From the Designers: Programming Modular Instruments

Jake Harnack, Product Manager
Marcos Kirsch, Principal Software Engineer
Driver Help Files

- Measurement fundamentals
- Connection diagrams
- Programming flow
- Supported properties by device
- Function and VI reference
Consistency Between Modular Instruments Drivers

- IVI-based
- Soft Front Panels
- Language support
  - LabVIEW
  - LabVIEW NXG
  - C/C++
  - LabWindows/CVI
  - C#
  - Python (coming soon!)
- Programming flow
- State model
Agenda

- Configuration Utilities
- Debugging
- Programming Flow
- Triggering and synchronization
- Data transfer
Configuration Utilities
General utilities

- Drivers:
  - Common functionality
    - Reset
    - Self-test
    - Calibration date
  - Driver-specific functions
    - Require valid session handle

- NI-ModInst
  - Device enumeration
System Configuration API

- One API for everything
  - Not only Modular Instruments
- No driver session needed
- Provides
  - Enumeration
  - Geographical info
  - Reset
  - Self-Calibration
  - Device Temperature
  - Renaming
  - …
Debugging
Soft Front Panel: Interactive Control and Debug

LabVIEW® application code

Soft Front Panel w/ Debug Driver Technology

Whitepaper and supported drivers
Probing Instrument Handle to View Session Properties

reset device
I/O Trace

- Formerly NI-SPY
- Logs NI API calls from an application
  - Locate incorrect API calls
  - Verify correct communication with instrument
- Combine with SFP
  - Learn how SFP uses the API
- Programmatic API available

Tips for a good I/O Trace capture
1. Set call history depth
2. Log to disk
3. Confirm API choices and column headers
Programming Flow
Modular Instruments State Model

1. Initialize a session
2. Configure properties, triggers, mode, etc.
3. Read data
4. Close the session
Modular Instruments State Model

1. **Initialize** a session
2. **Configure** properties, triggers, mode, etc.
3. **Initiate** generation and/or acquisition
4. **Fetch** your measurements
5. **Abort** generation and/or acquisition
6. **Close** the session
Modular Instruments State Model

1. **Initialize** a session
2. **Configure** properties, triggers, mode, etc.
3. **Commit** your settings
4. **Initiate** generation and/or acquisition
5. **Fetch** your measurements
6. **Abort** generation and/or acquisition
7. **Close** the session
Modular Instruments State Model

- **Initialize**
  - Opens a session
  - Reserves device or channels

- **Attribute verification**
  - Attribute values are looked as a whole
  - Driver will implicitly do this, but you can do it indirectly by getting an attribute

- **Commit**
  - Creates routes, reserves trigger lines
  - Sets hardware up (i.e. configure analog front end, download hardware sequences)
  - Driver will implicitly do this, but you can call it explicitly

- **Initiate**
  - Begins acquisition / generation
  - Puts you in running state
  - You can now call “on the fly” attributes

- **Abort**
  - Back to committed state
  - You can now re-configure
  - Driver can implicitly call this for you.

- **Close**
  - Stops ongoing acquisitions
  - Releases hardware
Optimization tips

- Re-use your sessions
  - Initialize can be slow

- Beware of Reset
  - Reset Device vs Reset
  - Initialize has Reset parameter, defaults to true
  - Session configuration doesn’t carry over

- Minimize “verifies”
  - Get attributes after you’ve configured your session

- Commit explicitly
  - Have control over where time is spent
  - … if you need Initiate to be fast
  - Useful if you have lots of instruments
Sessions

- **Initialize** reserves hardware
- **Close** releases hardware

**TIPS**
- *Close* stops generation, but the device holds the last "sample". This is very relevant for NI-DCPower.
- New sessions do not inherit state from previous sessions.
- Close can return errors (but still closes).
Sessions

- **Auto-close**: Open 2\textsuperscript{nd} session to same device, first session is closed.
  - Only when is same process. Otherwise error.
  - Per-session mutex on every function call

- **Session mutex**: Every driver call for a session is serialized
  - Thread safety
  - Recursive lock (not useful LabVIEW), i.e:
    \[
    \text{ViStatus } \_\text{VI\_FUNC niScope\_LockSession (ViSession vi, ViBoolean* callerHasLock);} \]

Tips: Clear zombie sessions…

- Open and Run “Close” VI
- Call *IVI Delete Session*
- Restart LabVIEW
Per-Channel Sessions: DCPower

Available in new FGEN hardware!
Triggering and Synchronization
Choose what’s right for your application

- **Jitter & latency**:<1ms
- **Program complexity**
  - **SW timing**: <1ms
  - **HW trigs**: <1μs
  - **TClk**: <1ns

- **SMU sequences**
- **Full rate digitizer acquisitions**
- **7 ½ digit DMM measurements**

