VeriStand

2023-05-23

Contents

What Is VeriStand and Who Is It For? . . . 7
VeriStand New Features and Changes . . . 9
Activating Your VeriStand Software . . . 27
- Activation FAQ . . . 27
- VeriStand Licensing Options . . . 29
- Activating Software . . . 31
- Deactivating Software . . . 34

Configuring the System . . . 35
- VeriStand Environment . . . 35
  - Components of a VeriStand Project . . . 37
  - APIs in VeriStand . . . 52
  - Keyboard Shortcuts . . . 52
  - VeriStand Directories and Aliases . . . 57
  - VeriStand File Extensions . . . 59
  - NI Hardware Support . . . 62
- Creating a New Project . . . 65
- Configuring a Project File . . . 66
  - Creating a VI Source Distribution . . . 66
  - Adding Custom Files . . . 68
  - Connecting Multiple Hosts to the Same Target . . . 69
- Integrating and Executing Models . . . 69
  - Models FAQs . . . 70
  - Using Models from C and C++ . . . 72
  - Using Models from Simulink . . . 75
  - Using Models from LabVIEW VIs . . . 84
  - FMI Support . . . 88
  - Controlling and Monitoring Model Execution . . . 91
  - Common Issues with Models in VeriStand . . . 94
- Configuring a System Definition File . . . 96
  - Creating a New System Definition File . . . 97
  - Adding and Activating a System Definition File . . . 97
  - Versioning a System Definition File . . . 98
  - Connecting to a Target System Definition . . . 99

Specifying a Target. 99
Configuring the VeriStand Engine. 100
Adding and Configuring a Procedure. 100
Adding and Configuring Alarms. 101
Adding and Configuring a Hardware Device. 107
Adding and Configuring a Custom Device. 158
Adding and Configuring a Model. 158
Running Tests. 181
Deploying the System Definition File to a Real-Time Target. 181
Downloading Support Files in MAX. 182
Running the VeriStand Gateway Silently. 182
Individual Target Management. 184
Configuring the Watchdog Timer when Deploying to a Real-Time Target. . . . 185
Running VeriStand Operations Using the Command Line. 186
VeriStand .NET Reference. 188
Data Logging Options. 189
Using NI-XNET Interfaces. 196
NI-XNET Overview. 197
Adding NI-XNET Databases. 199
Editing NI-XNET Databases. 200
Importing NI-XNET Frames. 201
Using NI-XNET Frame IDs. 202
Accessing Timing and ID Information for Incoming NI-XNET Frames. 203
Logging Incoming NI-XNET Frames. 205
Replaying Logged NI-XNET CAN Frame Data. 208
Configuring NI-XNET CAN Cyclic Frame Faulting. 211
Configuring Cyclic Redundancy Checks (CRCs) and Counters for Outgoing NI-XNET CAN Frames. 212
NI-XNET Bus Monitor. 214
How VeriStand Applies Scaling Factors to NI-XNET Signals. 215
ASAM XIL API - Generic Simulator Interface. 215
Accessing the VeriStand ASAM XIL Testbench. 216
Implementation Differences and Limitations with the ASAM XIL Interface. . . 216
ASAM XIL Framework C# Access. 217

Configuring the ASAM XIL Framework. 218
ASAM XIL Port Configuration Tag Reference. 218
Creating Real-Time Test Scenarios with Stimulus Profiles and Real-Time Test Sequences. 225
Navigating the Stimulus Profile Editor Environment. 226
Creating Stimulus Profiles. 229
Creating Real-Time Sequences. 257
Viewing Stimulus Profile Test Results. 319
Logging Real-Time Test Data with the Stimulus Profile Editor. 322
Communicating with the VeriStand Editor Using Stimulus Profile Arguments. 326
Getting Started with the Stimulus Profile Editor Tutorial. 326
Debugging the System. 361
VeriStand Error Codes. 361
Viewing Real-Time Target Logs. 382
Related Documentation. 383
Maximizing System Performance. 384
Streamlining the System Definition. 384
Configuring the BIOS Settings of the Controller. 386
Configuring the Ethernet Settings of the Controller. 387
Optimizing Hardware Performance. 388
Improving Model Performance. 388
Optimizing Reflective Memory. 389
Tailoring Channels. 393
System Channels. 393
Scaling a Channel on Hardware Devices. 398
Creating a Lookup Table Scale. 399
Creating a Polynomial Scale. 399
Creating a Thermocouple Scale. 400
Mapping Scales to Channels. 401
Importing Scales from Another Application. 402
Importing Scale Values from a Text File. 403
Faulting a Channel to a Specific Value. 404
Adding a User Channel. 405

Creating an Alias. 406
Mapping Channels and Aliases. 407
Adding a Calculated Channel. 408
Formula Calculated Channel Options. 409
Visualizing System State. 412
Viewing, Creating, and Interacting with Screens. 412
Adding and Configuring Components of a Screen. 413
Configuring Controls and Indicators to Send and Receive Data. 414
VeriStand Editor Tools. 416
Logging Data with the VeriStand Editor. 416
Viewing Model Values in the VeriStand Editor. 418
Importing and Managing Batches of Model Parameters with the VeriStand Editor. 418
Viewing Channel Values at Run Time. 420
Control Reference. 420
Actions Controls. 420
Array Controls. 422
Automotive Controls. 423
Booleans Controls. 423
Charts Controls. 424
Containers Controls. 425
Drawings Controls. 425
Numerics Controls. 426
Pull Downs Controls. 428
Text Controls. 429
Running the VeriStand Workspace. 429
Differences between Workspace and VeriStand Editor. 431
Adding and Configuring Controls and Indicators. 432
Modifying Control Mappings at Run Time. 433
Calibrating a Hardware Channel at Run Time. 433
Using Channel Value Forcing. 436
Logging Test Results with Stimulus Profiles. 436
Recording Commands VeriStand Sends to the Target. 437
Playing Back Commands Sent to the Target. 438

Setting Model Parameter Values in the Workspace. 439
Importing and Managing Batches of Model Parameters in the Workspace. . . 441
Displaying Waveform Data in a Graph. 445
Enhancing Your Workspace to View Data. 446
Configuring and Executing Host-Side Logging. 446
Adding Custom Software. 453
Logging Target Data with the Embedded Data Logger. 454
Creating Custom Devices. 456
Custom Device FAQs. 457
Custom Device Framework. 459
Planning a Custom Device. 497
Implementing a Custom Device. 523
Building a Custom Device. 553
Distributing Custom Devices. 555
Customizing an FPGA Target. 556
Copying the Sample FPGA VI and Project. 557
Customizing an FPGA VI. 558
Compiling a Custom FPGA VI into a Bitfile. 560
Creating a Custom FPGA Configuration File. 560
Glossary. 579

What Is VeriStand and Who Is It For?

VeriStand is software that engineers use to validate hardware and perform embedded software tests for Hardware-in-the-Loop applications. Use VeriStand to accelerate your product development with model integration, real-time stimulus generation, and an extensible open software environment.

Key Features

The features that set VeriStand apart.

- Import simulation models and control algorithms
- Configure real-time I/O
- Configure stimulus generation and deploy real-time tests
- Enable test automation with .NET, TestStand, Python, ASAM XIL, and other software
- Create custom plugins with an extensible ecosystem
- Respond to events with configurable alarms
- Configure a user interface during execution
- Monitor application data, alarm states, and system execution metrics

Components of a VeriStand System

A VeriStand system includes VeriStand, test hardware, add-ons, and extensions.

Component Type	Component	Notes
Software	VeriStand	Install VeriStand and recommended drivers using NI Package Manager.
Hardware	Real-time hardware device	Use any hardware that runs NI Linux Real-Time (Intel x64-

Component Type	Component	Notes
		based). For more information, refer to the release notes.
Optional Add-ons	VeriStand custom devices Simulink models	Install VeriStand custom devices using NI Package Manager and GitHub. For more information, refer to NI GitHub. You can also develop your own custom devices. For more information refer to the **VeriStand Custom Device Handbook**. To use Simulink CPU models, refer to the **VeriStand Model Generation Support** toolbox. For Simulink FPGA targeted models, refer to **HDL Coder Support Package for NI FPGA Hardware**.
Optional Extensions	System Definition API Execution API	Use the System Definition API to automate the configuration of a VeriStand system. Use the Execution API to automate the orchestration of a VeriStand system. You can access these APIs from C#, Python, and LabVIEW.

VeriStand New Features and Changes

Learn about updates — including new features and behavior changes — introduced in each version of VeriStand.

VeriStand 2023 Q2 New Features

VeriStand Engine Performance

In System Explorer, the new default for the PCL DAQ timing source is **Signal From Task (Sample Complete)**. This default applies to new projects when the timing source is set to **DAQ Timing** or **Automatic Timing** and there is a NI-DAQ device with at least one analog input hardware-timed single point channel.

The Signal From Task (Sample Complete) default option sends a tick to the PCL each time the Master DAQ device finishes acquiring an AI channel sample. This setting is beneficial for multiplexed DAQ modules with many channels. The setting has no impact on simultaneous sampled module behavior.

Other VeriStand Engine improvements include loop processing reductions for systems with a high count of channels, mappings, and faultable channels. Large systems, such as those with more than 20,000 channels and 10,000 mappings, can experience HP Loop duration reductions of more than 50 microseconds.

VeriStand Model Generation Support for Importing Signals

Use VeriStand Model Generation Support to import signals from Simulink models as channels. For more information, refer to How VeriStand Imports Models from the Model Generation Support MATLAB Add-on and the VeriStand Model Generation Support add-on documentation.

ESI File Importing

Import an EtherCAT Slave Information (ESI) file into VeriStand using the Scan Engine and EtherCAT Custom Device without installing LabVIEW. To begin importing the ESI file, open System Explorer and navigate to the custom device.

Screen Cluster Arrangement Options

Use additional cluster arrangement options on VeriStand screens to compact the appearance of large cluster hierarchies. To access these options in the VeriStand Editor, navigate to **File** » **Preferences** and click Screen to select a cluster arrangement.

IP to FPGA Conversion Utility supporting LV 2023 64-bit

IP to FPGA Conversion Utility 2023 Q2 now supports LabVIEW 2023 Q1 64-bit.

VeriStand 2023 Q1 New Features

LabVIEW 2023 Q1 Support

VeriStand is compatible with LabVIEW 2023 Q1 64-bit. All custom devices, LabVIEW models, and LabVIEW applications that contain the VeriStand API require this LabVIEW version. The VeriStand API will not install in previous versions of LabVIEW.

LabVIEW 2023 Q1 64-bit supports CompactRIO controllers and NI R Series modules. Custom devices and LabVIEW FPGA development can use LabVIEW 64-bit.



Note The HDL Coder Support Package for NI FPGA Hardware and IP to FPGA Conversion Utility only support LabVIEW 32-bit.

External Mode with XCP Communication

Enable external mode with XCP communication when building a MathWorks Simulink model to monitor signals and tune parameters from the Simulink environment. XCP provides additional capabilities such as viewing signals within referenced models. For more information, refer to the VeriStand Model Generation Support documentation on GitHub.

Terminal Specification for DAQmx Counter Operations

Use System Explorer or the System Definition .NET API to specify the following terminals for DAQmx counter tasks.

- Terminal for counter I/O tasks.
- A/B/Z terminal for position measurement tasks.

Specifying these terminals allows modules to use all counters through non-default PFI lines. For more information, refer to documentation for your module.

HDL Coder Support for PXIe R-series Kintex-7 and Simscape Models

HDL Coder Support Package for NI FPGA Hardware supports the following:

- PXIe R-series Kintex-7 modules — Use the support package to remain within HDL Coder while generating a bitfile from a Simulink model for compatible NI FPGA hardware.
- Simscape models.

VeriStand 2021 R3 New Features

Windows 11 Support

VeriStand now supports Microsoft Windows 11. For more information about NI support for Windows 11, refer to NI Product Compatibility for Microsoft Windows 11.

External Mode for Simulink Models

Enable **external mode** when building a model in MathWorks MATLAB Simulink to monitor signals in Simulink. To access external mode and relevant documentation, download the VeriStand Model Generation Support add-on.

For more information on VeriStand and Simulink models, refer to How VeriStand Imports Models from the Model Generation Support MATLAB Add-on.

HDL Coder Support Package for NI FPGA Hardware

HDL Coder Support Package for NI FPGA Hardware enables access to NI FPGA hardware from HDL Coder. Use this package to remain within HDL Coder while generating a bitfile from a Simulink model. That bitfile can then be imported into VeriStand with the FPGA Addon custom device. To access this package, visit GitHub.

IP to FPGA Conversion Utility

IP to FPGA Conversion Utility supports the conversion of Simulink models into NI FPGA bitfiles. This utility uses a command line interface to convert models without opening LabVIEW. For more information on the utility, refer to the supplemental documentation.

AIM MIL-STD-1553 Custom Device

Use the **AIM MIL-STD-1553** custom device to interact with AIM MIL-STD-1553 PXIe modules in VeriStand. To access this custom device, visit GitHub.

VeriStand 2021 R2 New Features

Deployment Command Center

Use the **Deployment Command Center** to execute HIL system validation tests by performing the following actions:

- Deploy and undeploy a system configuration.
- Connect and disconnect from local or remote gateways.
- Update gateway communication settings.
- Troubleshoot project and connection errors.

Access the Deployment Command Center from the VeriStand Editor by clicking the tab to the right of the home icon ().

Simulink Model Bus Support

Use virtual and non-virtual buses to organize Mathworks Simulink software models.

Virtual buses are user interface aids that do not generate code. **Non-virtual buses** group elements in memory and can generate code.

Bus element support in .vsmodel inports, outports, and parameters is limited to the following data types:

- Scalar
- Vector
- Numerical two-dimensional matrix

- Boolean two-dimensional matrix
- Scalar non-virtual bus

To access this support, generate models with version 1.1 of the VeriStand Model Generation Support add-on.

Custom Device LLB to PPL Conversion

Convert a LabVIEW library (LLB) custom device to packed project library (PPL) for the following benefits:

- Improve deployment time by creating a smaller disk footprint.
- Avoid load-time conflicts by granting each packaged custom device a copy of shared VI dependencies.
- Avoid internal naming conflicts by preserving the file hierarchy.

Use inline and inline-async templates to convert your custom devices to PPL and use LabVIEW advanced packaging features. You can access these templates and relevant documentation from GitHub.

For more information, refer to the Custom Device Developer Handbook.

AIM ARINC 429 Custom Device

Use the AIM ARINC 429 custom device to interact with ARINC 429 PXIe modules in VeriStand using the AIM API. You can access this custom device from GitHub.

VeriStand 2021 New Features

VeriStand Editor

- Add, delete, and configure alarms.
- Use the Target Log Viewer to troubleshoot unexpected engine and custom device behavior.

- View and adjust the target rate of a timing source from the VeriStand Editor configuration pane by clicking **Document**.

Supported Environments

VeriStand now supports the following LabVIEW environments.

- LabVIEW 64-bit
- LabVIEW RT 2021



Note All custom devices should be compiled in 64-bit.

VeriStand no longer supports the following real-time environments.

- PharLap
- Linux ARM Targets

Custom Devices

Build better custom devices with the following resources.

- Use the Custom Device Developer Handbook to extend the VeriStand environment with LabVIEW. The handbook provides background, design decisions, and technical information required to develop custom devices.
- Use templates to convert your custom devices to PPL and use LabVIEW advanced packaging features. You can access these templates and relevant documentation from GitHub.

VeriStand Model Generation Support

Use VeriStand Model Generation Support MATLAB add-on to create VeriStand compatible models within the MathWorks Simulink environment. Access this support natively from Simulink.

VeriStand 2020 R6 New Features

VeriStand Editor

Set the default appearance of channels on the screen. Click **File** » **Preferences** and select **Screen** to set the channel label location and cluster arrangement.

FPGA Addon Custom Device

FPGA Addon Custom Devices now support all FXP datatypes that are also supported by LabVIEW. You can access this custom device from GitHub.

NI-SWITCH Custom Device

Use the G scripting API to modify a NI-SWITCH Custom Device. For more information, refer to Scripting Examples.lvproj in <Application Data>\LabVIEW 20xx\examples\NI VeriStand Custom Devices\Routing and Faulting. You can access this custom device from GitHub.

VeriStand 2020 R5 New Features

VeriStand Editor

- Modify non-active mapping diagrams while a system definition is deployed. Editing in other tabs will not disturb the deployed system.
- Create duplicates of a system definition file. Use copies of a system definition to easily deploy and test modifications without losing the original file.
- Monitor array-type channels while the project is running.
- Use C# to customize and extend the VeriStand Editor. You can access examples of implemented C# code from GitHub.

Control Keyboard Shortcut Behavior from the Command Line

Use the noDeployKeys command to prevent the use of the deploy and undeploy keyboard shortcuts during a session. You can still deploy and undeploy from the Operate menu.

FPGA Add-on Custom Device

- Use the Scripting API to turn FPGA bitfiles into flexible and reusable system definition files.
- Load and reload bitfiles without losing a previous configuration. Scalars and waveforms from a former bitfile remain imported while those that no longer exist are removed from the new bitfile.

You can access this custom device from GitHub.

Collect Memory Usage Data with the Telemetry Custom Device

Use the Telemetry Custom Device to understand the CPU and RAM usage of your system. You can access this custom device from GitHub.

VeriStand 2020 R4 New Features

Launching VeriStand Silently

Deploy a system definition without launching the VeriStand Editor or System Explorer using the Windows Run command or from a language of your choice.

Instrument Add-on Custom Device

Use the Instrument Add-on custom device API to create a new instrument from a previously exported configuration. You can also use the API to add messages and to configure the command and response functionality. For more information, refer to the custom device quick start documentation in GitHub.

ASAM XIL Steps for TestStand

Use the <UndeployVeriStandProjectOnDisconnect> tag in the port configuration XML file to specify if VeriStand is undeployed when the Framework is cleaned up.

Alarm Status Channel

Monitor the state of one or more alarms in the VeriStand engine with an alarm status channel. You can use these channels in custom devices and real-time sequences to determine what actions to take when an alarm state changes.

Stopping Groups of Real-Time Sequences with Procedures

Create a procedure to stop a specified group of real-time sequences and skip to their clean-up sections. You can use the procedure to shut down sequences related to a specific device under test (DUT) in a multi-DUT scenario. Use the Stimulus Profile Editor to assign a group to a real-time sequence call step.

VeriStand 2020 R3 New Features

VeriStand Editor

- On the Mapping Diagram, use the Focus command (<Ctrl+,>) to emphasize all mappings and nodes connected to a selected item. Using Focus will

highlight all directly connected components while fading the rest of the system.

> **Note** You can interact with faded items while using Focus.

- In the System Definition pane, right-click an item and select **Locate in System Explorer** to navigate to the place in System Explorer where that item is configured.

XNET Status Channels

Monitor the XNET bus statuses for CAN, LIN, and FlexRay from VeriStand. In System Explorer, right-click an XNET port and select **Add Port Specific Channels** to add status channels to the system definition.

FMU Fixed Parameters

Set initial values for Functional Mockup Unit (FMU) initialization parameters during deployment through a text file. You can also set fixed parameter values before running a deployed model.

FPGA Add-on Custom Device

Use the FPGA Add-on custom device to run an FPGA bitfile without implementing the normal VeriStand FPGA template. The add-on allows the transfer of basic scalar data types inline with VeriStand's primary control loop (PCL) and supports reading and writing waveforms with FPGA DMA channels. You can access this custom device from GitHub.

Instrument Add-on Custom Device

Use the Instrument Add-on custom device to communicate with instrumentation, such as power supplies, environmental chambers, and emissions equipment, over various protocols. The device supports serial, GPIB (using NI VISA), ethernet (TCP,

UDP, and NI VISA), AK serial, and AK TCP. You can access this custom device from GitHub.

VeriStand 2020 R2 New Features

VeriStand Editor

- Manage aliases and user channels without using System Explorer.
- Use the Model Signal Viewer to check signal values while a project is running.
- Receive warnings when a disk model is out of sync with the loaded system definition model.

FlexRay

Enable the FlexRay Allow Passive to Active property to permit the transition from the normal-passive to the normal-active state.

CAN

Create channels to view timing information for individual CAN multiplexer modes independent of the overall frame.

VeriStand 2020 New Features

Mapping Diagram

The following tasks can be completed on the Mapping Diagram.

- Manage Calculated Channels — You can now create, modify, and delete calculated channels from the Mapping Diagram.

- View unmapped channels and mappings — Use the Mapping Diagram Table View to see unmapped channels and mappings in the system definition. You can filter the table by selecting a node or wire.

Embedded Data Logger Custom Device Available on GitHub

The Embedded Data Logger is now an open-source custom device that installs separately from VeriStand. You can access this custom device from GitHub. The 20.0 version includes the ability to automatically split and archive log files.

Scripting Routing and Faulting Custom Device Elements

Use the LabVIEW API to script Routing and Faulting Custom Device elements and automate fault insertion in VeriStand. The API is supported in LabVIEW 2017 and newer.

Support for DAQmx Waveform Logging

VeriStand now supports DAQmx Waveform Tasks on CompactRIO with DAQmx Controllers. This enables per-slot support with DAQmx, FPGA, XNET, and Scan Engine on supported controllers.

CAN FD Scripting

The VeriStand .NET API now supports scripting CAN ports with CAN FD baud rates and custom bitfields.

VeriStand 2019 R3 New Features

Mapping Diagram

The following tasks can be completed on the Mapping Diagram.

- Add and configure simulation models on the diagram from the palette.
- Evaluate items, such as channels and mappings, that have been added or removed on the Mapping Diagram. These items are highlighted after completing actions such as swapping a model or altering the system definition.

MathWorks Simulink Software Model Toolchain Support

VeriStand now supports models from Simulink software versions R2019a and R2019b for Windows and Linux only. VeriStand also supports compiling Simulink models for Windows using a MinGW-w64 compiler.

TDMS File Viewer Support for DIAdem

VeriStand now integrates DIAdem as its default TDMS file viewer.

Routing and Faulting Custom Device

Use the Routing and Faulting Custom Device's switching and fault insertion modules to inject faults.

The Routing and Faulting Custom Device also supports SLSC Switch hardware and additional hardware. The SLSC Switch Custom Device is a VeriStand add-on that supports SLSC routing modules. The following modules are supported:

- SLSC-12251
- SLSC-12252

- SET-2010

SLSC Switch routing modules can be found in System Explorer under **Targets** » **Controller** » **Hardware** » **SLSC** » **SLSC Chassis** » **Modules** » **Slot (n)**.

VeriStand 2019 R2 New Features

The following tasks can be completed on the **Mapping Diagram**.

- Map channels and aliases — Use wires to connect and disconnect mappings.
- Specify a target — Select a target and use the **Document** pane to designate the name, operating system, and IP address.
- Remove a model or calculated channel — Click and delete a model or calculated channel to remove it from the system definition.
- Customize terminal placement — Right-click a node and hover over **Terminal Placement** to choose the side of the node that inputs and outputs appear.
- Change a channel's name — Right-click a channel and select **Rename Channel**.
- Reorder channels — Click, hold, and drag the left corner of a channel to rearrange the list. You can also click a channel and use the Search dialog box to select another channel to switch them.

VeriStand 2019 New Features

VeriStand Editor

You can now use the **VeriStand Editor** to control aspects of your project. The UI Manager and Project Explorer windows from previous releases of VeriStand have been combined into one editor. Many of the features of the Project Explorer window can now be accessed through **Project Files** in the **Navigation pane** of the VeriStand Editor. The VeriStand Editor also contains the Mapping Diagram to help visualize channel mappings.

Note You can access the interface from previous versions of VeriStand by launching the VeriStand Project Explorer from the x86\NI\VeriStand 2019 directory.

Mapping Diagram

You can now visualize software mappable points within a VeriStand project with the Mapping Diagram. This graphical diagram allows you to see all VeriStand channels and the mappings between them.

Groups of mappings between two nodes are automatically bundled into a single wire by default to reduce clutter. If a wire represents multiple mappings, you can select the bundle to see more information in the **Configuration pane**.

Other features of the Mapping Diagram include the ability to:

- Display aliases and user channels
- Validate and debug mappings

PXI NI Linux RT Support

You can now deploy System Definition Files to PXI NI Linux RT targets similarly to previously supported targets. Benefits of supporting the NI Linux RTOS include:

- Support for newer compilation tools and advance C++ features.
- Support for Linux and other libraries.
- Improvement of memory handling and computational power.

Custom Device Support

VeriStand 2019 officially supports the following custom devices.

- Engine Simulation Toolkit — Provides a configuration-based experience for validating engine control units (ECUs).
- Scan Engine and EtherCAT — Supports easily read scanned I/O from C series modules in a CompactRIO or NI 914x EtherCAT chassis. This custom device also supports custom FPGA personalities to be used with a 914x chassis.
- SLSC 12201 DIO Module Custom — Integrates the SLSC-12201 33 V Digital I/O Conditioning Module into VeriStand.
- SLSC EDS Custom Device — Allows any generic SLSC Capabilities file to be used in VeriStand.
- Synchronization Custom Device — Synchronizes RT system time and PXI chassis clocks.

These custom devices can be downloaded with VeriStand 2019 using NI Package Manager. Check NI GitHub for updates and other custom devices.

CAN FD Support

VeriStand 2019 supports XNET implementation for the Controller Area Network Flexible Data-Rate (CAN FD) protocol. With a XNET implementation, you can now use databases that support CAN FD and CAN FD with Baud Rate Switching I/O modes. This implementation also allows calculating and using custom baud rates for a CAN interface. For more information on CAN FD, see Understanding CAN with Flexible Data-Rate (CAN FD).

FMI Support

VeriStand now has limited support for the Functional Mockup Interface (FMI) API standard on Windows and PXI LinuxRT. Use the FMI standard to create tool agnostic, portable solutions between modeling and simulation environments. VeriStand enables the configuration and execution of FMI 2.0 CoSimulation models on host and National Instruments Real-Time Linux64 systems.



Note This support requires the Functional Mockup Units (FMUs) to have the proper executable binaries available.

Activating Your VeriStand Software

Activate VeriStand on your computer.

After installing VeriStand, you must use a license to activate the software. For more information on installing your version of VeriStand, refer to the release notes.

1. Activation FAQ
   Refer to these common VeriStand activation questions.
2. VeriStand Licensing Options
   NI offers a variety of licenses for the different ways you can use VeriStand in development and deployment applications.
3. Activating Software
   Activate software products using NI License Manager.
4. Deactivating Software
   Use License Manager to deactivate a software license on a computer or virtual machine.

Activation FAQ

Refer to these common VeriStand activation questions.

What Is Activation?

Activation is the process of obtaining an activation code to enable your software to run on your computer. An activation code is an alphanumeric string that verifies the software, version, and computer ID to enable features on your computer. Activation codes are unique and are valid on only one computer.

What Is the NI Licensing Wizard?

The NI Licensing Wizard is a part of NI License Manager that leads you through the process of enabling software to run on your machine.

What Information Do I Need to Activate My NI Software?

You need your NI User account log-in, the product version and serial number, and a computer ID that uniquely identifies your computer. Certain activation methods may require additional information for delivery. This information is used only to activate your product. For complete disclosure of the NI software licensing information privacy policy, refer to the National Instruments Corporation Privacy Statement. If you optionally choose to register your software, your information is protected under the NI privacy policy, available at ni.com/privacy.

How Do I Find My Product Serial Number?

Your serial number uniquely identifies your purchase of NI software. For help finding the serial number on the Certificate of Ownership included in your software kit, refer to Finding the Serial Number or System Tag of My NI Product.

If you have installed a previous version using your serial number, you can find the serial number by selecting **Help** » **About** within the application.

What Is a Computer ID?

The computer ID is a 16-character value that uniquely identifies your computer. NI requires this information to enable your software. You can find your computer ID through the NI Licensing Wizard or by using NI License Manager, as follows:

- Launch the NI License Manager.
- Click **Computer Information** in the ribbon.

For more information about product activation and licensing, refer to **Activating NI Software Products**.

How Can I Evaluate NI Software?

You can evaluate most NI products in accordance with the license agreement. Evaluation terms vary, depending on which product you want to evaluate. Refer to your product documentation for specific information on the product's evaluation mode.

Moving Software after Activation

To transfer your software to another computer, uninstall the software on the first computer, then install and activate it on the second computer. You can transfer your software from one computer to another; you do not need to contact or inform NI of the transfer. Because activation codes are unique to each computer, you will need a new activation code. Refer to the **How do I Activate my Software?** section to learn how to acquire a new activation code and reactivate your software.

Deactivating a Product

To deactivate a product, open NI License Manager and navigate to the **Local Licenses** view. Select the product to be deactivated and click **Deactivate**. If the product was in evaluation mode before activation, the properties of the evaluation mode may not be restored.

If you cannot access NI License Manager on the host computer, contact NI to deactivate the product.

Using Windows Guest Accounts

NI License Manager does not support Microsoft Windows Guest accounts. You must log in to a non-Guest account to run licensed NI application software.

VeriStand Licensing Options

NI offers a variety of licenses for the different ways you can use VeriStand in development and deployment applications.

Licensing Options

Use the following descriptions to determine the VeriStand licensing option that fits your needs.



Note For more information on activating VeriStand licenses, refer to ni.com/activate. To purchase a VeriStand license, refer to ni.com/veristand. For questions about specific licensing needs, contact an NI representative.

License	Description
Evaluation Mode	Software runs as a Full Development system for 7 days. The evaluation period can be extended to 45 days. You can activate a VeriStand license at any point during or after the evaluation period.
Full Development	Enables full VeriStand functionality.
PC Development	Enables you to run simulations on a desktop PC.
Operator	Enables you to configure a project file by determining the preconfigured system definition file that runs on the target and defining the test environment that an operator interacts with.

Comparing License Option Features

The following table displays the features available with the Full Development, PC Development, and Operator licenses.



Note The Evaluation Package includes the same features as the Full Development license.

Feature	Full Development	PC Development	Operator
View system definition files as read-only	No	No	Yes
Create new system definition files	Yes	Yes	No
Configure supported NI hardware devices	Yes	Yes	No
Configure custom devices	Yes	Yes	No
Configure models	Yes	Yes	No
Add targets	Yes	Yes	No
Add user channels	Yes	Yes	No

Feature	Full Development	PC Development	Operator
Add calculated channels	Yes	Yes	No
Map channels	Yes	Yes	No
Add alarms	Yes	Yes	No
Add procedures	Yes	Yes	No
Deploy system definition file to a real-time target	Yes	No	Yes
Add standard and custom Tools menu utilities	Yes	Yes	Yes
Configure alarm responses	Yes	Yes	Yes
Add custom files to a project	Yes	Yes	Yes
Export channel resources	Yes	Yes	Yes
Create stimulus profiles	Yes	Yes	Yes
Run stimulus profiles	Yes	Yes	Yes
Log data	Yes	Yes	Yes
Run compiled models	Yes	Yes	Yes

Note You must install MathWorks Simulink software to run uncompiled models.

Activating Software

Activate software products using NI License Manager.

Before you begin, purchase a license for NI software and create a user account on ni.com.

If you are transferring software to another endpoint, review the NI Software License Agreement for information regarding transfer and seat limitations.

Note If you use NI Volume License Manager to manage NI software under a volume license agreement (VLA), notify your volume license administrator before activating your software on another computer.

1. Click **Activate Software**.
2. Activate NI software.

Option	Activation Type	Description
Check my account for licenses	Online	Log in to your NI user account to check for licenses associated with your account to activate software. If your account lacks the appropriate license for software, activate the product using an activation code or a serial number.
Enter a serial number	Online	Log in to your NI user account and enter a serial number to activate software. Find serial numbers in the following locations: ▪ On the Certificate of Ownership included in your software kit. ▪ On the product's packing slip or shipping label. ▪ On a previously installed version of your application software, by selecting **Help** » **About**. **Note** Visit the NI Info Codes web page and enter the Info Code

Option	Activation Type	Description
		SerialNumbers_en for the latest information regarding locating a serial number
Enter activation codes	Online/Offline	Enter an activation code to activate software. To receive an activation code, log in to your NI user account and click **Generate an activation code** or visit ni.com/activate. **Note** Your code is sent to your email inbox after you submit the Activation Code Generation form.
Connect to a volume license server	Online/Offline	Provide the volume license server address where NI software should check for licenses. **Note** For home use through a volume license agreement, request a license file from your volume license administrator.



Note If you were using a trial version of NI software before you began the activation process, you may need to restart the software.

License Manager attempts to automatically renew licenses when their subscription term ends.

Use the software immediately after activation.

Deactivating Software

Use License Manager to deactivate a software license on a computer or virtual machine.

If you intend to transfer software to another endpoint, review the NI Software License Agreement for information regarding transfer and seat limitations.



Note If you use NI Volume License Manager to manage NI software under a volume license agreement (VLA), notify your volume license administrator before transferring your software to another computer.

Contact NI to deactivate software offline.

1. Click **Local Licenses**.
2. Select a licensed software.
3. In the **Actions** section, click **Deactivate**.
4. In the **Deactivate** dialog box, click **Yes**.



Note Once software is deactivated, reuse the activation code to activate the software on another endpoint.

Configuring the System

Manage the system to fit your needs.

1. VeriStand Environment
   The VeriStand environment is configuration-based, consisting of several dialog boxes and windows that allow you to modify various components of a project.
2. Creating a New Project
   Create a new project in VeriStand to develop, prototype, and test control systems using hardware I/O and simulation models.
3. Configuring a Project File
   Configure a VeriStand project to complete tasks such as adding Tools menu items, services, alarm responses, and custom files.
4. Integrating and Executing Models
   Use models to mathematically represent real-world systems in your VeriStand project.
5. Configuring a System Definition File
   Modify a system definition file to complete tasks such as configuring the VeriStand engine, adding models, and creating aliases.

VeriStand Environment

The VeriStand environment is configuration-based, consisting of several dialog boxes and windows that allow you to modify various components of a project.

The following table describes the windows that make up the VeriStand environment.

Window	Common Tasks	How to Access
VeriStand Editor	▪ Accessing the system definition file ▪ Creating screens that serve as the user interface	Open a VeriStand project.

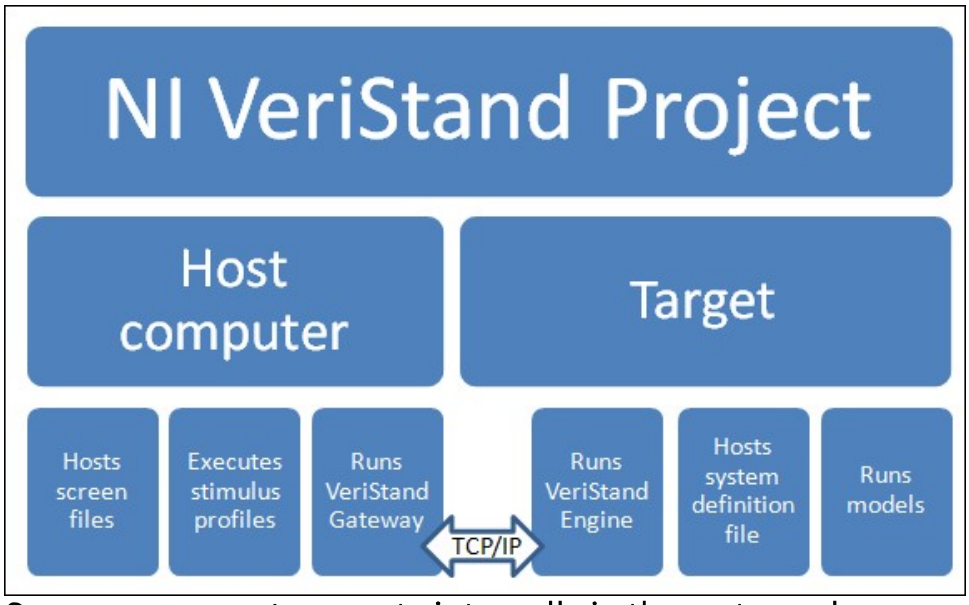
Window	Common Tasks	How to Access
	▪ Monitoring system data displaying in user interface indicators ▪ Manipulating variables, such as model parameters, through user interface controls ▪ Creating mappings to connect channels ▪ Interacting with various tools designed for monitoring alarms, viewing channel data, or running stimulus profiles ▪ Testing system behavior using channel faulting and error monitoring tools ▪ Controlling the execution of your system ▪ Launching System Explorer and the Workspace ▪ Configuring and running stimulus profiles ▪ Configuring services ▪ Configuring alarm responses	
System Explorer	Creating and modifying system definition files by: ▪ Representing the hardware and software components in the system	▪ Double-click the system definition file (.nivssdf) in the **Project Files** pane of the VeriStand Editor. ▪ Launch a system definition file from

Window	Common Tasks	How to Access
	▪ Creating alarms and procedures that execute on the target ▪ Creating mappings to connect channels ▪ Configuring settings of the VeriStand Engine, such as the rate at which the system runs	**Project Files** and click **Configure**.
Workspace	▪ Creating screens that serve as the user interface with which an operator interacts ▪ Monitoring system data displaying in user interface indicators ▪ Manipulating variables, such as model parameters, through user interface controls ▪ Interacting with various tools designed for monitoring alarms, viewing channel data, scaling and calibrating channels, or running stimulus profiles	Double-click **Workspace** in the **Project Files** pane of the VeriStand Editor.

Components of a VeriStand Project

A VeriStand project contains at least one project file (.nivsprj), one system definition file (.nivssdf), and one screen file (.nivsscr or .nivsscreen). These files are used to configure, deploy, and interact with your system.

The following figure illustrates the locations of these files and major components of an VeriStand project.



Some components operate internally in the system when you run a project. Other components are user-visible features you create and configure in the VeriStand environment.

Note The host computer and deployment target can be the same desktop PC. In this situation, you deploy a system definition file to host desktop PC like you would deploy the file to a remote target.

Related concepts:

- Host Computer
- Deployment Target
- Development Computer

Host Computer

A host computer hosts the screen files that serve as the user interface for operators and runs the VeriStand Gateway.

Note The host computer must be a PC running a supported version of Windows.

Internal Feature

VeriStand Gateway—Creates a TCP/IP communication channel that facilitates communication with the VeriStand Engine over the network. The VeriStand Gateway receives channel values from the VeriStand Engine and stores them in a table. You can view these values using the Channel Data Viewer tab in VeriStand Editor or the Channel Data Viewer workspace tool. If you run a project on a desktop PC, the VeriStand Gateway initiates the VeriStand Engine.

If you run a project on an real-time target, the VeriStand Gateway synchronizes with the system definition file that is running on the RT target. If the system definition file currently running on the VeriStand Engine does not match the system definition that the VeriStand Gateway expects, then the VeriStand Gateway does not synchronize with the system definition file running on the RT target.

Interactive Features

The following are features that you can modify.

- Project File—The .nivsprj file that defines high-level settings, such as:
  - The screen and system definition files to run
  - Available users and their permissions for the project
  - The list of tools you can launch from the **Tools Launcher**
  - Which services run when you deploy a project to the target
  - The IP address of the VeriStand Gateway
  - Stimulus profiles and real-time sequences
- Screen File—The .nivsscr or .nivsscreen files that define the configuration and settings for the screens and display items you view in the VeriStand Editor or Workspace, respectively
- Stimulus profile—A test executive that can call real-time sequences, open and close VeriStand projects, and perform data-logging and pass/fail analysis. It also connects real-time sequences to system definition files to bind channel data within the system definition file to variables in the real-time sequence.

Stimulus profiles execute on the host computer. You create and run a stimulus profile using the Stimulus Profile Editor.

- National Instruments Driver Software—You need the appropriate driver software to communicate with hardware installed on a target. For a list of the required driver software, see the VeriStand Readme located at <Program Files>\National Instruments\NI VeriStand\readme\readme.html. If you installed VeriStand to a different location, locate the readme directory in the install location you specified.

Development Computer

The development computer is the computer that contains the VeriStand software. The computer on which you develop a NI VeriStand project might be different from the host computer in the system. To extend the functionality of VeriStand, you might also use the following NI products on the development computer:

- LabVIEW Development System—If you want to create custom devices, workspace controls/indicators, timing devices, and/or Tools menu utilities, you need the LabVIEW Development System.
- LabVIEW Real-Time Module—You need this module to use RT functions in custom device VIs.
- LabVIEW FPGA Module—If you add a National Instruments FPGA target to a project, it must have an associated FPGA bitfile. VeriStand provides FPGA bitfiles for certain FPGA devices. If you want to customize these FPGA bitfiles or create a custom FPGA bitfile for another FPGA target, you need the FPGA Module.

Deployment Target

The deployment target in an VeriStand system is a desktop PC or RT target on which you run the system definition file and VeriStand Engine.

Internal Feature

The following is a feature that you cannot directly modify.

- VeriStand Engine—The non-visible execution mechanism that controls the timing of the entire system and the communication between the target and

the host computer. The VeriStand Engine consists of multiple timed loops that use RT FIFOs to transfer data between the loops.

Note To deploy a system definition file to an RT target, you must first download support files for VeriStand to the target.

Interactive Features

The following are features that you can modify.

- System Definition File—The .nivssdf file you configure in the System Explorer window. A system definition file contains the configuration settings of the VeriStand Engine, including:
  - The rate at which the system runs.
  - DAQ devices, NI-XNET devices, FPGA targets, or reflective memory devices and the task and channel configurations for each.
  - Simulation models to execute, and the rate at which they execute.
  - The list of active alarms. You can use alarms to trigger actions on the target, such as procedures, or to display dialog boxes that alert the user of an event.
  - The list of procedures that can execute on the target. A procedure is a script of commands that define a set of actions in the VeriStand Engine.
  - The system mappings that determine how channels are connected.
  - The list of channels for data objects in the system. The following table displays common channel types.

Channel Type	Examples
Hardware I/O channels	DAQ, FPGA, etc.
Model channels	Inputs, outputs, parameters, and signals
User channels	Used to store or map user-defined values in the system
Calculated channels	Channels that represent the result of a user-defined calculation of other channels in the system

- Model—A mathematical representation of a real-world system. A model responds to stimuli by producing outputs in a way that emulates the behavior of the modeled item. Models contain inputs and outputs, called inports and outports, that communicate with other parts of the control system. Build models using several different modeling environments, and then integrate the model into a system definition file.

