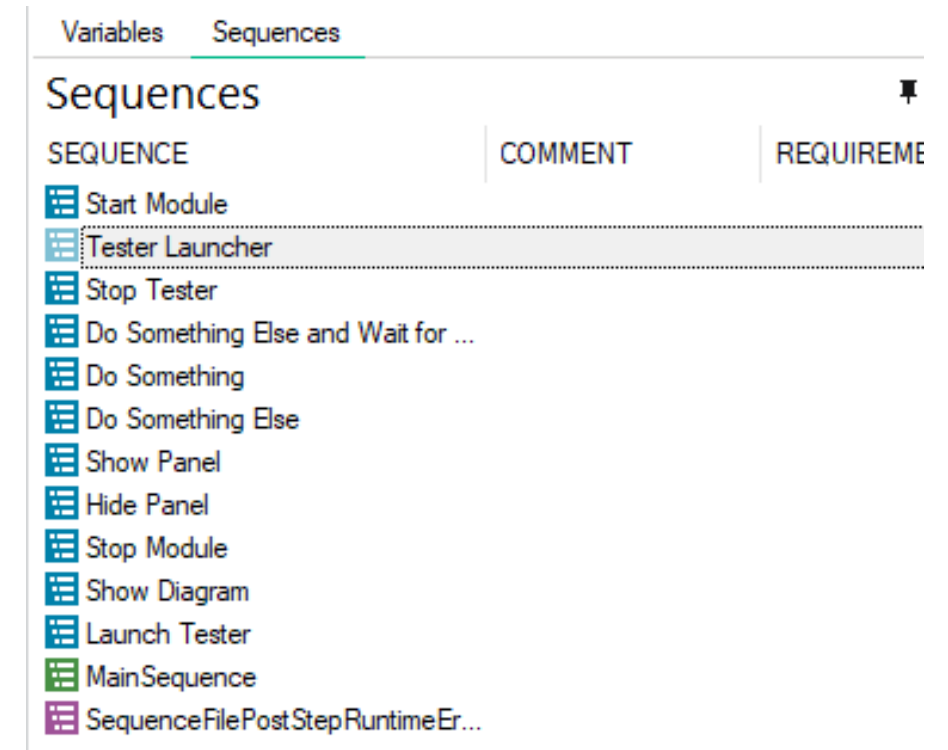




Using the output

Output is composed of 1 sequence file and 1 PPL

- Concentrate on TESTSTAND ! Sequence file is now a 'library' !
- Use the ability of a sequence call to execute actions in parallel
- Handle your errors in the created PostStepRunTimeError if needed
- Synchronize your Module ID to your TestSocket index
- Prefer launching / stopping your modules from Process Model Callbacks
- Don't forget: code runs in parallel to your sequence execution
- Use and abuse of the tester ! (advice: create Round Trips !)





Use it !

Available for free on VIPM !!

- Regularly maintained by Neosoft Technologies
- NI / VIPM Forum to post issues / questions / how much you like it !
 - <https://forums.ni.com/t5/Neosoft-Technologies/Support-Forum-Neosoft-DQMH-TestStandifier/td-p/4328567>
 - <https://forums.vipm.io/forum/102-dqmh-teststandifier-by-neosoft-technologies/>
- Open to sponsorship to add functionalities
- Maintain compatibility with new versions of DQMH®



NEOSOFT
Technologies



Acknowledgements - Questions

Developers, sponsors and supporters !

- Doyoung Kim, Neosoft Technologies
- Raphael Fortin, Neosoft Technologies
- DQMH® Consortium
- Darren Nattinger for advices on VI scripting
- VIShorts for sponsoring



NEOSOFT
Technologies

www.neosoft.ca

Cyril GAMBINI
cgambini@neosoft.ca