![Diagram showing choices based on jitter, latency, and program complexity](image-url)
## Hardware Timed Control

<table>
<thead>
<tr>
<th>Driver</th>
<th>HW Timing mode</th>
<th>NI-TCIk support</th>
</tr>
</thead>
<tbody>
<tr>
<td>NI-DMM</td>
<td>Multi-point</td>
<td></td>
</tr>
<tr>
<td>NI-DCPower</td>
<td>Sequencing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Advanced Sequencing</td>
<td></td>
</tr>
<tr>
<td>NI-FGEN</td>
<td>Scripting, frequency lists, arbitrary sequences</td>
<td>Yes</td>
</tr>
<tr>
<td>NI-Scope</td>
<td>Multi-record acquisition</td>
<td>Yes</td>
</tr>
<tr>
<td>NI-HSDIO</td>
<td>Scripting</td>
<td>Yes</td>
</tr>
<tr>
<td>NI-Digital</td>
<td>Pattern bursting</td>
<td>Yes</td>
</tr>
<tr>
<td>NI-SWITCH</td>
<td>Scanning</td>
<td></td>
</tr>
</tbody>
</table>

- **Speed**
- **Multi-device synchronization**
- **Minimum jitter**
- **Frees up host**
Trigger routing

**PXI Backplane**
8 lines per segment
3 segments with bridges between them

**Terminology**

**Triggers** are consumed by a device.
“Measure Trigger”

**Events** are generated by a device.
“Measure Complete Event”

Two methods of routing triggers
1. Manual (old and bad)
2. Dynamic / string-based routing (new and good)
Manual Trigger Routing

1. Choose and reserve specific trigger line in MAX
2. Configure segment bridging in MAX
3. Device 1: Export source complete event to specific trigger line
4. Device 2: Import trigger from specific trigger line

Use when you have no choice
- Driver(s) don’t support string based routing
- Third party devices
- External boxes (route through PXI DAQ or other)
Dynamic Trigger Routing

1. Choose trigger source terminal

- Routing system selects free PXI Trigger line
- Routing system configures PXI segment bridges
- Routing system reserves PXI Trigger line
  - Only while in use
  - Prevent double driving

Tip: abstract device name for code portability
Dynamic Trigger Routing

How do I know what the valid terminal strings are?

MAX

I/O controls
Software Triggers

niScope Configure Trigger Software  
niScope Send Software Trigger Edge

Software triggering for optimization
- Initiate to put device into running state and waiting on a software command
- Send a software trigger: faster than calling initiate

Use it to override a hardware trigger
Multi-session Synchronization (DCPower)

**CH0 (master)**
- **initialize**
- **configure**
- **commit**
- **initiate**

**CH1 (slave)**
- **initialize**
- **configure**
- **commit**
- **initiate**

**CH0 Trigger Settings**
- Start Trigger: None
- Measure Trigger: Automatically after source complete

**CH1 Trigger Settings**
- Start Trigger: From CH0
- Measure Trigger: From CH0
NI-TClk: High-Speed Synchronization Solution

- **Challenge**
  - Synchronizing high speed-clocks (>100MHz) provides challenges because of propagation delays

- **Solution**
  - NI-TClk secret sauce (see whitepaper)

- **Result**
  - Precise: Timing and synchronization with sub-nanosecond accuracy
  - Flexible: same or different clock on Instruments
  - Easy to use: Only 3 VIs or function calls necessary

NI-TCIk: High-Speed Synchronization Solution

- User configures the devices normally
- Three simple VI’s cover most use cases
- Signal routing done behind the scene
Data Transfer
Data Streaming

Continuous Acquisition

- Analyze data as it is acquired
- Reduce memory use by breaking up acquisition
- Record high bandwidth signals for offline-analysis

Continuous Generation

- Calculate data as it is generated
- Reduce memory use by breaking up waveform download
- Play back recorded (real-life) signals

Architecture of a Typical Data Streaming System
Data Streaming

- Fetch often to prevent overflows
  - Onboard memory or DMA buffers
  - Tip: You can configure size of DMA buffers
- Peer To Peer Streaming
  - NI-SCOPE, NI-FGEN, FlexRIO...
  - Devices DMA directly to/from each other
  - I.e: FlexRIO generates/processes data
  - No CPU usage!
- TDMS Streaming
  - File format
  - Optimized for streaming
Data Streaming

- Modular Instruments can stream at high speeds
  - NI PXIe-5122 (100MS/s, 14-bit) : ≤400 MB/s
  - NI PXIe-5162 (>1.25 GS/s, 10-bit) : ≤800 MB/s
  - NI 5171R (250 MS/s, 14-bit, 8 ch) : >3.2 GB/s

- Streaming resources
  - www.ni.com/white-paper/7665/en
Questions