VeriStand Engine

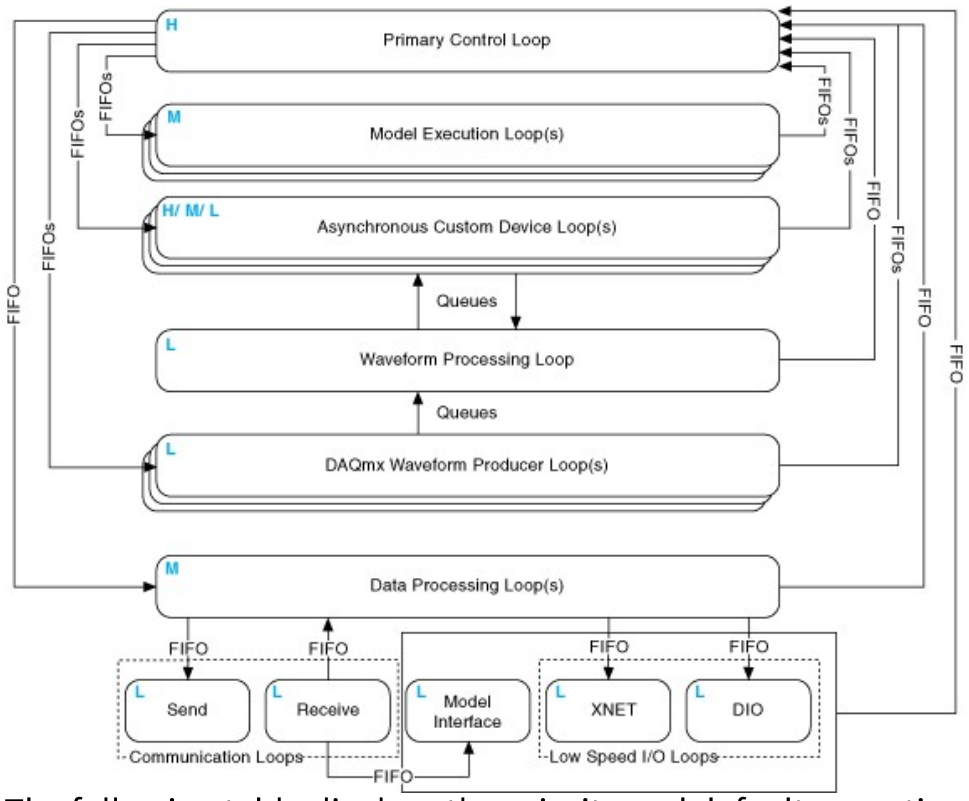
The VeriStand Engine is the execution mechanism that controls the timing of the entire system and the communication between the target and the host computer.

The VeriStand Engine executes hardware I/O, runs models, procedures, alarms, and tests, and computes values in the channel table based on the results of model execution and hardware I/O. This engine runs on either a desktop PC in simulation mode or as an embedded application on a RT system.

The VeriStand Engine consists of multiple timed loops that use real-time (RT) FIFOs to transfer data between the loops. Each loop performs designated tasks and has an assigned priority. Although you cannot change the priority or primary tasks of the engine loops, you can customize loop operations, such as the execution rate. The system definition file contains the configuration settings for the VeriStand Engine.

Note The VeriStand Engine determines which system definition file to run by communicating over the network with the VeriStand Gateway.

The following figure illustrates the operation of the VeriStand Engine.



The following table displays the priority and default execution rate for the loops of the VeriStand Engine.

VeriStand Engine loop	Description	Priority	Default execution rate
Primary Control Loop (PCL)	Controls the timing for the VeriStand Engine and maintains the most up-to-date table of channel values. Use System Explorer to set the execution mode of the PCL. Per iteration, the PCL executes the following tasks: ▪ Reads and writes high-speed FPGA I/O, analog and counter DAQ I/O,	High	100 Hz

VeriStand Engine loop	Description	Priority	Default execution rate
	and Asynchronous Custom Device Loop data. ▪ Applies scaling to the data. ▪ Executes one step of the real-time sequence of a test that is currently running ▪ Sends data to the Data Processing Loop to synchronize the table of channel values. ▪ Sends data to the Model Execution Loop(s). ▪ Prompts the Data Processing Loop, Model Execution Loop(s), and Asynchronous Custom Device Loop(s) to execute. ▪ Performs software fault insertion. ▪ Creates mapping connections. ▪ Executes inline custom devices.		

VeriStand Engine loop	Description	Priority	Default execution rate
	▪ Reads status information from the Waveform Processing Loop and DAQmx Waveform Producer Loop(s). **Note** For more information on the PCL, see Primary Control Loop Execution Steps		
Model Execution Loop(s)	Executes a corresponding compiled model. The number of models in the system definition determines the number of loops. Per iteration, each Model Execution Loop executes the following tasks: ▪ Reads the data sent by the PCL and maps this data to model inports. ▪ Executes one step of the model.	Medium	A decimation of the PCL rate

VeriStand Engine loop	Description	Priority	Default execution rate
	▪ Reads model outport values and sends this data to the PCL. ▪ Reads model inport signals and sends this data to the PCL. **Note** A Model Execution Loop handles high-speed, dynamic data associated with model inports and outports, while a Model Interface Loop reads and applies the lower-speed, asynchronous updates to model parameter values.		
Asynchronous Custom Device Loop(s)	Executes and transmits custom device inport data values per iteration of the PCL. The VeriStand Engine is responsible for	▪ High ▪ Medium ▪ Low	User-defined

VeriStand Engine loop	Description	Priority	Default execution rate
	initiating the Asynchronous Custom Device Loop.		
Waveform Processing Loop	Transfers waveform data through the system. Per iteration, the Waveform Processing Loop executes the following tasks: ▪ Reads waveform data from DAQmx Waveform Producer Loops. ▪ Sends waveform data to the VeriStand Gateway. ▪ Reads waveform data from custom devices. ▪ Sends waveform data to custom devices.	Low	Event driven
DAQmx Waveform Producer Loops	Acquires waveforms from DAQ devices. Each DAQmx Waveform Producer Loop corresponds to a waveform task in the system definition. Per iteration, Waveform Producer Loop executes the following tasks:	Low	10 Hz or user-defined

VeriStand Engine loop	Description	Priority	Default execution rate
	▪ Reads waveform data from analog input channels on DAQ devices. ▪ Sends waveform data to the Waveform Processing Loop. ▪ Logs acquired data to .tdms files.		
Data Processing Loop	Distributes the execution commands received by the Communication Receive Loop among the engine loops. Like the PCL, the Data Processing Loop maintains a complete copy of the channel values table. Per iteration, the Data Processing Loop executes the following tasks: ▪ Receives the table of channel values from the PCL. ▪ Executes procedures, alarms, and calculated channels. ▪ Transmits updated table of	Medium	A decimation of the PCL rate

VeriStand Engine loop	Description	Priority	Default execution rate
	channel values to the PCL. ▪ Sends data values to the Communication Send Loop.		
Communication Send Loop	Transmits channel values to the VeriStand Gateway.	Low	15 Hz
Communication Receive Loop	Listens for execution commands that the VeriStand Gateway sends.	Low	Event driven
XNET Loop	Reads and writes XNET data.	Low	100 Hz
DIO Loop	Reads and writes low-speed digital DAQ I/O data.	Low	A decimation of the PCL rate
Model Interface Loop	Reads and writes the lower-speed, asynchronous updates to model parameter values.	Low	Event driven

Primary Control Loop Execution Steps

The Primary Control Loop (PCL) controls the timing of the VeriStand Engine by performing several execution steps.

The PCL can run in **Parallel** mode or **Low Latency** mode. The difference between the modes is the timing of model-related steps:

- In **Parallel** mode, the PCL initiates execution of models and continues to its next iteration without waiting for models to finish executing. This causes a one-cycle delay between when a model executes and when the data it produces is available to the system
- In **Low** Latency mode, the PCL waits for the Model Execution Loop(s) to finish writing data to models before it reads and publishes model values to

the system. This occurs during every iteration of the system. When the model completes execution, the PCL provides data from the model to other loops during the same iteration that the model generated the data.

> **Note** NI recommends you select Low Latency mode only if you need to minimize the latency between your inputs, model execution, and outputs. Waiting for Model Execution Loops to read, execute, and write on each iteration can significantly slow the execution speed of the system.

You can use System Explorer to set the execution mode of the PCL.
The following table compares the execution steps of Parallel and Low Latency Modes.

> **Note** The VeriStand Engine in both modes executes inline custom devices in the order defined in the system definition file.

Step	Parallel mode	Low latency mode
1	Gets inputs from hardware devices in the system definition. **Note** If the system includes an inline hardware interface custom device, the PCL reads DAQ digital lines and counters after the **Read Data from HW** case of the custom device executes in step 3.	
2	Reads asynchronous custom device FIFOs from the previous iteration.	
3	Runs the **Read Data from HW** case of inline hardware interface custom devices. If you configured hardware scaling, VeriStand applies the scaling after acquiring all hardware inputs.	
4	Reads previous iteration data from models in the system definition. **Note** This step executes on the	—

Step	Parallel mode	Low latency mode
	second and subsequent iterations.	
5	Reads data from the previous iteration of the Data Processing Loop.	
6	Processes system mappings. **Note** VeriStand components (including custom devices) cannot read data from a previous step until the PCL processes system mappings, even if the previous step acquired the data the component needs.	
7	Runs the **Execute Model** case of inline model interface custom devices.	
8	Executes steps of running real-time sequences. **Note** VeriStand executes real-time sequences after input operations but before output operations and continues to run every step of the real-time sequence until the sequence is complete, reaches a Yield step, or completes an iteration of a loop with Auto Yield set to TRUE. If a sequence takes longer than the given time for an iteration of the PCL, the PCL runs late. To avoid errors, break up the timing of the steps by placing Yield steps throughout the sequence and enabling the Auto Yield property for any loops in the sequence.	
9	Processes system mappings.	
10	Writes data to models.	
11	Initiates asynchronous execution of models.	Initiates execution of models and waits for them to complete execution.
12	—	Reads data from models.
13	—	Processes system mappings.

Step	Parallel mode	Low latency mode
14	Writes data to the Data Processing Loop.	
15	Writes output data to hardware devices.	
16	Runs the **Write Data to HW** case of inline hardware interface custom devices.	
17	Writes data to asynchronous custom device FIFOs.	

APIs in VeriStand

VeriStand provides a variety of Application Program Interfaces (APIs) to programmatically create, deploy and interact with system definitions.
These APIs are made available by C# assemblies installed to the Global Assembly Cache (GAC). You can access the GAC from any .NET-compatible programming language or environment, such as LabVIEW, Python, and NI TestStand.
If you are using LabVIEW to program VeriStand, you can access these APIs within the **NI VeriStand** » **Execution** palette.

Keyboard Shortcuts

Navigate VeriStand using keyboard shortcuts.

File Operations

Action	Shortcut
Create a new screen document and add it to the existing project.	<Ctrl-N>
Open an existing project.	<Ctrl-O>
Close the current document.	<Ctrl-W>
Save the current file.	<Ctrl-S>
Save all open files.	<Ctrl-Shift-S>
Quit VeriStand.	<Alt-F4>

Basic Editing

Action	Shortcut
Cut.	<Ctrl-X>
	<Shift-Delete>
Copy.	<Ctrl-C>
	<Ctrl-Insert>
Paste.	<Ctrl-V>
	<Shift-Insert>
Undo.	<Ctrl-Z>
	<Alt-Backspace>
Redo.	<Ctrl-Y>
	<Alt-Shift-Backspace>

Editing Text

Action	Shortcut
Select a single word in a string.	Double-click text
Select the entire string.	Triple-click text
Move the cursor within a string by one word in the direction of the arrow.	<Ctrl-Right arrow>
	<Ctrl-Left arrow>
Move the cursor to the beginning of the current line.	<Home>
Move the cursor to the end of the current line.	<End>
Move the cursor to the beginning of the string.	<Ctrl-Home>
Move the cursor to the end of the string.	<Ctrl-End>
Cancel text entry.	<Esc>
Submit text entry.	<Ctrl-Enter>
Open shortcut menu for selected item.	<Shift-F10>
Add free label or comment to the project screen.	<Double-click open area>
Find and replace text or objects within a project.	<Ctrl-Shift-F>

Action	Shortcut
Find and replace text or objects within a document.	<Ctrl-F>

Selecting and Moving Objects

Action	Shortcut
Select multiple objects.	<Shift-Click>
Add object to the current selection.	
Select all objects.	<Ctrl-A>
Move selected objects in grid-sized increments.	<Arrow keys>
Move selected objects four grid units.	<Shift-Arrow key>
Copy and drag selected object.	<Ctrl-Drag>
Copy selected object and move it along one axis.	<Ctrl-Shift-Drag>
Resize selected object while maintaining aspect ratio.	<Shift-Resize>
Resize selected object while maintaining center point.	<Ctrl-Resize>
Resize selected object while maintaining both aspect ratio and center point.	<Shift-Ctrl-Resize>
Create additional blank space along the axis you drag the mouse.	<Ctrl-Drag open area>

Navigating the Environment

Action	Shortcut
Search document for next instance of text or an object. This command is only available when in Find mode.	<Enter>
	<F3>
	<Ctrl-G>
Search document for previous instance of text or an object. This command is only available when in Find mode.	<Shift-Enter>
	<Shift-F3>
	<Shift-Ctrl-G>

Action	Shortcut
Cycle through document tabs in the order in which they appear onscreen.	<Ctrl-Tab>
Cycle through document tabs in the opposite order in which they appear onscreen.	<Ctrl-Shift-Tab>

Navigating the Screen and Mapping Diagram

Action	Shortcut
Highlight all mappings and nodes connected to a selected item.	<Ctrl+,>
Shift focus to the palette search bar.	<Ctrl-Spacebar>
	<Ctrl-Alt-Spacebar> for Chinese keyboards
Scroll the document horizontally.	<Shift-Mouse Wheel>
Shift focus from one control to another in tabbing order while the code is running.	<Tab>
Shift focus from one control to another in reverse tabbing order while the code is running.	<Shift-Tab>
Pan across the project screen.	<Spacebar-Drag>

Deployment Commands

Action	Shortcut
Deploy the active system definition.	<F6>
Undeploy the active system definition.	<F7>
Connect to the system definition that is currently running on the VeriStand Gateway.	<F8>
Disconnect from the current running system definition. This leaves the VeriStand Gateway running for other clients.	<F9>

Help Commands

Action	Shortcut
Display the Context Help.	<Ctrl-H>
Access additional information on selected item.	<F1>

Wiring

Action	Shortcut
Delete all broken wires from the diagram.	<Ctrl-B>
Delete a wire you are in the process of creating.	<Esc>
	<Ctrl-Z
Select one wire segment.	Single-click wire
Select a wire branch.	Double-click wire
Select the entire wire.	Triple-click wire
Create a new wire branch from an existing wire.	<Ctrl-Click> wire
Tack down the wire segment and start a new wire segment.	Single-click while wiring
End the wire without connecting it to a node.	Double-click while wiring
Switch the direction of a wire between horizontal and vertical.	<Tap spacebar> while wiring
Organize the diagram or the selected code to make it easier to understand.	<Ctrl-U>

Navigating the Project Files Navigation Pane

Action	Shortcut
Expand everything in the selected folder.	<*> on the numeric keypad
Expand the selected folder.	<+> on the numeric keypad
Collapse the selected folder.	<-> on the numeric keypad
Expand the selected folder if it is closed. Otherwise, this keyboard shortcut selects the first child.	<Right arrow>

Action	Shortcut
Collapse the selected folder if it is open. Otherwise, this keyboard shortcut selects the parent.	<Left arrow>
Select the item beginning with the entered letter(s).	<Any printable key>
Open the selected document.	<Enter>
Rename the selected item.	<F2>
Move the selection to the first item in the tree.	<Home>
Moves the selection to the last item in the tree.	<End>
Move the selection to the first visible item in the tree.	<Page up>
Move the selection to the last visible item in the tree.	<Page down>

Zooming

Action	Shortcut
Zoom in and out.	<Ctrl-Mouse wheel>
Zoom in.	<Ctrl-+>
Zoom out.	<Ctrl-->
Zoom to fit.	<Ctrl-0>
Zoom to fit the selection.	<Ctrl-9>

VeriStand Directories and Aliases

VeriStand uses directories and aliases for project files, models, and custom devices.

The following tables list paths to common VeriStand directories by operating system. The heading before each table indicates how NI documentation refers to the directory. For directories with aliases listed, the alias is the text that appears with a relative path in an API or XML file. This text defines the directory that the path is relative to.

Note The aliases in this topic refer to locations on disk and are not related to aliases you define for channels in a system definition file.

<Common Data>

Alias: To Common Doc Dir

Operating system	Path
Windows	<Public Documents>\National Instruments\NI VeriStand <xxxx>

Note <xxxx> is the VeriStand version number.

<Application Data>

Alias: To Application Data Dir

For internal use only. Certain custom device development tools, including Custom Device XML and the Device Properties VIs, can reference this directory. However, NI recommends you avoid storing or modifying files in this directory.

Operating system	Path
Windows	<Application Data>\National Instruments\VeriStand

<Base>

Alias: To Base

Operating system	Path
Windows	<Program Files>\National Instruments\VeriStand <xxxx>

Note <xxxx> is the VeriStand version number.

<Custom Device Engine Destination>

Alias: None

Operating system	Path
Phar Lap/ETS	C:\ni-rt\NIVeristand\custom devices\<custom device name>\

<Model Framework>

Alias: None

Operating system	Path
Windows	C:\VeriStand\<xxxx>\ModelInterface

Note <xxxx> is the VeriStand version number.

VeriStand File Extensions

VeriStand projects use a variety of file types.

Use the following table to find information on a file extension.

File extension	Description
.bt1	A batch stimulus profile file created using VeriStand 2009, 2010, or NI Dynamic Testing Software 1.0.
.cah	Calibration History file.
.csv	Comma Separated Values file.
.dbc	Database file that describes signals and associated messages for a collection of controller area network (CAN) nodes.
.depvs	Support file that lists any dependencies of a compiled model. This file ensures the dependencies deploy to real-time targets.

File extension	Description
.dll	Shared library. VeriStand requires that models be compiled into a dynamic link library (DLL) before they can run on a Phar Lap RT target.
.et1	A step-based stimulus profile file created using VeriStand 2009 or NI Dynamic Testing Software 1.0.
.et2	A table-based stimulus profile file created using VeriStand 2009 or NI Dynamic Testing Software 1.0.
.fpgaconfig	FPGA Configuration file. XML-based file that specifies the content of direct memory access first-in-first-out memory buffers (DMA FIFOs).
.ini	Standard Windows configuration settings file.
.ldf	A local interconnect network (LIN) description file.
.llb	A LabVIEW VI library (LLB). Custom devices that run in VeriStand consist of LabVIEW VIs distributed in LLBs.
.lvbitx	LabVIEW-generated bitfile. Defines the available I/O on the FPGA. A bitfile is a compiled version of an FPGA VI.
.lvlib	LabVIEW project library. A collection of LabVIEW VIs, type definitions, shared variables, palette files, and other files, including other project libraries. For example, the VeriStand LabVIEW APIs are organized into project libraries.
.lvmodel	Compiled model that runs on Windows PCs. Generate models from a LabVIEW VI or simulation subsystem.
.lvmodelso	Compiled model that runs on Linux x64 or Linux ARM. Generate models from a LabVIEW VI or simulation subsystem. **Note** You must install additional software to enable LabVIEW models for targets running a Linux Real-Time OS. For more information about how

File extension	Description
	to use LabVIEW models with Linux, visit the NI website. VeriStand is not supported on x64 Intel-based cDAQ controllers running NI Linux Real-Time.
.lvproj	LabVIEW project file. A LabVIEW file type you can use to build custom devices for VeriStand.
.m	Text-based m-script file that initializes variables and can provide other information needed by models.
.mdl	Model you developed in the MathWorks Simulink simulation environment.
.ncd	NI-CAN database. Database file that describes signals and associated messages for a collection of CAN nodes.
.ncl	NI-XNET logfile. Binary file for the storage of CAN, FlexRay, and LIN data.
.nivscal	Calibration file deployed to targets. Binary file that VeriStand deploys, and then deletes when the project is undeployed.
.nivscf	Calibration file on host. XML-based file you can import a .nivscf file into other projects to reuse the calibrations.
.nivsmacro	Macro file. A recording of the commands sent to the target.
.nivsproj	Legacy VeriStand project file.
.nivsprj	VeriStand project file.
.nivsscreen	VeriStand screen file.
.nivssdf	VeriStand system definition file.
.nivsseq	Real-time sequence.
.nivsseqt	Real-time sequence template.
.nivsstimprof	Stimulus profile.
.nivsstimproft	Stimulus profile template.
.nivstest	A stimulus profile file created using VeriStand 2009, 2010, or NI Dynamic Testing Software 1.0.

File extension	Description
.out	Shared library.
.tdms	Technical Data Management Streaming file. A binary measurement file that contains waveform data. Stores descriptive information at the file, group, and channel levels. When you run a stimulus profile file using the Stimulus Profile Editor, VeriStand logs data to a TDMS file.
.vi	LabVIEW Virtual Instrument.
.xml	Extensible Markup Language file. VeriStand requires a corresponding XML file for any custom device that you add to a project. An XML file might also refer to a Stimulus Profile Editor test results file or a FIBEX file, a database storage file for use with the XNET Database Editor.
.xsd	XML Schema Document file. This file determines the validity of that XML file. It contains a set of rules to which the corresponding XML document must adhere in order to be considered valid according to that schema. VeriStand ships with .xsd files that you can use to validate FPGA configuration files and custom device XML files.

NI Hardware Support

VeriStand supports NI-XNET devices, NI-DAQ devices, and NI FPGA targets.

For the best performance, use NI-DAQ devices that support hardware-timed single point (HWTSP) I/O or on-demand sampling. Refer to DAQ Devices with Hardware-Timed Single Point Sampling Mode Support for a list of NI-DAQ devices that support HWTSP I/O. Refer to the custom device's GitHub repository for a list of supported modules.

Note VeriStand does not support HWTSP on cRIOs with DAQmx, such as 904x and 905x. For single-point I/O on these devices, use the Scan Engine and EtherCAT custom device. Refer to the custom device's GitHub repository for a list of supported modules.

Hardware Feature	NI-XNET Devices	NI-DAQ Devices	NI FPGA Targets
Custom FPGA VIs	No	No	Yes
Hardware timing	No	Yes	Yes
Analog I/O	No	Yes	Yes
Digital I/O	No	Yes	Yes
Pulse Width Modulation (PWM) I/O	No	Yes	Yes
Auto-discovery of installed devices and I/O modules	Yes	Yes	Yes

Refer to the **FPGA Targets with Built-in Support** section for a list of NI FPGA targets for which VeriStand provides bitfiles and FPGA configuration files.

Device support for PWM I/O includes:

- Generating pulses—If HWTSP sample mode is enabled (default) for a counter output channel, only X Series PCI and PXI devices support generating pulses with that channel. However, if that property is disabled for a Counter Output (CO) channel, all DAQ devices support generating pulses.
- Measuring pulses—Only X Series PCI and PXI devices support measuring pulses with counter input channels.

Supported Targets

VeriStand supports deploying a project to the following targets:

- PXI/PCI—Any PXI controller with at least 256 MB of RAM.
- CompactRIO/Single-Board RIO—Any CompactRIO or Single-Board RIO device with at least 128 MB of RAM.
- Desktop PC—Any PC running Windows 7, Vista, XP Service Pack 3 or later, Server 2003 R2 (32-bit), or Server 2008 R2 (64-bit).

FPGA Targets with Built-in Support

VeriStand ships with bitfiles and FPGA configuration files for the following FPGA targets:

- NI cRIO-9103

> **Note** The files provided for the NI cRIO-9103 are configured for the NI 9215, NI 9263, NI 9411, and NI 9474 I/O modules.

- NI PXI/PCI-7811R
- NI PXI/PCI-7813R
- NI PXI/PCI-7831R
- NI PXI/PCI-7833R
- NI PXI/PCIe-7841R
- NI PXI/PCIe-7842R
- NI PXI/PCIe-7851R
- NI PXI/PCIe-7852R
- NI PXI-7853R
- NI PXI-7854R

All bitfiles use direct memory access (DMA) to transfer data.

You can use the LabVIEW FPGA Module to create custom FPGA bitfiles for these or other FPGA targets.

Supported SCXI Modules

VeriStand supports adding the following SCXI DAQ modules to system definitions:

- SCXI-1100
- SCXI-1102
- SCXI-1102B
- SCXI-1102C
- SCXI-1104
- SCXI-1104C
- SCXI-1112
- SCXI-1120
- SCXI-1120D
- SCXI-1121

- SCXI-1122
- SCXI-1124
- SCXI-1125
- SCXI-1126
- SCXI-1127
- SCXI-1128
- SCXI-1140
- SCXI-1141
- SCXI-1142
- SCXI-1143
- SCXI-1160
- SCXI-1161
- SCXI-1162
- SCXI-1162HV
- SCXI-1163
- SCXI-1163R
- SCXI-1190
- SCXI-1191
- SCXI-1192
- SCXI-1520
- SCXI-1530
- SCXI-1531
- SCXI-1540
- SCXI-1581

Related information:

- Creating a Custom FPGA Bitfile

Creating a New Project

Create a new project in VeriStand to develop, prototype, and test control systems using hardware I/O and simulation models.

1. Open VeriStand.
2. Click **Default Project**.
3. Enter a **Project Name**.
4. Select a **Location** to save the project.
5. Select a **System Definition**.

> **Note** If you do not have a system definition, leave this field as None chosen. VeriStand will create a new system definition file with the same name as the project.

6. Click **Create**.

Configuring a Project File

Configure a VeriStand project to complete tasks such as adding Tools menu items, services, alarm responses, and custom files.

1. Launch your project in the VeriStand Editor.
2. Depending on your goal, complete any of the following tasks.

Goal	Task
Adding a custom Workspace Tools menu item to the project file.	Add items to the Tools menu of the Workspace to display custom dialog boxes.
Adding custom VIs.	Add custom VIs by building a LabVIEW project into a source distribution.
Adding custom files to the project file.	Add custom files, such as documentation, to the project file.
Connecting multiple hosts to the same target.	Configure one or more host computers to communicate with the same target using the VeriStand Gateway.

Creating a VI Source Distribution

Integrate custom VIs into VeriStand by building a LabVIEW project into a source distribution.

Use the LabVIEW Application Builder to build a source distribution to call custom VIs in VeriStand. This creates a local copy of the VI dependencies.

1. Open the LabVIEW project containing the VI in Project Explorer.
2. Right-click **Build Specifications** and select **New** » **Source Distribution**.
3. In Source Distribution Properties, set where to build the output.
    a. Under the Category list, click **Information**.
    b. Enter a **Build specification name**.
    c. Select the **Destination directory**.
4. Include the VI source files.
    a. Under the Category list, click **Source Files**.
    b. Add the custom VI and any supporting sub-VIs to Always Included.
5. Specify the output type.
    a. Under the Category list, click **Destinations**.
    b. Click **Add Destination** to create a new destination.
    c. Enter a **Destination label** that matches the VI name.
    d. Set the Destination type as **Directory**.
6. Remove additional exclusions.
    a. Under the Category list, click **Additional Exclusions**.
    b. Disable **Exclude file from vi.lib**, **Exclude files from instr.lib**, and **Exclude files from user.lib**.
7. Click **Build**.

When the build finishes, LabVIEW creates the VI and all necessary dependencies in the destination path.

After building the custom VI, add it to the VeriStand project or to the Tools menu.

Adding a Standard or Custom Tools Menu Item

Add items to the Tools menu of the **Workspace** to display custom dialog boxes.

Before you begin, build the custom Tools menu item into a source distribution.

Use these dialog boxes to perform operations such as monitoring alarms, viewing channel data, scaling and calibrating channels, or running stimulus profiles.

1. Open the VeriStand Editor.
2. In the Navigation pane, click **Project Files**.
3. Right-click **Workspace** and select **Configure Tools**.
4. Use the Tools Properties dialog box to add and configure **Tools** menu items.
5. Click **OK**.

Adding Custom VIs

Add custom VIs by building a LabVIEW project into a source distribution.

Before you begin, build the VI into a source distribution.

1. Open the VeriStand Editor.
2. In the Navigation pane, click **Project Files**.
3. Click **New** » **Add File.**
4. In the Select Item window, select a custom VI and click **OK**.
5. **Optional:** In the Project Files tab, right-click the custom VI and click **Run On Deploy**.

   Note Enabling this option allows VeriStand to automatically run the VI when the system definition deploys.

6. Click **File** » **Save all**.

Adding Custom Files

Add custom files, such as documentation, to the project file.

1. Launch your project in the VeriStand Editor.
2. In the Navigation pane, click **Project Files**.
3. Click **New** » **Add File.**
4. In the Select Item window, select a file, and click **Open**.

The file appears in the Project Files tab.

Connecting Multiple Hosts to the Same Target

Configure one or more host computers to communicate with the same target using the VeriStand Gateway.

Before you begin, you will need the IP address of the host that you want to run the VeriStand Gateway.

1. On the host that you want to connect, open the VeriStand Editor.
2. In the Navigation pane, click **Project Files**.
3. Double-click your project file (.nivsprj) to open it as a new tab.
4. In the Configuration pane, click the **Document** tab.
5. In **Gateway address**, enter the IP address of the host running the VeriStand Gateway.

   Note The default is localhost, which specifies that the VeriStand Gateway is running on the local machine.

6. Click **File** » **Save all**.

The host you configured now connects to the instance of the VeriStand Gateway that runs on the host whose IP address you entered.

Integrating and Executing Models

Use models to mathematically represent real-world systems in your VeriStand project.

Simulate both the plant and controller to create a closed-loop control system within a software model. Use models for signal generation, signal analysis, and control. For more information on using models in VeriStand, refer to Models FAQs.

1. Prepare your model for use.
   - Use a model from C/C++—Create a model in C or C++ that NI software can load and execute through the VeriStand Model Framework.
   - Use a model from MathWorks Simulink software—Use Simulink software to convert your model for use on real-time targets.

     Note VeriStand only supports 64-bit operating systems.

   - Use a model from LabVIEW VIs—Convert LabVIEW VIs into compiled .lvmodel or .lvmodelso files.
   - Use the Functional Mockup Interface (FMI)—Decouple modeling environments from model consumers.
2. Add and configure the model—Connect a model to other parts of the system and run the model on a hardware target.
3. Control and monitor model execution—Use model execution channels to interact with models.

If you have problems after integrating your model with VeriStand, refer to Common Issues with Models in VeriStand.

Models FAQs

Answers to common questions on models in VeriStand.

Which Simulink addon should I use to compile my model?

Use the VeriStand Model Framework under these circumstances:

- You have existing models that have been adapted for the VeriStand Model Framework in previous versions of VeriStand.
- Your model uses buses for top-level Inports or Outports. Modifying the model to add a Bus Selector node for every bus is not feasible.
- You need to map model signals that are not inports, outports, or parameters.

Use VeriStand Model Generation Support under these circumstances:

- You are beginning a new model design that has not been previously integrated into a VeriStand system.
- Your model parameters are defined in the Base Workspace or Model Workspace through a MATLAB script.
- You need support for Simulink.Parameter objects.
- Your model will need to use Simscape.

What versions of MathWorks software and the LabVIEW Development System are compatible with my version of VeriStand?

Refer to VeriStand Version Compatibility for a list of software you can use with each VeriStand distribution.

What determines the rate a model runs?

The rate at which a model runs is a function of the system rate and a model-specific decimation of the system rate, where actual model rate = Primary Control Loop rate / decimation. You can use VeriStand to set the model timing.

How do I control the latency of models and the rest of my system?

The Primary Control Loop (PCL) of the VeriStand Engine provides two execution modes. The default mode, **Parallel**, applies a one-cycle delay between when a model executes and when the data it produces is available to the system. Alternatively, **Low Latency** mode ensures data from models is available to the rest of the system on the same iteration of the PCL as it is generated.

How do I make my models execute in a particular order?

Multiple models in a system execute in parallel unless you define an execution order. If you want one model to wait until a second model finishes executing before the first model runs, you must define an execution order. Execution orders ensure data can transfer between models during the same iteration of the PCL.

How can I improve the performance of my system as it relates to models?

Avoid importing parameters and signals that the system does not use. The presence of many parameters and signals can have a negative impact on the performance of the system even if the model is not running.

Using Models from C and C++

Create a model in C or C++ that NI software can load and execute through the VeriStand Model Framework.

Before you begin, install the VeriStand Model Framework on the computer you are compiling the model. The framework adds tools you will use during the compile process.

1. Based on the operating system (OS), choose a compiler.

OS	Compiler
Windows	Microsoft Visual C++
NI Linux RT x64	C/C++ Development Tools for NI Linux Real-Time, Eclipse Edition

2. Create a model header file—Create a model.h header file that contains the type definitions for model properties and all user-visible parameters in your model.
3. Adapt the template to model code—Use the template.c file that the VeriStand Model Framework installs as a starting point for your model code.
4. Create a makefile to compile model code—Create a makefile for the compiler and operating system your model will use.

Locate example .c and model.h files and makefiles in <RootDrive>\VeriStand\<xxxx>\ModelInterface\custom\examples.

> **Note** <RootDrive> is the drive where NI software installs and <xxxx> is the VeriStand version number. If you have problems locating the files, check NI Package Manager to make sure that the VeriStand Model Framework was installed.

After compiling the model, add it to a system definition.

For more information on simulating models in LabVIEW with the Model Interface Toolkit, open LabVIEW and select **Help** » **LabVIEW Help**. Click **Contents** and browse to **Toolkits** » **Model Interface Toolkit**.

Creating a Model Header File

Create a model.h header file that contains the type definitions for model properties and all user-visible parameters in your model.

1. Create a header file and name it model.h.
2. Create code in the file similar to the following code.

```
#ifndef MODEL_h
# define MODEL_h
typedef struct {
  double a[2][2];
  double b11;
  double c12;
  double idleRPM;
  double redlineRPM;
  double temperature_timeConstant;
  double temperature_roomTemp;
  double temperature_operatingTempDelta;
  double temperature_redlineTempDelta;
} Parameters;
#endif
```

   The example code contains definitions for both scalar and vector double parameters.

For information about defining parameters whose data type is something other than double, refer to the TO DO comments in the template.c file installed by the VeriStand Model Framework.

3. Save the file.

After creating a model header file, adapt the template to your model code.

Adapting the C Template to Model Code

Use the template.c file that the VeriStand Model Framework installs as a starting point for your model code.

Before you begin, create a model header file.

1. Browse to <RootDrive>\VeriStand\<xxxx>\ModelInterface\custom\examples, create a copy of template.c, and give the copy a new name.

> **Note** <RootDrive> is the drive where NI software installs and <xxxx> is the VeriStand version number.

2. Modify the following files.

File	Description
template.c copy	Lists the code you must customize, which is marked with TO DO comments. This file also contains information about how to instantiate and access parameters.
ni_modelframework.h	Lists definitions for properties of outward-facing components of your model, such as inports, outports, parameters, and signals. **Note** This file is located in the RootDrive:\VeriStand\xxxx\Model Interface\ directory.

After you adapt the template, create a makefile to compile the model.

Creating a Makefile and Compiling Model Code

Create a makefile for the compiler and operating system your model will use.

Before you begin, adapt the C template to your model code and determine which real-time (RT) operating system your RT target runs.

For your model code to work with the VeriStand Model Framework, create an appropriate makefile to compile your model code.

1. Create a makefile (.mak) to compile the model source code.
   For examples of makefiles designed to compile models that work with the VeriStand Model Framework, refer to the example .mak and .mk files installed in the <RootDrive>\VeriStand\<xxxx>\ModelInterface\custom\examples directory.

   Note <RootDrive> is the drive where NI software installs and <xxxx> is the VeriStand version number.

2. Place your model code components in the same directory as the makefile.
3. Place your model framework components, ni_modelframework.h and ni_modelframework.c, in the same directory as the makefile.
4. Run the makefile to compile your model code.

Using Models from Simulink

Use Simulink software to convert your model for use on real-time targets.

Depending on your version of MathWorks Simulink Software, use one of the following resources to compile your model.

Simulink Version	Resource
R2017b to R2020b	VeriStand Model Framework **Note** For more information on how the framework and Simulink interact, refer to How VeriStand

Simulink Version	Resource
	Imports Simulink Models using the Model Framework.
R2020a and newer	VeriStand Model Generation Support MATLAB Addon **Note** For more information on how the addon and Simulink interact, refer to How VeriStand Imports Models from the Model Generation Support MATLAB Add-on.

After compiling the model, add it to a system definition.

Using the VeriStand Model Framework for Simulink

Run Simulink models written in C/C++ with the VeriStand Model Framework.

Before you begin, install the VeriStand Model Framework on the computer you are compiling the model. The framework adds tools you will use during the compile process. You should also install a supported version of Simulink.

Note The Simulink user interface differs between versions of the software. For more information on how to implement the following steps, refer to the Simulink documentation.

1. Launch Simulink and load the model you want to convert.
2. Configure the model parameters with the following settings.
   - **Stop time: inf**
   - **Type: Fixed-step**
3. Generate the model code.
4. **Optional:** If the model specifies any .c or .h files, include the locations of the source files and directories.

5. Based on your target's operating system, select the correct TLC file and compiler.

OS	TLC File	Compiler
Windows	NIVeriStand.tlc	Microsoft Visual C++ **Note** NI recommends installing Microsoft Visual Studio 2017 with C++ Development option enabled.
NI Linux Real-Time Intel x64-based targets	NIVeriStand_Linux_64.tlc	C/C++ Development Tools for NI Linux Real-Time, Eclipse Edition

6. Build the model.

The MATLAB software command window displays the status of the build process and indicates when the Simulink Coder software has completed compiling the model.

After compiling, add the model to the system definition.

VeriStand Model Framework

The VeriStand Model Framework is used to design and compile models created in third-party modeling environments and C/C++.
VeriStand and the LabVIEW Model Interface Toolkit allow you to run simulation models written in C/C++. To enable your models to interact with this NI software, you must design them to work with the VeriStand Model Framework. The framework includes files that allow your model to interact with VeriStand and the Model Interface Toolkit.
When you run your VeriStand application, the application executes functions defined in VeriStand Model Framework files. These functions call your model code to execute the model.

Model Framework Components

The model framework includes files that define properties, identify functions for export, implement an interface, and create interdependent structures.
Use the following table to locate files in the Model Framework to help implement your model code.

File	Description	Installed location
ni_modelframework.h	A header file that includes the following components: ▪ Type definitions to define properties—such as inports, outports, parameters, and signals—of your model's outward-facing components. ▪ Functions that the VeriStand Model Framework exports to your compiled model.	<RootDrive>\VeriStand\<xxxx>\ModelInterface\
ni_modelframework.c	Implements the common interface between your test application and your model code.	<RootDrive>\VeriStand\<xxxx>\ModelInterface\custom\src
template.c	A template for your model code. Use this file to create code that maintains interdependent structures between your model and ni_modelframework.c.	<RootDrive>\VeriStand\<xxxx>\ModelInterface\custom\examples

Note <RootDrive> is the drive where NI software installs and <xxxx> is the VeriStand version number.

Model Code Components

The files model.h and model.c are used to implement your models.
Create the following two files to implement your C/C++ model:

- model.h—Contains the type definitions for your model parameters.

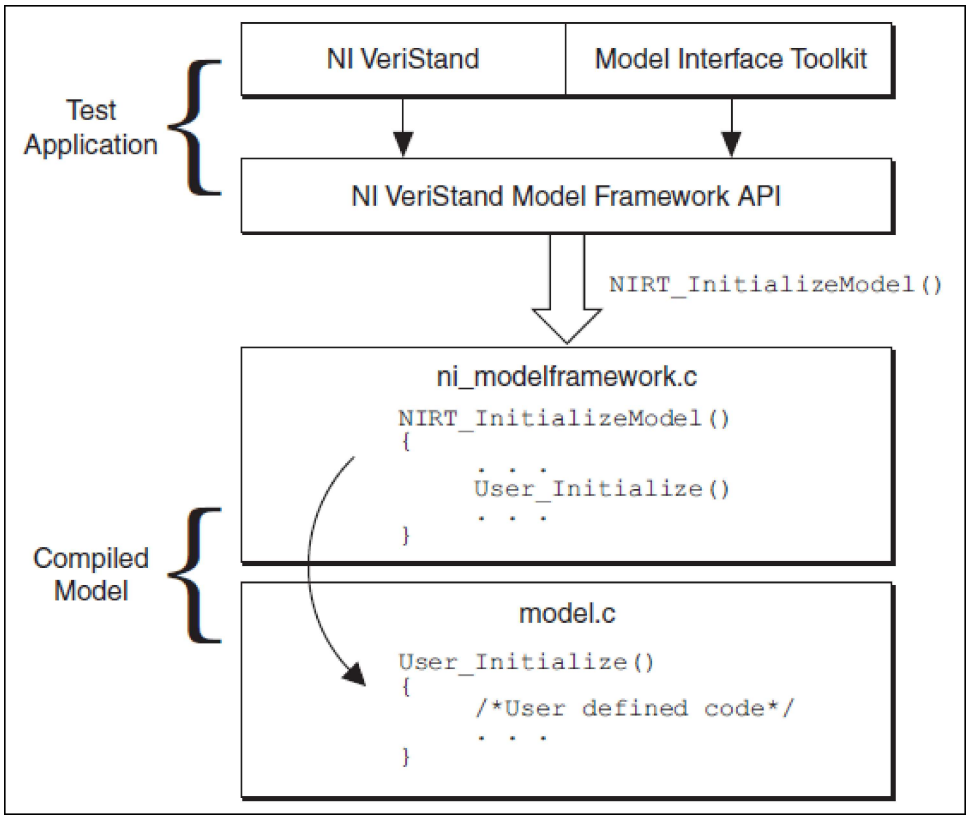
> **Note** You must name the file model.h and include all user-visible parameters in your model.

- model.c—Contains your model code. Make a copy of template.c and modify the copy to ensure you maintain interdependent structures, such as headers, imported and exported symbols, and functions that NI software recognizes, between your model code and ni_modelframework.c.

> **Note** You can name this .c file anything.

Model Framework and Model Code Interaction

When you run your VeriStand or Model Interface Toolkit test application, the application executes functions that the VeriStand Model Framework files define. These functions call functions in your model code, which convert user-defined data types, initialize your model, and increment a time step in VeriStand.
The following illustration shows how NI software, the Model Framework API, and code in your model interact.

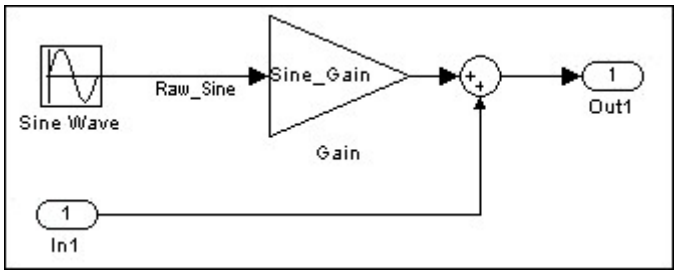


Your test application calls a function that the Model Framework exports. The exported function calls a function in your model code. As the model executes, the test application can interact with the model in the following ways:

- Writing data to model inports
- Reading data from model outports
- Allowing you to adjust model parameter values
- Allowing you to probe model signals

How VeriStand Imports Simulink Models using the Model Framework

VeriStand identifies inports, outports, parameters, and signals in models you created in Simulink software according to their configuration in Simulink. A simple model in Simulink might contain the following components.



When you add this model to a system definition in VeriStand, its components appear as shown in the following table.

Component in model	Type of component in VeriStand
Raw_Sine	Signal
Sine_Gain	Parameter
In1	Inport
Out1	Outport

Note VeriStand organizes named components of bus objects, such as signals, into folders that match the hierarchy of the bus.

Signal Importing

Certain optimizations you enable in Simulink can make a signal unavailable in VeriStand. You can disable these options for the entire model to make all signals available for probing, but the memory footprint of the model increases as a result. Alternatively, you can mark individual signals as test points in Simulink to maintain a reduced memory footprint while keeping the test-point signals available for probing.

Note If you mark new signals as test points, you must recompile your model.

Parameter Importing

Compiled models contain one of two types of parameters in VeriStand:

- **Global parameters**, by default, applies to the current model and to any global parameters with the same name in other models on the target. This parameter is similar to a workspace variable in MathWorks MATLAB software. If you set inline parameters in the Simulink software, MathWorks Real-Time Workshop software converts MATLAB workspace variables to global parameters in the compiled model.
- **Local parameters** only apply to the specific model and block or subsystem that they belong to. If you do not set inline parameters in Real-Time Workshop, block parameters remain block parameters in the compiled model.

A Simulink model can contain only one type of parameter. However, a system definition can contain a model with global parameters and a model with block parameters.

VeriStand supports model references to submodels, but you cannot access parameters in submodels. Submodels execute in a VeriStand system, but their parameters are not available for mapping.

In Simulink, you can inline parameters. In VeriStand, inlined parameters are not available for configuration. However, even if you inline a parameter in Simulink, you still can allow users to influence the parameter by configuring a variable that affects the parameter to be tunable.

For example, consider a model with the following characteristics:

- Contains a constant configured with an expression (x + 3).
- Contains a sine wave block whose amplitude and frequency parameters are inlined, and therefore unavailable in VeriStand.
- Adds the result of the constant to the output from the sine wave block.

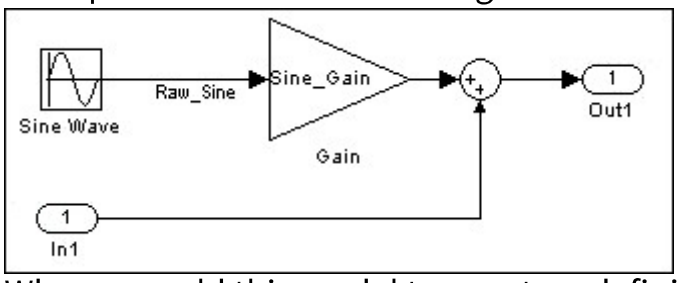
Although the parameters of the sine wave block are unavailable, you can influence the operation by specifying that the x variable is tunable in Simulink. After you add the model to VeriStand, when x is tunable, it will appear in the **Parameters** list in System Explorer. You can change the value of x as the model executes.

Inport and Outport Importing

Top-level Simulink inports and outports become VeriStand inports and outports. Submodel inports and outports in Simulink import only if you place VeriStand inport and outport blocks within the submodel in Simulink.

How VeriStand Imports Models from the Model Generation Support MATLAB Add-on

VeriStand identifies inports, outports, parameters, and signals in models you created in Simulink software according to their configuration in Simulink.
A simple model in Simulink might contain the following components.



When you add this model to a system definition in VeriStand, its components appear as shown in the following table.

Component in model	Type of component in VeriStand
Sine_Gain	Parameter
In1	Inport
Out1	Outport

Signal Importing

To import signals, enable the signal for export before compiling the model. In Simulink, navigate to **Code Generation** » **Interface** to enable signal export and designate the required signals as test points.

Note VeriStand does not support the use of buses as signals.

Simulink External Mode

To visualize signals and tune parameters, enable Simulink external mode before building an imported model.



Note Enabling external mode can negatively impact model execution performance.

After importing and deploying a model in VeriStand, connect to the model in Simulink by specifying the target IP address and port number. Determine the port number by referring to the ExternalMode/Port parameter. Once connected, use Simulink to visualize signals and tune parameters.



Note When importing a model compiled using Simulink external mode, VeriStand opens a port in the dynamic range of 1025 to 65536.

Parameter Importing

Simulink variables that are defined in the model workspace, Base Workspace, or in a data dictionary will be available as parameters in VeriStand if they meet the following requirements.

- They are of a supported datatype.
- They are referenced by at least one Simulink block or Model Mask.

Inport and Outport Importing

Top-level Simulink inports and outports become VeriStand inports and outports. Submodel inports and outports in Simulink are not accessible.

Importing Models with Referenced Models and Model Masks

VeriStand supports importing models that use Referenced Models and models under Model Masks. Their Inports and Outports will not be available, but any variable referenced by those models will be available as a parameter.

Using Models from LabVIEW VIs

Convert LabVIEW VIs into compiled .lvmodel or .lvmodelso files.

Before you begin, verify VeriStand supports your hardware target and prepare your LabVIEW VIs for conversion.
In order to generate compiled models from LabVIEW VIs, you need the following products:

- LabVIEW development system that is the same year version as your VeriStand installation.
- LabVIEW Application Builder

You can convert LabVIEW VIs or simulation subsystems you create using the LabVIEW Control Design and Simulation Module. The converted files add system simulation, closed-loop control, and other functionality to VeriStand applications.

1. In LabVIEW, select **Tools** » **NI VeriStand** » **Generate Model from VI**.
2. On the **Generate NI VeriStand Model from VI** dialog box, enter the **Source VI Path** where you saved the source file.
3. Enter a **Destination Folder** where you want to save the generated model.
4. **Optional:** If your source file is a simulation subsystem, click **Next**.
    a. Specify a **model time step (sec)** that indicates the interval between the times the ODE Solver evaluates the model and updates the model output.

> **Note** For your compiled model to run in real-time, the model time step (sec) value must equal the controller period multiplied by the model Decimation, or the model Decimation divided by the Target Rate. These two methods are represented by the following equations.
>
> - model time step (sec) = Controller Period * Decimation where Controller Period = 1/Target Rate
> - model time step (sec) = Decimation/Target Rate

> In System Explorer, use the Controller Configuration page to specify the **Target Rate** and the Model Configuration page to specify the **Decimation**.

b. Specify the **ODE Solver**.

5. Click **Build**.

> **Note** If you experience errors when converting a VI, refer to LabVIEW VI Model Conversion Preparation.

If you deploy the compiled .lvmodel or .lvmodelso to a real-time target, VeriStand automatically copies required .dlls to the target. However, if you manually copy the file to a new location, such as a different host computer, you must move the following support files to maintain relative paths:

- A .depvs file that LabVIEW creates to reference the dependencies.
- Any LabVIEW .dll located in a subdirectory named data in the destination folder where LabVIEW generates the compiled model.

LabVIEW VI Model Hardware Target Support

Verify your hardware target will support a LabVIEW VI compiled into a VeriStand model.
Use the following table to determine if your hardware target supports a VeriStand model file type.

> **Note** For a list of real-time targets and the real-time operating systems (RTOS) that each runs, see Real-Time Controllers and Real-Time Operating System Compatibility.

Computer type	RTOS	Supported VeriStand model file type
Windows	—	.lvmodel
Real-Time Target	NI Linux Real-Time	.lvmodelso

On applicable real-time targets, .lvmodelso and .lvmodel files are supported if the source LabVIEW VI does not contain code with Windows function calls that are not supported by the RTOS.

LabVIEW VI Model Conversion Preparation

You must assign front panel controls and indicators in LabVIEW VIs to the connector pane so VeriStand can identify them as inports, outports, and parameters when you add the compiled .lvmodel or .lvmodelso to a system definition.
Use the following table to build the LabVIEW VI connector pane according to how you want each control or indicator to work in VeriStand.

Desired VeriStand component	VI connector pane assignment	Is LabVIEW default value imported to VeriStand?
Inport	Required input	No
Outport	Any output	No
Parameter	Optional or recommended input	Yes

Supported LabVIEW Data Types

Assign the following supported data types to the controls and indicators in the connector pane.

- Numerics
- Booleans
- 1D arrays of numerics
- 1D arrays of Booleans
- Clusters containing the previous data types

If you use an unsupported data type, LabVIEW returns an error when you try to convert the VI to a compiled model. Controls and indicators not assigned to the connector pane can have other data types.

Global and Local Parameters

A compiled model you add to a system definition can contain global parameters and block parameters.

If you want a LabVIEW VI front panel control to become a local parameter in VeriStand, place that control in a cluster shell before you compile. To make a LabVIEW VI front panel control a global parameter in VeriStand, do not place the control in a cluster.

Considerations for LabVIEW VIs with Array Terminals

If a LabVIEW VI contains an array control or indicator you want to include in the model, enter a value in the n element of the array, where n is the desired number of elements. Right-click the array control and select **Data Operations** » **Make Current Value Default**. Otherwise, the array appears in VeriStand with a single element.

FMI Support

The Functional Mock-up Interface (FMI) is an API standardization for exchanging dynamic system models.
You can use FMI to decouple modeling environments from model consumers. This decoupling helps create tool-agnostic, portable solutions between modeling and simulation environments.
The FMI standard defines two model interaction patterns:

1. Model Exchange—The package contains the mathematical representation of the model and the simulation environment solves the equations of the model
2. Co-Simulation—The model contains the solver for the model and can directly provide outputs based on inputs and time slice.

FMI also defines the distribution packaging of the model and decouples the interface description from the actual model binaries. You can have support for several platforms, like Windows 32-bit and 64-bit, Linux 32-bit and 64-bit, and source code, in the same package. A model that implements this interface is called a Functional Mockup Unit (FMU).

Model Configuration and Execution Support

VeriStand enables configuration and execution of FMI 2.0 CoSimulation models on Windows and NI Real-Time Linux 64-bit systems. This support requires FMUs to have the proper executable binaries available.

Supported modeling environments include:

- AVL Boost
- FMU SDK
- Wolfram SystemModeler
- MapleSim
- Altair Activate

Note NI can validate other modeling environments on request if provided with sample models.

Model Configuration

You can use the same general model import and configuration steps in System Explorer to import FMUs into VeriStand.

The following table displays how a variable appears based on its causality when a FMU is imported into VeriStand.

FMU Variable Causality	VeriStand Node Type
input	inport
output	outport
parameter	parameter that can be imported as a channel or accessed through Model Parameter Manager
calculatedParameter	not visible
independent	not visible
local	signal

The rate of the model is defined by the **stepSize** attribute in the **DefaultExperiment.** Decimation must be configured based on the rate at which the VeriStand Engine is running.

Model Execution

The following table displays the model execution target architectures that FMI Early Access supports on the NI platform.

NI Target Architecture	Folder in .zip	Early Access Support
PXI Linux	linux64	Yes
Linux64 cRIO	linux64	No
linuxArm	arm-linux-gnueabi	No
Win	win32	Yes
Pharlap	win32	No
sources	sources	No

Note Linux64 cRIO is expected to work by default due to its similar OS to PXI Linux.

FMI Support Limitations

Early Access FMI support has several limitations.

- All FMI numeric data types are supported but cast to doubles internally.
- String variables are not accessible.
- A **DefaultExperiment** section with a non-zero **stepSize** is mandatory. Models without the **DefaultExperiment** section will fail to run.

  Note An example of a working **stepSize** is <DefaultExperiment stepSize="0.01"/>.

- Changing **stepSize** in VeriStand is not supported. The only way to change **stepSize** is by modifying the modelDescription.xml file inside the FMU.
- FMU logging is not supported.

Note Minimal logging to the console will be done in the case of **fmi2Fatal** results.

Controlling and Monitoring Model Execution

Use model execution channels to interact with models.

Before you begin, add a model to the system definition.

Use model execution channels to accomplish tasks such as manually starting the execution of a model rather than allowing it to execute upon deployment. When you add a model to your VeriStand project, model execution channels are created.

Depending on your goal, map a model execution channel to model controls on the Workspace to control or monitor the state of the model after deployment.

Note You can find execution channels for a model in the configuration tree by clicking **Targets** » **Controller** » **Simulation Models** » **Models**, selecting a model, and clicking **Execution**.

Goal	Model execution channel
Run, pause, or stop the simulation	Model Command
Save the model state to file or restore the model state to file	
View the model execution status	Model Status
View the time in the model	Model Time
View how long the current time step of the model has been running	Time Step Duration

You can also use system channels to monitor the system while it is deployed and running.

Model Command

Use the Model Command channel to change the model execution state at run time.

If you configured your model to be initially paused, you can use this channel to start it. If your model gets into a desirable state for testing purposes, you can pause the model, save that state to a file, and then restore the state later.
Refer to the following table to determine what values VeriStand associates with each status.

Value	Status
0	Start the model. **Note** If the model is paused, it starts from the current state.
1	Pause the model.
2	Reset the model.
3	Prompt to save the model state to file after setting the model to idle. **Note** The model must be running or paused when you set this command. Otherwise, VeriStand ignores the command.
4	Prompt to restore the model state from file after setting the model to idle. **Note** The model must be running or paused when you set this command. Otherwise, VeriStand ignores the command.

Model Status

Use the Model Status channel to monitor how the model is operating at run time. The Model Status channel returns the model's state, such as running, resetting, and stopped.
Refer to the following table to determine what values VeriStand associates with each status.

Value	Status
0	Running
1	Paused
2	Resetting
3	Idle
4	Stopped
5	Restoring
6	Saving

Model Time

Use the Model Time channel to verify that the model is running at the correct rate. You can compare the model time to the value of the System Time system channel to determine if the model is executing slower or faster than real time.
The following equation describes how the VeriStand Engine determines the rate to execute each model.
model rate = Primary Control Loop (PCL) rate / model decimation
If the rate for a model is incorrect, adjust its decimation and monitor the model time again.

Time Step Duration

Use the Time Step Duration channel to monitor how long a model takes to execute. If a model takes a long time to execute, the PCL might run slower than the target rate or the model might not run on schedule during the next PCL iteration. If a system contains multiple models and you need to determine which one delayed the system, monitor the Time Step Duration execution channel for each model. During each PCL iteration, a model executes one time step, so this channel allows you to see if a model executes longer than the expected rate, given the following equation.
model rate = PCL rate / decimation



Note The Windows operating system supports only millisecond resolution. This channel does not provide exact timing for microsecond values and instead provides an approximation that averages to the requested time over the course of the timing interval. Windows does not

> support resolutions under one millisecond and rounds them up to one millisecond.

Common Issues with Models in VeriStand

If your model crashes or does not execute as expected, isolate the issue and determine if its source is within the model or due to your system definition.
To identify the source of an issue, replace your model in the system definition with a simple model, and then redeploy the system definition. If the simple model executes as expected, the source of the issue is within your model. However, if the simple model also experiences issues, the source of the issue is due to settings for your system definition.
The following table lists common model issues and solutions.

Issue	Solutions
Model is crashing	Models often crash when an inport receives a value of 0 and the model attempts to divide by the inport value. This issue will occur during deployment if the initial state of the model is to run and the default value for an inport is 0. Depending on your system, complete the following troubleshooting solutions: ▪ Change the default value for the inport. ▪ Rewrite the model to remove the possibility of dividing by 0. ▪ Change the initial state of the model to be paused in System Explorer and implement a start-up procedure that ensures that the inport values are acceptable before you start the model.

Model runs too fast or slow	If your model is unstable because it runs too fast or too slow, ensure the actual model rate matches the rate at which the model was compiled to run. If the rates do not match, adjust the settings that determine the actual model rate until the following expressions are correct:
	`actual model rate = compiled model rate`
	`actual model rate = PCL rate / decimation`

Issue	Solutions
	Adjust the model timing by configuring the following settings where specified: ▪ Compiled loop rate—Displays on the **Model Configuration** page, in **Simulation model info**. ▪ Model decimation—Set on the **Model Configuration** page, in the **Decimation** control. ▪ PCL rate—Set on the **Controller Configuration** page with the **Target Rate** control.
Model generated data is delayed	If other parts of your system that are mapped to your model do not receive data when you expect, consider adjusting the following system definition settings: ▪ PCL execution mode—Change the execution mode to low latency if data from models must be available for mapping during the same PCL iteration the model generates the data. The default mode, parallel, applies a one-cycle delay between when a model executes and when the data it produces is available for mapping. ▪ Execution order—Multiple models in a system execute in parallel unless you define an execution order. If you map an outport from one model to an inport of a second model and you want the second model to wait until the first model finishes executing before it runs, you must define the execution order.
Decreased system performance	If you suspect that models are causing your system to run slower than you desire, consider the following solutions to improve performance: ▪ Import only parameters and signals your system requires. Importing many parameters and signals can have a negative impact on the performance of the model even if the model is not running. ▪ If you do not need data from models to be available to the rest of the system on the same iteration of the PCL, set the PCL execution mode to parallel instead of low latency. Parallel mode causes a one-cycle delay between when a model executes and when the data it produces is available

Issue	Solutions
	to the system, but increases the execution speed of the entire system.

If you continue experiencing issues with your model, contact NI Support.

Configuring a System Definition File

Modify a system definition file to complete tasks such as configuring the VeriStand engine, adding models, and creating aliases.

Before you begin, you must create a new system definition file.

The system definition file contains the configuration settings of the VeriStand Engine, such as the system rate and the list of channels.

1. Open a project in VeriStand.
2. Depending on your goal, complete any of the following tasks.

Goal	Task
Adding and activating a system definition file	Reuse or add new system definition files to VeriStand projects.
Versioning a system definition file	Create and test modifications to your system definition file by duplicating the file in the VeriStand Editor.
Connecting to a target system definition	Connect to the target's deployed system definition to send and receive data.
Specifying a target	Select a target and designate its name, operating system, and IP address.
Configuring the VeriStand Engine	Control the timing of the system and the communication between the target and host computer.
Adding and configuring a procedure	Set the actions the VeriStand Engine executes in response to an alarm, when called from another procedure, or as a startup procedure.
Adding and configuring an alarm	Notify users when a channel value is outside a specified range.

Goal	Task
Adding and configuring a hardware device	Add and configure NI hardware, timing and sync devices, and custom devices.
Scaling a channel on hardware devices	Create scales to convert hardware channel unit measurements to transducer/actuator scaled units.
Adding and configuring a custom device	Add and configure third-party custom devices to execute user-defined actions, determined by LabVIEW VIs.
Adding and configuring a model	Connect a model to other parts of the system and run the model on a hardware target.
Adding a user channel	Store a single value as a user channel to use as a variable in procedures, stimulus profiles, and other operations.
Adding a calculated channel	Produce a new value based on calculations performed on other channels in the system using the Mapping Diagram.
Creating an alias	Set an alternate name for channels in a system definition file.
Mapping channels and aliases	Connect channels or aliases to one another.

Creating a New System Definition File

You can create a new system definition file without creating a new project with **System Explorer**.

1. Open VeriStand.
2. Create or open a VeriStand project.
3. Click **Configure...** to open **System Explorer**.
4. Click **File** » **New** to open the **New System Definition** dialog box.
5. Enter a **System Definition name**.
6. Click **OK**.

After creating the system definition file, add it to the project.

Adding and Activating a System Definition File

Reuse or add new system definition files to VeriStand projects.

Before you begin, create a new system definition file.

1. Launch your project in the VeriStand Editor.
2. Click **New** » **Add File.**
3. In the Select Item window, select a system definition file (.nivssdf), and click **Open**.
4. In the VeriStand Editor, click the **Project Files** tab.
5. Right-click the new system definition file and click **Make Active**.

Versioning a System Definition File

Create and test modifications to your system definition file by duplicating the file in the **VeriStand Editor**.

Before you begin, create a VeriStand project.

VeriStand supports the versioning of project documents, including the system definition. Copy an existing system definition to test changes. The original file serves as a restoration point whereas the copy can be modified.

1. Open a project in the VeriStand Editor.
2. In the Navigation pane, click **Project Files**.
3. Right-click the existing system definition file (.nivssdf) and select **Duplicate**.

> **Note** If the system definition has unsaved changes, you will be prompted to save the file.

4. **Optional:** Update the name of the new system definition.
    a. Right-click the file and select **Rename**.
    b. Enter a new name for the file.
5. Modify the new system definition file.

6. Right-click the modified system definition and select **Make Active**.
7. Select **Operate** » **Deploy** to deploy the modified system definition.
8. **Optional:** Remove a system definition file by right-clicking the file and selecting **Delete**.

> **Note** You cannot remove an active system definition. Right-click the system definition and select **Make Inactive** before deleting the file.

Connecting to a Target System Definition

Connect to the target's deployed system definition to send and receive data.

Before you begin, you must deploy the system definition to a real-time target.

1. Open a project in VeriStand.
2. In the VeriStand Editor, select **Operate** » **Connect**.

Specifying a Target

Select a target and designate its name, operating system, and IP address.

Update the target specification when reusing a system definition with a new real-time target.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, double click the system definition file (.nivssdf) to open the **Mapping Diagram**.
3. In the **Select Target** drop-down, select a target.

> **Note** The **Select Target** drop-down only appears when you have more than one target in your system definition.

4. In the Configuration pane, click **Document**.
5. Enter the target's **Name**.
6. In the Operating System drop-down, select the target's operating system.

7. Enter the target's **IP Address**.

> **Note** VeriStand does not support remote Windows targets. The IP address will remain localhost.

8. Click **File** » **Save all**.

Configuring the VeriStand Engine

Control the timing of the system and the communication between the target and host computer.

Before you begin, you should understand the VeriStand Engine and the PCL execution mode steps.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Select **Controller** in the configuration tree.
4. Use the Controller Configuration page that appears to the right of the tree to configure the VeriStand Engine.
5. Save the system definition file.

Adding and Configuring a Procedure

Set the actions the VeriStand Engine executes in response to an alarm, when called from another procedure, or as a startup procedure.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Procedures** in the configuration tree.
4. Click **Add Procedure** to add an empty procedure to the configuration tree.
5. Use the Procedure Configuration page that appears to the right of the configuration tree to configure the procedure.
6. Save the system definition file.

Now that you have added a procedure, you can call that procedure from multiple alarms.

Calling One Procedure from Multiple Alarms

Configure an Alarm Command step to trigger an alarm that executes a procedure without resetting/disabling the alarm and exiting the subroutine.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Procedures** in the configuration tree.
4. Click a procedure.
5. Click **Add** » **Alarm Command**.
6. Double-click the alarm command you added.
7. In the Alarm Command Step Configuration page, click the **Function** drop-down, and select an option that resets or disables the alarm.

   Note The Alarm field should be grayed out, indicating that the step will execute for the calling alarm. If it is not, enable **Apply settings to the alarm that tripped this procedure**.

8. Save the system definition file.

Adding and Configuring Alarms

Notify users when a channel value is outside a specified range.

Add and configure alarms with the VeriStand Editor, System Explorer, and Workspace.

Depending on your goal, access the following locations to add and configure alarms.

Goal	Location	How to Access
▪ Add a new alarm to the system definition file.	VeriStand Editor	Use the System Definition palette.

Goal	Location	How to Access
▪ Assign alarms to groups. ▪ Set alarm priorities. ▪ Configure alarm settings.		
Configure an alarm to trigger for a specific channel value.		Use the Mapping Diagram.
▪ Manage alarms at run time in the VeriStand Editor. ▪ View the current status of alarms in a deployed project. ▪ Acknowledge and unacknowledge alarms. ▪ Enable and disable alarms. ▪ View the history of triggered alarms and export the history to a file.		Click **View** » **Alarm Monitor**,
▪ Add a new procedure to the system definition file. ▪ Configure the automated actions that occur during a procedure.	System Explorer	Click **Targets** » **Controller** » **Procedures**.
▪ Manage alarms at run time in the Workspace. ▪ View the current status of alarms in a deployed project.	Workspace	Open the Alarm Monitor tool.

Goal	Location	How to Access
▪ Acknowledge alarms or mark them as unacknowledged. ▪ Enable and disable alarms. ▪ View the history of triggered alarms and export the history to a file.		

Related reference:

- VeriStand Environment

Adding an Alarm

Add alarms to notify users that the value of a channel is outside a specified range or to trigger a procedure.

1. Launch your project in the VeriStand Editor.
2. Click **View** » **System Definition**.
3. **Optional:** In the System Definition Palette, right-click **Alarms** and select **New** » **Folder** to create a sub-folder to organize your alarms.
4. Right click **Alarms** or the sub-folder you created and select **New** » **Alarm**.
5. In the Create a New Alarm dialog box, configure the alarm.

   Note If you need the **Re-arm behavior** to trigger based on a procedure, create the procedure first.

6. Click **OK**. The new alarm appears under **Alarms** or in the subfolder you selected.

   Note You can update the alarm configuration by right-clicking the alarm and selecting **Properties**.

7. Save the system definition file.

After adding an alarm, you can right-click the alarm and select **Add status channel**. Use a status channel to monitor when an alarm triggers or clears.

Related tasks:

- Setting an Alarm Priority
- Assigning an Alarm Group

Assigning an Alarm Group

Assign an alarm group to execute one alarm procedure at a time.

Alarm groups are used to determine the execution order of re-arm procedures. You can assign alarms to different groups. If a system definition file contains multiple alarm groups, one procedure per alarm group executes simultaneously.

1. Launch your project in the VeriStand Editor.
2. Click **View** » **System Definition**.
3. Right-click an alarm and select **Properties**.
4. In the Properties dialog box, set the **Re-arm behavior** to **Procedure**.
5. Enter a **Group** number.

   Note One alarm group is automatically created for each alarm that uses procedural re-arm behavior. All alarms default to group 0.

6. **Optional:** Enter a **Priority** number.

   Note Procedures execute differently in a group based on the priority you set for each procedure.

7. Click **OK**.
8. Save the system definition file.

Alarm Group Execution

Alarm groups and priorities determine when procedures execute. Only one alarm procedure executes at a time in an alarm group.

Use the following table to learn the execution order of procedures based on the order of the alarms.

Note If an alarm is set to **Indicate Only** mode, the alarm trips when the monitored channel goes out of range but does not execute a procedure.

Order of alarms	Order of procedures
Multiple alarms go out of range simultaneously.	The alarm with the highest priority executes its procedure first.
A low-priority alarm trips before a high-priority alarm.	The high-priority procedure interrupts the low-priority alarm. After the high-priority procedure finishes and resets, the low-priority procedure resumes.
A high-priority alarm trips before a low-priority alarm.	The high-priority procedure executes, finishes, and resets the high-priority alarm. If the low-priority alarm condition is still met after the high-priority alarm resets, the low-priority alarm trips. **Note** If the channel value for the low-priority alarm returns to a normal range before the high-priority alarm resets, the low-priority procedure does not execute.

Setting an Alarm Priority

Set an alarm priority to ensure high severity alarms execute immediately, interrupting other alarms that may be executing when multiple alarms are monitoring the same system.

1. Launch your project in the VeriStand Editor.
2. Click **View** » **System Definition**.
3. Right-click an alarm and select **Properties**.
4. In the Properties dialog box, set the **Re-arm behavior** to **Procedure**.

5. Enter a **Priority** number.

> **Note** Zero (0) is the highest priority number. For more information on priority execution, refer to Alarm Group Execution.

6. Click **OK**.
7. Save the system definition file.

Triggering an Alarm for a Specific Channel Value

Create an alarm that trips when a channel reaches an exact value.

1. Launch your project in the VeriStand Editor.
2. Create and configure a conditional calculated channel to represent your alarm trigger.
    a. On the Mapping Diagram, create a mapping wire from the conditional calculated channel's **X** terminal to the channel you want to monitor.
    b. In the configuration pane, use the **Condition** drop-down menu to select **=**.
    c. Enable **Use constant Y** value and enter a **Y value** that will trip the alarm.
    d. Enable **Use constant W** value and set the **W value** as 1.
    e. Enable **Use constant Z** value and set the **Z Value** as 0.
3. Add an alarm.
    a. Map the **Source** to the calculated channel.
    b. In the **Upper Limit** drop-down menu, select **Constant** and enter the **Value** as 0.000.
    c. In the **Lower Limit** drop-down menu, select **Constant** and enter the **Value** as 0.000.

> **Note** When the calculated channel detects the channel value you specified (y value), the value of the calculated channel changes to 1 and the alarm trips.

4. Save the system definition file.

Adding and Configuring a Hardware Device

Add and configure NI hardware, timing and sync devices, and custom devices.

1. Open a project in VeriStand.
2. Depending on your hardware goal, complete any of the following tasks in System Explorer.

Goal	Task
Adding and configuring an SLSC device	Use SLSC devices to introduce signal conditioning and fault insertion into a real-time testing scheme.
Adding and configuring DAQ devices	Use DAQ devices to support analog, digital, and counter I/O functions such as acquiring waveform data.
Adding NI FPGA targets	Use NI FPGA targets to create customizable I/O, help with data preprocessing and postprocessing, add high-speed closed-loop control, and simulate a variety of sensors for hardware-in-the-loop testers
Adding NI-XNET devices	Use the NI-XNET platform to communicate with hardware using the CAN, LIN, and FlexRay protocols.
Adding reflective memory networks	Use a reflective memory card to split up a simulation model to execute simultaneously on different target systems.
Adding and configuring timing and sync devices	Use a timing and sync device to synchronize more than one chassis.
Synchronizing hardware and software	Synchronize the hardware and software components of a system to ensure

Goal	Task
	consistency and optimal performance, enable data analysis, and time correlation.
Setting chassis master hardware synchronization devices	Use a chassis master hardware synchronization device to control the synchronization of all hardware in a chassis.

Adding and Configuring an SLSC device

Use SLSC devices to introduce signal conditioning and fault insertion into a real-time testing scheme.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **SLSC** in the configuration tree.
4. Right-click **SLSC** and click **Add SLSC Chassis**.
5. In the **Add SLSC Chassis** dialog box, use the **Connect with** drop-down to specify whether to connect to the chassis using the chassis name or the host name or IP address.
6. Enter the chassis name, host name, or IP address.
7. Enter a **Username** and **Password** for the engine to log into the SLSC chassis.

   Note Entering a user name and password removes the requirement for custom devices to separately log into the SLSC chassis. If the credentials are incorrect when the engine attempts to log into the SLSC chassis, VeriStand will undeploy the system definition. By default, the user name is anonymous and the password left empty.

8. Click **OK**.
9. In the configuration tree, navigate to **SLSC Chassis** » **Modules** under your SLSC chassis.
10. Use the drop-down menus under SLSC Modules Settings to select the modules in each of the chassis slots.

11. Save the system definition file.

Related concepts:

- SLSC Systems in Real-Time Schemes

SLSC Systems in Real-Time Schemes

A SLSC system is composed of chassis and module devices.
A **SLSC chassis** interfaces with the network and provides multiple slots for SLSC modules. A **SLSC module** contains circuitry for switching, loads, and signal conditioning.
The device under test and the measurement device send signals to the SLSC system for signal conditioning. The SLSC system passes the signals on to the next stage in the real-time scheme. Signals can pass back and forth through this process from either the measurement device or the device under test. The direction of the signals depends on the set up of the real-time scheme.
SLSC devices are assigned names using the following user-defined format.

Device	Name Format
Chassis	SLSC-<chassis_model>-<chassis_serial_number>
Module	<chassis_name>-Mod<slot_number>

Adding and Configuring a DAQ Device

Use DAQ devices to support analog, digital, and counter I/O functions such as acquiring waveform data.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **DAQ** in the configuration tree.
4. Choose to add one or all discoverable DAQ devices.

Number of Devices	How to Add
One	a. Click **Add DAQ Device**.

Number of Devices	How to Add
	b. Use the Add DAQ Device dialog box to identify the DAQ device you want to add. c. Click **OK**.
All	a. Click **Hardware Discovery Wizard** and follow the onscreen instructions. b. Right-click a device and select **Add Channels** to display the Add DAQ Channels dialog box.

5. Use the Add DAQ Channels dialog box to configure the type of physical channel to add and their properties.

> **Note** For analog input channels, you must choose to acquire the signal a single point at a time or over a period of time as waveform.

6. Click **Next**.
7. Select the specific channels on the device you want to add to the system definition.

> **Note** If the DAQ device does not contain channels of the type you specified, such as AO or DI, no channels are available to select.

8. Click **Finish**.
9. Save the system definition file.

After you add a DAQ device, you can add and configure more channels. Add an internal channel to a DAQ device by right-clicking the device and selecting **Add Internal Channel**.

You can also add a SCXI module to a DAQ device by right-clicking the device and selecting **Add SCXI Modules**. Use the Add SCXI Module dialog box to configure the module.

Adding and Configuring DAQ Device Channels

Define how VeriStand performs measurements using DAQ channels.

Before you begin, add a DAQ device.

Each measurement type has configurable properties. For example, counter input channels that count up have a count edge property that sets whether the channel counts rising edges or falling edges.



Note Some properties are only available on certain devices. Refer to the documentation for your hardware device to find out what properties your device supports.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **DAQ** in the configuration tree.
4. Right-click a DAQ device and click **Add Channels**.
5. Determine the channel type you want based on the measurement type.

Measurement type	Channel type
▪ Accelerometer ▪ Bridge ▪ Current ▪ Force ▪ Pressure ▪ Resistance Temperature Detector (RTD) ▪ Strain ▪ Thermistor Iex ▪ Thermistor Vex ▪ Thermocouple ▪ Torque	Analog Input (AI)

Measurement type	Channel type
▪ Voltage	
▪ Current ▪ Voltage	Analog Output (AO)
Digital Input	Digital Input (DI)
Digital Output	Digital Output (DO)
▪ Count Up/Down ▪ Frequency ▪ Position ▪ Pulse Measurement ▪ Time Period	Counter Input (CI)
Pulse Generation	Counter Output (CO)

6. In the Add DAQ Channels dialog box, use the **Select channel type to add** pull-down to select the channel type.
7. Use the **Select measurement type** pull-down to select the measurement type.
8. Click **Next**.
9. Select the channel(s) you want to add and click **Finish**.

> **Note** If the DAQ device does not contain channels of the type you specified, no channels are available to select.

10. Save the system definition file.

Accelerometer Channel Properties (AI)

Configure an accelerometer's analog input (AI) channel properties to measure acceleration.

For more information on accelerometers, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Input Configuration	Specifies the input terminal configuration to apply to the device channels. **Note** If you select any configuration other than Same as device, it overrides the configuration you specify for the device on the DAQ Device Configuration page in System Explorer. ▪ **Same as device**—The same configuration specified for the DAQ device itself. To set the input terminal configuration at the device level, use the **Input Configuration** pull-down menu on the DAQ Device Configuration page. ▪ **Default**—At run time, NI-DAQmx chooses the default terminal configuration for the channel. ▪ RSE—Referenced single-ended mode. ▪ NRSE—Non-referenced single-ended mode. ▪ Differential—Differential mode. ▪ Pseudodifferential—Pseudodifferential mode.
Sensitivity	The sensitivity of the sensor in the units you specify. Refer to the sensor documentation to determine this value.
Current Excitation Source	Specifies the source of excitation:

Property/Section	Description
	▪ **External**—Use an external excitation source instead of the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. ▪ **None**—Supply no excitation to the channel. For both internal and external current excitation sources, you must use the Current Excitation Value property.
Current Excitation Value	The amount of excitation to supply to the sensor. Refer to the sensor documentation to determine this value.
dB Reference	Specifies the decibel reference level. When you read samples as a waveform, the decibel reference level is included in the waveform attribute.
Coupling Mode	Specifies the coupling for the channel: ▪ **AC**—Removes the DC offset from the signal. ▪ **DC**—Allows VeriStand to measure all of the signal.

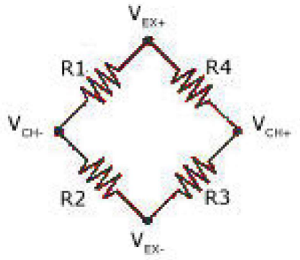
Bridge Channel Properties (AI)

Configure a bridge-based sensor's analog input (AI) channel properties to measure its output.

For more information on bridge-based sensors, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.

Property/Section	Description
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Bridge Configuration	Specifies what type of bridge configuration to use: ▪ **Full Bridge** ▪ **Half Bridge** ▪ **Quarter Bridge**
Excitation Source	Specifies the source of excitation: ▪ **External**—Use an excitation source other than the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. For both internal and external excitation sources, you must use the Excitation Value property.
Excitation Value	Specifies the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine this value.
Nominal Bridge Resistance	Specifies the resistance of the bridge while not under load.
Enable Offset Null	Specifies whether to perform a bridge offset nulling calibration.
Enable Shunt Calibration	Specifies whether to perform a shunt calibration.
Shunt Element Location	Specifies the location of the shunt resistor: ▪ **R1**—Between V_{ch-} and V_{ex+} ▪ **R2**—Between V_{ch-} and V_{ex-} ▪ **R3**—Between V_{ch+} and V_{ex-} ▪ **R4**—Between V_{ch+} and V_{ex+}

Property/Section	Description
	
Shunt Resistance	Specifies the shunt resistance.
Filter Type (supported devices only)	Specifies whether to apply a digital filter to the input signal: ▪ **Disabled**—No filter. ▪ **Lowpass**—Eliminates all signal frequency components above the cutoff frequency.
Filter Cutoff Frequency (supported devices only)	Specifies the cutoff frequency of the digital filter.
In-Situ Calibration (supported devices only)	Specifies whether to perform an in-situ calibration and when to perform the calibration: ▪ **Before deployment**—Perform an in-situ calibration before the system definition is deployed. ▪ **After undeployment**—Perform an in-situ calibration after the system definition is no longer deployed. ▪ **Both**—Perform an in-situ calibration before the system definition is deployed and after the system definition is no longer deployed. ▪ **None**—Do not perform an in-situ calibration.
In-Situ Minimum Value (supported devices only)	The minimum value you expect to measure when VeriStand performs an in-situ calibration.
In-Situ Maximum Value (supported devices only)	The maximum value you expect to measure when VeriStand performs an in-situ calibration.

Property/Section	Description
In-Situ Input Configuration (supported devices only)	Specifies the input terminal configuration to use when performing the in-situ calibration: ▪ **Same as device**—The same configuration specified for the DAQ device itself. To set the input terminal configuration at the device level, use the **Input Configuration** pull-down menu on the DAQ Device Configuration page. ▪ **Default**—At run time, NI-DAQmx chooses the default terminal configuration for the channel. ▪ RSE—Referenced single-ended mode. ▪ NRSE—Non-referenced single-ended mode. ▪ Differential—Differential mode. ▪ Pseudodifferential—Pseudodifferential mode.

Current Channel Properties (AI)

Configure analog input (AI) channel properties to measure current.

For more information on measuring current, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Input Configuration	Specifies the input terminal configuration to apply to the device channels. **Note** If you select any configuration other than Same as device, it overrides the configuration you

Property/Section	Description
	specify for the device on the DAQ Device Configuration page in System Explorer. ▪ **Same as device**—The same configuration specified for the DAQ device itself. To set the input terminal configuration at the device level, use the **Input Configuration** pull-down menu on the DAQ Device Configuration page. ▪ **Default**—At run time, NI-DAQmx chooses the default terminal configuration for the channel. ▪ RSE—Referenced single-ended mode. ▪ NRSE—Non-referenced single-ended mode. ▪ Differential—Differential mode. ▪ Pseudodifferential—Pseudodifferential mode.
External Shunt Resistance	Specifies the external shunt resistance.

Force Channel Properties (AI)

Configure analog input (AI) channel properties to measure force.

For more information on measuring force, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Bridge Configuration	Specifies what type of bridge configuration to use: ▪ **Full Bridge**

Property/Section	Description
	▪ **Half Bridge** ▪ **Quarter Bridge**
Excitation Source	Specifies the source of excitation: ▪ **External**—Use an excitation source other than the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. For both internal and external excitation sources, you must use the Excitation Value property.
Excitation Value	Specifies the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine this value.
Nominal Bridge Resistance	Specifies the resistance of the bridge while not under load.
Electrical Units	Specifies from which electrical unit to scale the data. Select the same unit that the sensor data sheet or calibration certificate uses for electrical values.
Scale: 1st Electrical Value	Specifies the first electrical value used to calculate the slope and y-intercept of a two-point linear equation to scale electrical values to physical values.
Scale: 1st Physical Value	Specifies the physical value that corresponds to the first electrical value.
Scale: 2nd Electrical Value	Specifies the second electrical value used to calculate the slope and y-intercept of a two-point linear equation to scale electrical values to physical values.
Scale: 2nd Physical Value	Specifies the physical value that corresponds to the second electrical value.
Filter Type (supported devices only)	Specifies whether to apply a digital filter to the input signal:

Property/Section	Description
	▪ **Disabled**—No filter. ▪ **Lowpass**—Eliminates all signal frequency components above the cutoff frequency.
Filter Cutoff Frequency (supported devices only)	Specifies the cutoff frequency of the digital filter.

Pressure Channel Properties (AI)

Configure analog input (AI) channel properties to measure pressure.

For more information on measuring pressure, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Bridge Configuration	Specifies what type of bridge configuration to use: ▪ **Full Bridge** ▪ **Half Bridge** ▪ **Quarter Bridge**
Excitation Source	Specifies the source of excitation: ▪ **External**—Use an excitation source other than the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. For both internal and external excitation sources, you must use the Excitation Value property.

Property/Section	Description
Excitation Value	Specifies the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine this value.
Nominal Bridge Resistance	Specifies the resistance of the bridge while not under load.
Electrical Units	Specifies from which electrical unit to scale the data. Select the same unit that the sensor data sheet or calibration certificate uses for electrical values.
Scale: 1st Electrical Value	Specifies the first electrical value used to calculate the slope and y-intercept of a two-point linear equation to scale electrical values to physical values.
Scale: 1st Physical Value	Specifies the physical value that corresponds to the first electrical value.
Scale: 2nd Electrical Value	Specifies the second electrical value used to calculate the slope and y-intercept of a two-point linear equation to scale electrical values to physical values.
Scale: 2nd Physical Value	Specifies the physical value that corresponds to the second electrical value.
Filter Type (supported devices only)	Specifies whether to apply a digital filter to the input signal: ▪ **Disabled**—No filter. ▪ **Lowpass**—Eliminates all signal frequency components above the cutoff frequency.
Filter Cutoff Frequency (supported devices only)	Specifies the cutoff frequency of the digital filter.

RTD Channel Properties (AI)

Configure analog input (AI) channel properties to measure temperature from a Resistance Temperature Detector (RTD).

Platinum RTDs use a linearization curve known as the Callendar-Van Dusen equation to measure the temperature of RTDs. For more information on measuring temperature, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Resistance Configuration	Specifies the number of wires to use for resistive measurements: ▪ 2-Wire ▪ 3-Wire ▪ 4-Wire
Excitation Source	Specifies the source of excitation: ▪ **External**—Use an external excitation source instead of the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. ▪ **None**—Supply no excitation to the channel. For both internal and external current excitation sources, you must use the Excitation Value property.
RTD Type	Specifies the type of RTD connected to the channel: ▪ **Custom**—Use a custom RTD. You must use the three Custom RTD Const properties to supply the coefficients for the Callendar-Van Dusen equation. ▪ **Pt3750**

Property/Section	Description
	▪ Pt3851 ▪ Pt3911 ▪ Pt3916 ▪ Pt3920 ▪ Pt3928
r0	The sensor resistance at 0 degrees Celsius. The Callendar-Van Dusen equation requires this value. Refer to the sensor documentation to determine this value.
Excitation Value	Specifies the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine this value.
Custom RTD A const	Specifies the A constant of the Callendar-Van Dusen equation. VeriStand requires this value when you use a custom RTD and specify **Custom** for the RTD Type property.
Custom RTD B const	Specifies the B constant of the Callendar-Van Dusen equation. VeriStand requires this value when you use a custom RTD and specify **Custom** for the RTD Type property.
Custom RTD C const	Specifies the C constant of the Callendar-Van Dusen equation. VeriStand requires this value when you use a custom RTD and specify **Custom** for the RTD Type property.

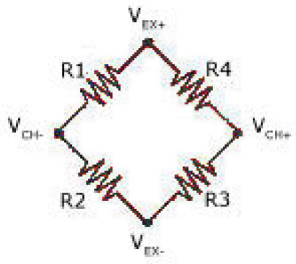
Strain Channel Properties (AI)

Configure analog input (AI) channel properties to measure strain.

For more information on measuring strain, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.

Property/Section	Description
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Strain Configuration	Specifies what type of bridge configuration to use for the strain gages: ▪ Full Bridge I ▪ Full Bridge II ▪ Full Bridge III ▪ Half Bridge I ▪ Half Bridge II ▪ Quarter Bridge I ▪ Quarter Bridge II
Excitation Source	Specifies the source of excitation: ▪ **External**—Use an external excitation source instead of the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. ▪ **None**—Supply no excitation to the channel. For both internal and external excitation sources, you must use the Excitation Value property.
Excitation Value	Specifies the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine this value.
Lead Resistance	Specifies the amount of resistance in the lead wires. Ideally, this value is the same for all leads.
Initial Bridge Voltage	Specifies the output voltage of the bridge in the unloaded condition. VeriStand subtracts this value from any measurements before applying scaling equations. Perform a voltage measurement on the bridge with no strain applied to determine this value.

Property/Section	Description
Gage Factor	Specifies the sensitivity of the strain gages and relates the change in electrical resistance to the change in strain. Each gage in the bridge must have the same gage factor. Refer to the sensor documentation to determine this value.
Nominal Gage Resistance	Specifies the resistance of the gages in an unstrained position. Each gage in the bridge must have the same nominal gage resistance. The resistance across arms of the bridge that do not have strain gages must also be the same as the nominal gage resistance. Refer to the sensor documentation to determine this value.
Poisson Ratio	Specifies the ratio of lateral strain to axial strain in the material you are measuring.
Enable Offset Null	Specifies whether to perform a bridge offset nulling calibration.
Enable Shunt Calibration	Specifies whether to perform a shunt calibration.
Shunt Element Location	Specifies the location of the shunt resistor: ▪ **R1**—Between V_{ch-} and V_{ex+} ▪ **R2**—Between V_{ch-} and V_{ex-} ▪ **R3**—Between V_{ch+} and V_{ex-} ▪ **R4**—Between V_{ch+} and V_{ex+} 
Shunt Resistance	Specifies the shunt resistance.
Filter Type (supported devices only)	Specifies whether to apply a digital filter to the input signal: ▪ **Disabled**—No filter.

Property/Section	Description
	▪ **Lowpass**—Eliminates all signal frequency components above the cutoff frequency.
Filter Cutoff Frequency (supported devices only)	Specifies the cutoff frequency of the digital filter.
In-Situ Calibration (supported devices only)	Specifies whether to perform an in-situ calibration and when to perform the calibration: ▪ **Before deployment**—Perform an in-situ calibration before the system definition is deployed. ▪ **After undeployment**—Perform an in-situ calibration after the system definition is no longer deployed. ▪ **Both**—Perform an in-situ calibration before the system definition is deployed and after the system definition is no longer deployed. ▪ **None**—Do not perform an in-situ calibration.
In-Situ Minimum Value (supported devices only)	The minimum value you expect to measure when VeriStand performs an in-situ calibration.
In-Situ Maximum Value (supported devices only)	The maximum value you expect to measure when VeriStand performs an in-situ calibration.
In-Situ Input Configuration (supported devices only)	Specifies the input terminal configuration to use when performing the in-situ calibration: ▪ **Same as device**—The same configuration specified for the DAQ device itself. To set the input terminal configuration at the device level, use the **Input Configuration** pull-down menu on the DAQ Device Configuration page. ▪ **Default**—At run time, NI-DAQmx chooses the default terminal configuration for the channel. ▪ RSE—Referenced single-ended mode.

Property/Section	Description
	▪ NRSE—Non-referenced single-ended mode. ▪ Differential—Differential mode. ▪ Pseudodifferential—Pseudodifferential mode.

Thermistor Iex Channel Properties (AI)

Configure analog input (AI) channel properties to measure current excitation temperature using a thermistor.

NI-DAQmx scales the resistance of a thermistor to a temperature using the Steinhart-Hart thermistor equation. For more information on measuring temperature, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Resistance Configuration	Specifies the number of wires to use for resistive measurements: ▪ 2-Wire ▪ 3-Wire ▪ 4-Wire
Excitation Source	Specifies the source of excitation: ▪ **External**—Use an external excitation source instead of the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. ▪ **None**—Supply no excitation to the channel.

Property/Section	Description
	For both internal and external current excitation sources, you must use the Current Excitation Value property.
Excitation Value	Specifies the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine this value.
A	Specifies the A constant for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.
B	Specifies the B constant for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.
C	Specifies the C constant for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.

Thermistor Vex Channel Properties (AI)

Configure analog input (AI) channel properties to measure voltage excitation temperature using a thermistor.

NI-DAQmx scales the resistance of a thermistor to a temperature using the Steinhart-Hart thermistor equation. For more information on measuring temperature, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Resistance Configuration	Specifies the number of wires to use for resistive measurements:

Property/Section	Description
	▪ 2-Wire ▪ 3-Wire ▪ 4-Wire
Excitation Source	Specifies the source of excitation: ▪ **External**—Use an external excitation source instead of the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. ▪ **None**—Supply no excitation to the channel. For both internal and external current excitation sources, you must use the Current Excitation Value property.
r1	Specifies the value of the reference resistor.
Excitation Value	Specifies the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine this value.
A	Specifies the A constant for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.
B	Specifies the B constant for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.
C	Specifies the C constant for the Steinhart-Hart thermistor equation. Refer to the sensor documentation to determine values for these constants.

Thermocouple Channel Properties (AI)

Configure analog input (AI) channel properties to measure temperature using a thermocouple.

Thermocouples require cold-junction compensation (CJC) for temperature references. For more information on measuring temperature, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Thermocouple Type	Specifies the type of thermocouples connected to the channel. Thermocouple types differ in composition and measurement range. ▪ **B**—B-type thermocouple. ▪ **E**—E-type thermocouple. ▪ **J**—J-type thermocouple. ▪ **K**—K-type thermocouple. ▪ **N**—N-type thermocouple. ▪ **R**—R-type thermocouple. ▪ **S**—S-type thermocouple. ▪ **T**—T-type thermocouple.
CJC Value	Specifies the temperature of the cold-junction if you set the CJC Source property to **Constant Value**.
CJC Source	Specifies the source of cold-junction compensation: ▪ **Constant Value**—Use the CJC Value property to specify the cold-junction temperature. ▪ **Internal**—Use a cold-junction compensation channel built into the terminal block.

Torque Channel Properties (AI)

Configure analog input (AI) channel properties to measure torque.

For more information on measuring torque, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Bridge Configuration	Specifies what type of bridge configuration to use: ▪ **Full Bridge** ▪ **Half Bridge** ▪ **Quarter Bridge**
Excitation Source	Specifies the source of excitation: ▪ **External**—Use an excitation source other than the built-in excitation source of the device. ▪ **Internal**—Use the built-in excitation source of the device. For both internal and external excitation sources, you must use the Excitation Value property.
Excitation Value	Specifies the amount of excitation supplied to the sensor. Refer to the sensor documentation to determine this value.
Nominal Bridge Resistance	Specifies the resistance of the bridge while not under load.
Electrical Units	Specifies from which electrical unit to scale the data. Select the same unit that the sensor data

Property/Section	Description
	sheet or calibration certificate uses for electrical values.
Scale: 1st Electrical Value	Specifies the first electrical value used to calculate the slope and y-intercept of a two-point linear equation to scale electrical values to physical values.
Scale: 1st Physical Value	Specifies the physical value that corresponds to the first electrical value.
Scale: 2nd Electrical Value	Specifies the second electrical value used to calculate the slope and y-intercept of a two-point linear equation to scale electrical values to physical values.
Scale: 2nd Physical Value	Specifies the physical value that corresponds to the second electrical value.
Filter Type (supported devices only)	Specifies whether to apply a digital filter to the input signal: ▪ **Disabled**—No filter. ▪ **Lowpass**—Eliminates all signal frequency components above the cutoff frequency.
Filter Cutoff Frequency (supported devices only)	Specifies the cutoff frequency of the digital filter.

Voltage Channel Properties (AI)

Configure analog input (AI) channel properties to measure voltage.

For more information on measuring voltage, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.

Property/Section	Description
Input Configuration	Specifies the input terminal configuration to apply to the device channels. **Note** If you select any configuration other than Same as device, it overrides the configuration you specify for the device on the DAQ Device Configuration page in System Explorer. ▪ **Same as device**—The same configuration specified for the DAQ device itself. To set the input terminal configuration at the device level, use the **Input Configuration** pull-down menu on the DAQ Device Configuration page. ▪ **Default**—At run time, NI-DAQmx chooses the default terminal configuration for the channel. ▪ RSE—Referenced single-ended mode. ▪ NRSE—Non-referenced single-ended mode. ▪ Differential—Differential mode. ▪ Pseudodifferential—Pseudodifferential mode.
Filter Type (supported devices only)	Specifies whether to apply a digital filter to the input signal: ▪ **Disabled**—No filter. ▪ **Lowpass**—Eliminates all signal frequency components above the cutoff frequency.
Filter Cutoff Frequency (supported devices only)	Specifies the cutoff frequency of the digital filter.
In-Situ Calibration (supported devices only)	Specifies whether to perform an in-situ calibration and when to perform the calibration:

Property/Section	Description
	▪ **Before deployment**—Perform an in-situ calibration before the system definition is deployed. ▪ **After undeployment**—Perform an in-situ calibration after the system definition is no longer deployed. ▪ **Both**—Perform an in-situ calibration before the system definition is deployed and after the system definition is no longer deployed. ▪ **None**—Do not perform an in-situ calibration.
In-Situ Minimum Value (supported devices only)	The minimum value you expect to measure when VeriStand performs an in-situ calibration.
In-Situ Maximum Value (supported devices only)	The maximum value you expect to measure when VeriStand performs an in-situ calibration.
In-Situ Input Configuration (supported devices only)	Specifies the input terminal configuration to use when performing the in-situ calibration: ▪ **Same as device**—The same configuration specified for the DAQ device itself. To set the input terminal configuration at the device level, use the **Input Configuration** pull-down menu on the DAQ Device Configuration page. ▪ **Default**—At run time, NI-DAQmx chooses the default terminal configuration for the channel. ▪ RSE—Referenced single-ended mode. ▪ NRSE—Non-referenced single-ended mode. ▪ Differential—Differential mode. ▪ Pseudodifferential—Pseudodifferential mode.
Lowpass Cutoff Filter Frequency (supported devices only)	Specifies the frequency that corresponds to the -3dB cutoff of the analog filter.

Current Channel Properties (AO)

Configure analog output (AO) channel properties to generate current.

For more information on generating current, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.

Note VeriStand limits the values it writes to AO channels to values between the minimum and maximum values.

Voltage Channel Properties (AO)

Configure analog output (AO) channel properties to generate voltage.

For more information on generating voltage, refer to the NI-DAQmx Manual.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.

Note VeriStand limits the values it writes to AO channels to values between the minimum and maximum values.

Digital Input Channel Properties (DI)

Configure DI channel properties to measure digital signals.

For more information on measuring digital signals, refer to the NI-DAQmx Manual.

Property/Section	Description
Invert Lines	Specifies whether to reverse digital line polarity: ▪ **False**—Do not invert lines. ▪ **True**—Invert lines. ▪ **Same as port**—Use the same option as the port that contains the channel. To set this option, on the Port Configuration page, click **Invert digital lines**.

Digital Output Channel Properties (DO)

Configure DO channel properties to generate digital signals.

For more information on generating digital signals, refer to the NI-DAQmx Manual.

Property/Section	Description
Invert Lines	Specifies whether to reverse digital line polarity: ▪ **False**—Do not invert lines. ▪ **True**—Invert lines. ▪ **Same as port**—Use the same option as the port that contains the channel. To set this option, on the Port Configuration page, click **Invert digital lines**.

Count Up/Count Down Channel Properties (CI)

Configure counter input (CI) channel properties to count the rising or falling edges of a digital signal.

For more information on counting edges, refer to the NI-DAQmx Manual.

Property/Section	Description
Count Edge	Specifies on which edges of the input signal to increment or decrement the count:

Property/Section	Description
	▪ **Falling**—Count falling edges. ▪ **Rising**—Count rising edges.
Count Direction	Specifies whether to increment or decrement the counter on each edge of the type you specify with the Count Edge property: ▪ **Count down**—Decrement counter. ▪ **Count up**—Increment counter. ▪ **Externally Controlled**—The state of a digital line controls the count direction. Each counter has a default count direction terminal.

Note For device specific information about the default terminals used for counter measurements and generations, refer to Connecting Counter Signals.

Frequency Channel Properties (CI)

Configure counter input (CI) channel properties to measure the frequency of a digital signal.

VeriStand measures the frequency of a digital signal by counting observed edges and performing a software calculation over a period of time determined by your expected frequency range.

Note This method uses a software calculation that may not provide the most reliable accuracy or update rate. Alternatively, if your DAQ card supports Pulse Measurement, use values from the Frequency channel located under the counter channels in System Explorer.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.

Property/Section	Description
	Note The minimum value of frequency determines, and is equal to, the measurement frequency resolution. The measurement time is equal to the reciprocal of the minimum frequency value.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Count Edge	Specifies on which edges of the input signal to increment or decrement the count: ▪ **Falling**—Count falling edges. ▪ **Rising**—Count rising edges.

Note For device specific information about the default terminals used for counter measurements and generations, refer to Connecting Counter Signals.

Position Channel Properties (CI)

Configure counter input (CI) channel properties for channels that use a linear encoder to measure linear position.

For more information on encoders and measuring linear displacement, refer to the NI-DAQmx Manual.

Property/Section	Description
Z Index Mode	Specifies the states at which signal A and signal B must be while signal Z is high for VeriStand to reset the measurement. **Note** If signal Z is never high while signal A and signal B are high, you

Property/Section	Description
	must choose a phase other than A High B High. ▪ **A High B High**—Reset the measurement when signal A and signal B are high. ▪ **A High B Low**—Reset the measurement when signal A is high and signal B is low. ▪ **A Low B High**—Reset the measurement when signal A is low and signal B high. ▪ **A Low B Low**—Reset the measurement when signal A and signal B are low. When signal Z transitions to high and how long it stays high varies from encoder to encoder. Refer to the documentation for the encoder to determine the timing of signal Z with respect to signal A and signal B.
Decoding	Specifies how to count and interpret the pulses the encoder generates on signal A and signal B. X1, X2, and X4 are valid for quadrature encoders only. ▪ **2 Pulse Counting**—Increment the count on rising edges of signal A. Decrement the count on rising edges of signal B. If you select this value, the Z Index Mode property is ignored. ▪ **X1**—If signal A leads signal B, count the rising edges of signal A. If signal B leads signal A, count the falling edges of signal A. ▪ **X2**— Count the rising and falling edges of signal A

Property/Section	Description
	▪ **X4**—Count the rising and falling edges of signal A and signal B. **Note** 2 Pulse Counting is valid only for two-pulse encoders. X2 and X4 decoding are more sensitive to smaller changes in position than X1 encoding, with X4 being the most sensitive. However, more sensitive decoding is more likely to produce erroneous measurements if vibration exists in the encoder or other noise exists in the signals.

Note For device specific information about the default terminals used for counter measurements and generations, refer to Connecting Counter Signals.

Pulse Measurement Channel Properties (CI)

Configure counter input (CI) channel properties to measure pulse specifications.

A **pulse specification** is a pairing of high time and low time values.

Note Only X Series DAQ devices support this measurement type.

VeriStand returns measurements for pulse specifications as pairs of frequency and duty cycle values. You can access these values for each counter input channel that performs pulse measurements through two additional channels, Frequency and Duty Cycle, that appear under the counter channels in System Explorer.

Note If you assign this measurement type to one or more counter input channels, you cannot assign a different measurement type to other counter input channels on the same device.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure before VeriStand performs any scaling or calibration.
Sample Clock Source	Specifies the name of the source terminal of the sample clock. You can use an internal counter timebase when performing counter measurements or an external timebase.
Sample Clock Rate	Specifies in hertz the sampling rate in samples per channel per second. If you use an external source for the sample clock, set this input to the maximum expected rate of that clock.
Active Edge	Specifies whether a timebase cycle is from rising edge to rising edge or from falling edge to falling edge: ▪ **Falling**—Falling edge(s). ▪ **Rising**—Rising edge(s).
Timeout Value	Specifies an amount of time to wait for the channel to return valid data. VeriStand considers invalid data to be repeated values, which might occur if the system attempts to read data faster than the Sample Clock Rate property. When VeriStand reads invalid data, it continues to read from the channel while it counts until the Timeout Value. VeriStand will return values of NaN until the channel returns valid data again.

Note For device specific information about the default terminals used for counter measurements and generations, refer to Connecting Counter Signals.

Time Period Channel Properties (CI)

Configure counter input (CI) channel properties to measure the period of a digital signal.

With this measurement type, VeriStand measures the period of a digital signal by counting observed edges and performing a software calculation over a period of time determined by your expected range.

Note This method uses a software calculation that may not provide the most reliable accuracy or update rate. Alternatively, if your DAQ card supports Pulse Measurement, use values from the Frequency channel located under the counter channels in System Explorer.

Property/Section	Description
Minimum Value	The minimum value you expect to measure before VeriStand performs any scaling or calibration.
Maximum Value	The maximum value you expect to measure, before VeriStand performs any scaling or calibration. **Note** The maximum value of the time period determines, and is equal to, the measurement time required. The maximum update rate of the measurement is equal to the reciprocal of the maximum time period.
Count Edge	Specifies on which edges of the input signal to increment or decrement the count: ▪ **Falling**—Count falling edges. ▪ **Rising**—Count rising edges.

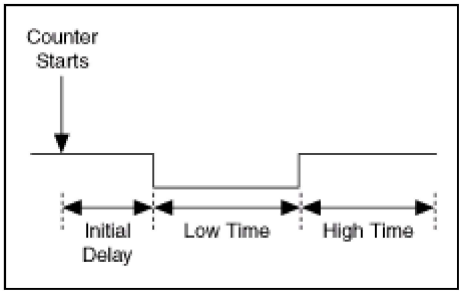
Note For device specific information about the default terminals used for counter measurements and generations, refer to Connecting Counter Signals.

Pulse Generation Channel Properties (CO)

Configure counter output (CO) channel properties to generate digital pulses.

VeriStand generates digital pulses that are defined by pairs of frequency and duty cycle values. Provide these values for each counter output channel that generate pulses through two additional channels, Frequency and Duty Cycle, that appear under the counter channels in System Explorer.

The following illustration shows the parts of a pulse.



For more information on generating pulses, refer to the NI-DAQmx Manual.

Note To assign this measurement type to channels on a non-X Series device, set Enable HWTSP to False. If Enable HWTSP is True, only X Series DAQ devices support this measurement type.

Property/Section	Description
Idle State	Specifies the resting state of the output terminal: ▪ **High**—Terminal is at a high state at rest. A pulse with a high idle state starts high, pulses to low, and returns to high. ▪ **Low**—Terminal is at a low state at rest. A pulse with a low idle state starts at the low value, pulses high, and returns to low.

Property/Section	Description
Initial Delay	Specifies the amount of time the output remains at the idle state before generating the pulse. The idle state always replaces high time or low time for the first pulse of a generation, depending on the idle state.
Enable HWTSP	▪ **True**—Generate pulses continuously using hardware timing without a buffer. This timing type is called **hardware-timed single-point sample mode**. ▪ **False**—Generates pulses continuously without specifying timing. This timing type is called **implicit** because the signal being measured is itself the timing signal or the timing is implicit in the rate of the generated pulse train. **Note** Implicit timing is appropriate when the measurement does not require sample timing, such as with counters for buffered frequency measurement, buffered period measurement, or pulse train generation.
HWTSP Clock Source	If the Enable HWTSP property is True, this property specifies the name of the source terminal of the sample clock. Otherwise, this property is ignored. You can use an internal counter timebase when performing counter measurements or an external timebase. **Note** A DAQ analog channel with hardware-timed sample mode or an FPGA device must be configured as the chassis master hardware synchronization device on the Chassis Configuration page in

Property/Section	Description
	System Explorer. Otherwise, the clock source is not available, and VeriStand returns an error during deployment.
HWTSP Clock Rate	If the Enable HWTSP property is True, this property specifies the sampling rate in samples per channel per second. Otherwise, this property is ignored. If you use an external source for the sample clock, set this input to the maximum expected rate of that

Note For device specific information about the default terminals used for counter measurements and generations, refer to Connecting Counter Signals.

VeriStand limits how quickly you can update the frequency and duty cycle values that define the pulses it generates. At least one complete pulse must elapse with a set of frequency and duty cycle values before you can change one of these values. If you update a value too quickly, VeriStand reacts in one of the following ways:

- If the Enable HWTSP property is set to False, VeriStand ignores the new value and continues using the latest value you successfully set.
- If Enable HWTSP is set to True, VeriStand returns an error.

Accessing Cold-Junction Compensation Channels on SCXI Accessories

Manually add a Cold-Junction Compensation (CJC) channel to a system definition.

Before you start, add a DAQ device and use MAX to configure the SCXI chassis, module, and accessory.

Note For information on adding and configuring SCXI chassis and modules, refer to the **Measurement & Explorer Help**.

CJC channels on SCXI accessories, such as the SCXI-1303, do not automatically appear under SCXI modules in System Explorer. You must manually add them.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **DAQ** in the configuration tree.
4. Right-click a DAQ device and select **Add SCXI Modules**.
5. In the Add SCXI Module dialog box, specify the module type to add, set # **Internal channels** to 1, and click **OK**.
6. Under SCXI Chassis, click the SCXI Module you added.
7. Click **Internal Channels** » **Channel 0**.
8. In physical channel name field, enter _cjTemp.

Map a thermocouple scale to the _cjTemp internal channel to convert the acquired voltage values to temperature units.

Related tasks:

- Creating a Thermocouple Scale

Setting Up Timing and Logging Properties for Waveform Acquisitions

Use tasks to define properties for when a system starts and stops acquisitions and how to perform data logging.

Before you begin, add a DAQ device to the system definition.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **DAQ** in the configuration tree.
4. Under a DAQ device, click **Analog Input**.
5. In the Sample Mode drop-down menu, select **Waveform**.
6. In the Analog Input Task drop-down menu, select **Create new...**

7. In the Create DAQ Task dialog box, name the task and set basic timing properties and click **OK**.
8. Click **Waveform Tasks** and select the task you created.
9. Depending on your goal, access one of the following areas to configure how VeriStand performs waveform acquisitions.

Goals	Location
Perform finite acquisitions and identify an external sample clock.	Click the task and use the Task Configuration page.
Allow logging and configure to save data in .tdms log files.	Click **Logging** and use the Logging Configuration page.
Control waveform acquisitions by DAQ devices.	Use waveform task channels.
Define triggers to for acquisitions to start and stop when a certain analog or digital value occurs or a software command is received.	Click **Triggers** and use the Triggers Configuration page.

10. Save the system definition file.

Related concepts:

- Single-Point Versus Waveform Acquisition

Related information:

- TDMS File Format

Adding Waveform Task Channels

Use Analog Input (AI) channels with waveform tasks to control waveform acquisitions by DAQ devices, such as how VeriStand logs waveform data and defines triggers that start and stop acquisitions.

As part of setting up an AI channel(s) to acquire waveforms, you need to specify the rate, size, and other properties of the acquisitions you want to perform. You set these properties in a group called a **task**, and then assign the task back to the AI channel(s). You must assign one task to each DAQ device that performs waveform acquisitions.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **DAQ** in the configuration tree.
4. Depending on your goal, add one of the following task channels.

> **Note** You can write to these channels at run time.

Goals	Task Channel	How to Add
Set a task to an active, running state.	Task Enabled	Always available under each task.
Specify if logging occurs during acquisitions.	Logging Enabled	In the Logging Configuration page, click **Allow TDMS logging** .
Specify when logging stops and begins with a new file.	Start New File	
Set a finite task to automatically restart itself and wait for a new trigger when the acquisition is complete.	Retriggerable	In the Triggers Configuration page, set **Acquisition mode** to **Finite**.
Set a software start trigger to begin an acquisition.	Start Trigger	In the Triggers Configuration page, set **Trigger type** to **Software**.

5. Save the system definition file.

Single-Point Versus Waveform Acquisition

Specify if an Analog Input (AI) channel performs single-point or waveform acquisitions when you add the channel to a system definition.
In **single-point** acquisitions, channels acquire a single point at a time and return the value directly to the system. A single-point acquisition is an immediate, non-buffered operation that occurs at the rate at which the system runs.

In **waveform** acquisitions, channels acquire signals over a period of time as waveforms.
The following table displays the use case for each acquisition type.

Acquisition Type	Use Case
Single-Point	Useful for implementing closed-loop control, as the system reads one value per iteration of the Primary Control Loop and can produce appropriate outputs during the same iteration.
Waveform	Useful for reading data at a rate faster than the rate a system runs. For example, when you need to monitor or log a value that changes quickly, such as the pressure in an engine cylinder, acquiring the signal as a waveform allows you to achieve the high sampling rate required to represent the signal adequately. When you configure AI channels to perform waveform acquisitions, you can achieve rates up to the maximum sampling rate for the DAQ device without being restricted by the system rate.

Related tasks:

- Setting Up Timing and Logging Properties for Waveform Acquisitions

Adding NI FPGA Targets

Use NI FPGA targets to create customizable I/O, help with data preprocessing and postprocessing, add high-speed closed-loop control, and simulate a variety of sensors for hardware-in-the-loop testers

You can add an FPGA target and corresponding FPGA configuration file to a system definition file without installing the LabVIEW FPGA Module. However, you must install the LabVIEW FPGA Module to create a custom FPGA bitfile and configuration file.

1. Launch your project in the VeriStand Editor.

2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **FPGA** in the configuration tree.
4. Choose to add one or all discoverable FPGA targets.

Number of Devices	How to Add
One	a. Click **Add FPGA Target**. b. Select an FPGA configuration file c. Click **OK**.
All	a. Click **Hardware Discovery Wizard** and follow the onscreen instructions. b. Select a target to display its configuration page. c. In the path control, select the FPGA configuration file.

5. Save the system definition file.

You can configure the FPGA device and its individual channels with the FPGA Target and FPGA Channel configuration pages.

Related information:

- Customizing an FPGA Target

Adding NI-XNET Devices

Use the NI-XNET platform to communicate with hardware using the CAN, LIN, and FlexRay protocols.

You can configure an NI-XNET session by adding ports for your devices under the protocol you want to use.

Depending on your goal, complete any of the following tasks.

Goal	Task
Adding any type of NI-XNET port	Discover all NI-XNET devices on connected targets and automatically add ports for them to the system definition file.
Adding a CAN, FlexRay, or LIN port	Add a CAN, FlexRay, or LIN port to a system definition file.

Related information:

- Using NI-XNET CAN, LIN, and FlexRay Interfaces

Adding an NI-XNET Port

Discover all NI-XNET devices on connected targets and automatically add ports for them to the system definition file.

1. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
2. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** in the configuration tree.
3. Click **Hardware Discovery Wizard** and follow the onscreen instructions. The new NI-XNET device(s) appears under the appropriate section in the configuration tree.

   Note You can restrict the type of NI-XNET device to discover. Select **CAN**, **FlexRay**, or **LIN**, in the configuration tree and click **Hardware Discovery Wizard**.
4. Select each new port to display its configuration page.
5. Specify the XNET database and cluster within the database to associate with the port.
6. Save the system definition file.

Expand the port to view sections for adding incoming and outgoing frames, data logging files, and other options. Use the CAN, FlexRay, or LIN Port configuration page to configure additional settings for the newly added port.

Related information:

- Using NI-XNET CAN, LIN, and FlexRay Interfaces

Adding a CAN, FlexRay, or LIN Port

Add a CAN, FlexRay, or LIN port to a system definition file.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** in the configuration tree.
4. Select the type of port you want to add.
5. Click **Add CAN/FlexRay/LIN Port** to display the **Add New NI-XNET CAN/FlexRay/LIN Port** dialog box.
6. Enter a port name, address, XNET database, and cluster within the database to associate with the port, and click **OK**.
7. Save the system definition file.

Expand the port to view sections for adding incoming and outgoing frames, data logging files, and other options. Use the CAN, FlexRay, or LIN Port configuration page to configure additional settings for the newly added port.

Related information:

- Using NI-XNET CAN, LIN, and FlexRay Interfaces

Adding Reflective Memory Networks

Use a reflective memory card to split up a simulation model to execute simultaneously on different target systems.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **Data Sharing** in the configuration tree.

4. Choose to add one or all discoverable reflective memory cards.

Number of Devices	How to Add
One	a. Right-click **Data Sharing**. b. Select **Add Reflective Memory**.
All	a. Click **Hardware Discovery Wizard** and follow the onscreen instructions. **Note** The wizard lists reflective memory cards in the GE category.

5. Configure the reflective memory network using the Reflective Memory Configuration page.
6. Save the system definition file.

Adding and Configuring Timing and Sync Devices

Use a timing and sync device to synchronize more than one chassis.

You must add a timing and sync device to the Timing and Sync directory of VeriStand in order to add it to the system definition.

VeriStand includes one example timing and sync device. You can add a timing and sync device to a system definition file without installing LabVIEW. However, you must install the LabVIEW development environment to create a custom timing and sync device.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **Timing and Sync** in the configuration tree.
4. Right-click **Timing and Sync**.
5. From the drop-down menu, select a timing and sync device.

6. Use the Timing and Sync Configuration page to configure the device.
7. Save the system definition file.

Related information:

- Creating Custom Timing and Sync Devices

Synchronizing Hardware and Software

Synchronize the hardware and software components of a system to ensure consistency and optimal performance, enable data analysis, and time correlation.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Depending on your goal, complete any of the following tasks.

Goal	Tasks
Configure timing of the system	a. Click **Targets** » **Controller**. b. Under Timing Source Settings, in the Primary Control Loop timing source drop-down, select the device that will time the system by sending ticks to the Primary Control Loop of the VeriStand Engine to start loop iterations. c. Enter a **Target Rate** and a **Timing Source Timeout**.
Synchronize hardware-timed single-point devices in a single chassis.	a. Click **Targets** » **Controller** » **Hardware** » **Chassis**. b. In the Chassis master hardware synchronization device drop-down, select the device that will synchronize signal-based hardware devices in the chassis by distributing a Sample Clock to them.

Goal	Tasks
	 **Note** You must configure this for each chassis in your system definition. Additionally, you can only synchronize hardware in a PXI chassis, and all devices must be connected to the PXI backplane because the Sample Clock is routed from the chassis master using PXI_Trig0. c. Configure the device.
Synchronize hardware-timed single-point devices in a multiple chassis.	a. Share the chassis Reference Clocks between chassis with the 10 MHz REF IN and OUT BNC connectors on the backplanes of the PXI chassis. b. Click **Targets** » **Controller** » **Hardware** » **Chassis** c. Configure a chassis to export a start trigger. a. In the Chassis master hardware synchronization device drop-down, select the device that will synchronize signal-based hardware devices in the chassis by distributing a Sample Clock to them. b. In the Export start trigger on line drop-down, select which line the chassis will export a start trigger. d. Configure a chassis to import the start trigger. a. In the Chassis master hardware synchronization device drop-

Goal	Tasks
	down, select the device that will synchronize signal-based hardware devices in the chassis by distributing a Sample Clock to them. b. In the Trigger line drop-down, select which line the chassis will import the start trigger.
Synchronize complex systems	Synchronizing more complex systems may require additional system and software configuration and additional hardware. For detailed help on synchronizing complex systems, see the white paper on Building Synchronized VeriStand Systems.

4. Save the system definition file.

Related concepts:

- Hardware Synchronization Types

Related information:

- Building Synchronized VeriStand Systems

Hardware Synchronization Types

Use Time-Based and Signal-Based Synchronization to coordinate hardware device timing in a system.

In **Time-Based Synchronization**, each piece of time-based hardware shares a common wall-clock time reference. For test and measurement hardware, there are many industry standards for time such as IEEE 1588, GPS, network time protocol (NTP), pulse per second (PPS), and inter-range instrumentation group (IRIG) time codes.

With **Signal-Based Synchronization**, the system uses physical hardware pulses as a reference for events. Each piece of signal-based hardware in the system shares a hardware clock. The clock can be a shared Sample Clock or a high-frequency Reference Clock. Each signal-based device derives a Sample Clock and a shared start trigger from the shared clock. When multiple signal-based devices are

synchronized, they update I/O simultaneously and also drift by the same number of samples over a time period.
The following table displays common use cases for each type of hardware synchronization.

Synchronization Type	Use Case
Time-Based	▪ Correlate data logged by time-based devices, such as XNET devices. ▪ Initiate I/O sampling across time-based devices.
Signal-Based	▪ Simultaneously sample I/O across several different data acquisition devices. ▪ Update the PWM duty cycle of an FPGA device. ▪ Update analog outputs of data acquisition devices.

Setting Chassis Master Hardware Synchronization Devices

Use a chassis master hardware synchronization device to control the synchronization of all hardware in a chassis.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Choose to add and configure one of the following hardware devices to the system definition.

Device Type	Device Configuration
NI-DAQ	The NI-DAQ device must have at least one analog input or output channel.
NI FPGA	Any NI FPGA device.
Timing and Sync	The timing and sync device must have the capability to drive the 0 line.

Device Type	Device Configuration
	Note The RTSI 0 line is a digital line that sends a clock signal that synchronizes all hardware I/O devices in the system.

4. Click **Targets** » **Controller** » **Hardware** » **Chassis** in the configuration tree to open the Chassis Configuration page.
5. Click **Chassis master hardware synchronization device** to select the hardware device.
6. Save the system definition file.

Related information:

- Using Synchronization to Build Integration Test Systems with VeriStand

Adding and Configuring a Custom Device

Add and configure third-party custom devices to execute user-defined actions, determined by LabVIEW VIs.

Note You can add a custom device to a system definition file without installing LabVIEW. However, you must install LabVIEW to create a custom device.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** in the configuration tree.
4. Right-click **Custom Devices** and select a device from the drop-down menu.
5. Configure the custom device.
6. Save the system definition file.

Adding and Configuring a Model

Connect a model to other parts of the system and run the model on a hardware target.

Before you begin, you must compile the model to use it. To learn more about models, refer to Components of a Model and Primary Control Loop Step Execution in Models.

VeriStand executes each model in its own loop. If a system definition contains multiple models referencing the same compiled model, VeriStand makes a temporary copy of the model so each loop has its own compiled model to execute.

> **Note** Adding more than one instance of the same compiled model causes errors if the model accesses a shared resource, such as a dependency. Contact your model provider for information about whether the model accesses such resources.

1. Launch your project in the VeriStand Editor.
2. On the Mapping Diagram on the Palette, click **Software** » **Simulation Model** and drop the model node on the diagram.
3. In the Open dialog box, select a simulation model and click **Open**.
   The Mapping Diagram will load the model.
4. In the Configuration pane, use the **Item** tab to modify the model.
   a. Set the Initial state of the model to **Running** or **Paused**.

      > **Note** To change the values of model parameters before the first time step, set the **Initial state** to **Paused**.

   b. Set the **Decimation** of the model rate.
   c. Click **Import parameters**.
   d. In the Import Parameters dialog box, select the parameters you want to import as channels and click **OK**.

The parameters you selected appears inside the node when you expand it.

> **Note** Importing too many parameters negatively impacts system performance.

e. Click **Import signals**.

f. In the Import Signals dialog box, select the signals you want to import as a channels and click **OK**.
The signals you selected appears inside the node when you expand it.

> **Note** Importing too many signals negatively impacts system performance.

g. **Optional:** If your model contains a vector inport or outport, set the **Vector port specification** as **Segment into scalar channels** or **Maintain as vector channel**.

> **Note** Scalar channels provide greater flexibility than vector channels. Vector channels only map to models that contain a vector channel of the same size. You cannot map a vector channel to controls or indicators on your Workspace or use it with calculated channels, alarms, procedures, and others.

5. Wire the channels to create mappings.
6. Depending on your goal, complete the following tasks to configure the model further.

Goal	Task
Set model timing	Adjust the step size to have the model run at a different rate.
Set model parameters	Use the VeriStand Editor, Workspace, Model Parameter Manager, and Stimulus Profile Editor to set the values of model parameters.

Goal	Task
Scope global parameters	Update the scope of all global parameters in a model to the target-level or model-level.
Set the default values for inports	Change the default value for an inport to prevent your models from using invalid values.
Configure the execution order of multiple models	Define the order that the Primary Control Loop (PCL) executes models.

7. Save the system definition file.

After adding the model to the system definition, use model execution channels to interact with models.

Components of a Model

Models run on hardware targets and contain inports, outports, parameters, and signals to respond to stimuli from other parts of the system by producing outputs in a way that simulates the modeled item.
The following table displays common model components that connect to other parts of the system or allow you to interact with the model.

Component	Description
Inports and Outports	Communicates with other parts of the control system. You can map inports and outports directly to hardware inputs and outputs, other models in the system, system channels, and more. Inports and outports are dynamic values the simulation updates each time the model executes.
Parameters	Acts like variables in the model. You can manipulate parameters to tune the behavior of the simulation. For example, an operator can set a parameter before the model starts executing or update its value between the execution of discrete tests.
Signals	Serves as probes, or test points, of a model as it executes.

For example, consider a system with a model that runs a physical controller on a hardware target to represent a DC motor. Such a model might contain the following components:

- An inport that accepts the motor command from the motor controller
- An outport that returns the motor speed from the model
- Parameters that adjust the load on the motor
- A signal that returns internal data that aids in debugging

Primary Control Loop Step Execution in Models

The Primary Control Loop (PCL) executes steps in models differently if it is in parallel mode or in low latency mode.
Setting the PCL execution mode to parallel or low latency affects the steps that the VeriStand engine takes each iteration. The following table displays the main differences.

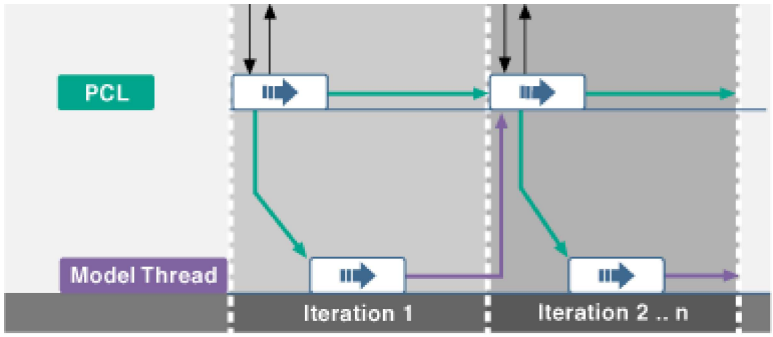
PCL iteration	Parallel mode steps	Low latency mode steps
First	1. Writes values to model. 2. Initiates execution of model.	1. Writes values to model. 2. Waits for model to finish executing. 3. Reads values from model.
Second and after	1. Reads values from previous execution of model. 2. Writes values to model. 3. Initiates execution of model.	

In addition, the PCL timing differs in models depending on if it is running in parallel mode or low latency mode.

Parallel Mode Primary Control Loop Timing in Models

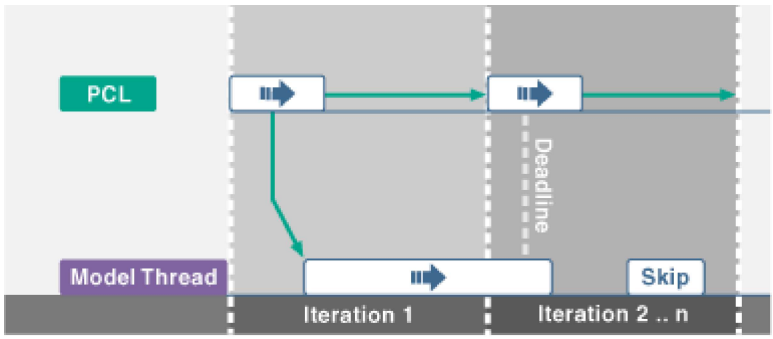
Models that have a Primary Control Loop (PCL) set to parallel mode have a one-cycle delay between when a model executes and when the data it produces is available to the system.

As seen in the following illustration, the PCL will not wait for models to finish executing before it executes other steps.



Model Execution Deadlines

In parallel mode, because the PCL does not wait for the models to finish executing before it executes other steps, models can continue executing even after the PCL starts its next iteration. During the next iteration, the model must finish executing before the PCL can read data from models. The following illustration shows how the VeriStand engine imposes this deadline in parallel mode.



If a model does not finish executing by the deadline, the VeriStand engine does not schedule any models to execute during that iteration. The Model Count system channel also increments.

Note To identify the model that was late, monitor the Time Step Duration execution channel for each model. This is useful when a system contains multiple models.

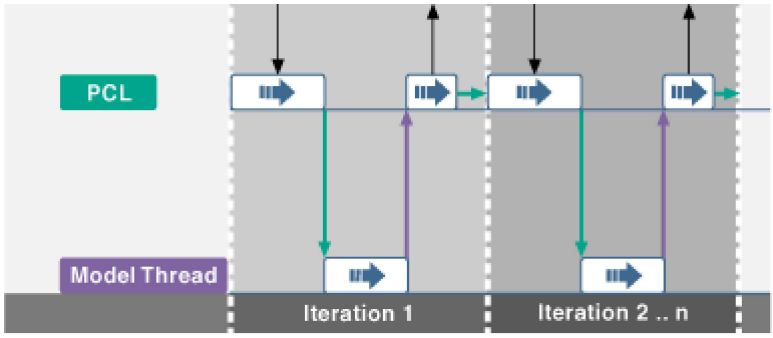
Deadlines When Model Rate Differs from Target Rate

If you configure a model to run at a decimation of the PCL rate, the VeriStand engine enforces a deadline when the decimation specifies it to finish executing. For example, if the target rate for the PCL is 100 Hz and the decimation for a model is 2 (therefore, it runs at 50 Hz), the VeriStand Engine does not impose a deadline after the first 100 Hz PCL iteration because, according to the decimation, the model is not scheduled to finish executing.

However, after the second 100 Hz PCL iteration, when the model stops executing, the VeriStand Engine imposes a deadline. This happens on a per-model basis. A different model in the system with a decimation of 1 has a deadline imposed at every 100 Hz PCL iteration.

Low Latency Primary Control Loop Timing in Models

Models in VeriStand that have a Primary Control Loop (PCL) set to low latency mode wait for the model to finish executing so other loops can access the data generated before executing again.
As seen in the following illustration, the PCL will let models transfer data before executing the next step.



Decimated Models

A model that is decimated returns values on every N iteration of the PCL, when N represents the decimation factor rather than every iteration as shown in the previous illustration. Additionally, passing data between decimated models causes an expected N tick delay where N represents the decimation factor.

Note Since decimated models run in parallel, they ignore the execution order you set in the system definition file. If you want to implement an execution order, add handshaking code to the decimated models.

Model Execution Deadlines

In low latency mode, the VeriStand engine does not enforce deadlines. The PCL waits for models to finish executing before moving to the next iteration. In other words, even though the late models delay the execution of VeriStand engine components, you can access data from models when needed. The Model Count and HP Count system channels increment when a model makes the PCL late.

Note To identify the model that made the PCL late, monitor the Time Step Duration execution channel for each model. This is useful when a system contains multiple models.

Setting Model Timing

Adjust the step size to have the model run at a different rate.

Before you begin, learn about the VeriStand system.

A model is set to run at a rate as defined in the build options when the model is compiled. Depending on configuration settings for the system definition, the rate at which the VeriStand Engine executes models can differ from the compiled rate. The following equation describes how the VeriStand Engine executes models:

actual model rate = Primary Control Loop (PCL) rate / model decimation

Modify the system definition to alter this equation.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Set the PCL rate.
    a. Click **Targets** » **Controller** in the configuration tree.

b. Under Timing Source Settings, set a **Target Rate**.

4. Set the model decimation.
   a. Click **Targets** » **Controller** » **Simulation Models** » **Models** in the configuration tree.
   b. Click a model.
   c. Under Model Settings, enter a **Decimation**.
5. Save the system definition file.

If both the controller and plant are simulated, such as during a model-in-the-loop test, you can run your model at a different rate than specified when it was compiled. For example, you can set the Primary Control Loop of a model compiled to run at 100 Hz to 1 kHz, or 10 times faster than real time (assuming model decimation is 1).

This results in more simulations and more data in a shorter amount of time. However, a model that does not run in real time is potentially unstable. When testing with hardware, such as hardware-in-the-loop testing, run your model at the rate it was compiled to accurately simulate the system.

Setting Model Parameters

Use the **VeriStand Editor**, **Workspace**, **Model Parameter Manager**, and **Stimulus Profile Editor** to set the values of model parameters.

Depending on your goal, complete the following task to set model parameters.

Goal	Task
Change the initial value of model parameters	Configure VeriStand to apply initial values for model parameters from a .txt file when a system definition file deploys.
Manually set individual parameters at run time.	Use model calibration controls in the Workspace to view and modify the values for any model parameters in the system definition.
Lock model parameters at run time	Lock a parameter in the Workspace to prevent its value from updating.
Declare temporary variables in a model parameter file	Declare temporary variables within a .txt file and use those temporary variables as new

Goal	Task
	parameter values or as parts of expressions that define new parameter values.
Call a subscript from a model parameter file	Call additional parameter files from within a .txt file to encapsulate certain elements of a test in separate files.
Alias parameter names in a model parameter file	Use an alias file to define syntactically correct aliases for model parameter names.
Import and manage batches of model parameters in the VeriStand Editor	Import model parameters with the Model Parameter Manager to apply values defined in an external .txt file to a model.
Import and manage batches of model parameters in the Workspace	
Update model parameters during a stimulus profile test	Use the **Update Model Parameters from File** step in a stimulus profile to apply model parameter values defined in a text file to a simulation model that is deployed and running on a target.

Changing the Initial Value of Model Parameters

Configure VeriStand to apply initial values for model parameters from a .txt file when a system definition file deploys.

Before you begin, format the initial values in the .txt file to VeriStand supported syntax.

By default, the initial values of parameters are the values that were compiled into the model. However, VeriStand can automatically apply parameter values from a file when you deploy the system definition file. You can switch initial parameter values between tests without recompiling models.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Simulation Models** in the configuration tree.
4. On the **Model Configuration** page, click **Apply parameter values from a file at initialization**.
5. Configure the paths to the parameter file and any dependent files.

Note You can only select one model parameter file for VeriStand to apply to models on the target that contain matching parameters. Use the main model parameter file to call subscripts saved in separate files.

6. Save the system definition file.

Locking Model Parameters at Run Time

Lock a parameter in the **Workspace** to prevent its value from updating.

Before you begin, you must deploy and connect to a system definition with models before you can lock model parameters.

Locking a parameter is useful if you want an expression to determine the initial value of a parameter, but do not want its value to change based on the expression. For example, in the expression b = a * 2, changing the value of variable a results in a change to the value of the parameter, b. If you lock b, its value does not update if a changes.

Note Locking a parameter does not prevent users from directly changing its value in the VeriStand Editor, Workspace controls, or the Model Parameter Manager. Locked parameters ignore only expression-based changes.

VeriStand always unlocks parameters when you deploy a system definition. You cannot lock parameters when VeriStand initializes parameter values prior to running the model.

1. Open the Workspace.
2. Right-click a parameter.
3. On the Item Properties dialog box, enable **Lock**.
4. Click **OK**.

A glyph ⛔ appears next to the parameter to indicate you locked it.

Supported Syntax in Model Parameter Files

Format text (.txt) files to the correct syntax before you apply them to a simulation model.
The text files must be in the following format.

```
parameter1 delimiter value1
parameter2 delimiter value2
...
```

Note Text files cannot contain any column headers.

The following table describes valid entries for the elements of a text file.

Note For examples of valid model parameter text files, refer to the text files in the <Common Data>\Examples\Stimulus Profile\Engine Demo\Stimulus Profiles\Update Model Parameters directory.

Element Type	Processing Notes	Valid Entries
parameter	Valid parameters begin with a letter and contain only alphanumeric characters or underscores. Enclose a parameter whose expression does not fit these conventions in curly braces ({ }). For example, represent a block parameter that includes the model and block names separated by slashes in the following way: {model1/sine/parameter1}.	▪ An expression of a model parameter, as displayed on the **Model Parameter** configuration page in System Explorer. ▪ An alias to a model parameter path. ▪ The command subscript to call another model parameter file. ▪ The name of a temporary variable. **Note** You can use temporary variables elsewhere in

Element Type	Processing Notes	Valid Entries
		the same parameter file, but they are local to that file.
delimiter	—	▪ A Tab entry, by pressing Tab on your keyboard. ▪ An equals sign (=). ▪ A comma (,).
value	Regardless of localized setting, periods (.) in a value are processed as decimal separators and commas (,) are processed as a thousands separator. In the VeriStand Editor, the localization setting is considered when displaying values. This may lead to situations where values that contain a period in the file will be displayed with commas.	▪ A numeric constant (double). ▪ A matrix in row-major form with the following element types. ▪ Numeric constants (doubles). ▪ Fractions of the form x/y, where x and y are doubles. ▪ A temporary variable that represents a scalar value. **Note** For example, if a and b are declared variables, then [1 -2/3; a b] is a valid 2 x 2 matrix. ▪ The constants Infinity and -Infinity.

Element Type	Processing Notes	Valid Entries
		▪ An expression that follows the VeriStand expression syntax. ▪ A path to another model parameter file. **Note** A path is only valid if the corresponding parameter entry is subscript.

Declaring Temporary Variables in a Model Parameter File

Declare temporary variables within a .txt file and use those temporary variables as new parameter values or as parts of expressions that define new parameter values.

Before you begin, learn about the supported syntax for .txt files.

1. Open a formatted .txt file.
2. Create a new line for the temporary variable.
3. Enter the temporary variable name in the parameter column and the variable value in the value column.

   Note The variable name must start with a letter and contain only alphanumeric characters or underscores.

4. Save the text file.
5. Configure VeriStand to allow temporary variables based on where you call the text file.

Location	How to configure
Stimulus Profile Editor	a. Open a stimulus profile.

Location	How to configure
	b. On the Edit tab, select **Update Model Parameters from File**. c. Set the **Allow Temporary Variables** property to true.
Initializing parameters	a. Use System Explorer to apply initial values for model parameters from a .txt file. b. On the Simulation Models page, click **Allow temporary variables**.
Model Parameter Manager tab	a. Use the VeriStand Editor to import model parameters. b. In the Configure Parameter Import dialog box, click **Allow local variables**.
Model Parameter Manager Workspace tool	a. Use the Workspace to import model parameters. b. In the Select Model Calibration File dialog box, click **Allow temporary variables**.

Note Allowing temporary variables causes VeriStand to assume that any parameter column entries that do not exactly match model parameter names are temporary variables. For example, if Allow Temporary Variables is TRUE and you enter a model parameter name incorrectly, VeriStand creates a temporary variable with the new name and uses that variable instead of the model parameter. You do not receive an error notification about the name mismatch.

The following excerpt from the ParameterUpdate2.txt example file shows how you can declare a temporary variable, tempConversionFactor. The variable is used in an

expression that defines the value of the environment temperature (C) model parameter.

> **Note** This file uses a tab delimiter.

```
tempConversionFactor     0.5
{environment temperature (C)}     50 * tempConversionFactor
```

This file is part of the Update Model Parameters example, available in the <Common Data>\Examples\Stimulus Profile\Engine Demo\Stimulus Profiles\ directory.

Calling a Subscript from a Model Parameter File

Call additional parameter files from within a .txt file to encapsulate certain elements of a test in separate files.

Before you begin, learn about the supported syntax for .txt files.

1. Open a formatted .txt file.
2. Create or open a second text file that updates parameters from the same model.

   > **Note** This is the subscript file. The subscript file must use the same delimiter as the calling file.

3. In the original file, create a new line to call the subscript.
4. Enter subscript in the parameter column and the path to the file to call in the value column.

   > **Note** The path can be absolute or relative to the directory that contains the calling file.

When the call to the top-level text file executes, VeriStand inserts the contents of the subscript file into the calling file at the line that contains the subscript call. You also can use subscript calls recursively to call subscripts from within subscripts.

The second line of the following excerpt from the ParameterUpdate2.txt example file shows how you can call a subscript file, in this case subfile.txt, from a model parameter file.

Note This file uses a tab delimiter.

```
{environment temperature (C)}     50 * tempConversionFactor
subscript     subfile.txt
```

This file is part of the Update Model Parameters example, available in the <Common Data>\Examples\Stimulus Profile\Engine Demo\Stimulus Profiles\ directory.

Aliasing Parameter Names in a Model Parameter File

Use an alias file to define syntactically correct aliases for model parameter names.

In addition to creating cleaner parameter files, aliasing parameter names can extend the reusability of a test by allowing you to simply update the alias file if you want to use the same test on a model with different parameter names.

1. Open a formatted text file (.txt or .m).
2. Replace the parameter names with syntactically correct aliases.

   Note Each alias must start with a letter and contain only alphanumeric characters or underscores.

3. Create a new text (.txt) file and enter the aliases and their corresponding model parameter paths in the same format you used for the model parameter file.

   Note The alias file must have a .txt file extension and use the same delimiter as the parameter file with which it corresponds.

4. Save the text files.
5. Configure VeriStand to use the parameter alias file path based on where the original model parameter text file is called.

Location	How to enter
Stimulus Profile Editor	a. Open a stimulus profile. b. On the Edit tab, select **Update Model Parameters from File**. c. Enter the **Alias File** path.
Initializing parameters	a. Use System Explorer to apply initial values for model parameters from a .txt file. b. On the Simulation Models page, enter the **Parameter alias file path**.
Model Parameter Manager tab	a. Use the VeriStand Editor to import model parameters. b. In the Configure Parameter Import dialog box, enter the **Alias File Path**.
Model Parameter Manager Workspace tool	a. Use the Workspace to import model parameters. b. In the Select Model Calibration File dialog box, enter the **Parameter alias file path**.

6. Set the **Delimiter** property to the delimiter type that both the parameter file and the alias file use.

When the feature that calls the top-level text file executes, VeriStand uses the alias file to map the aliases in the parameter file to actual model parameters.

The following examples show how to format a parameter file with aliases and its corresponding alias file.



Note These files use a tab delimiter.

File type	Example
Parameter	`a 25` `b 900`
Alias	`a {model/environment temperature (C)}` `b {model/idle speed (RPM)}`

Scoping Global Parameters

Update the scope of all global parameters in a model to the target-level or model-level.

Before you begin, identify your parameter's type and scope.

In VeriStand, parameters have two scopes.

- **Target-level** scope—VeriStand applies updates to all global parameter values with the same name in other models that run on the same target.
- **Model-level** scope—VeriStand restricts the scope of updates to just the parameter in the owning model.

All global parameters in a particular model share the same scope.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Simulation Models** » **Models** in the configuration tree.
4. Click a model.
5. Click **Parameters**.
6. On the Parameters Configuration page, use the Scope for Global Parameters drop-down to select **Target** or **Model**.
7. Save the system definition file.

Identifying Local and Global Parameters

Use VeriStand to determine if your parameters are local or global.

Before you begin, add a model and import parameters.

Local parameters apply to a specific subsystem, or block, in the owning model. Local parameters allow you to independently adjust a common parameter for multiple instances of the same block.

Global parameters, by default, apply to the current model and to any global parameters with the same name in other models on the target. You can restrict global parameters in a model from applying to other models by configuring the scope of the global parameters. Global parameters are similar to workspace variables in MathWorks MATLAB software.

A parameter **expression** contains the model name that appears in the System Explorer configuration tree. Every parameter also has an associated path that contains the name the model was compiled under. The expression of a global parameter also indicates whether its scope is at the target-level or the model-level.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Simulation Models** » **Models** in the configuration tree.
4. Click a model.
5. Click **Parameters** and select a parameter.
6. On the Parameter Configuration page, review the parameter information.
7. Based on the model's expression and characteristics, determine the parameter type.

Example expression	Characteristic	Parameter type
Sine Wave/Block1/Amplitude	The root of the Amplitude parameter is a block rather than a model.	Local
Sine Wave/Amplitude	The root of Amplitude is the owning model.	Global, model-level scope

Example expression	Characteristic	Parameter type
Amplitude	The root of Amplitude is not a specific model.	Global, target-level scope

Note The name of the owning model in the system definition is Sine Wave.

Setting Default Values for Inports

Change the default value for an inport to prevent your models from using invalid values.

The default value of all inports is 0. Change the default if your model divides by an inport to avoid an invalid operation.

Note Verify if your system configuration will cause the initial default value to go unused. For example, if the Primary Control Loop execution mode is in Low Latency, the model is assigned to the first execution group, and an inport is mapped to hardware, the inport will always receive its value from hardware.

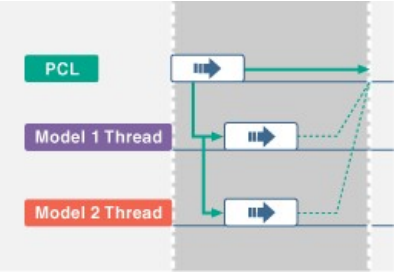
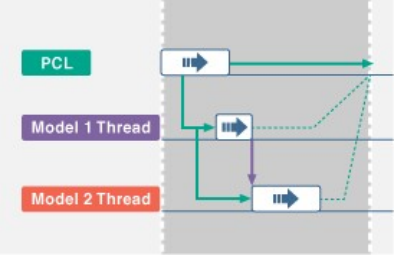
1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Simulation Models** » **Models** in the configuration tree.
4. Click a model.
5. Click **Inports** and click an inport.
6. On the Inport Configuration page, set the **Default Value**.
7. Save the system definition file.

Configuring the Execution Order of Models

Define the order that the Primary Control Loop (PCL) executes models.

1. Use the following table to determine the order that your models execute.

> **Note** The PCL execution mode does not affect the execution order for multiple models.

Model execution	Description	Diagram
In parallel (default)	Models read data from the previous PCL iteration and execute in parallel. If you map one model to another, the second, dependent, model always receives data the first model generated during the previous iteration of the Model Execution Loop. It does not receive data the first model generates during the current iteration.	
In series	Models execute relative to each other. If you map one model to another, defining an execution order allows you to ensure that the second, dependent model receives data the first model generates during the same iteration.	

2. Launch your project in the VeriStand Editor.
3. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
4. Click **Targets** » **Controller** » **Simulation Models** » **Execution Order** in the configuration tree.

5. On the **Execution Order Configuration** page, drag the red cells representing the **Models** from one group to another to arrange them in the order that you want the models to execute.

   Note Models in the same execution groups execute in parallel.

6. Click **Refresh** to reorder the **Models** and **Execution order** lists based on the changes you have made.
7. Save the system definition file.

Running Tests

Execute and automate tests.

Deploying the System Definition File to a Real-Time Target

Deploy the system definition file to the real-time (RT) target to run a project.

Before you begin, install the VeriStand engine to the RT target using MAX.

1. Open your project in the VeriStand Editor.

   Note You can deploy the system definition without launching the VeriStand Editor.

2. In the Project Files pane, double-click a system definition file (.nivssdf) to open the Mapping Diagram.
3. In the Configuration pane, click **Document**.
4. Set the target's operating system to **Linux_x64**.
5. Enter the **IP Address** of the RT target.
6. Save the system definition file.
7. Click **Operate** » **Deploy**.

If a system definition file is already running on the RT target, one of the following happens:

- If the system definition file on the host computer is the same as the system definition deployed to the target, the host connects to the target and launches the VeriStand Editor without deploying the system definition file.
- If the system definition file on the host computer is different from the system definition deployed to the target, clicking **Operate** » **Deploy** stops the system definition on the target and deploys the system definition on the host.

You can also manage individual targets at run time and configure a watchdog timer for VeriStand to execute during deployment.

Downloading Support Files in MAX

Before you can deploy VIs or a system definition file to a real-time target, use NI Measurement & Automation Explorer (MAX) to download support files.

1. Launch MAX.
2. In the configuration tree, click **Remote Systems**.
3. Select the real-time target you want to deploy to and click **Software**.

> **Note** If the real-time target does not have a **Software** category, then it does not support the required software.

4. On the toolbar, click **Add/Remove Software** to launch the LabVIEW Real-Time Software Wizard.

> **Note** You will be prompted to enter the real-time system's administrator name and password if it has one. For more information, refer to the Logging into your System topic in the **Measurement & Automation Explorer Help**.

5. **Optional:** Select **Custom software installation (currently installed)** and click **Next** to install recommended software if prompted.
6. Click the icon next to **VeriStand RT Engine**.
7. Select **Install the feature** and click **Next**.
8. Confirm that you want to install VeriStand support and click **Next**.

MAX displays the progress of the installation and reboots the target so it is ready for a system definition file.

Running the VeriStand Gateway Silently

Deploy a system definition without launching a VeriStand user interface, such as **VeriStand Editor** or **System Explorer**.

By silently running the VeriStand Gateway, you can deploy a system definition, close the user interface, and then reconnect to the still-deployed project upon re-opening the user interface. This allows you to leave the system definition deployed while being able to connect and disconnect testing user interfaces as needed.

If you run the VeriStand Gateway silently without first launching VeriStand, the VeriStand Editor will indicate Legacy Project Functionality Disabled in the window title. While in this state, you will not be able to access the following features:

- Workspace
- Workspace Tools dialog box
- Console Viewer
- XNET Bus Monitor
- Channel Calibration
- System Explorer Options dialog box

While the Gateway is running silently, you can check on the Gateway's status and use programmatic APIs.

1. Open a command prompt.

   Note You can also use the Windows Run command or a language of your choice.

2. Execute one of the following commands using the following syntax: "<Base>\veristand-server.exe" <command>.

Command	Description
help	Displays command options.
start	Starts the server if it is not already running.
stop	Stops the server if it is running.
status	Displays the status of the server.

Command	Description
deploy	Deploys a system definition to the server.
undeploy	Undeploys a system definition from the server if not password-locked.

Individual Target Management

You can manage individual targets at run time by connecting and disconnecting individual targets and undeploying a system definition.
The following table displays the ways you can manage targets at run time.

Management Action	Description
Connect Individual Targets	Connect to a new target for the first time or reconnect to an old target after maintenance.
Disconnect Individual Targets	Disconnect a target for maintenance if it returns an error or develops a physical fault.
Undeploy a System Definition	Undeploy the system definition from an old target.

Connecting Individual Targets

Connect to a new target for the first time or reconnect to an old target after maintenance.

Connecting to a target deploys the system definition to the target. You must define the target in the system definition before connecting to a new target. Otherwise, you must undeploy the system definition from all targets, define the new target, and redeploy to all targets.

1. In the VeriStand Editor, click **Tool Launcher** » **Manage Targets**.
2. In the Tool window's Manage Targets tab, select the target or targets that you want to connect.
3. Click **Deploy**.

> **Note** If you are reconnecting a target that already has the system definition, select the target and click **Connect**.

In the Manage Targets tab, verify that the **State** column displays the correct running target.

Disconnecting Individual Targets

Disconnect a target for maintenance if it returns an error or develops a physical fault.

1. In the VeriStand Editor, click **Tool Launcher** » **Manage Targets**.
2. In the Tool window's Manage Targets tab, select the target or targets that you want to disconnect.
3. Click **Disconnect**.

In the Manage Targets tab, verify that the **State** column displays that the target is disconnected.

Undeploying the System Definition from an Individual Target

Undeploy the system definition from an old target.

Undeploying the system definition from a target also disconnects the target.

1. In the VeriStand Editor, click **Tool Launcher** » **Manage Targets**.
2. In the Tool window's Manage Targets tab, select the target or targets from which you want to undeploy the system definition.
3. Click **Undeploy**.

In the Manage Targets tab, verify that the State column displays that the target is disconnected. Once the system definition is undeployed, you must redeploy it before you can reconnect the target.

Configuring the Watchdog Timer when Deploying to a Real-Time Target

You can modify the timing watchdog that stops the execution of the VeriStand Engine if too much time elapses when you deploy a project to a real-time target.

The timing watchdog is controlled by the Watchdog Timer Loop. This loop executes at a rate of 10 Hz and at a lower priority than the other loops in the VeriStand Engine.
The Watchdog Timer Loop executes and resets the Watchdog Timer system channel every 500 milliseconds. When the Primary Control Loop executes, it evaluates whether the Watchdog Timer system channel reports a time greater than 1 second. If it does, VeriStand throws an error and stops execution of the VeriStand Engine.

To enable or disable watchdog functionality, complete the following steps:

1. Open **System Explorer**.
2. Select **Controller** from the configuration tree to display the **Controller Configuration** page.
3. Select **Other Settings** » **Filter Watchdog Errors**.

When you enable this option, VeriStand will filter errors reported by the timing watchdog. You can then monitor and respond to the Watchdog Timer system channel with custom alarms and procedures.

Running VeriStand Operations Using the Command Line

Use the command line to execute processes in VeriStand, such as deploying or closing a project.

Commands can be used while VeriStand is open or closed.

1. Open a command prompt.
2. Change directories to the location of the VeriStand executable.
3. Execute one of the following commands using the following syntax: VeriStand.exe /<command>.

> **Note** You can use a slash (/), a dash (-), or two dashes (--) before each command. Commands are not case sensitive. For example, /deploy and --DePloY are equivalent commands.

Command	Description
connect	Connects to a deployed system on the gateway. This command executes after commands for creating the project, opening the project, and modifying the system definition. This command is ignored if deploy is also specified. An error returns if a system is not deployed but a project is open.
disconnect	Disconnects from a deployed system on the gateway.
deploy	Deploys the system to the gateway and connects to it. This command executes after commands for creating the project, opening the project, and modifying the system definition.
editScreen	Turns operate mode off to unlock the screen for editing. This can be used to reverse the operateScreen or operateOnly commands while VeriStand is running.
gateway <IP address or hostname>	Specifies the given IP address or hostname as the gateway.
help	Opens the VeriStand manual to **Running VeriStand Operations Using the Command Line**.
noDeployKeys	Disables the keyboard shortcuts for deploy (F6) and undeploy (F7). This command does not disable the Operate menu or the deploy and undeploy commands.
nivsprj <file path to .nivsprj file>	Opens the specified project. If the project does not exist, VeriStand creates it using the default project template. If VeriStand has a different project open, a dialog box will ask you to save and close it.
openProject <file path to .nivsprj file>	
operateScreen	Turns operate mode on to lock the screen. When operate mode is on, you cannot resize, move, or edit the controls in the screen document. This command disables the screen document unlock button.
operateOnly	

Command	Description
openDocument <file path to legacy .nivsproj file>	Converts a legacy .nivsproj VeriStand project into a .nivsprj file and opens it.
sysDef <file path to .nivssdf system definition file>	Loads the system definition into an open or specified project. If you create a new project with the nivsprj or openProject commands, the specified system definition is used instead of the default template. This command does not work if the gateway is currently connected or connecting. This command executes before the connect command.
undeploy	Removes deployed system from the gateway. This command works on connected and disconnected systems.

VeriStand .NET Reference

Use .NET APIs to programmatically control software operations.

VeriStand includes the following .NET APIs.

Note These APIs are documented in the VeriStand .NET API Help.

API	Assembly	Description
Execution API	NationalInstruments.VeriStand.ClientAPI (in NationalInstruments.VeriStand.ClientAPI.dll)	Automates the operation of an VeriStand application on the target. For example, you can read and write channel data, control running models, configure alarm states and read data from alarms, and access Workspace tools.
System Definition API	NationalInstruments.VeriStand.SystemDefinitionAPI (in NationalInstruments.VeriStand.SystemDefinitionAPI.dll)	Automates the operation and configuration of a system definition file. This API performs the same operations as configuring the file in the System Explorer window.

API	Assembly	Description
Stimulus Profile Definition API	NationalInstruments.VeriStand.StimulusProfileDefinitionApi (in NationalInstruments.VeriStand.RealTimeSequenceDefinitionApi.dll)	Automates the operation and configuration of stimulus profiles. This API performs the same operations as configuring stimulus profiles in the Stimulus Profile Editor.
Real-Time Sequence Definition API	NationalInstruments.VeriStand.RealTimeSequenceDefinitionApi (in NationalInstruments.VeriStand.RealTimeSequenceDefinitionApi.dll)	Automates the operation and configuration of real-time sequences. This API performs the same operations as configuring real-time sequences in the Stimulus Profile Editor.
Data Types API	NationalInstruments.VeriStand.Data (in NationalInstruments.VeriStand.DataTypes.dll)	Represents data types and resources used by VeriStand.

You can access these assemblies from any .NET-compatible programming language or environment, including LabVIEW and NI TestStand.

Data Logging Options

Log data with tools such as the Embedded Data Logger, Stimulus Profile Editor, and DAQ devices.

Use the following table to determine the best data logging option for your needs.

Note If you need to log data from varying sources, use DIAdem to combine and time correlate all of your data logs. For more information, refer to Viewing Time Correlated NI VeriStand Data Logs.

Option	Location	Rate	Strengths	Weaknesses	Use cases
Embedded Data Logger	Target	Medium	▪ Adds structured metadata to your log files and allows	▪ Limits logging	▪ Log more data

Option	Location	Rate	Strengths	Weaknesses	Use cases
			you to organize logged channels in groups. ▪ Configures dynamic start and stop trigger conditions. By using the Embedded Data Logger via a channel in the system definition, you can trigger logging through mappings to outputs from models, real-time sequences, procedures, etc. ▪ Retains logging data if you lose connection to the target.	rate to the rate of the PCL. ▪ Requires configuration of the Embedded Data Logger before deploying. You cannot change configuration at run time.	than the bandwidth of the connection between a host and target allows. ▪ Disconnect the host from the target after deploying a system definition and continue to

Option	Location	Rate	Strengths	Weaknesses	Use cases
					log data.
NI-XNET Raw Frame Data Logging	Target	Logs as data arrives	▪ Logs potentially at rates faster than the PCL. ▪ Allows you to specify whether to log all frames or specific frames by ID. ▪ Retains logging data if you lose connection to the target. ▪ Configures dynamic start and stop trigger conditions. As you can control when to start and stop NI-XNET raw frame data logging via a channel in the system definition, you can trigger logging through mappings to outputs from models, real-time sequences, procedures, etc.	▪ Logs data in a low-level format, so you must perform post-processing of the data to convert it to readable units. ▪ Cannot change configuration at	Log frame data during an NI-XNET session.

Option	Location	Rate	Strengths	Weaknesses	Use cases
				run time.	
Logging waveform acquisition from a DAQ device	Target	High, potentially up to the rate the DAQ board can run.	▪ Produces smaller log files. ▪ Log potentially at rates faster than the PCL. ▪ You will not lose logging data if you lose connection to the target.	▪ Logs via waveforms, and you cannot scale or calibrate waveforms as you might channels. ▪ Limited start and stop trigger functionality. ▪ Cannot chan	Log waveform acquisitions from a DAQ device.

Option	Location	Rate	Strengths	Weaknesses	Use cases
				ge configuration at run time.	
Logging with the Stimulus Profile Editor	Host	Medium	▪ Allows advanced triggering functionality. ▪ Capture responses of your unit under test (UUT) to a real-time test.	▪ Limited to logging results from tests you execute in the Stimulus Profile Editor. ▪ The host must be connected to the targe	Save the responses of a UUT to specific scenarios.

Option	Location	Rate	Strengths	Weaknesses	Use cases
				t to log. ▪ Logging bandwidth limited to the bandwidth of the connection between the host and the target.	
VeriStand Editor logging specification file	Host	Medium	▪ Reconfigure your logging settings and add new specification files at run time. ▪ Provides dynamic configuration options. ▪ Automates post-processing of your log data.	▪ Logging bandwidth limited to the bandwidth of	▪ Configure and execute host-side data logging from

Option	Location	Rate	Strengths	Weaknesses	Use cases
				the connection between the host and the target. ▪ The host must be connected to the target to log.	the VeriStand Editor. ▪ Automate post-processing actions.
Data Logging workspace control	Host	Medium	▪ Reconfigure logging settings at run time. ▪ Provides dynamic configuration options. ▪ Automates post-processing of your log data.	▪ Logging bandwidth limited to the bandwidth of the conn	▪ Perform dynamic logging on a host. ▪ Record behaviors duri

Option	Location	Rate	Strengths	Weaknesses	Use cases
				ection between the host and the target. ▪ The host must be connected to the target to log.	ng a test without undeploying a system definition.

Using NI-XNET Interfaces

Use NI-XNET interfaces to communicate and interact with applications that require real-time, high-speed manipulation of hundreds of Controller Area Network (CAN), Local Interconnect Network (LIN), and FlexRay frames and signals.

Popular application types to use the NI-XNET platform include hardware-in-the-loop (HIL) simulation, rapid control prototyping, bus monitoring, and automation control.

The NI-XNET platform includes a series of high-performance CAN, LIN, and FlexRay communication protocol interfaces used by automotive and industrial networks.

Depending on your goal, complete any of the following tasks.

Goal	Task
Add an NI-XNET database	Create a standardized file for embedded system communication in a FIBEX (.xml), CANdb (.dbc), NI-CAN (.ncd), or LDF (.ldf) format.
Edit an NI-XNET database	Use the NI-XNET Database Editor to configure a basic network, define frames and exchanged signals, and assign frames to Electronic Control Units (ECUs).
Import NI-XNET frames	Import incoming or outgoing frames from an NI-XNET database.
Use NI-XNET frame IDs	Use frame IDs to prioritize event-triggered frames, filter log file frames, and filter CAN data replay file frames.
Access timing and ID information for incoming NI-XNET frames	Create Frame Information channels to track timestamps and frame IDs.
Log incoming NI-XNET frames	Create TDMS (.tdms) or NI-XNET log (.ncl) files to record incoming frame data during an NI-XNET session.
Replay logged NI-XNET CAN frame data	Add and replay TDMS (.tdms) or NI-XNET log (.ncl) files on a CAN bus.
Configure cyclic NI-XNET CAN frame faulting	Configure outgoing cyclic frames of NI-XNET CAN interfaces by adding Skip Cyclic Frames and Transmit Time channels.
Configure cyclic redundancy checks (CRCs) and counters for outgoing NI-XNET CAN frames	Specify the bytes for outgoing frames of NI-XNET CAN interfaces to include in CRCs and add counters that increment each time the frame transmits across the bus.

Related concepts:

- NI-XNET Bus Monitor
- How VeriStand Applies Scaling Factors to NI-XNET Signals

Related information:

- NI-XNET CAN, LIN, and FlexRay Platform Overview

NI-XNET Overview

Use sessions, clusters, and frames to set up your NI-XNET interfaces.



Note Refer to the **NI-XNET Hardware and Software Help** that installs with your hardware for more detailed documentation about NI-XNET interfaces.

Sessions

An **NI-XNET session** represents a connection between your NI CAN, FlexRay, or LIN hardware and hardware products on the external network. Sessions include the following configuration components:

- **Port**—A port in NI-XNET refers to the physical connector on an NI hardware device.
- **Interface**—An interface represents the software CAN, FlexRay, or LIN connector on an NI hardware device. Use the interface name as an alias for your ports so you can avoiding changing your application if the physical hardware configuration changes.
- **XNET Database**—An XNET database is a standardized file that describes embedded communication. XNET database file formats include CANdb (.dbc) for CAN, FIBEX (.xml) for FlexRay, and LIN Description File (.ldf) for LIN. For the NI-XNET interface to communicate with hardware products on the external network, NI-XNET must understand the communication in the actual embedded system.
  You can edit NI-XNET databases directly from VeriStand by launching the NI-XNET Database Editor from System Explorer.
- **Session Mode**—A session mode specifies the data type (signals or frames), direction (input or output), and how your application and network transfer data. VeriStand supports the following NI-XNET session modes:
  - Signal Input Single-Point
  - Signal Output Single-Point
  - Frame Input Single-Point

- Frame Output Single-Point
- Frame Input Stream



Note Refer to the **NI-XNET Hardware and Software Help** that installs with your hardware for more information about session modes.

Clusters

A **cluster** is a description of a single network, such as a CAN bus, within an XNET database. For importing frames, each port in VeriStand is associated with a single cluster within an XNET database. A cluster can contain an arbitrary number of frames.

Frames

A **frame** is a message that transmits across an embedded network. In VeriStand, frames are either inputs (incoming frames) or outputs (outgoing frames), and are classified according to their transmission characteristics. For example, event-triggered frames transmit only when a specific event occurs. Frames also contain information such as ID numbers and timing data that you can access through Frame Information channels in VeriStand.

You can import frames into VeriStand in either signal or raw data format. Signal format frames contain **signals** and raw data format frames contain **channels**. These terms refer to the basic data exchange unit on the network.

The specific format and characteristics of frames vary based on the communication protocol you use. Refer to the **NI-XNET Hardware and Software Help** for detailed information about frame formats.

Adding NI-XNET Databases

Create a standardized file for embedded system communication in a FIBEX (.xml), CANdb (.dbc), NI-CAN (.ncd), or LDF (.ldf) format.

Note Refer to the NI-XNET Hardware and Software Help for detailed information about NI-XNET databases.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **XNET Databases** in the configuration tree.
4. Click **Add Database File** and choose where you want to add the database from.

Database location	How to add
A registered NI-XNET database	a. Click **Registered XNET Database**. b. Select an **Existing Alias** from the drop-down menu.
A database on disk	a. Click **New XNET Database**. b. Enter an **Alias**. c. Click the **Path** folder. d. In the Select a File or Folder window, select a database file. e. Click **OK**.

5. Click **OK**.

Note If OK is disabled, confirm that you selected a valid database file and that the database does not already appear under XNET Databases.

6. Save the system definition file.

Editing NI-XNET Databases

Use the NI-XNET Database Editor to configure a basic network, define frames and exchanged signals, and assign frames to Electronic Control Units (ECUs).

Before you begin, add an NI-XNET database.

The NI-XNET Database Editor creates and maintains embedded network databases.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **XNET Databases** in the configuration tree.
4. Select the database you want to edit and click **Launch XNET Database Editor**  .

   > **Note** The editor cannot open LIN description files (.ldf) or FIBEX databases with LIN content.

5. Use the editor to make changes to the database.
6. Select **File** » **Save** to save the database file and close the editor.
7. Select the database again and click **Refresh Databases**  .

   > **Note** VeriStand will ask you to delete any signals removed from the database or multiplexed signals that changed in the database. Import the changed multiplexed signals again for VeriStand to read them correctly.

8. Save the system definition file.

Importing NI-XNET Frames

Import incoming or outgoing frames from an NI-XNET database.

Before you begin, add an NI-XNET database and port.

Frames are the messages transmitted across an embedded network. These messages are sorted into clusters within an NI-XNET database. You can import frames from the database to VeriStand.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Hardware** » **Chassis** » **NI-XNET** in the configuration tree.
4. Expand an NI-XNET LIN, FlexRay, or CAN port.
5. Expand **Incoming** or **Outgoing**.
6. Right click a frame type you want to import and select **Import Frames**.
7. In the Import NI-XNET Frames dialog box, click the XNET Frames tab, and select the frames you want to import.
   You can also use the options on the Import Settings and General Options tabs to temporarily modify the default settings for importing frames. For example, you can import the frames in signal or raw data format, create information channels for the frames, or allow disable and trigger channels for frame transmission.

   Note Configure default settings for importing frames on the XNET page of the Options dialog box.
8. Click **OK**.
9. Save the system definition file.

Using NI-XNET Frame IDs

Use frame IDs to prioritize event-triggered frames, filter log file frames, and filter CAN data replay file frames.

The frames that transmit across the network are assigned unique identifiers. You can use these frame IDs for more than just identifying the frame.

Note Refer to the **NI-XNET Hardware and Software Help** for more information about frame identifiers.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Depending on your goal, complete any of the following tasks.

Goal	Tasks
Prioritizing the order that event-triggered outgoing frames transmit.	a. Add a CAN, FlexRay, or LIN port. b. Click the port you added. c. On the port configuration page, click the **Transmission Order** tab. d. Under Pending Transmit Order, click **By Identifier**.
Filtering specific frame IDs to include or exclude from the log file.	a. Set up logging for incoming NI-XNET frames. b. Click the XNET logging file you added. c. Use the XNET Data Logging Configuration page to configure filtering in the Data Logging Settings section.
Filtering CAN data replay file frames to specify frame IDs to include or exclude from replay.	a. Add a CAN data replay file. b. Click the data replay file you added. c. In the File Replay IDs section, use the File Replay Configuration page to configure filtering.

4. Save the system definition file.

Accessing Timing and ID Information for Incoming NI-XNET Frames

Create Frame Information channels to track timestamps and frame IDs.

Before you begin, import an incoming XNET frame.

Frame information channels store incoming NI-XNET frame information. You can create the following types of Frame Information channels.

- **Receive Time**—Contains the timestamp of the most recent frame.
- **Time Difference**—Contains the difference between the two most recent Receive Time timestamps.
- **Frame ID**—Contains the ID number that identifies the frame. This channel is for raw data format frames only.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** in the configuration tree.
4. Select an NI-XNET LIN, FlexRay, or CAN port.
5. Click **Incoming** and select a frame.
6. Click **Add Information Channels**  .
7. Expand **Frame Information** to view the channels.
8. **Optional:** Specify a dedicated start value for a channel.

Channel type	How to specify start value
Receive Time	a. Select a channel. b. On the Receive Time configuration page, enter an initial value.
Time Difference	a. Select a channel. b. On the Time Difference configuration page, enter an initial value.

> **Note** Raw data format frames also include Frame ID channels. VeriStand reads the value for this channel from the XNET database. You cannot modify it or specify an initial value through System Explorer. However, you can use frame IDs to prioritize event-triggered frames and to include or exclude frames from a data log file.

9. Save the system definition file.

After creating Frame Information channels, you can use them like other channels in VeriStand. For example, you can map them to other channels or to controls and indicators in the VeriStand Editor or Workspace.

Use the XNET page of the Options dialog box to configure VeriStand to create Frame Information channels when you import NI-XNET frames. You can also use the Import NI-XNET Frames dialog box to automatically create channels on a one-time basis when you import frames.

Logging Incoming NI-XNET Frames

Create TDMS (.tdms) or NI-XNET log (.ncl) files to record incoming frame data during an NI-XNET session.

When you deploy and run the project, VeriStand logs the frame data you specify to the log file when the trigger condition you specify is met. You can add multiple log files to a single system definition file. A single log file can store data from multiple clusters within an XNET database. If you do not configure filters on the log file, it simply stores everything that is received on the port. You can also replay the log file with a CAN interface across a CAN bus.

> **Note** Refer to the **NI-XNET Logfile Specification**, available in the NI-XNET\Documentation directory, for more information about the .ncl file format.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.

3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** in the configuration tree.
4. Select an NI-XNET LIN, FlexRay, or CAN port.
5. Click **Incoming** » **Raw Frame Data Logging**.
6. Click **New Raw Frame Data Logging**  .
7. In the Add NI-XNET Logging File dialog box, configure the data logging file.
    a. Enter a XNET data logging file name.
    b. Select a file type from the drop-down.
    c. For TDMS files, enter a group name and channel name for storing the logged data.
    d. Select a destination for the log file.

> **Note** You can choose a destination on either the host computer or real-time target. For real-time targets, the destination automatically populates.

8. Click **OK**.
   The new file appears under Raw Frame Data Logging in the configuration tree.
9. On the XNET Data Logging Configuration page, enter a **Buffer time** [s] that is large enough to avoid a buffer overflow while reading data.

> **Note** The larger the buffer, the more memory is required for data logging. To avoid additional dynamic memory allocation, determine the bus load for your session before starting a data logging operation.

10. **Optional:** Select a filter from the drop-down to configure filtering of the frames to log.
    Filtering uses frame IDs to include or exclude specific frames from the log file.
11. Click **Trigger and File**.
12. Configure when data logging starts, stops, and the operation that performs when logging restarts after a stop.

For example, on a trigger channel, you can specify to start logging when value becomes zero and to stop logging when it is not zero.

13. Save the system definition file.

Related concepts:

- Raw Frame Data Logging FAQs

Related tasks:

- Monitoring Incoming NI-XNET Frame Logging

Monitoring Incoming NI-XNET Frame Logging

Check the status of your logging of incoming NI-XNET frames by using the Error, Finished Files, and Status channels.

Before you begin, you must add a raw frame data logging file.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** in the configuration tree.
4. Select an NI-XNET LIN, FlexRay, or CAN port.
5. Click **Incoming** » **Raw Frame Data Logging** and select a log file.
6. Depending on your goal, click a channel.

Goal	Channel
Review the most recent logging error.	Error
Determine what files are ready for use by other processes.	Finished Files
Find the current status of the data logging process.	Status

You can monitor these channels from the VeriStand Editor and Workspace, or map them to other channels within the project.

Raw Frame Data Logging FAQs

Answers to common questions on using raw frame data logging in VeriStand.

When should I use raw frame data logging for NI-XNET CAN interfaces?

Raw frame logging for CAN interfaces is useful when you need the exact timestamps of when messages are sent and received from the hardware. You can use a program such as DIAdem for post-processing of the data. Additionally, you can use the log files you create to replay the frames.

How does raw frame logging affect processor load?

The processor load added by raw frame logging depends on the bus and the controller. However, raw frame logging is a background process. It should not interrupt higher priority items, such as the Primary Control Loop (PCL) or models.

Are timestamps added to the log files?

Timestamps are added to log files, but the type of timestamps added depends on the file format of your log file.

Log file format	Type of timestamp
NI-XNET (.ncl)	Contains timestamps from the NI-XNET hardware.
TDMS (.tdms)	Contains timestamps from the NI-XNET hardware and from the VeriStand System Time channel.

How can I view the raw frame data in real time?

You can use the NI-XNET Bus Monitor to display CAN, FlexRay, or LIN network data.

Replaying Logged NI-XNET CAN Frame Data

Add and replay TDMS (.tdms) or NI-XNET log (.ncl) files on a CAN bus.

Before you begin, add a CAN port.

You can replay the frames within the file in the same order and timing as they were initially received. You can also filter specific frames from the file. A data replay file can be any valid TDMS or NI-XNET log file that contains CAN data.

Note VeriStand can only replay TDMS files that contain specific header information. To ensure your files contain the correct information, use VeriStand to create TDMS files you want to replay.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** » **CAN** in the configuration tree.
4. Click a port.
5. Click **Outgoing** » **Data Replay**.
6. Click **Add a Replay File**  to display the Select an XNET Data File dialog box.
7. Click the **Path** folder to select a log file that contains CAN data and click **OK**.
8. Under Data Replay, select the log file to display the File Replay Configuration page.
9. Under File Replay Settings, specify a trigger channel for triggering the replay of the file.

   Note Replay starts as soon as the value of the trigger channel is not zero. You can select a channel that triggers multiple replays of the file.

10. (TDMS only) Specify the group name and channel name for the TDMS channel that contains the data to replay.
11. Specify the cache # of frames and loop rate (Hz).
12. Under file replay IDs, specify the replay behavior.

The default behavior is to replay the entire file. You can configure this behavior to include or exclude specific frame IDs.

> **Note** If you select **Include Frame IDs**, you cannot specify IDs of single-point frames that already appear as outgoing frames under the port. If you select one of the other behavior options, you cannot have any outgoing, single-point frames specified under the port. Specifying a single-point frame as both an output and a frame to replay causes a run-time error.

13. Save the system definition file.

> **Note** After you deploy and run the project, VeriStand also deploys the data replay file. If a data replay file with the same name already exists on the target when you deploy the project, VeriStand overwrites the existing file.

Monitoring Replay Status

Check the status of your replaying logged NI-XNET CAN frame data using the **Error**, **Pending Frames**, and **Status** channels.

Before you begin, you must add a data replay file.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** » **CAN** in the configuration tree.
4. Click a port.
5. Click **Outgoing** » **Data Replay** and click a log file.
6. Depending on your goal, click a channel.

Goal	Channel
Review the most recent replay error.	Error
Determine whether data is replaying as expected	Pending Frames
Find the current status of the replay process.	Status

You can monitor these channels from the Workspace or map them to other channels within the project.

Configuring NI-XNET CAN Cyclic Frame Faulting

Configure outgoing cyclic frames of NI-XNET CAN interfaces by adding Skip Cyclic Frames and Transmit Time channels.

Before you begin, import an outgoing CAN cyclic frame.

In embedded networks, cyclic frames are frames that transmit at regular intervals. You can add the following frame fault channels:

- **Skip Cyclic Frames**—Skips transmission of a specified number of cyclic frames across the bus.
- **Transmit Time**—Specifies the amount of time that must elapse between subsequent transmissions of a cyclic frame.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** » **CAN** in the configuration tree.
4. Click a port.
5. Click **Outgoing** » **Cyclic**.
6. Select a frame and click **Add Faulting Channels**  .
   VeriStand creates a Frame Faulting section under the frame in the configuration tree.
7. Click **Frame Faulting**.

8. Depending on your goal, complete the following tasks.

Goal	Tasks
Skip the transmission of a certain number of frames.	a. Click **Skip Cyclic Frames** to display the Skip Cyclic Frames Configuration page. b. Specify the **Skip N cycles** for the number of cycles to skip . c. Specify the **Trigger channel** to use to start skipping channels.  **Note** Skipping begins when this channel does not equal zero.
Specify an amount of time that must elapse between frame transmissions.	a. Click **Transmit Time** to display the Transmit Time Configuration page. b. Specify the time that must elapse in the **Transmit time [sec]** field, or select **Use trigger channel to set transmit time?** and specify a trigger channel from which to get the time.  **Note** VeriStand uses the value of the channel as the transmit time, in seconds.

9. Save the system definition file.

After creating Frame Faulting channels, you can use them like other channels in VeriStand. For example, you can map them to other channels or to controls and indicators in the VeriStand Editor or Workspace.

Use the XNET page of the Options dialog box to configure VeriStand to always create faulting channels when you import NI-XNET frames. You can also use the Import NI-XNET Frames dialog box to automatically create channels on a one-time basis when you import frames.

Configuring Cyclic Redundancy Checks (CRCs) and Counters for Outgoing NI-XNET CAN Frames

Specify the bytes for outgoing frames of NI-XNET CAN interfaces to include in CRCs and add counters that increment each time the frame transmits across the bus.

Before you begin, import an outgoing CAN frame.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **NI-XNET** » **CAN** in the configuration tree.
4. Click a port.
5. On the CAN Port Configuration page, click the **Automatic Frame Processing** tab.
6. Click the **Installed automatic frame processing binary files** drop-down to select a file to use for frame processing.

   Note VeriStand supports CRC-8 and CRC-16 length polynomials.

7. Configure the CRC.
   a. In the Polynomial field, enter a generator polynomial for the CRC.
   This field requires a decimal number. VeriStand converts this number to binary form to create the polynomial.
   b. In the Initial CRC field, enter an initial value to use for the CRC calculation.
   This field requires a decimal number. VeriStand converts this number to binary form.
   c. In the Final XOR field, enter a value to XOR with the calculated CRC.
   d. **Optional:** If you want to reverse the order of the bits in the CRC before writing data into the data stream, select **Reflected**.

8. In the configuration tree, under the CAN port, right-click an outgoing frame and select **Add Automatic Frame Processing Data**.
9. Expand **Automatic Frame Processing** to view the configuration pages for the CRC and counter.
10. Click **CRC** to display the CRC Configuration page, and configure CRC settings.
    a. Under CRC Settings, specify the first included byte and last included byte for the CRC.
    b. In the Storage offset [byte] field, specify a number of bytes to offset the byte where CRC data is stored.

    > **Note** The CRC is always the last byte in the data. Set this value to n-1, where n is the number of bytes in the data, to avoid dropping frame data from the CRC.

    c. **Optional:** If you want to specify a trigger channel to trigger writing data, click **Use alternate channel?**.
11. Select **Counter** to display the XNET Counter Configuration page, and configure the counter.
    a. Under Counter Settings, enter the **Width [bit]** of the counter. The maximum width is 8.
    b. Specify the **Initial value** to start the counter.
    c. **Optional:** If you want to specify offsets for counter data, enter a **Storage offset [byte]** and a **Bit offset [bit]**.
    d. **Optional:** If you want to specify a trigger channel to use to trigger writing data, click **Use alternate channel?**.
12. Save the system definition file.

NI-XNET Bus Monitor

Use the NI-XNET Bus Monitor to display and log CAN, FlexRay, or LIN network data as either last recent data or historical data view.
Use the Workspace to launch the NI-XNET Bus Monitor.

To identify more detailed frame information, assign a network database to the NI-XNET Bus Monitor. You can display the message name and comment information in the Monitor view or ID Log view if you find a received frame in the database.
In addition to the network data, the NI-XNET Bus Monitor can provide statistical information. For offline data analysis, you can stream all received network data to disk in two log file formats.
You can launch the NI-XNET Bus Monitor from Measurement and Automation Explorer (MAX) or from the Windows **Start** menu.
For more information about the NI-XNET Bus Monitor, refer to the **NI-XNET Tools and Utilities Help**.

How VeriStand Applies Scaling Factors to NI-XNET Signals

In NI-XNET software, you can assign a scaling factor to a signal.
When VeriStand reads values from a signal that has a scaling factor, VeriStand scales the raw, prescaled values according to the scaling factor. For example, if you assign a scaling factor of 0.1 to a signal and Veristand reads a value of 3 from the signal, it scales the value to 0.3.
Conversely, VeriStand converts values to prescaled values before writing to signals. For example, if you write a value of 0.3 to a signal that has a scaling factor of 0.1, VeriStand writes a value of 3 to the bus.

ASAM XIL API - Generic Simulator Interface

VeriStand includes a Generic Simulator Interface for the Framework, MAPort and EESPort based on the 2.1 version of the ASAM XIL Standard
The Association for Standardization of Automation and Measuring Systems (ASAM) is an organization that facilitates interoperability between measurement and automation testing tools from different vendors.
Use the **MAPort** to read and write data and to capture and generate signals in the simulation model. The **EES port** provides a general API for electrical error simulation. The **Framework** is an abstraction layer above the testbench.



Note Although the EESPort and MAPort use the 2.1 version of the ASAM XIL API, they have their functionality implemented for the 2.0.1 API version. The TargetScript in MAPort also implements the 2.1 API.

Locate example VeriStand ASAM XIL testbench and framework files in <Common Data>\Examples\DotNet4.6.2\ASAM XIL.

Accessing the VeriStand ASAM XIL Testbench

To access the VeriStand ASAM XIL testbench in your C# application, you must create a vendor-specific testbench.
Use the following values as an example on how to create a testbench for VeriStand:

```
ITestbenchFactory testbenchFactory = new TestbenchFactory();
ITestbench testbench = testbenchFactory.CreateVendorSpecificTestbench(
vendorName:"National Instruments",
productName:"NI VeriStand ASAM XIL Interface",
productVersion:"2020");
```

Implementation Differences and Limitations with the ASAM XIL Interface

VeriStand's ASAM XIL implementation deviates from the ASAM XIL standard in areas such as signal generation and capturing.

Implementation Differences

The following table displays implementation differences.

Process	Implementation differences
Signal Generation	▪ ConditionWatcher definitions cannot be saved in the STI format when a SignalDescriptionSet contains ConditionWatchers as StopTriggers. ▪ Signal generation operations cannot write to model parameters. ▪ Signal generation processes create temporary real-time sequence files in the temporary directory.
Capturing	▪ When Capture.DurationUnit is set to eSAMPLES, VeriStand ASAM XIL assumes

Process	Implementation differences
	the data capture rate was set during capture creation. ▪ Capture.Stop() may take longer than expected to complete execution. The function only returns a value after flushing log data into files and finishing post-processing. ▪ Capture.Fetch() might not return the latest samples because of buffering. ▪ VeriStand logs data to TDMS despite what capture result writer you choose. The data is later processed by either an MDF or in-memory capture result writer.
Miscellaneous	▪ Condition Watchers conditions support the syntax, operators, and expression functions that VeriStand real-time sequences support. ▪ Signal generation and capturing are not supported when the gateway is running on a machine other than localhost.

Limitations

The following table displays known limitations.

Process	Limitation
Capturing	Capture sessions ignore data logging errors. ASAM XIL API users will not receive an error notification. Instead, the capture will immediately enter the eFINISHED state and cease logging data. Untriggered captures also cause this state change. Capture results and .mdf files contain logging data until when the error occurred.

ASAM XIL Framework C# Access

To access the VeriStand ASAM XIL framework in your C# application, you must create a vendor-specific framework.
Use the following values to create a framework for VeriStand.

> **Note** To run this code, you must reference the ASAM.XIL.Implementation.FrameworkFactory assembly in order to have access to the FrameworkFactory class. This assembly installs with VeriStand.

```
IFrameworkFactory frameworkFactory = new FrameworkFactory();
IFramework framework = frameworkFactory.CreateVendorSpecificFramework(
vendorName:"National Instruments",
productVersion: "2020");
```

Configuring the ASAM XIL Framework

The ASAM XIL framework is the central class you use to manage and configure testbench ports, record data, and stimulate variables.

You can use the ASAM XIL framework to complete tasks such as logging variables and stimulating values for variables. You can also create units, data types, and framework variables, and map framework variables to testbench variables.
To configure the ASAM XIL framework, create the following XML configuration files:

- Port configuration XML file—Configures one or more VeriStand ASAM XIL testbench ports.
- Framework configuration XML file—Defines your ports, port configuration files, and mapping files.
- Framework mapping XML file—Defines the labels for the testbench and framework variables and then map them together.

Locate example VeriStand ASAM XIL testbench and framework files in <Common Data>\Examples\DotNet4.6.2\ASAM XIL.

ASAM XIL Port Configuration Tag Reference

Use specific XML elements and complex elements in the port configuration XML file to configure one or more VeriStand ASAM XIL testbench ports.

The following table displays the XML elements you can use in a VeriStand ASAM XIL port configuration XML file.

Tool	Required?	Element Type	Min/Max Occurrences	Description
<NIVSPortConfig>	Yes	Top-level	1/1	The type of configuration file.
<Version>	Yes	VersionType	1/1	The version of the port configuration file.
<Project>	Yes	NonEmptyStringType	1/1	The path to the VeriStand project you want to connect to through the ASAM XIL Interface.
<ShowWorkspace>	No	xs:boolean	0/1	Specifies whether to display the workspace when a project is deployed.
<ShowVeriStandScreen>	No	xs:boolean	0/1	Specifies whether to display the VeriStand screen when a project is deployed.
<UndeployVeriStandProjectOnDisconnect>	No	xs:boolean	0/1	Specifies whether to undeploy the running VeriStand project when the port is disconnected.
<EESPortConfig>	No	EESPortConfigType	0/1	A container for the EESPort

Tool	Required?	Element Type	Min/Max Occurrences	Description
				configuration options.
<HWTriggerChannelList>	No	HWTriggerChannelListType	0/1	The list of channels you want to use to fire EESPort hardware triggers.
<HWTriggerChannel>	No	HWTriggerChannelType	0/unbounded	The path to the channel you want use as a hardware trigger. You can only set one hardware trigger channel per target. **Note** The hardware trigger only occurs if the channel value changes from 0 to any other value.
<SWTriggerChannelList>	No	SWTriggerChannelListType	0/1	A list of channels you want to use to fire EESPort software triggers.

Tool	Required?	Element Type	Min/Max Occurrences	Description
<SWTriggerChannel>	No	SWTriggerChannelType	0/unbounded	A path to the software channel you want to trigger. You can only set one software trigger channel per target. **Note** The software trigger only occurs if the channel value changes from 0 to any other value. Set the AutoReset attribute of this element to true to automatically set the channel back

Tool	Required?	Element Type	Min/Max Occurrences	Description
				to zero after the trigger fires. If AutoReset is set to false, you must manually set the channel value back to zero.
<MAPortConfig>	No	MAPortConfigType	0/1	The container for the MAPort configuration options.
<LogFilePath>	No	NonEmptyStringType	0/1	Specifies the path of the temporary TDMS log file that is created during capturing.
<TaskList>	No	TaskListType	0/1	A list of user-defined tasks.
<Task>	No	TaskType	0/unbounded	Specifies the task name and frequency. Use the format <TaskN>, where N is the frequency of the task in Hz.

Tool	Required?	Element Type	Min/Max Occurrences	Description
<LogPostProcessPeriodMS>	No	xs:int	0/1	The length, in milliseconds, of the post-processing period.  **Note** ▪ If you call Fetch() before post-processing completes, Fetch()returns an empty result. ▪ If post-processing occurs periodically, a new

Tool	Required?	Element Type	Min/Max Occurrences	Description
				TDMS file is created for each period, which adds overhead to the operation
<ChannelDataTypeOverrideList>	No	ChannelDataTypeOverrideListType	0/1	A list of channels that you want to override the default data type. In VeriStand, channels are double-precision, floating-point numerics by default.
<ChannelDataTypeOverride>	No	ChannelDataTypeOverrideType	0/unbounded	A container for the channel name and data type. Channel names inside this container must be unique.
<ChannelName>	Yes*	NonEmptyStringType	0/1	The name of the channel for which

Tool	Required?	Element Type	Min/Max Occurrences	Description
				you want to change the data type.
<DataType>	Yes*	DataTypeType	0/1	The data type you want the channel value to change to.
*Required child elements of optional parent elements are only required if you use the parent element.				

Types

- VersionType—Specifies version information.

Attribute	Required?	Attribute Type
Major	Yes	xs:unsignedInt
Minor	Yes	xs:unsignedInt
Fix	Yes	xs:unsignedInt
Build	Yes	xs:unsignedInt

- NonEmptyString—Restricts the element to allow only non-empty strings.

Restrictions	Minimum Length
xs:string	1

- DataTypeType—Restricts the element to only allow int, uint, bool and float data types. Only scalars, vectors, and matrices are allowed from the data types.

Creating Real-Time Test Scenarios with Stimulus Profiles and Real-Time Test Sequences

Use the **VeriStand Stimulus Profile Editor** to create specific test scenarios for use in your real-time test applications.

In addition to using the Stimulus Profile Editor environment, you can use the LabVIEW VIs on the **Stimulus Profile** palette in LabVIEW to execute and control stimulus profile and real-time sequence files programmatically. VeriStand includes examples of using these VIs in the <LabVIEW>\examples\VeriStand\API\Execution API\Sequences directory.

1. Create a stimulus profile—Use stimulus profiles to act as the test executive that defines the stimuli to apply to a unit under test (UUT).
2. Create a real-time sequence—Use real-time sequences to define specific tasks for a unit under test (UUT).
3. View test results—Enable the Stimulus Profile Editor to automatically open stimulus profile test results.
4. Log-real-time test data—Use stimulus profiles to log real-time test data to the host computer while a test executes on a target.
5. Use stimulus profile arguments to communicate with the VeriStand Editor—Send commands to the VeriStand Editor through a stimulus profile to specify a VeriStand project, a system definition, VeriStand Gateway IP address, or connect to a target from the system definition.

If you want to script your test sequences in Python, you can use the VeriStand Python Integration Package as an alternative to the Stimulus Profile Editor. This open-source package allows you to execute Python scripts as if they were VeriStand real-time sequences. To learn more about this feature, view the VeriStand Python Documentation.

Navigating the Stimulus Profile Editor Environment

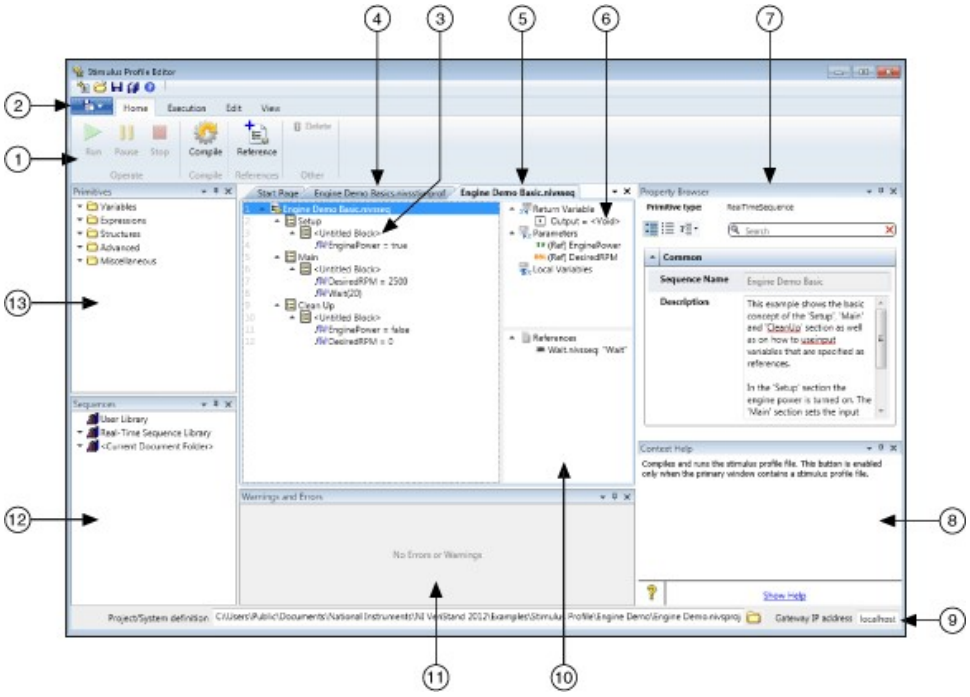
Use the Stimulus Profile Editor to create, modify, and execute stimulus profiles and real-time sequences.

The **Stimulus Profile Editor** is a highly customizable environment consisting of various elements that you can show, hide, resize, or rearrange to suit your needs. The editor is a separate executable from System Explorer, VeriStand Editor, or Workspace. It allows you to develop your test profiles in parallel with the system definitions and user interfaces that they interact with. However, a stimulus profile must be associated with a deployed VeriStand system definition in order to execute.

In the Stimulus Profile Editor, you can start multiple concurrent stimulus profile executions. Each stimulus profile execution performs sequential execution of one or more real-time sequences.

To open the Stimulus Profile Editor, use the VeriStand Editor and click **Tool Launcher** » **Stimulus Profile Editor**.

Use the following sections of the Stimulus Profile Editor to modify your code.



1	Ribbon—Perform actions such as editing the environment and running profiles and sequences.
2	File menu—Perform actions such as creating, saving, and opening profiles and sequences.
3	Real-time sequence code.
4	Stimulus profile—Acts as the test executive that defines the stimuli to apply to a unit under test (UUT).
5	Real-time sequence—Defines specific tasks for a UUT.

6	Variables pane—displays all the variables the selected real-time sequence has access to at run time and uses in expressions.
7	Property Browser—Displays the editable properties of stimulus profile steps, real-time sequences, primitives, and references. Select an item in the code of a stimulus profile or real-time sequence to configure that item.
8	Context Help—Displays suggested help for an item that you hover over.
9	Status Bar—Displays the **Project/System definition** that you launched the editor from and the **Gateway IP Address**. By default, VeriStand associates stimulus profiles with the system definition file of the project that you create them. You can **Browse** to a different project or system definition from the editor.
10	References pane—Contains a list of all the real-time sequences that the selected real-time sequence references.
11	Warning and Errors list—Lists any errors, warnings, or messages in a stimulus profile or real-time sequence file. Click an error, warning, or message to display a description.
12	Sequences palette—Contains real-time sequences you add to a stimulus profile or real-time sequence.
13	Primitives palette—Contains programming building blocks, such as statements and variables, you can add to a real-time sequence.

Creating a Stimulus Profile Editor Layout

Customize the individual sections of the **Stimulus Profile Editor** environment and save that layout for later use.

Before you begin, you should familiarize yourself with the Stimulus Profile Editor environment.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. In the Stimulus Profile Editor, modify the layout by dragging sections to other parts of the editor, close sections, or use the **View** tab to show or hide a section.
3. In the **View** tab, click **Save Layout**.
4. Enter a **Layout File Name** and click **OK**.

> **Note** You can save as many layouts as you need, but you cannot rename or delete a layout after you save it.

To access a saved layout, in the **View Tab**, click **Available Layouts** and select the layout you want to apply.

Creating Stimulus Profiles

Use stimulus profiles to act as the test executive that defines the stimuli to apply to a unit under test (UUT).

Before you begin, you should familiarize yourself with the Stimulus Profile Editor environment.

In real-time test, a **stimulus** is a physical or logical input that incites a reaction from the UUT. A typical stimulus might be a change in temperature, voltage, power, or any other variable that might affect the behavior of the UUT.
A **stimulus profile** contains specific tasks, called real-time sequences, that deploy to the UUT and execute in real-time. Stimulus profiles execute on the host computer and control some actions of the VeriStand environment. For example, stimulus profiles can open and close projects and user interface windows and log the results of test scenarios to TDMS files.

Stimulus profiles appear on the VeriStand Editor in the Project Files tab with the file extension .nivsstimprof. A single VeriStand project can contain multiple stimulus profiles that define different test scenarios. You can execute multiple stimulus profiles concurrently, and each stimulus profile can contain multiple real-time sequences.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. In the Stimulus Profile Editor, click **File** » **New** » **Stimulus Profile**.
   The profile organizes your code into the **Setup**, **Main**, and **Clean Up** blocks.
3. Drag steps from the Steps palette to the appropriate block of the stimulus profile.
   For example, you might launch a user interface as part of your **Setup** code and close the user interface as part of your **Clean Up** code. The **Main** block typically contains calls to real-time sequences, which are programs that define specific tasks to execute on the UUT. This block might also contain steps that configure data logging to TDMS.
4. For each step you add, click the step in the stimulus profile code and use the Property Browser to configure the step.
5. Click **Compile**.
   a. Resolve any Warnings and Errors.

   > **Note** A stimulus profile can run with warnings, but not errors. You can save the profile and resolve errors later.

6. **Save** the stimulus profile.

After you configure and save the stimulus profile and any real-time sequences it calls, you can run the profile. Deploy its associated system definition to run the stimulus profile.

Calling a Real-Time Sequence from a Stimulus Profile

Call a real-time sequence file from within a stimulus profile to define a specific task to execute on a unit under test (UUT).

Before you begin, create a stimulus profile.

1. Open the stimulus profile from which you want to call the sequence in the Stimulus Profile Editor.
2. In the **Steps** palette, expand **Real-Time Sequences**.

3. Select Real-Time Sequence Call and drag it to the stimulus profile code.
4. In the **Property Browser**, enter a **File Path** where the real-time sequence (.nivsseq) file is located.
5. Use the **Parameters** section of the Property Browser to map the sequence input parameters to channels.

> **Note** If the stimulus profile you are using is saved in the same directory as a real-time sequence,you can drag the real-time sequence file from the **Sequences** palette directly to the stimulus profile code to create a Real-Time Sequence Call step that calls that sequence.

If you make any changes to a real-time sequence called by a Real-Time Sequence Call step, select the step, and in the **Property Browser**, click **Update Parameters**. This will refresh the parameter mappings for the sequence call based on the current contents of the specified sequence. This ensures that parameter mappings exist for all parameters, that the data types of all parameter mappings match the data types in the corresponding sequence, and any parameter mappings that have no corresponding entry in the sequence are deleted.
VeriStand also supports CSV files as real-time sequences. The process for calling real-time sequences defined in CSV files is the same as the process for calling .nivsseq files. However, you cannot view the real-time sequence defined by a CSV file in the Stimulus Profile Editor. If you want to make edits to the real-time sequence, you must use a text editor.

Updating Model Parameters During a Stimulus Profile Test

Use the **Update Model Parameters from File** step in a stimulus profile to apply model parameter values defined in a text file to a simulation model that is deployed and running on a target.

By using this step to automate model parameter value updates, you can update your model at known points within the execution of the stimulus profile.

1. Create a stimulus profile that is associated with the system definition file that contains the model you want to update.

2. In the Steps palette, click to **VeriStand Control** » **Workspace** and drag the Update Model Parameters from File step to the stimulus profile code.
3. In the Property Browser, configure the **Update Model Parameters from File** step.

> **Note** The **Update Model Parameters from File** step also supports .m files generated by the Model Parameter Manager workspace tool.

   a. Enter a **Source** path to a text file (.txt) that conforms to the model parameter file format.
   b. Confirm that the **Target** property shows the target that the model is deployed to.
   c. Confirm that the **Delimiter** property is set to the delimiter that the **Source** file uses to separate parameter/value pairs.
   d. If you are using an alias file to define model parameters, enter the path to the file in the **Alias File** property.
   e. If you are using temporary variables within the **Source** file, enable **Allow Temporary Variables**
4. Deploy the system definition.
5. Compile and run the stimulus profile.

When the **Update Model Parameters from File** step executes, the model parameters update to the values in the **Source** file.

Stimulus Profile Steps

Use steps to call real-time sequences, configure data logging operations, and interact with other elements of VeriStand, such as the project or **Workspace**.

A **step** is an element of a stimulus profile that performs a specific action.

Subpalette	Description
Real-Time Sequences Steps	Call real-time sequences from a stimulus profile.

Subpalette	Description
Logging Steps	Log data from channels in a system definition.
VeriStand Control Steps	Interact with VeriStand, including the Workspace and the VeriStand project.
Other Steps	Add functionality to a stimulus profile.

Real-Time Sequences Steps

Call real-time sequences from a stimulus profile.

Palette Object	Description
Real-Time Sequence Call	Calls a real-time sequence, executes the sequence on the specified **Target**, and returns information about whether the sequence execution passes or fails.
Real-Time Sequence Group	Groups Real-Time Sequence Call steps so that each sequence within the group executes in parallel.

Real-Time Sequence Call Step

Calls a real-time sequence, executes the sequence on the specified **Target**, and returns information about whether the sequence execution passes or fails.

You can call a real-time sequence from any section of a stimulus profile (Setup, Main, or Clean Up) or from within a Real-Time Sequence Group.

Property/Section	Description
File Path	Specifies the path to the real-time sequence to execute. You can specify a real-time sequence file (.nivsseq) or a properly formatted CSV file.
Target Name	Specifies the name of the target on which the sequence executes. The target must be defined in the system definition file associated with the stimulus profile.
Timeout [ms]	Specifies the amount of time in milliseconds within which the real-time sequence must complete each timestep. A zero or negative

Property/Section	Description
	value indicates an infinite timeout. If the sequence does not complete a timestep within the specified amount of time, the VeriStand Engine aborts sequence execution and returns an error.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
Pass/Fail Evaluation	Includes properties you can use to configure pass/fail evaluation for the sequence: ▪ **Type**—Specifies the type of pass/fail evaluation to perform. This step evaluates the return variable for the real-time sequence to determine whether the sequence passes or fails. You can select from the following options: ▪ **AlwaysPass**—(Default) The evaluation always passes, regardless of the value of the return variable. ▪ **Boolean**—Evaluates a Boolean value for pass/fail status. True is pass, unless you invert the evaluation. ▪ **NumericBoundsCheck**—Evaluates an integer value relative to specified high and low boundaries. ▪ **Parameters**—Defines parameters for the pass/fail evaluation. The options that appear depend on the **Type** of evaluation you specify. **AlwaysPass** evaluations have no parameters to define, because they always pass. You can set the following parameters: ▪ **Invert**—[Type: Boolean] If TRUE, inverts a **Boolean** evaluation so that FALSE is pass and TRUE is fail.

Property/Section	Description
	▪ **Type**—[Type: NumericBoundsCheck] Specifies the type of bounds check: ▪ **Inbounds**—The evaluation passes if the value is within the specified bounds. ▪ **OutOfBounds**—The evaluation passes if the value is outside the specified bounds. ▪ **High**—[Type: NumericBoundsCheck] Specifies the high limit of the bounds. A value is within bounds if it is less than or equal to this value and greater than or equal to the **Low** value. ▪ **Low**—[Type: NumericBoundsCheck] Specifies the low limit of the bounds.
Group Number	Defines the group associated with this real-time sequence. Valid group numbers include 0 to 27. A group number of -1 does not refer to any group.
Parameters	Contains properties that define the parameter assignments for the real-time sequence. When you specify the **File Path** to the real-time sequence, this section updates to display all the parameters defined in the sequence. ▪ **Parameter Assignment**—Specifies the value to assign to a real-time sequence parameter. You can specify a constant value or a channel in the system definition file. ▪ **Update Parameters**—Refreshes the **Parameter Assignments** for the sequence call based on the current contents of the specified sequence. This ensures parameter assignments exist for all parameters and that the data types of all parameter assignments match the

Property/Section	Description
	data type of the corresponding sequence. This also deletes any parameter assignments that have no corresponding entry in the sequence. Use this option if you make any changes to the real-time sequence after you configure the Real-Time Sequence Call step.

Real-Time Sequence Group Step

Groups Real-Time Sequence Call steps so that each sequence within the group executes in parallel.

This step does not complete until each real-time sequence in the group executes.

Property/Section	Description
Sequence Group Name	Specifies the name of the sequence group.
Description	Specifies a description for the current item. This text appears in the Context Help when you hover over the item in the Stimulus Profile Editor.

Logging Steps

Log data from channels in a system definition.

Palette Object	Description
Channel Group	Groups channels in a logging configuration.
Start Logging	Starts logging data from the system definition channels you specify to a TDMS file.
Stop Logging	Stops a currently running logging configuration.

Channel Group Step

Groups channels in a logging configuration.

TDMS files arrange log data into groups of channels. The Channel Group Name you specify and the Channels you add to the group correspond to the group and channels that appear in the TDMS file. You can create multiple channel groups under a single logging configuration to create multiple groups in one TDMS file.



Note You must use this step as a child of a Start Logging step.

Property/Section	Description
Channel Group Name	Specifies the name of the channel group.
Channels	Specifies the channels that belong to the channel group. Click the search button to display an interactive system definition hierarchy tree. Place check marks next to the channels you want to add to the channel group.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Start Logging Step

Starts logging data from the system definition channels you specify to a TDMS file.

When you add this step to a stimulus profile, VeriStand automatically creates a **Channel Group** step to associate with the logging configuration. You can add multiple channel groups under a single logging configuration.

Property/Section	Description
Configuration Name	Specifies the name of a specific data logging operation. You use this name to identify and control the operation. For example, to start and stop data logging, both the Start Logging step and the Stop Logging step must use the same Configuration Name.
File Path	Specifies the name and location of the log file to save data to. The Stimulus Profile Editor creates log files in the TDMS file format.

Property/Section	Description
	You can enter an absolute or relative path. VeriStand treats relative paths as relative to the directory that contains the ATML results file for the stimulus profile.
Timestamp Filename	If True, specifies to append a timestamp to the data log filename that indicates when the file is created.
Replace Existing File	If True, specifies to overwrite an existing file at the location specified by File Path. If False, specifies to append data to the existing log file. If you choose to append data to an existing TDMS file, VeriStand logs new data for existing channels under the existing channel and group names, and the file only shows the initial creation date in the file header. The TDMS file does not retain the timestamps of subsequent logging operations that append data to an existing file.
Log Rate [Hz]	Specifies the rate in hertz at which to log data. VeriStand logs at the closest possible rate to this value that does not exceed the rate at which a target produces data. The default is 100.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
Triggered Logging	Includes properties you can use to configure a trigger that starts logging: ▪ **Trigger Condition**—Specifies the condition under which to register start and stop triggers: ▪ **none**—(Default) No trigger is specified. Logging starts when this step executes and continues until a stop logging command is received or the stimulus profile stops executing. ▪ **in_limits**—Registers a start trigger when the **Trigger Channel** value

Property/Section	Description
	enters the bounds specified by **High Limit** and **Low Limit** and a stop trigger when the channel value leaves the bounds. ▪ **out_of_limits**—Registers a start trigger when the Trigger Channel value leaves the bounds specified by High Limit and Low Limit and a stop trigger when the channel value re-enters the bounds. ▪ **Trigger Channel**—Path to the channel, as specified in the system definition file, to watch for the specified Trigger Condition. ▪ **High Limit**—High limit for trigger limit analysis. ▪ **Low Limit**—Low limit for trigger limit analysis. ▪ **Pre-Trigger Duration**—The number of seconds of data to retain in the buffer in case a start trigger occurs. When the start trigger occurs, any buffered data is included in the log. ▪ **Post-Trigger Duration**—The duration in seconds to continue logging data after a stop trigger occurs.
File Segmenting	Includes properties you can use to control if and when a log file is split into separate files: ▪ **Segment Options**—Specifies whether and how the log file is segmented: ▪ **DoNotSegment**—Does not segment the log file, regardless of how large it gets. ▪ **OnStartTrigger**—Starts a new log file each time a start trigger occurs.

Property/Section	Description
	▪ **SizeLimit**—Starts a new log file when the current file reaches a specified size. ▪ **Segment Size**—[Segment Options: SizeLimit] Specifies the maximum size of a log file segment, in bytes.

Stop Logging Step

Stops a currently running logging configuration.

To stop logging, the Configuration Name must match the name you specified in the Start Logging step.

Property/Section	Description
Configuration Name	Specifies the name of a specific data logging operation. You use this name to identify and control the operation. For example, to start and stop data logging, both the Start Logging step and the Stop Logging step must use the same Configuration Name.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

VeriStand Control Steps

Interact with VeriStand, including the **Workspace** and the VeriStand project.

For example, you can use Control steps to open and close the project, open and close the Workspace, and access Workspace tools.

Palette object	Description
Launch NI VeriStand	Launches the VeriStand executable.

Subpalette	Description
Project Steps	Open, communicate with, or close a VeriStand project (.nivsprj) file.
Workspace Steps	Interact with the Workspace and Workspace tools.

Launch NI VeriStand Step

Launches the VeriStand executable.

The Stimulus Profile Editor runs as a separate executable from the main VeriStand application. Ensure VeriStand is running to execute tests. However, you can edit stimulus profiles and real-time sequences without running VeriStand.

Property/Section	Description
Timeout	Specifies the time in seconds to wait for VeriStand to launch before returning a timeout error.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Project Steps

Open, communicate with, or close a VeriStand project (.nivsprj) file.

Palette object	Description
Open VeriStand Project	Opens the VeriStand project (.nivsprj) file you specify.
Close Active VeriStand Project	Closes the active VeriStand project (.nivsprj) file.
Control Active VeriStand Project	Sends a command to the active VeriStand project (.nivsprj) file. The project must be open for it to receive commands.

Open VeriStand Project Step

Opens the VeriStand project (.nivsprj) file you specify.

A stimulus profile must be associated with an open and deployed project to execute real-time sequences. Use this step to open a project file directly from a stimulus profile. You can then use the Control Active VeriStand Project step to deploy the project.

Property/Section	Description
Show Project	If TRUE, specifies to display the project in the VeriStand Editor.
Project Path	Specifies the path to the project file. This can be an absolute or relative path. Relative paths must be relative to the directory that contains the stimulus profile.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
Security	Includes the following properties: ▪ **Username**—(Optional) Specifies the username to use to access the project. If the project has only one defined user, such as the Administrator, and that user has no password, you do not need to enter the username. ▪ **Password**—(Optional) Specifies the password associated with the Username. Not all usernames have associated passwords. If a username does have an associated password and you leave this property empty, VeriStand prompts you to enter the password when this step executes.

Close Active VeriStand Project

Closes the active VeriStand project (.nivsprj) file.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Control Active VeriStand Project Step

Sends a command to the active VeriStand project (.nivsprj) file.

Property/Section	Description
Command	Specifies the command that the step sends to the project: ▪ **Run**—Runs the project. ▪ **Deploy**—Deploys the system definition file associated with the project to the target. ▪ **Connect**—Connects the project on the host computer to a target. This option only establishes a connection to a target. It does not deploy the system definition. ▪ **Disconnect**—Disconnects the project from the target. This option does not stop execution of the system definition file on the target. ▪ **Undeploy**—Undeploys the system definition file associated with the project from the target. Undeploying the system definition file stops execution on the target.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Workspace Steps

Interact with the **Workspace** and **Workspace** tools.

Palette object	Description
Close VeriStand Workspace	Closes the VeriStand workspace.
Open VeriStand Workspace	Opens the VeriStand Workspace and loads the screen (.nivsscreen) file associated with the currently active VeriStand project.
Update Model Parameters from File	Updates parameter values for a simulation model to the values specified in a text (.txt) file.

Subpalette	Description
Workspace Tools Steps	Opens and communicates with Workspace tools directly from a stimulus profile.

Close VeriStand Workspace

Closes the VeriStand workspace.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Open VeriStand Workspace

Opens the VeriStand **Workspace** and loads the screen (.nivsscreen) file associated with the currently active VeriStand project.

Open the Workspace to see the affects of the test on channel values while the stimulus profile executes.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Update Model Parameters from File Step

Updates parameter values for a simulation model to the values specified in a text (.txt) file.

Use this step to synchronize parameter value updates with real-time sequences without stopping the execution of a stimulus profile.

Property/Section	Description
Source	Specifies the path to the text file that contains the new model parameter values. The source file contains a set of key/value pairs that represent model parameter names and the corresponding values to assign each parameter. This file must be a .txt file, and must follow the expected model parameter file format. This step also provides limited support for .m files that follow the exact format generated by the Model Parameter Manager Workspace tool.
Target	Specifies the name of the target, as it appears in the system definition file, on which the model executes. This step can only update model parameters on one target at a time.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
Advanced	Includes the following properties: ▪ **Alias File**—Specifies the path to an alias file to define mappings for aliased parameter names in the source file. Alias files allow you to easily reuse the same source file for multiple models. For example, you can write a source file with parameter names a, b, and c, and then use an alias file to map a, b, and c to parameters in the current model.

Property/Section	Description

	Alias files must be .txt files that use the same Delimiter as the source file.
	▪ **Delimiter**—Specifies the delimiter used to separate model parameter names and values in the source file and, if used, parameter aliases and names in the alias file:
	▪ **Tab** (Default)
	▪ **Equals**
	▪ **Comma**
	▪ **Allow Temporary Variables**—If True, specifies to allow temporary variables in the source file. For example, the following snippet defines a temporary variable, tempX, and then uses tempX in an expression that defines a model parameter value:
	`tempX 0.5`
	`{parameter} 10 * tempX`
	Note Model parameter names and alias names are case-sensitive. You will not receive error messages for variables that do not map to model parameters. Instead, the step discards the corresponding value when updating model parameters. This step assumes that keys in the source file that do not match a model parameter name or an alias defined in the alias file are temporary variables.

Workspace Tools Steps

Opens and communicates with **Workspace** tools directly from a stimulus profile.

Palette object	Description
Open Workspace Tool	Opens the workspace tool specified by **VI Path**.
Send Workspace Tool Message	Sends a message or command to a workspace tool VI, as well as data required by the case in the tool that handles the command.

Open Workspace Tool Step

Opens the workspace tool specified by **VI Path**.

Property/Section	Description
Initialization Time	Specifies the time in milliseconds to allow for the workspace tool to initialize after it loads. The stimulus profile does not proceed to the next step until the initialization time elapses.
VI Path	Specifies the path to the workspace tool VI.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Send Workspace Tool Message

Sends a message or command to a workspace tool VI, as well as data required by the case in the tool that handles the command.

The tool at the **VI Path** you specify must be capable of receiving messages.

Property/Section	Description
VI Path	Specifies the path to the workspace tool VI.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
Message Details	Includes the following properties:

Property/Section	Description
	▪ **Command**—Specifies the message or command to send to the workspace tool. The tool VI must include code to handle the command you send. This property expects a string. ▪ **Data**—Specifies data to send to the workspace tool. This can be any data that the code that handles the **Command** accepts, but must be a string with a format that the tool can process. ▪ **Timeout [ms]**—Specifies the amount of time in milliseconds to wait for a response from the workspace tool before timing out.

Other Steps

Add functionality to a stimulus profile.

Examples include displaying a message to the user, replaying a previously recorded macro (.nivsmacro) file, or grouping steps to better organize and reuse code.

Palette object	Description
Command Shell	Invokes the Windows Command Prompt, calls the application specified by Filename, and passes that application the specified arguments.
Group	Groups steps with no impact on execution.
Macro Player	Replays a previously recorded macro (.nivsmacro) file.
Message Box	Displays a pop-up message to the stimulus profile operator.

Subpalette	Description
FTP Steps	Interact with files on an FTP server, such as a real-time target.

Command Shell Step

Invokes the Windows Command Prompt, calls the application specified by Filename, and passes that application the specified arguments.

You can configure whether the rest of the stimulus profile waits for the Command Prompt to return before completing execution.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
Command Prompt	Includes the following properties that configure interaction with the Command Prompt: ▪ **Filename**—The name of the application to pass arguments to. You can enter a system command (for example, cmd notepad), a system variable, or the fully-qualified path to the executable. ▪ **Arguments**—The arguments to pass to the called application. ▪ **Hide command shell**—If True, runs the application in an inaccessible Command Prompt window. **Note** If Redirect Standard Error and Redirect Standard Output are both False, the stimulus profile ignores a True value for this option and always shows the Command Prompt.
Execution	Includes the following properties that configure step execution: ▪ **Wait to Complete**—If True, blocks execution of the remainder of the

Property/Section	Description
	stimulus profile until the Command Prompt returns. **Note** The Command Prompt does not return until all applications it calls are properly closed. ▪ **Wait Timeout**—Specifies the time in milliseconds to wait for the Command Prompt to return before returning a timeout error. The default is -1 to specify an infinite timeout. ▪ **Redirect Standard Error**—If True, writes the standard error output of the Command Prompt to the ATML test results file for the stimulus profile. ▪ **Redirect Standard Output**—If True, writes the standard output of the Command Prompt to the ATML test results file for the stimulus profile.

Macro Player Step

Replays a previously recorded macro (.nivsmacro) file.

Property/Section	Description
VeriStand Macro File	The path to the macro file to play back.
Playback Mode	Specifies the speed at which to play back the macro file: ▪ **Ignore Timing**—(Default) Plays back the macro file as fast as possible. ▪ **Use Timing**—Plays back the macro file at the speed at which it was recorded.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Message Box Step

Displays a pop-up message to the stimulus profile operator.

Property/Section	Description
Message	The message that the pop-up displays to the operator.
Dialog Title	The text that appears in the title bar of the pop-up.
Default Text	The text that appears on the button the operator uses to close out of the pop-up. For example, OK.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Group

Groups steps with no impact on execution.

You can use groups to organize your code. Because a **Group** is a single item, using groups makes it easy to move or duplicate a set of steps.

Property/Section	Description
Group Name	The name of the group.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

FTP Steps

Interact with files on an FTP server, such as a real-time target.

Palette object	Description
FTP Download	Downloads files from an FTP server, such as a real-time target.
FTP Upload	Uploads a file to an FTP server, such as a real-time target.

FTP Download

Downloads files from an FTP server, such as a real-time target. .

Note Some versions of Microsoft Windows block incoming FTP traffic by default. If prompted by the OS, allow VeriStand to use port 21 (FTP) to transfer data.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
FTP Server	Includes the following properties that configure access to the FTP server: ▪ **URL**—The address of a file or directory on the FTP server. **Note** The address cannot contain any characters that define the protocol. For example, 10.0.72.66/samplefile.txt is a valid address, but ftp://10.0.72.66/samplefile.txt is not. ▪ **Is Folder?**—If TRUE, specifies that **URL** is a folder. If FALSE, specifies that **URL** is a specific file. ▪ **Filter**—If the path to the files to transfer is a folder, specifies a regular

Property/Section	Description
	expression to use to filter the files in the folder by filename. **Note** This step only acts on the files that match the regular expression. The match is not case-sensitive. ▪ **Username**—Specifies the username to use to access the FTP server. ▪ **Password**—Specifies the password for **Username**. ▪ **Timeout**—Specifies the time in seconds to wait for a response from the FTP server before returning a timeout error. ▪ **Passive Connection**—If TRUE, specifies that this step initiates a connection on the FTP data port. If FALSE, specifies that this step listens for a connection on the FTP server. ▪ **SSL**—If TRUE, specifies that data transmission is encrypted using an SSL protocol.
FTP Transfer	Includes the following properties that configure details of the file transfer: ▪ **Destination**—The destination directory for downloaded files. If you do not specify a destination, this step downloads files to the directory that contains the stimulus profile. ▪ **Binary**—If TRUE, transmits data in binary form. If FALSE, transmits data as text.

Property/Section	Description
	▪ **Behavior**—Specifies the action to take if a file of the same name already exists in the **Destination** directory. ▪ **Overwrite**—Overwrites the existing file. This option does not change the creation date of the file, but it does change the most recent access date. ▪ **Skip**—Skips the file completely and does not include it in the download. ▪ **Unique**—Downloads the file and appends a number to the filename to create a unique name. ▪ **Buffer Length**—The size of the buffer, in bytes, to use for streaming data. The minimum is 2.
Proxy	Includes the following properties for configuring a proxy server: ▪ **Enable Proxy**—If TRUE, specifies to transmit data through a proxy server. ▪ **Proxy URL**—Specifies the URL of the proxy server. ▪ **Proxy Username**—Specifies the username for the proxy server. ▪ **Proxy Password**—Specifies the password for the **Proxy Username**.
Port	Includes the following property that configures which port the proxy server uses: ▪ **Proxy Port**—Specifies the port the FTP proxy uses. If **Enable Proxy** is FALSE, this value is ignored.

FTP Upload

Uploads a file to an FTP server, such as a real-time target.

Note Some versions of Microsoft Windows block incoming FTP traffic by default. If prompted by the OS, allow VeriStand to use port 21 (FTP) to transfer data.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
FTP Server	Includes the following properties that configure access to the FTP server: ▪ **URL**—The address of a file or directory on the FTP server. **Note** The address cannot contain any characters that define the protocol. For example, 10.0.72.66/samplefile.txt is a valid address, but ftp://10.0.72.66/samplefile.txt is not. ▪ **Username**—Specifies the username to use to access the FTP server. ▪ **Password**—Specifies the password for **Username**. ▪ **Timeout**—Specifies the time in seconds to wait for a response from the FTP server before returning a timeout error. ▪ **Passive Connection**—If TRUE, specifies that this step initiates a connection on the FTP data port. If FALSE, specifies that this step listens for a connection on the FTP server.

Property/Section	Description
	▪ **SSL**—If TRUE, specifies that data transmission is encrypted using an SSL protocol.
FTP Transfer	Includes the following properties that configure details of the file transfer: ▪ **Source**—The path to the source file(s) to upload. This can be a file or a folder containing multiple files. ▪ **Is Folder?**—If TRUE, specifies that **URL** is a folder. If FALSE, specifies that **URL** is a specific file. ▪ **Filter**—If the path to the files to transfer is a folder, specifies a regular expression to use to filter the files in the folder by filename. **Note** This step only acts on the files that match the regular expression. The match is not case-sensitive. ▪ **Binary**—If TRUE, transmits data in binary form. If FALSE, transmits data as text. ▪ **Behavior**—Specifies the action to take if a file of the same name already exists in the **Destination** directory. ▪ **Overwrite**—Overwrites the existing file. This option does not change the creation date of the file, but it does change the most recent access date. ▪ **Skip**—Skips the file completely and does not include it in the download.

Property/Section	Description
	▪ **Unique**—Downloads the file and appends a number to the filename to create a unique name. ▪ **Buffer Length**—The size of the buffer, in bytes, to use for streaming data. The minimum is 2.
Proxy	Includes the following properties for configuring a proxy server: ▪ **Enable Proxy**—If TRUE, specifies to transmit data through a proxy server. ▪ **Proxy URL**—Specifies the URL of the proxy server. ▪ **Proxy Username**—Specifies the username for the proxy server. ▪ **Proxy Password**—Specifies the password for the **Proxy Username**.
Port	Includes the following property that configures which port the proxy server uses: ▪ **Proxy Port**—Specifies the port the FTP proxy uses. If **Enable Proxy** is FALSE, this value is ignored.

Creating Real-Time Sequences

Use real-time sequences to define specific tasks for a unit under test (UUT).

Before you begin, you should familiarize yourself with the Stimulus Profile Editor environment.

Real-time sequences deploy to the target and can deterministically execute. Sequences test the UUT by reading from and writing to channels defined in the system definition file. These sequences can include a wide variety of programming constructs, including while loops, for loops, variables, and conditional statements.

Real-time sequences appear on the VeriStand Editor in the Project Files tab with the file extension .nivsseq. A real-time sequence cannot execute on its own and must belong to a stimulus profile.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. In the Stimulus Profile Editor, click **File** » **New** » **Real-Time Sequence**. The sequence organizes your code into the **Setup**, **Main**, and **Clean Up** blocks. The Primitives palette contains programming building blocks, such as statements and variables, you can use to define the task the real-time sequence completes.
3. Create the variables your sequence code can access.
    a. Drag variables from the Primitives palette to one of the following sections of the Variables pane.

Section	Description
Return Variable	The value the real-time sequence returns after it executes.
Parameters	The inputs and outputs that the sequence accepts when called and can pass out to other sequences or the stimulus profile.
Local Variables	Variables you want to access from statements within the current sequence.
Channel References	References that allow you to read/write system definition channels directly from a real-time sequence. **Note** Channel references bind to specific system definition channels when added to a real-time sequence. The sequence that contains them is bound to that system definition file. If you want to write a sequence that you can reuse across

Section	Description
	multiple system definition files, use Parameters.



Note Use Double or Boolean primitives to create parameters that you want to map to system definition channels. All channels in VeriStand are 64-bit floating point numbers (doubles), but many can also accept Boolean values.

b. To add channel references, right-click **Channel References** and click **Insert Channels**.

4. Map parameters to system definition channels.
   a. Select a parameter in the Variables pane.
   b. In the Property Browser, give the parameter a unique **Identifier**.
   c. Next to **Default Assignment**, click **Browse** and navigate to the appropriate channel or click **View aliases** in the window to select a channel by its alias.
5. If you want to use **Local Variables** in statements or expressions, use the Property Browser to configure a unique **Identifier** and a **Default Value** for those variables
6. If you want to call another sequence from an expression in the current sequence, add the new sequence to the **References** pane.
   You can drag variables from the Variables pane or sequences from the Sequences palette directly to the sequence code to automatically create expressions that act on those elements. In the case of sequences, VeriStand also creates references to the sequence.
7. Use expressions, structures, and other primitives to build the sequence code. Select any item in the sequence code to configure it in the Property Browser.
8. Click **Compile**.
   a. Resolve any Warnings and Errors.

> **Note** A sequence can run with warnings, but not errors. You can save the profile and resolve errors later.

9. **Save** the sequence.

After creating the real-time sequence, add a call for the real-time sequence to a stimulus profile. A sequence cannot execute independently of a stimulus profile. Run the stimulus profile to run the sequence.

Declaring Variables in a Real-Time Sequence

Declare and define the variables a real-time sequence can access and act on.

Before you begin, create a real-time sequence.

Real-time sequences can contain four different sections of variables that appear in the Variables pane.

Section	Description
Return Variable	The value the real-time sequence returns after it executes.
Parameters	The inputs and outputs that the sequence accepts when called and can pass out to other sequences or the stimulus profile.
Local Variables	Variables you want to access from statements within the current sequence.
Channel References	References that allow you to read/write system definition channels directly from a real-time sequence. **Note** Channel references bind to specific system definition channels when added to a real-time sequence. The sequence that contains them is bound to that system definition file. If you want to write a sequence that you can reuse across multiple

Section	Description
	system definition files, use Parameters.

1. Open a real-time sequence.
2. Declare a **Return Variable**.
    a. Determine the data type you want the sequence to return:
        - Boolean—Returns a true/false value that indicates if the sequence passes.
        - Void Return Value—If you do not need to perform a pass/fail evaluation on the sequence.
    b. In the Primitives palette, drag a data type from the **Variables** folder to the **Return Variable** section of the Variables pane.
    c. In the Variables pane, select the variable and use the Property Browser to configure its name and, for a Boolean, default value.
3. Declare a **Parameter**.
    a. To conserve memory, select the smallest possible data type that can hold the value you want to pass into or out of the sequence.

    > **Note** Use Double or Boolean primitives to create parameters to map to system definition channels. All channels in VeriStand are 64-bit floating point numbers (doubles), but many can also accept Boolean values.

    b. In the Primitives palette, drag a data type from the **Variables** folder to the **Parameters** section of the Variables pane.
    c. In the Variables pane, select a variable and use the Property Browser to configure its **Default Assignment**, or the channel it maps to.
4. Declare a **Local Variable**.

a. To conserve memory, select the smallest possible data type that can hold the value you want to pass into or out of the sequence.
b. In the Primitives palette, drag a data type from the **Variables** folder to the **Local Variables** section of the Variables pane.
c. In the Variables pane, select a variable and use the Property Browser to configure its **Default Value**.

5. Declare a **Channel Reference**.
   a. In the **Variables** pane, right-click the **Channel References** section and click **Insert Channels**.
   b. Use the configuration tree to select the channels you want to add as **Channel References** and click **OK**.

After you declare variables, you can include them in expressions that you add to the sequence code. When you begin entering an **Expression** in the Property Browser, VeriStand automatically includes the names of defined variables to the list of values that appears.

Using CSV Files as Real-Time Sequences

Use Comma Separated Values (.csv) files within stimulus profiles to stimulate, fault, and evaluate channels.

1. Format your CSV file.
2. Call the CSV file from a stimulus profile file as a real-time sequence or by reference from another real-time sequence file.

CSV files that you call as real-time sequences return a Boolean return value. This Boolean is false if the CSV file evaluates a channel that fails to meet any expected value defined within the CSV file. Otherwise, the Boolean returns true. CSV files that do not evaluate channels for pass/fail requirements always return true.

CSV File Formatting for Stimulus Profiles

Format your CSV file column headers to define inputs, values to assign to those inputs, and timestamps to update the input values.
Based on your use case, use the following column headers in your CSV file.

Use Case	Column header	Description
All	timestamp	Specifies the relative time from the start of the test, in milliseconds. This column specifies when to execute the action that a row defines, such as when to update a parameter value.
Stimulating	ParameterName	Specifies the name of an input parameter for the real-time sequence. This column contains the values to set for the parameter at the specified timestamp. ParameterName can be any string, but the name you specify effects how VeriStand creates the default channel assignment for the parameter. As with a standard real-time sequence file, you must assign channels in your system to each input parameter you specify. You can specify an unlimited number of parameters.
Faulting	#FLT_STATE#ParameterName	Specifies a fault state for ParameterName at the specified timestamp. Valid values are 0 (no fault) and 1 (fault). You must use this header together with #FLT_VALUE#ParameterName.
Faulting	#FLT_VALUE#ParameterName	Specifies the value to force ParameterName to when #FLT_STATE#ParameterName is 1. You must use this header together with #FLT_STATE#ParameterName.

Use Case	Column header	Description
Evaluating	#EXP(+tol;-tol;ms delay)#ParameterName	Specifies the expected pass/fail value for the ParameterName channel. VeriStand compares the actual values of ParameterName to the values in this column to determine whether to pass or fail the test. +tol and -tol are absolute values that specify the tolerance of the test, or the range above and below the expected value within which ParameterName must be to pass. ms delay is the delay in milliseconds to add to each timestamp value before updating the expected value of ParameterName. If you specify a delay on a value at a particular timestamp, VeriStand waits for timestamp + ms delay before updating the expected value from the one at the previous timestamp.



Note For all headers that include a ParameterName, VeriStand replaces any characters that would make the name invalid, according to the following rules.

Use Case	Column header
+	Plus
-	Minus
All other invalid characters	_ (underscore)

For all headers that include a ParameterName, VeriStand creates a default channel assignment based on the format of ParameterName. If ParameterName is a fully

qualified channel path or channel alias, VeriStand uses the full channel or alias as the default channel assignment. Otherwise, VeriStand treats ParameterName as a partially qualified alias and creates a fully qualified alias using ParameterName as the alias name. This new alias is the default channel assignment.

CSV File Formatting Examples

Format CSV files to run as real-time sequences or subroutines within a sequence in a stimulus profile.

The following table displays examples of CSV files based on the goal. The formatting of the CSV file depends on whether you want to stimulate, fault, and/or evaluate a channel.

Goal	Example

Stimulating a Channel	The following CSV file stimulates a channel with the alias channelX by updating the value of the channel every 100 milliseconds.
	`timestamp,channelX`
	`0,0`
	`100,5`
	`200,10`
	`300,20`
	`400,30`
	`500,40`

Stimulating Multiple Channels	The following CSV file stimulates two channels, channelX and channelY, by updating the channel values every 100 milliseconds.
	`timestamp,channelX,channelY`
	`0,0,-50.5`
	`100,5,-49`
	`200,10,-46`
	`300,20,-40`
	`400,30,-28`
	`500,40,-4`

Faulting a Channel	The following CSV file alternates between faulting and clearing a fault on channelX every

Goal	Example

	100 milliseconds, and forces the value of channelX to 100 whenever it faults the channel.
	`timestamp,#FLT_STATE#channelX,#FLT_VALUE#channelX`
	`0,0,0`
	`100,1,100`
	`200,0,0`
	`300,1,100`
	`400,0,0`
	`500,1,100`

Evaluating a Channel	The following CSV file tests channelX for an expected value, and updates the expected value every 100 milliseconds. In this example, the actual value of channelX can be within .05 of the expected value for a passing test, but no delay in reaching the value is allowed.
	`timestamp,#EXP(.05;.05;0)#channelX`
	`0,0`
	`100,1`
	`200,0`
	`300,-1`
	`400,0`
	`500,1`
	Note Because you can specify delay on evaluation tasks, multiple evaluations can run at different intervals. If you evaluate multiple channels, VeriStand runs each channel evaluation in a separate, parallel task.

Stimulating, Faulting, and Evaluating a Channel	The following CSV file stimulates channelX, faults channelY, and evaluates channelZ every 100 milliseconds.

Goal	Example

	`timestamp,channelX,#FLT_STATE#channelY,#FLT_VALUE#channelY,#EXP(.05;.05;50)#channelZ`
	`0,0,0,0,0`
	`100,1,1,100,10`
	`200,0,0,0,0`
	`300,-1,1,100,-10`
	`400,0,0,0,0`
	`500,1,1,100,10`
	
	Note VeriStand runs the channel evaluation in one task and the faulting and stimulation in a separate, parallel task.

Calling a Real-Time Sequence from Another Sequence

Call real-time sequences by reference to run multiple sequences in parallel.

You can add references to real-time sequences or CSV files that you call as sequences, and then call those references from expressions within an existing sequence.

1. Create a real-time sequence to serve as the calling file.
2. In the References pane, select **References**.
3. On the Home tab of the ribbon, click **Reference**.
4. Use the File dialog box to navigate to the real-time sequence (.nivsseq) or CSV file you want to reference.

Note If the real-time sequence or CSV file is saved in the same directory as the sequence you are editing, you can also drag the real-time sequence or CSV file from the Sequences pane to **References** to create the reference.

After you add the reference, you can use an expression to call the referenced real-time sequence.

Generating Errors in a Real-Time Sequence

Configure a real-time sequence to return user-defined error codes and messages in the stimulus profile test results file.

1. Create a real-time sequence.
2. Add a Generate Errors primitive to the sequence code.

> **Note** This primitive also allows you to stop the sequence and skip to the clean-up tasks or to immediately abort the sequence without performing clean-up tasks.

3. If you need to clear or access an error later in the sequence, create an expression in the Property Browser that implements a function.

Goal	Function
Clear the last error so it does not appear in the test results file.	clearlasterror
Return the numeric error code of the last error.	getlasterror

The test results file displays the error information in the section that corresponds to the sequence that generated the error. The following example shows how an error appears in the test results file:
Outcome: Error: 55. Details: <append>========== VeriStand: Alert! The Engine Temperature is outside the critical range. Shutting down the engine.

Adding and Editing Expressions in a Real-Time Sequence

Use expressions to access and act on the variables you define for a real-time sequence.

Expressions are the building blocks of real-time sequence code.

1. Depending on your goal, complete the following tasks.

Goal	Tasks
Create a new, empty expression	Drag an Expression primitive from the Expressions palette to the sequence code.
create an expression that automatically includes a variable.	Drag a declared variable from the Variables pane to the sequence code.
create an expression that automatically includes a sequence	Drag a real-time sequence from the Sequences palette or the References pane to the sequence code.

2. In the Property Browser, build the expression as a mathematical operation.



Note As you begin typing in the Expression field, a type-ahead menu appears with valid functions or variables you can include in the expression. You can connect these functions and variables with operators.

If the new expression contains a variable that is not already declared in the sequence, declare the variable.

Expression Functions

Use functions when editing expressions in the **Property Browser**.



Note All function names must be lowercase.

Function	Description	Return type
abs(x)	Returns the absolute value of x.	Returns a value of the same type as the input.
abstime( )	Returns the current date and time in seconds relative to 12:00 a.m., Friday, January 1, 1904, Universal Time [01-01-1904 00:00:00].	Double

Function	Description	Return type
acos(x)	Returns the inverse cosine of x in radians.	Double
acosh(x)	Returns the inverse hyperbolic cosine of x.	Double
acot(x)	Returns the inverse cotangent of x in radians.	Double
acoth(x)	Returns the inverse hyperbolic cotangent of x.	Double
acsc(x)	Returns the inverse cosecant of x in radians.	Double
acsch(x)	Returns the inverse hyperbolic cosecant of x.	Double
arraysize(x)	Returns the number of elements in x, where x is an array.	Int64
asec(x)	Returns the inverse secant of x in radians.	Double
asech(x)	Returns the inverse hyperbolic secant of x.	Double
asin(x)	Returns the inverse sine of x in radians.	Double
asinh(x)	Returns the inverse hyperbolic sine of x.	Double
atan(x)	Returns the inverse tangent of x in radians.	Double
atan2(y,x)	Returns the arctangent of y/x in radians.	Double
atanh(x)	Returns the inverse hyperbolic tangent of x.	Double
ceil(x)	Rounds x to the next higher integer (smallest integer ≥ x) and returns the rounded value.	Double
clearfault(x)	Clears any fault set on x. x must be a reference to a channel and should not be a reference to a local variable. If x references a	Void

Function	Description	Return type
	local variable, clearfault performs no operation.	
clearlasterror(x)	Clears the last error set by the Generate Error primitive so the error does not appear in the test results file.	Void
cos(x)	Returns the cosine of x, where x is in radians.	Double
cosh(x)	Returns the hyperbolic cosine of x.	Double
cot(x)	Returns the cotangent of x (1/tan(x)), where x is in radians.	Double
coth(x)	Returns the hyperbolic cotangent of x (1/tanh(x)).	Double
csc(x)	Returns the cosecant of x (1/sin(x)), where x is in radians.	Double
csch(x)	Returns the hyperbolic cosecant of x (1/sinh(x)).	Double
deltat( )	Returns the duration of the current system timestep in seconds. To perform equality or comparison operations, use deltatus instead.	Double
deltatus( )	Returns the duration of the current system timestep in microseconds.	Int64
exp(x)	Returns the value of e raised to the x power.	Double
expm1(x)	Returns one less than the value of e raised to the x power ((e^x) – 1).	Double
fault(x,c)	Faults x with a value of c. x must be a reference to a channel and should not be a reference to a local variable. If x references a local variable, fault performs no operation.	Boolean

Function	Description	Return type
fix(x)	Rounds x to the nearest integer between x and zero and returns the rounded value.	Double
floor(x)	Truncates x to the next lower integer (largest integer ≤ x) and returns the truncated value.	Double
getlasterror( )	Returns the numeric error code of the last error set by the Generate Error primitive.	Int32
hypot(x,y)	Returns sqrt((x * x) + (y * y)) without overflowing if x and y are large values.	Double
isnan(x)	Determines whether x is NaN. Returns true if x is NaN. Otherwise, returns false.	Boolean
iteration( )	Returns the number of iterations since the virtual machine started.	Int64
ln(x)	Returns the natural logarithm of x (to the base of e).	Double
lnp1(x)	Returns the natural logarithm of (x + 1).	Double
log(x)	Returns the logarithm of x to the base of 10.	Double
log10(x)	Returns the logarithm of x to the base of 10.	Double
log2(x)	Returns the logarithm of x to the base of 2.	Double
max(x,y)	Compares x and y and returns the larger value.	Returns a value of the same type as the input with the largest data type.
min(x,y)	Compares x and y and returns the smaller value.	Returns a value of the same type as the input with the largest data type.
mod(x,y)	Returns the remainder of x/y, when the quotient is rounded toward –Infinity.	Returns a value of the same type as the input with the largest data type.

Function	Description	Return type
pow(x,y)	Returns x raised to the y power.	Double
pow10(x)	Returns 10 raised to the x power.	Double
pow2(x)	Returns 2 raised to the x power.	Double
quotient(x,y)	Returns floor(x/y), the number of times y evenly divides into x.	Returns a value of the same type as the input with the largest data type.
rand(max)	Returns a floating-point number between 0 and the maximum value.	Double
recip(x)	Returns 1/x.	Double
rem(x,y)	Returns the remainder of x/y, when the quotient is rounded toward zero.	Returns a value of the same type as the input with the largest data type.
round(x)	Rounds x to the nearest integer and returns the rounded value.	Double
sec(x)	Returns the secant of x (1/cos(x)), where x is in radians.	Double
sech(x)	Returns the hyperbolic secant of x (1/cosh(x)).	Double
seqtime( )	Returns the number of elapsed seconds since the virtual machine started. To perform equality or comparison operations, use seqtimeus instead.	Double
seqtimeus( )	Returns the number of elapsed microseconds since the virtual machine started.	Int64
sign(x)	Returns 1 if x is greater than 0, returns 0 if x is equal to 0, and returns –1 if x is less than 0.	Returns a value of the same type as the input.
sin(x)	Returns the sine of x, where x is in radians.	Double
sinh(x)	Returns the hyperbolic sine of x.	Double

Function	Description	Return type
sqrt(x)	Returns the square root of x.	Double
tan(x)	Returns the tangent of x, where x is in radians.	Double
tanh(x)	Returns the hyperbolic tangent of x.	Double
tickcountms( )	Returns the current value of the millisecond counter.	Int64
tickcountus( )	Returns the current value of the microsecond counter.	Int64
ythroot(x,y)	Returns x^(1/y), the y root of x.	Double

Expression Syntax

Use expression syntax similarly to the syntax used in text-based programming languages.
Your expression syntax must conform to the allowed functions and operator precedence in the **Expression** component of the Property Browser.
The expression syntax is summarized below using Backus-Naur Form (BNF notation). The summary includes non-terminal symbols: identifier, number, array-size, floating-point-type, integer-type, left-hand-side, assignment-operator, and function.

> **Note** Italicized symbols are terminal symbols given exactly as how you should reproduce them. The symbol # denotes any number of the term following it.

assignment:

- expression
- left-hand-side assignment-operator assignment

expression:

- expression binary-operator expression
- unary-operator expression
- expression unary-operator

- expression ? expression : expression
- (expression)
- identifier
- constant
- function-name (argument-list)

left-hand-side:

- identifier
- identifier array-subscription

array-subscription:

- [assignment]

assignment-operator: one of

- = += –= *= /= >>= <<= &= ^= |= %= **=

binary-operator: one of

- + – * / ^ != == > < >= <= >> << && || & | % **

unary-operator: one of

- – ! ++ –– ~

argument-list:

- expression
- expression , argument-list

constant:

- pi
- true
- false
- number

non-digit: one of

- _ a~z A~Z

digit: one of

- 0 1 2 3 4 5 6 7 8 9

nonzero-digit: one of

- 1 2 3 4 5 6 7 8 9

binary-digit: one of

- 0 1

hex-digit: one of

- 0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

identifier:

- non-digit [non-first-character]

non-first-character:

- non-digit [non-first-character]
- digit [non-first-character]

number:

- integer-constant
- float-constant

integer-constant:

- decimal-constant
- binary-constant
- hex-constant

decimal-constant:

- nonzero-digit #digit

binary-constant:

- 0b #binary-digit
- 0B #binary-digit

hex-constant:

- 0x #hex-digit
- 0X #hex-digit

float-constant:

- fraction exponent-part
- decimal-constant exponent-part

fraction:

- #digit . digit #digit

exponent-part:

- e [sign] #digit
- E [sign] #digit

sign: one of

- + -

Expression Operator Precedence

Operators in real-time sequence expressions have an order to when they execute. The following table lists the order of precedence for operators from highest to lowest. Operators on the same line have the same precedence.

Operator	Description
**	Exponentiation
-, !, ~, ++, and ––	Unary negation, logical not, bit complement, pre- and post-increment, pre- and post-decrement
*, /, %	Multiplication, division, modulus (remainder)
+ and –	Addition and subtraction
>> and <<	Arithmetic shift right and shift left
>, <, >=, and <=	Greater, less, greater or equal, and less or equal
!= and ==	Inequality and equality
&	Bit and

Operator	Description
^	Bit exclusive or
|	Bit or
&&	Logical and
||	Logical or
? :	Conditional evaluation
= op=	Assignment, shortcut operate and assignop can be +, –, *, /, >>, <<, &, ^, |, %, or **.

The assignment operator = is right associative (groups right to left), as is the exponentiation operator **. All other binary operators are left associative.
The numeric value of TRUE is 1, and FALSE is 0 for output. The logical value of 0 is FALSE, and any nonzero number is TRUE. The logical value of the conditional expression

```
<lexpr> ? <texpr>: <fexpr>
```

is <texpr> if the logical value of <lexpr> is TRUE and <fexpr> otherwise.

Calling a Real-Time Sequence from an Expression

Use expressions to call real-time sequences from within other real-time sequences.

1. Add a reference to the real-time sequence you want to call.
2. Enter the sequence alias, which appears to the right of the file name in the References pane, into the Expression component of the Property Browser.
3. In parentheses following the alias, list each parameter of the sequence. For example, if the alias is Sequence, and the sequence contains the parameters Parameter1 and Parameter2, the expression must contain:

   ```
   Sequence(Parameter1, Parameter2)
   ```

The process for calling CSV files from an expression is the same as for real-time sequence (.nivsseq) files. You must add a reference to the CSV file, and each channel in the CSV file requires a parameter in the function call.

Note If you are evaluating a channel in a CSV file for pass/fail requirements, you must list two parameters in the function call, one for the

evaluated channel, and one for the channel where you want to store the results.

Performing Division with Expression Functions and Operators

Use the division operator or quotient function to divide expression functions and operators.

The **division operator (/)** method divides all numbers as doubles and results in a double quotient, regardless of the data type of the dividend or divisor.

Note If the dividend, divisor, and result are all of type I32, the division operator performs an integer division and returns a result of data type I32.

The **quotient function** method rounds the result of the division towards minus infinity.

Depending on your goal, use a division method with your expression function or operator.

Goal	Method
Perform an integer division and want the result as a double	Division Operator (/)
Perform an integer division and want the result as an integer rounded towards minus infinity	Quotient Function
Divide U64 integers	

The result of the function (operator) is either the data type of the dividend or the divisor. This is determined by the largest data type, according to this order: Double > U64 > I64 > U32 > I32.
If the result is passed to another data type, VeriStand performs an implicit cast. An implicit cast converts one data type to another. For example, if a double is converted to an integer, the resulting number is rounded toward zero. This may lead to unexpected values.

The following table shows the results of different functions in the case of division by zero.

Function/Operator	Double	Integer (I32, I64, U32, U64)
mod(x,y)	Nan	The sequence is aborted with error -8
quotient(x,y)	-Inf/Nan/Inf	
rem(x,y)	Nan	
/	-Inf/Nan/Inf	
%	Nan	

Faulting Channels in a Real-Time Sequence

Use a software fault insertion by using the fault(x,c) and clearfault(x) functions.

A **software fault insertion** allows you to test the behavior of a system when a channel reaches a certain value. Use the following functions to perform a software fault insertion.

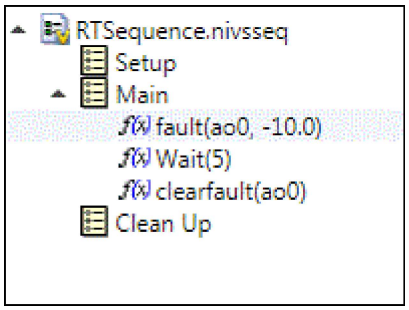
- The **fault(x,c)** function faults the channel referenced by the specified parameter (x) to the specified value (c).

> **Note** While the fault is active, it overrides any attempts to set the value of the faulted channel, whether from mappings or from the VeriStand Editor or Workspace.

- The **clearfault(x)** function clears any faults from the specified channel (x). Once cleared, any mappings resume.

1. Create a real-time sequence.
2. Add expressions that call the fault(x,c) and clearfault(x) functions.

In the following real-time sequence, the parameter ao0 is mapped to the system definition channel PXI FPGA AO0. The real-time sequence faults the value of PXI FPGA AO0 to -10.0, waits five seconds, and then clears the fault.



Variables for Reading and Writing Channels in a Real-Time Sequence

Use parameter and channel reference variables to read or write system definition channels.

There are key differences between how parameters and channel references access channels. Use the following table to determine the best variable type to read or write channels in your real-time sequence.

Variable	Use case	Limitations
Parameter	Use parameters to create real-time sequences that you can use across multiple system definition files.	Parameters are difficult to manage if a stimulus profile contains nested real-time sequences or real-time sequences that access many channels.
Channel Reference	Use channel references if you do not need to use a real-time sequence with multiple system definition files. In general, channel references are easier to manage than parameters.	Channel references bind a real-time sequence to a specific system definition file.

To illustrate how channel references are easier to manager than parameters, consider a real-time sequence, SeqA, that calls another sequence, SeqB. The stimulus profile, MyProfile, manages both sequences. You need to read a channel, named MyChannel, from SeqB.

The following illustration contrasts how reading MyChannel with a channel reference differs from reading MyChannel with a parameter.

Parameters		Channel Reference	
	▪ Add a parameter and assign MyChannel to it. ▪ Add the parameter to the real-time sequence code.		▪ Add a channel reference for MyChannel. ▪ Add the channel reference to the real-time sequence code to read or write the system definition channel. **Note** You do not need to configure the channel reference in SeqA or MyProfile.
	▪ Add a parameter and assign it to MyChannel. ▪ Call SeqB and declare the MyChannel parameter in the real-time sequence call.		
	▪ Configure the parameter assignments so that the appropriate SeqA parameter is assigned to MyChannel.		

Real-Time Sequence Primitives

Use real-time sequence primitives to define variables, create expressions, and add structures such as loops and conditional statements to your real-time sequence code.

A **primitive** is a programming element you can use in a real-time sequence.

Subpalette	Description
Advanced Primitives	Configure advanced operations in the real-time sequence code.
Expressions Primitives	Assign values to and perform operations on variables in a real-time sequence.
Miscellaneous Primitives	Add cosmetic and informational elements to real-time sequence code.
Structures Primitives	Add programming structures, such as loops and conditional statements, to the real-time sequence code.
Variables Primitives	Create and configure variables that a real-time sequence can access and act on.

Advanced Primitives

Configure advanced operations in the real-time sequence code.

Palette object	Description
Yield	A statement that causes the real-time sequence to pause while the Primary Control Loop iterates and then resumes executing the real-time sequence.

Yield Primitive

A statement that causes the real-time sequence to pause while the Primary Control Loop iterates and then resumes executing the real-time sequence.

In a multi-tasking real-time sequence, this statement causes the current task to complete the current step and yield control of the CPU to the next task, if one exists.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Expressions Primitives

Assign values to and perform operations on variables in a real-time sequence.

Palette object	Description
Assignment	A simple expression statement that assigns a value to a variable.
Expression	A statement that can operate on variables in a real-time sequence.

Assignment Primitive

A simple expression statement that assigns a value to a variable.

For example, VarName = 150 is an assignment that sets the value of a variable, VarName, to 150.

Property/Section	Description
Expression	Specifies the expression to evaluate. You can include real-time sequence variables, supported functions, and operators in an expression.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Expression Syntax

A statement that can operate on variables in a real-time sequence.

For example, VarName * 2 is an expression that multiplies the value of a variable, VarName, by 2. Expressions are the building blocks of real-time sequence code, and must follow the expected syntax.

Property/Section	Description
Expression	Specifies the expression to evaluate. You can include real-time sequence variables,

Property/Section	Description
	supported functions, and operators in an expression.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Miscellaneous Primitives

Add cosmetic and informational elements to real-time sequence code.

The primitives on this palette do not affect the execution of sequence code.

Palette object	Description
Block	Organize your real-time sequence code.
Comment	Document in your real-time sequence code.
Generate Error	Create and return a user-defined error code and message in the stimulus profile test results file.

Block

Organize your real-time sequence code.

A **block** is a list of functional statements to execute. Because a block is itself a single statement, you also can use blocks to easily duplicate or move related statements. The predefined sections of a real-time sequence, Setup, Main, and Clean Up, are examples of code blocks. Blocks have no effect on the execution of the code.

Property/Section	Description
Name	The name of the block.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Comment

Document in your real-time sequence code.

Comments have no effect on the execution of the code.

Property/Section	Description
Comment	The text of the comment.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Generate Error

Create and return a user-defined error code and message in the stimulus profile test results file.

You can also configure this primitive to stop the sequence and skip to the clean-up tasks or to immediately abort the sequence without performing clean-up tasks. Only the most recent error appears in the test results file because the most recent error overwrites any previous error that occurred.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.
Error Code	Specifies the numeric value to return.
Message	Specifies a string that you want to append to the **Error Code** in the test results file.
Action to Take	Specifies how to proceed through the remainder of the real-time sequence: ▪ **ContinueSequenceExecution**—Continues normal execution of the real-time sequence. ▪ **StopSequence**—Halts execution and runs the tasks under the **Clean Up** node. ▪ **AbortSequence**—Immediately halts execution without running tasks under the **Clean Up** node.

The test results file displays the error code and message in the section that corresponds to the sequence that generated the error. The following example shows how an error appears in the test results file:

```
Outcome: Error: 55. Details: <append> ========================= NI VeriStand:
Alert! The Engine Temperature is outside the critical range. Shutting down the
engine.
```

Structures Primitives

Add programming structures, such as loops and conditional statements, to the real-time sequence code.

Subpalette	Description
Conditional Statements	Use the Conditional statements to execute different code under different, specified conditions.
Loops	Use Loops to add structures that repeat a section of real-time sequence code a specified number of times or while a specified condition is TRUE.
Multitasking Primitives	Use the Multitasking primitives to divide real-time sequence code into multiple tasks that execute in parallel.

Conditional Statements

Use the Conditional statements to execute different code under different, specified conditions.

Palette object	Description
If Else	A statement that defines an expression to evaluate to determine whether to execute one section of code or another.

Subpalette	Description
Switch Statement Primitives	Use the Switch Statement primitives to create different cases of code to execute based on the value of a test expression.

If Else

A statement that defines an expression to evaluate to determine whether to execute one section of code or another.

When you add an If Else statement to your real-time sequence code, the editor automatically generates **Then** and **Else** blocks under the statement. If the **Test Expression** evaluates to TRUE, the statement executes the code under the **Then** block. If the expression evaluates to FALSE, the statement executes the code under the **Else** block.

To specify the code that each block executes, drag expressions and other primitives to the blocks and configure them as you would any other section of sequence code.

Property/Section	Description
Test Expression	Specifies the expression to evaluate to determine the code to execute.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Switch Statement Primitives

Use the Switch Statement primitives to create different cases of code to execute based on the value of a test expression.

Palette object	Description
Case Statement	A functional statement under which you add code that executes if a test condition is met.
Switch	A statement that defines an expression to evaluate to determine which section of code, or Case Statement, to execute.

Case Statement

A functional statement under which you add code that executes if a test condition is met.

A case statement executes if its **Case Value** matches the value of the **Test Expression** for the Switch statement that contains the case.

To specify the code that the case executes, drag expressions and other primitives to the case and configure them as you would any other section of sequence code.

Property/Section	Description
Case Value	The value to check against the **Test Expression** for the Switch statement. If the two values match, this case executes.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Switch Primitive

A statement that defines an expression to evaluate to determine which section of code, or Case Statement, to execute.

The Switch statement executes the case with the **Case Value** that matches the value of the **Test Expression**. If no cases match the test value, a default case executes instead. When you add a Switch to your real-time sequence code, the editor automatically creates a Cases section for the test cases and a DefaultCase, which is another Case Statement.

Build your test code by adding additional cases to the Cases section. You can then add code to define the execution of each case, including the default.

Property/Section	Description
Test Expression	Specifies the expression to evaluate to determine the code to execute.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Loops

Use Loops to add structures that repeat a section of real-time sequence code a specified number of times or while a specified condition is TRUE.

Palette object	Description
DoWhile Loop	A collection of statements that execute once and then continue executing for as long as the **Repeat While** expression evaluates to TRUE.
For Loop	A collection of statements that execute continuously for a specified number of loop iterations.
ForEach Loop	A collection of statements that executes one time for each element in the array specified by the **Array Expression**.
While Loop	A collection of statements that execute continuously for as long as the **Repeat While** expression evaluates to TRUE.

DoWhile Loop

A collection of statements that execute once and then continue executing for as long as the **Repeat While** expression evaluates to TRUE.

Configure the code that executes in the loop by dragging expressions and other primitives to the loop and configuring them as you would any other section of sequence code.

Property/Section	Description
Repeat While	Specifies the expression to evaluate to determine if the loop continues to execute. The loop executes as long as this expression evaluates to TRUE.

Property/Section	Description
Auto Yield	If TRUE, specifies that the loop automatically yields control of the CPU to the next task at the end of each iteration.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

For Loop

A collection of statements that execute continuously for a specified number of loop iterations.

Configure the code that executes in the loop by dragging expressions and other primitives to the loop and configuring them as you would any other section of sequence code.

Property/Section	Description
Iteration Count	The number of iterations the loop executes.
Loop Variable	Specifies the identifier, or name, for the variable that holds the current iteration count for the loop. You can use this variable in other expressions within the real-time sequence.
Auto Yield	If TRUE, specifies that the loop automatically yields control of the CPU to the next task at the end of each iteration.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

ForEach Loop

A collection of statements that executes one time for each element in the array specified by the **Array Expression**.

Configure the code that executes in the loop by dragging expressions and other primitives to the loop and configuring them as you would any other section of sequence code.

Property/Section	Description
Loop Variable	Specifies the identifier, or name, for the variable that holds the current iteration count for the loop. You can use this variable in other expressions within the real-time sequence.
Array Expression	Defines the array of items that the loop iterates over.
Auto Yield	If TRUE, specifies that the loop automatically yields control of the CPU to the next task at the end of each iteration.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

While Loop

A collection of statements that execute continuously for as long as the **Repeat While** expression evaluates to TRUE.

Configure the code that executes in the loop by dragging expressions and other primitives to the loop and configuring them as you would any other section of sequence code.

Property/Section	Description
Repeat While	Specifies the expression to evaluate to determine if the loop continues to execute. The loop executes as long as this expression evaluates to TRUE.
Auto Yield	If TRUE, specifies that the loop automatically yields control of the CPU to the next task at the end of each iteration.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Multitasking Primitives

Use the Multitasking primitives to divide real-time sequence code into multiple tasks that execute in parallel.

Palette object	Description
MultiTask	A structure that branches real-time sequence code execution into one or more child tasks.
Stop Task	Stops a Task in a MultiTask structure.
Task	A block of code in a MultiTask structure.

MultiTask Structure

A structure that branches real-time sequence code execution into one or more child tasks.

On each time step, the MultiTask structure iteratively executes code from each child task until the task either terminates or yields execution to the next time step. MultiTask structures have two child tasks by default, but you can add more Task primitives from the Multitasking palette.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Stop Task

Stops a Task in a MultiTask structure.

When a task stops, any subsequences executing in that task immediately execute their Clean Up block and any sub-tasks stop.

Property/Section	Description
Task Name	Specify the task name.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Task

A block of code in a MultiTask structure.

On each time step, the MultiTask structure iteratively executes code from each child Task that it contains. A Task cannot exist outside of a MultiTask structure. Configure the code that executes as part of the task by dragging expressions and other primitives to the task and configuring them as you would any other section of sequence code.

Property/Section	Description
Task Name	Specify the task name.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Variables Primitives

Create and configure variables that a real-time sequence can access and act on.

Drag a variable to the Variables pane to configure its properties. You can drag most variables directly to the sequence code to create an expression that sets the value of a variable.

Palette object	Description
Boolean	A Boolean value.
Double	A double-precision, floating point number.
Int32	A 32-bit signed integer.
Int64	A 64-bit signed integer.
UInt32	A 32-bit unsigned integer.
UInt64	A 64-bit unsigned integer.
Void Return Value	Returns void, or no value, when used as the return variable for a real-time sequence.

Subpalette	Description
Array Variables Primitives	Use the Array Variables primitives to create variables that are arrays of values of a certain data type.

Boolean Variable

A Boolean value.

If you want to configure pass/fail notification for a real-time sequence, use a Boolean as the return variable.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter.

Property/Section	Description
	▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Double Variable

A double-precision, floating point number.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling

Property/Section	Description
	sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Int32 Variable

A 32-bit signed integer.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can

Property/Section	Description
	be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Int64 Variable

A 64-bit signed integer.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where

Property/Section	Description
	parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.

Property/Section	Description
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

UInt32 Variable

A 32-bit unsigned integer.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter.

Property/Section	Description
	▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type.  **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile.

Property/Section	Description
	You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

UInt64 Variable

A 64-bit unsigned integer.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling

Property/Section	Description
	sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type.  **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.

Property/Section	Description
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Void Return Value

Returns void, or no value, when used as the return variable for a real-time sequence.



Note You cannot use this variable for anything other than a return variable.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Array Variables Primitives

Use the Array Variables primitives to create variables that are arrays of values of a certain data type.

Palette object	Description
Boolean Array	An array of Boolean values.
Double Array	An array of double-precision, floating point numbers.

Palette object	Description
Imported Double Array	An array of double-precision, floating point numbers imported from a file.
Int32 Array	An array of 32-bit signed integers.
Int64 Array	An array of 64-bit signed integers.
UInt32 Array	An array of 32-bit unsigned integers.
UInt64 Array	An array of 64-bit unsigned integers.

Boolean Array Variable

An array of Boolean values.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter.

Property/Section	Description
	▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Double Array Variable

An array of double-precision, floating point numbers.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling

Property/Section	Description
	sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Imported Double Array Variable

An array of Boolean values.

To create this type of variable, right-click **Local Variables** in the **Variables** pane and select **Import Double Array from File**. In the file dialog box that displays, select a file whose data you want to import. The Import Double Array from File dialog box displays, which you use to select which channels you want to import as local variables, specify how much data to import, and preview the channel data. When you click **OK**, the variable(s) appear in the list of local variables.

Note You cannot add this type of variable from a palette or drag it to the Return Variable or Parameters sections in the Variables pane.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Properties	Includes the following properties that allow you to select an import file or that display information about the data that VeriStand will import from File Path: ▪ **File Path**—Specifies the path of the file from which to import values. You can change the import file after you create the variable; however, you cannot change other properties you set in the Import Double Array from File dialog box when you create the variable, such as the Number of Values ▪ **Channel**—Displays the name of the channel in the import file that contains data you want to import. ▪ **Channel Group**—Displays the name of the group in the import file that owns the **Channel**. ▪ **Subset Start Value**—Displays the index of the first value imported from the file.

Property/Section	Description
	▪ **Number of Values**—Displays the number of values imported from the file, starting at the **Subset Start Value** index. ▪ **Offset**—Displays the amount by which channel values are offset along the y-axis. ▪ **Scale**—Displays the multiplier by which channel values are scaled along the y-axis.
Units	Specifies the units to associate with the variable value. If the channel has associated units in the import file, VeriStand uses those units. You can change the **Units** after you import the local variable.

Int32 Array Variable

An array of 32-bit signed integers.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling

Property/Section	Description
	sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.

Property/Section	Description
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Int64 Array Variable

An array of 64-bit signed integers.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be

Property/Section	Description
	called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.

Property/Section	Description
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

UInt32 Array Variable

An array of 32-bit unsigned integers.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling

Property/Section	Description
	sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

UInt64 Array Variable

An array of 64-bit unsigned integers.

Property/Section	Description
Identifier	Specifies the name of the variable. Use this string to identify the variable in expressions.
Evaluation Method	Specifies whether to evaluate the parameter by value or by reference. **ByReference** is appropriate for most use cases, where parameters map to channels in a system definition, because calling by reference allows changes to the value of the channel to propagate across all sequences that use the parameter. ▪ **ByReference**—When another real-time sequence calls this real-time sequence as a subsequence, the calling sequence operates directly on the mapped variable value. If the subsequence updates the parameter value, the mapped variable in the calling sequence also is updated. If the calling sequence updates the mapped variable value while the subsequence executes from another task, the parameter value in the subsequence updates as well. **ByReference** parameters can only be called by variables of the same data type as the parameter. ▪ **ByValue**—When another real-time sequence calls this real-time sequence as a subsequence, the parameter maps a copy of the variable value. If the calling sequence updates the mapped variable value while the subsequence executes, the parameter value in the subsequence is not affected. If the subsequence modifies the parameter value while the subsequence executes, the value of the mapped variable in the calling sequence is not affected. **ByValue** parameters can

Property/Section	Description
	be called by variables of any logical data type. **Note** This property only appears if you use the variable as a parameter.
Default Assignment	Specifies the default channel in the system definition to assign to this parameter. The real-time sequence uses the **Default Assignment** unless you override the parameter value when you call the real-time sequence from a stimulus profile. You can specify a channel by its alias or by the path to the channel in the system definition, for example: Targets/Controller/System Channels/Model Count This property only appears if you use the variable as a parameter.
Default Value	Specifies the default or initial value of the local variable. This property only appears if you use the variable as a local variable.
Units	Specifies the units to associate with the variable value.
Description	Specifies a description for the current item. This text appears when you hover over the item in the Stimulus Profile Editor.

Viewing Stimulus Profile Test Results

Enable the **Stimulus Profile Editor** to automatically open stimulus profile test results.

Each time you run a stimulus profile, VeriStand produces a test results file that adheres to the Automatic Test Markup Language (ATML) standard.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. Click the **Execution** tab and click **Show Test Result File**.

After a stimulus profile finishes running, the ATML Test Report will open in a web browser.

Alter the appearance of the test report by modifying its style sheet.

Automatic Test Markup Language (ATML) Standard

ATML is a military and aerospace industry standard for sharing data between different components of a test system and supports test program, test asset, and unit under test (UUT) interoperability within an automatic test environment. ATML accomplishes this through a standard XML schema for exchanging UUT test and diagnostic information between components of the test system. ATML specifies standards for test environments that encompass the total product life cycle. ATML defines an integrated set of test-related information that supports the information needs of test environments for testing applications.
ATML standards focus on the following areas:

- Diagnostics
- Instrument Description
- Test Adapter
- Test Configuration
- Test Description
- Test Results and Session Information
- Test Station
- UUT Description

Purpose

ATML is intended to accomplish the following objectives:

- Facilitate the communication, sharing, and reuse of product design and test information for the purpose of testing the product.
- Facilitate test program set (TPS) portability and interoperability.
- Facilitate instrument interchangeability.
- Facilitate the development, integration, and use of test software and test software development tools.

- Support the application of integrated diagnostics.
- Support modular software architectures based upon a framework that supports reusable software products.

Test Results and Session Information

The ATML Test Results and Session Information schema provides the definition for the data collected when you execute a test or tests of a UUT using test procedures in an automated test environment, including the measured values, pass/fail results, and accompanying data, such as test operator, station information, environmental conditions, and so on. ATML Test Results is a component standard of IEEE 1636 Software Interface for Maintenance Information Collection Analysis (SIMICA).

VeriStand generates XML reports that conform to the approved version 6.0.1 of the Test Results and Session Information schema the ATML standard defines. The <Application Data>\Data Storage\ATML directory contains a copy of version 6.0.1 of the schema.

Customizing ATML Test Result Appearance

Switch and edit style sheets to change the ATML report's fonts, colors, tables, and more when it appears in a web browser or external viewer application.

When VeriStand finishes executing a real-time sequence, it generates an XML report that conforms to the approved version of the Test Results and Session Information schema the ATML standard defines. VeriStand applies a style sheet to that report to define how a web browser displays the content of the report. When you open the report in a web browser, the web browser uses this file to transform the XML report into formatted HTML.
VeriStand contains two style sheets you can use. Both template XSL files are located in the <Application Data>VeriStand\Data Storage\ATML directory.

- TRML.xsl—Displays reports in a vertical, tabular, indented format with data for each step in multiple rows. This is the default style sheet.
- TR_Horizontal.xsl—Displays reports in a concise tabular format with expand and collapse sections.

To switch between these style sheets, use the following steps.

1. In the <Application Data>\NI VeriStand\Data Storage\ATML directory, copy the TR_Horizontal.xsl and paste it in the same directory as the XML report.
2. Open the XML report in a text editor.
3. At the top of the XML report, change the code that reads <?xml-stylesheet type="text/xsl" href="TRML.xsl"?> to <?xml-stylesheet type="text/xsl" href="TR_Horizontal.xsl"?>.
4. Save the XML report.

Open the XML report in a web browser to view the changes.

You can edit the style sheet in a text editor to further modify the report's appearance.

Logging Real-Time Test Data with the Stimulus Profile Editor

Use stimulus profiles to log real-time test data to the host computer while a test executes on a target.

Before you begin, you should familiarize yourself with the Stimulus Profile Editor environment.

Logging test data enables you to easily review and save the responses of a unit under test (UUT) to specific scenarios. VeriStand saves logged data in the TDMS file format, which you can later view and analyze using the TDMS File Viewer workspace tool, other National Instruments software such as NI DIAdem, or Microsoft Excel. The Stimulus Profile Editor provides support for both triggered and segmented logging to help you manage large data sets and long test scenarios.

1. Create a stimulus profile that calls a real-time sequence.
2. Add a Start Logging step to the stimulus profile, before the **Real-Time Sequence Call** step.

   Note The step appears as **Start Logging Configuration** with a Channel Group substep.

3. Click **Start Logging Configuration** in the stimulus profile code and use the **Property Browser** to configure the following properties.

Property	Description
Configuration Name	The name you want to use to start and stop logging.
File Path	The name and location for a resulting log file.
Timestamp Filename	Whether to append the start time of the logging operation to the name of the log file.
Replace Existing File	Whether to replace an existing file with the same filename. If you disable this property, the Stimulus Profile Editor appends any new log data to the existing file. This property performs a basic append, so you will need to use the channel data in the final file to determine where new data is appended.  **Note** Consider adding a time channel to your log to easily identify breaks in data logging.
Log Rate [Hz]	The Stimulus Profile Editor logs data at the closest possible rate to this value without exceeding the rate at which the target produces data.
Triggered Logging	Configure trigger conditions to specify when data logging starts and stop. If you do not configure triggers, the **Stimulus Profile Editor** continuously logs all specified channel data beginning when the Start Logging step executes.

4. Click the **Channel Group** step and configure the following properties.

Property	Description
Channel Group Name	The name of the channel group used in the TDMS file.
Channels	Adds channels or aliases to the channel group.

5. If you want to log data in multiple channel groups, add additional Channel Group steps under **Start Logging Configuration**.
6. Add a Stop Logging step after the **Real-Time Sequence Call** step, and set the **Configuration Name** to the name you specified in the **Start Logging** step.
7. Save and run the stimulus profile.

The Stimulus Profile Editor logs data on channels using the triggers and file segmenting you specified.

Configuring Triggered Logging in Stimulus Profiles

Use triggered logging in stimulus profiles if you want to see channel data under certain conditions, such as those that you might expect to cause the unit under test (UUT) to fail.

When you use the Stimulus Profile Editor to log real-time test data, you can configure start and stop triggers for logging. The editor logs channel data while the start trigger condition is TRUE and the stop trigger condition is FALSE.
You can also configure pre-triggered and post-triggered data logging. This saves the channel data immediately before a start trigger and after a stop trigger. For example, if the start trigger represents a fault in the system, and the stop trigger represents a return to expected values, it can be useful to see the behavior of the system that lead to the fault and how well the system recovers.

1. Create a stimulus profile that logs real-time test data.
2. Select the Start Logging step for a logging configuration.
3. In the Property Browser, set the **Trigger Channel** to a channel you want to watch for logging conditions.

   Note This can be any channel in the system definition associated with the stimulus profile.

4. Set the **Trigger Condition** and the corresponding **High Limit** and **Low Limit** values.

Note You can set the **Trigger Condition** to **in_limits** or **out_of_limits** to configure logging to start and stop when the value of **Trigger Channel** either enters or leaves the value range specified by the limits. For example, if you set **Trigger Condition** to **in_limits**, the stimulus profile registers a start trigger when the value of Trigger Channel is greater than or equal to the Low Limit and less than or equal to the High Limit. It registers a stop trigger, and stops logging, when the value of **Trigger Channel** leaves this window.

5. Depending on your desired result, configure the step to handle a situation where multiple start triggers occur.

Desired Result	Configuration
One file containing all logged data.	Disable **Replace Existing File**.
One file containing only data from the last occurrence of a start trigger.	Enable **Replace Existing File**.
Separate files for each set of logged data.	a. Enable **Timestamp Filename**. **Note** You must enable timestamps on segmented filenames to avoid filename conflicts. b. Set **Segment Options** to **OnStartTrigger**

6. If you want to specify an amount of channel data to log immediately before a start trigger occurs, set a value for **Pre-Trigger Duration**.

Note When you execute the profile, the VeriStand Gateway maintains a buffer that always contains this amount of data.

7. If you want to specify an amount of channel data to log after a stop trigger occurs, set a value for **Post-Trigger Duration**.

8. Click **Save**.

Communicating with the VeriStand Editor Using Stimulus Profile Arguments

Send commands to the **VeriStand Editor** through a stimulus profile to specify a VeriStand project, a system definition, VeriStand Gateway IP address, or connect to a target from the system definition.

Depending on your goal, use an argument to send a command to the VeriStand Editor.

Goal	Argument	Example
Specify the VeriStand project to run.	/nivsprj	/nivsprj "<Common Data>\Examples\Stimulus Profile\Engine Demo\Engine Demo.nivsprj"
Specify the system definition file to use.	/sysdef	/sysdef "<Common Data>\Examples\Stimulus Profile\Engine Demo\Engine Demo.nivssdf"
Specify the IP address of the VeriStand Gateway.	gateway	/gateway 10.0.38.64
Tell the VeriStand editor to connect to the target defined by the system definition file.	/connect	/connect

Getting Started with the Stimulus Profile Editor Tutorial

Use tutorial examples to become familiar with the **Stimulus Profile Editor**.

Before you begin, learn how to navigate the stimulus profile editor.

1. Set up a basic test—Create a real-time sequence that turns on a car engine, sets the engine speed to 2500 RPM, holds this speed for 20 seconds, and then turns the engine off.
    a. Reading and writing channels directly from a real-time sequence—Use channel references, parameters, and variables to turn on a car engine,

set the engine speed to three different RPM values, measure how long the engine takes to settle at each specified RPM, and turn the engine off.

2. Execute multiple parallel tasks—Create a stimulus profile and a real-time sequence that incorporates multitasking, which executes multiple parallel tasks.
3. Configure failure notification—Modify a real-time test by adding a return value that reflects the success or failure of a warm-up task.
4. Log data to a file—Log channel data from a running stimulus test to a TDMS file.
5. Call a CSV file as a real-time sequence—Use Comma Separated Values (.csv) files within stimulus profiles to stimulate, fault, and evaluate channels.
6. Play back previously recorded test data—Create and run a stimulus profile that plays back data from a previously recorded VeriStand macro file.
7. Update model parameter values during test execution—Call text files in a stimulus profile to update the values of engine model parameters while a test is running.

You can find complete examples of these tutorials in the <Common Data>\Examples\Stimulus Profile\Engine Demo\Stimulus Profiles directory.

Deploying the Engine Demo

Deploy the engine demo's system definition before running a stimulus profile.

1. Launch VeriStand and double-click **Engine Demo**.
2. In the Engine Demo dialog box, click **Create**.
3. Select **Operate** » **Deploy** and wait for the system definition to deploy.

Setting up a Basic Stimulus Profile Editor Test

Create a real-time sequence that turns on a car engine, sets the engine speed to 2500 RPM, holds this speed for 20 seconds, and then turns the engine off.

1. Deploy the Engine Demo—Deploy the engine demo's system definition before running a stimulus profile.

2. Create a real-time sequence—Create a real-time sequence that starts, stalls, and stops the engine demo.
3. Create a stimulus profile—Configure a stimulus profile to execute a real-time sequence.
4. Run the stimulus profile—Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

Creating a Basic Real-Time Sequence

Create a real-time sequence that starts, stalls, and stops the engine demo.

Before you begin, deploy the engine demo's system definition.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. In the Stimulus Profile Editor, click **New Real-Time Sequence**.
3. Save the sequence as Engine Demo Basics tutorial.nivsseq in the <Common Data> \VeriStand Projects\Engine Demo\Stimulus Profiles\Basic Engine Demo directory.
4. Add blocks to organize your code.
    a. In the Primitives palette, expand **Miscellaneous** and drag a **Block** into the Setup section of the real-time sequence.
    b. In the Property Browser Name field, enter Turn on engine.
    c. Add a block to the Main section and name it Set engine speed to 2500 and wait.
    d. Add a block to the Clean Up section and name it Turn off engine.
5. Create variables.

> **Note** For an example of how to use channel references to access channels, refer to Reading and Writing Channels Directly from a Real-Time Sequence Tutorial.

    a. In the Primitives palette, expand Variables and drag a **Boolean** into the Parameters section of the Variables pane.

b. In the Property Browser next to Default Assignment, click **Browse** to display the system definition channel tree.

c. Click **View aliases** to display the aliases defined in the system definition.

d. Double-click **EnginePower** to assign this alias to the parameter.

e. In **Identifier**, enter EnginePower.

f. In **Units**, enter On/Off.

g. Select **Double** from the Primitives palette and drag it to Parameters to add a double-precision numeric parameter after EnginePower.

h. In the Property Browser, name the new parameter DesiredRPM, map the parameter to the corresponding alias, and enter the units as RPM.

6. Add variables to the sequence code and set their values.

a. In the Variables pane, drag **EnginePower** into the **Turn on engine** block.

b. In the Property Browser, edit the **Expression** to EnginePower = true. This value will turn the engine on when the block executes.

c. Drag **DesiredRPM** into the **Set engine speed to 2500 and wait** block.

d. Set **DesiredRPM** to 2500.

e. Press <Ctrl> and drag **DesiredRPM** from the **Set engine speed to 2500 and wait** block into the **Turn off engine** block.

f. Set **DesiredRPM** to 0.

g. In the Variables pane, right-click EnginePower and select **Copy**.

h. Right-click **Turn off engine** and select **Paste**.
Leave the value as false to turn the engine off when the block executes.

7. Set the expression to wait before it executes.

a. In the Sequences palette, expand **Real-Time Sequence Library** » **Standard** » **Timing** and drag **Wait** into the **Set engine speed to 2500 and wait** block.

b. In the Property Browser, edit the Expression to replace **Duration** with 20.
   This will hold the **DesiredRPM** value at 2500 for 20 seconds when this step executes.

8. Set a return value.

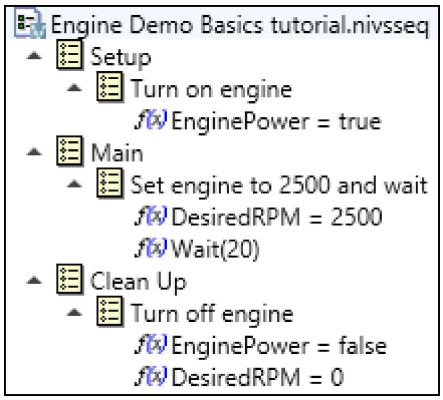
   a. In the Primitives palette, expand Variables and drag **Void Return Value** into the Return Variable section of the Variables pane.

   > **Note** By default, the return value node is named Pass.

   b. In the Property Browser, enter the **Identifier** as Output to change the name of the return value node.

9. **Save** the real-time sequence.

The real-time sequence code will look like the following image.



After creating this real-time sequence code, you must add it to a stimulus profile.

Creating a Basic Stimulus Profile

Configure a stimulus profile to execute a real-time sequence.

Before you begin, create a real-time sequence for the engine demo.

1. In the Stimulus Profile Editor, click the **Start Page** tab.
2. Click **New Stimulus Profile**.

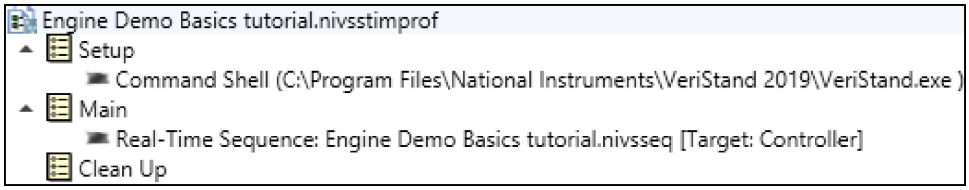
3. Save the stimulus profile as Engine Demo Basics tutorial.nivsstimprof in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Basic Engine Demo directory.
4. Add a step to launch the VeriStand Editor.
    a. In the Steps palette, expand **Other** and drag **Command Shell** into **Setup**.
    b. In the Property Browser, specify the full path to VeriStand.exe as the **Filename**.

    > **Note** You can add arguments in the Property Browser for this step to specify the VeriStand project, system definition file, and VeriStand gateway IP address to connect to.

    When the stimulus profile runs, this step opens the VeriStand Editor or Workspace so you can watch the stimulus profile execute.
5. In the Steps palette, expand **Real-Time Sequences** and drag **Real-Time Sequence Call** into **Main**.
   The Real-Time Sequence Call calls the real-time sequence you specify in the Property Browser.
6. Select the **Real-Time Sequence Call** step in the stimulus profile code to specify the real-time sequence to call.
   The Property Browser displays several properties you can use to configure the step. Each step in a stimulus profile contains properties. These properties are editable attributes that determine how the step executes.
7. In the Property Browser, browse the **File Path** to the real-time sequence you created.
   The Parameters section of the Property Browser displays the parameters in the called real-time sequence as well the channels assigned to them. If you use the parameters to read or write system definition channels, you must assign those channels to the parameters in the stimulus profile. When you add a real-time sequence call, the stimulus profiles uses the default assignment for each parameter.

8. Click the **Target Name** pull-down and select **Controller** to execute the real-time sequence on the Controller target.
9. Click the **Pass Fail** pull-down and select **AlwaysPass**.
10. **Save** the stimulus profile.

The stimulus profile code will look like the following image.



After creating the stimulus profile, run it.

Reading and Writing Channels Directly from a Real-Time Sequence Tutorial

Use channel references, parameters, and variables to turn on a car engine, set the engine speed to three different RPM values, measure how long the engine takes to settle at each specified RPM, and turn the engine off.

You can use channel references to read/write channels in a system definition directly from a real-time sequence. Unlike a parameter, which you must assign to a system definition channel, a channel reference automatically binds to a channel when you add the channel reference to a real-time sequence. This makes channel references easier to manage than parameters in stimulus profiles that access many channels, especially when accessing channels in nested sequences.

Note Channel references bind to specific system definition channels, and therefore a real-time sequence that contains channel references can only be used with the system definition file that contains those channels.

1. Deploy the Engine Demo—Deploy the engine demo's system definition before running a stimulus profile.
2. Create a real-time sequence with a channel reference—Create a real-time sequence that uses a channel reference to read/write a engine power channel and a parameter to specify the value to read/write.

3. Create a real-time sequence with channel references and local variables—Create a real-time sequence that uses channel references to measure how long the engine takes to settle at a specified RPM.
4. Create a stimulus profile that calls a channel referencing sequence—Configure a stimulus profile to turn on the engine and measure how long it takes the engine to settle at various RPMs.
5. Run the stimulus profile—Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

Create a Real-Time Sequence with a Channel Reference

Create a real-time sequence that uses a channel reference to read/write a engine power channel and a parameter to specify the value to read/write.

Before you begin, deploy the engine demo's system definition.

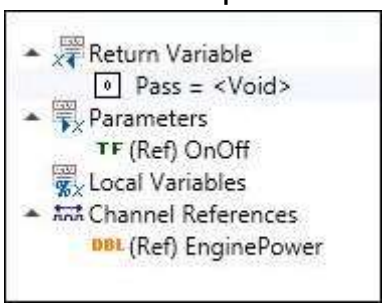
1. In the VeriStand Editor, click **Tool Launcher**  » **Stimulus Profile Editor**.
2. In the Stimulus Profile Editor, click **New Real-Time Sequence**.
3. Save the sequence as Set Engine Power tutorial in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Channel References directory.
4. Create variables.
    a. In the Primitives palette, expand Variables and drag a **Boolean** into the Parameters section of the Variables pane.
    b. In the Property Browser, enter the **Identifier**, as OnOff.
    c. Click the **Evaluation Method** pull-down, select **ByReference**.
    This parameter specifies whether to turn the engine on or off. You set the value when you call this sequence from the stimulus profile.
    d. In the Variables pane, right-click **Channel References** and click **Insert Channels**.
    e. In the Select channels dialog box, expand **Aliases**, enable **EnginePower**, and click **OK**.

This channel reference writes the value of the OnOff parameter to the EnginePower channel.

f. In the Primitives palette under Variables, drag **Void Return Value** into the Return Variable section of the Variables pane.
This variable returns no value. Instead, you call this sequence from a stimulus profile to turn the engine on and off.

> **Note** By default, the return value node is named Pass.

The Variables pane will look like the following image.



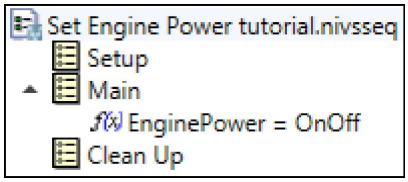
5. Configure the real-time sequence to turn the engine on or off based on the value of the OnOff parameter.

   a. In the Variables pane, drag **EnginePower** into Main.

   b. In the Property Browser, enter the **Expression** as EnginePower = OnOff.
   This allows you to toggle the engine on or off by calling this sequence from a stimulus profile.

6. **Save** the real-time sequence.

The real-time sequence code will look like the following image.



After creating this real-time sequence, create another real-time sequence.

Create a Real-Time Sequence with Channel References and Local Variables

Create a real-time sequence that uses channel references to measure how long the engine takes to settle at a specified RPM.

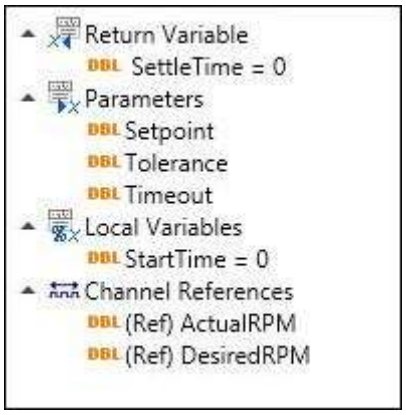
Before you begin, create a real-time sequence with a channel reference.

1. In the Stimulus Profile Editor, click the **Start Page** tab.
2. Click **New Real-Time Sequence**.
3. Save the sequence as Measure Set Preference tutorial in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Channel References directory.
4. Create variables.
    a. In the Primitives palette, expand Variables, drag a **Double** into the Return Variable section of the Variables pane, and enter its **Identifier** as SettleTime
    b. Create and configure the following Double parameters and local variable.

Variable Type	Name	Evaluation Method	Units
Parameter	Setpoint	ByValue	rpm
Parameter	Tolerance	ByValue	rpm
Parameter	Timeout	ByValue	s
Local variable	StartTime	ByValue	s

    c. In the Variables pane, right-click **Channel References** and click **Insert Channels**.
    d. In the Select channels dialog box, expand **Aliases**, enable **ActualRPM** and **DesiredRPM**, and click **OK**.

The Variables pane will look like the following image.



5. Add a setup expression for the set point of the engine.
    a. In the Primitives pane, expand **Miscellaneous** and drag **Comment** into **Setup**.
    b. In the Property Browser, enter the **Comment** as Set the desired set point for the engine.
    c. In the Variables pane, drag **DesiredRPM** into **Setup**.
    d. In the Property Browser, enter the **Expression** as DesiredRPM = Setpoint.

    The Setup block of the sequence code uses the DesiredRPM channel reference to change the value of the DesiredRPM channel to the specified Setpoint. You specify the value of the Setpoint parameter when you call this sequence from the stimulus profile. By configuring the real-time sequence this way, you can call this sequence from the stimulus profile to measure the settle time of various set points.
6. Add main expressions to measure how long it takes for the engine to settle at its RPM set point.
    a. In the Primitives pane, drag **Comment** into **Main**.
    b. In the Property Browser, enter the **Comment** as Measure how long it takes for the engine to settle at its RPM set point.
    c. In the Variables pane, drag **StartTime** into **Main**.
    d. In the Property Browser, enter the **Expression** as StartTime = abstime().

e. In the Sequences pane, expand **Real-Tim Sequence Library** » **Standard** » **Timing** and drag **WaitUntilSettled** into **Main**.

f. In the Property Browser, enter the **Expression** as WaitUntilSettled(ActualRPM, DesiredRPM + Tolerance, DesiredRPM - Tolerance, 1.0, Timeout).

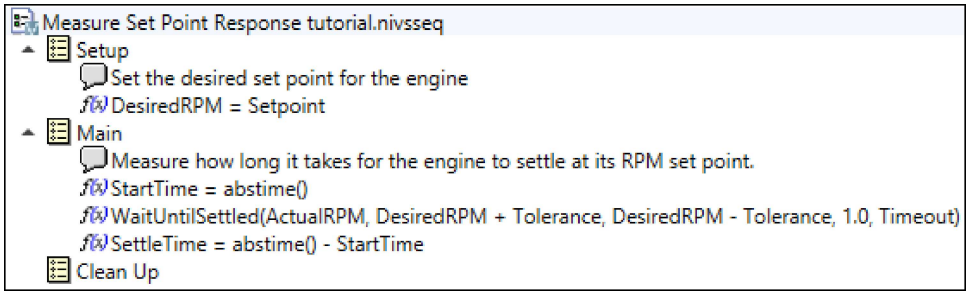
g. In the Variables pane, drag **SettleTime** into **Main**.

h. In the Property Browser, enter the **Expression** as SettleTime = abstime() - StartTime.

The Main block of the sequence stores the absolute time at which the sequence starts to the StartTime local variable. The code waits for the RPM, as read from the ActualRPM channel, to settle into a range between the DesiredRPM plus or minus the specified Tolerance for at least one second or until it reaches the specified Timeout. When the RPM either settles or times out, the code returns how long it took the engine to settle at the specified RPM.

7. **Save** the real-time sequence.

The real-time sequence code will look like the following image.



After creating this real-time sequence, add it to a stimulus profile.

Creating a Stimulus Profile that Calls a Channel Referencing Sequence

Configure a stimulus profile to turn on the engine and measure how long it takes the engine to settle at various RPMs.

Before you begin, create a real-time sequence with channel references and local variables.

1. In the Stimulus Profile Editor, click the **Start Page** tab.
2. Click **New Stimulus Profile**.
3. Save the sequence as Test Engine Set Points tutorial in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Channel References directory.
4. Add a setup step to call the sequence to turn on the engine.
    a. In the Steps palette, expand **Real-Time Sequences** and drag **Real-Time Sequence Call** into **Setup**.
    b. In the Property Browser, navigate to the Set Engine Power real-time sequence as the **Filename**.
    c. Under **Parameters**, click the **OnOff** pull-down and select True.

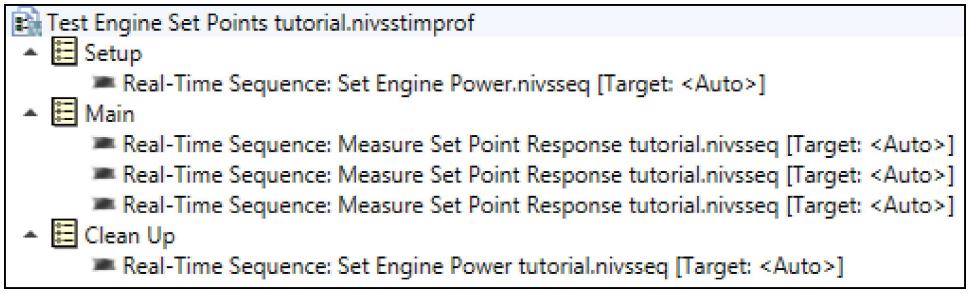
    > **Note** You do not need to configure any channel mappings from the stimulus profile. The real-time sequence uses a channel reference to change the value of the EnginePower channel to the value specified by the OnOff parameter.

5. Add main steps.
    a. Add a **Real-Time Sequence Call** into **Main**.
    b. In the Property Browser, navigate to the Measure Set Point Response real-time sequence as the **Filename**.
    c. Under **Parameters**, enter the **Setpoint** as 2500, **Timeout** as 60, and **Tolerance** as 100.
    d. Add another **Real-Time Sequence Call** to the Measure Set Point Response real-time sequence and enter the **Setpoint** as 6000, **Timeout** as 60, and **Tolerance** as 100.

e. Add another **Real-Time Sequence Call** to the Measure Set Point Response real-time sequence and enter the **Setpoint** as 3000, **Timeout** as 60, and **Tolerance** as 100.

6. Add clean up steps.
   a. Add a **Real-Time Sequence Call** into **Clean Up**.
   b. In the Property Browser, navigate to the Set Engine Power real-time sequence as the **Filename**.
   c. Under **Parameters**, click the **OnOff** pull-down and select False.
7. **Save** the stimulus profile.

The stimulus profile code will look like the following image.



After creating the stimulus profile, run it.

Executing Multiple Parallel Tasks Using the Stimulus Profile Editor Tutorial

Create a stimulus profile and a real-time sequence that incorporates multitasking, which executes multiple parallel tasks.

This real-time test turns on a car engine, sets the engine speed to 2500 RPM, and holds this speed for 25 seconds. It then raises the engine speed to 8000 RPM and holds this speed for another 25 seconds. In addition to controlling the engine speed, the sequence monitors engine temperature. If the engine temperature exceeds 110 degrees, the engine shuts down and the sequence aborts.

1. Deploy the Engine Demo—Deploy the engine demo's system definition before running a stimulus profile.

2. Create a multitasking real-time sequence—Create a real-time sequence that will warm-up and monitor the demo engine in separate tasks.
3. Create a stimulus profile—Configure a stimulus profile to execute a multitasking real-time sequence.
4. Run the stimulus profile—Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

Creating a Multitasking Real-Time Sequence

Create a real-time sequence that will warm-up and monitor the demo engine in separate tasks.

Before you begin, you need to deploy the Engine Demo's system definition.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. In the Stimulus Profile Editor, click **New Real-Time Sequence**.
3. Save the sequence as Engine Demo Advanced tutorial in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Advanced directory.
4. Add blocks to organize your code.
    a. In the Primitives palette, expand **Miscellaneous** and drag a **Block** into the Setup section of the real-time sequence.
    b. In the Property Browser Name field, enter Turn on engine.
    c. Add a block to the Main section and name it Set engine speed to 2500 and wait.
    d. Add a block to the Clean Up section and name it Turn off engine.
5. Create variables.
    a. In the Primitives palette, expand Variables and drag a **Boolean** into the Parameters section of the Variables pane.
    b. In the Property Browser next to Default Assignment, click **Browse** to display the system definition channel tree.
    c. Click **View aliases** to display the aliases defined in the system definition.

d. Double-click **EnginePower** to assign this alias to the parameter.

e. In **Identifier**, enter EnginePower.

f. In **Units**, enter On/Off.

g. Select **Double** from the Primitives palette and drag it to Parameters to add a double-precision numeric parameter after EnginePower.

h. In the Property Browser, name the new parameter DesiredRPM, map the parameter to the corresponding alias, and enter the units as RPM.

i. Add another Double primitive after **DesiredRPM**, name it ActualRPM, and map it to the corresponding alias.

j. Add another Double primitive after **ActualRPM**, name it EngineTemp, and map it to the corresponding alias.

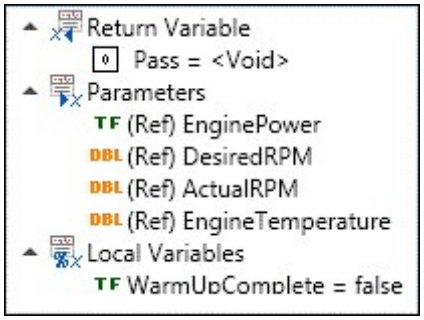
k. Drag a Boolean primitive into the Local Variables section of the Variables pane and name it WarmUpComplete.

> **Note** Local variables are variables you use within the real-time sequence as a way to hold values you get or set in statements.

l. In the Primitives palette under Variables, drag **Void Return Value** into the Return Variable section of the Variables pane.

> **Note** By default, the return value node is named Pass.

The Variables pane will look like the following image.



6. Add variables to the sequence code and set their values.

a. In the Variables pane, drag **EnginePower** into the **Turn on engine** block.

b. In the Property Browser, edit the **Expression** to EnginePower = true. This value will turn the engine on when the block executes.

c. Drag **WarmUpComplete** into **Setup** so that this variable is at the same level in the tree as **Turn on engine**.
Adding WarmUpComplete to Setup initializes the variable to a known value so that the real-time sequence can then write values to the variable during execution.

d. Drag **EnginePower** into **Turn off engine**.
Leave the value as false to turn off the engine when the block executes.

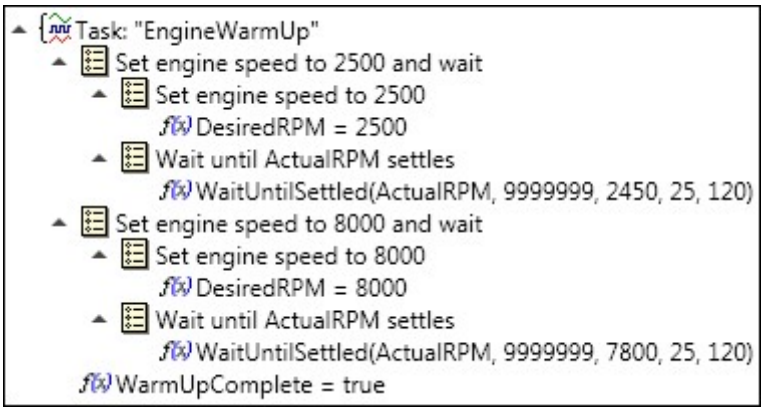
e. Drag **DesiredRPM** into **Turn off engine**.
Leave the value as 0 to decrease the RPM to 0 when the block executes.

7. Enable Multitasking

a. In the Primitives palette, expand **Structures** » **Multitasking** and drag **MultiTask** into **Main**.

b. In the Property Browser, rename **Task1** as EngineWarmUp and **Task2** as MonitorEngineTemperature.

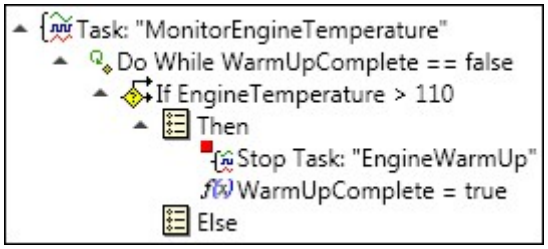
c. Modify the **EngineWarmUp** task by adding the following code.



Note In the Sequences palette, expand **Real-Time Sequence Library** » **Standard** » **Timing** to find **WaitUntilSettled**.

The EngineWarmUp task sets the engine speed (DesiredRPM) to 2500 RPM, and then waits until the RPM (ActualRPM) settles into a range between 2450 and 9999999 RPM for 25 seconds. Then the task will raise the engine speed to 8000 RPM and wait until the RPM settles into a range between 7800 and 9999999 RPM for another 25 seconds. When the task successfully completes, the task sets the WarmUpComplete variable to true for the MonitorEngineTemperature task.

8. Modify the **MonitorEngineTemperature** task with the following code.

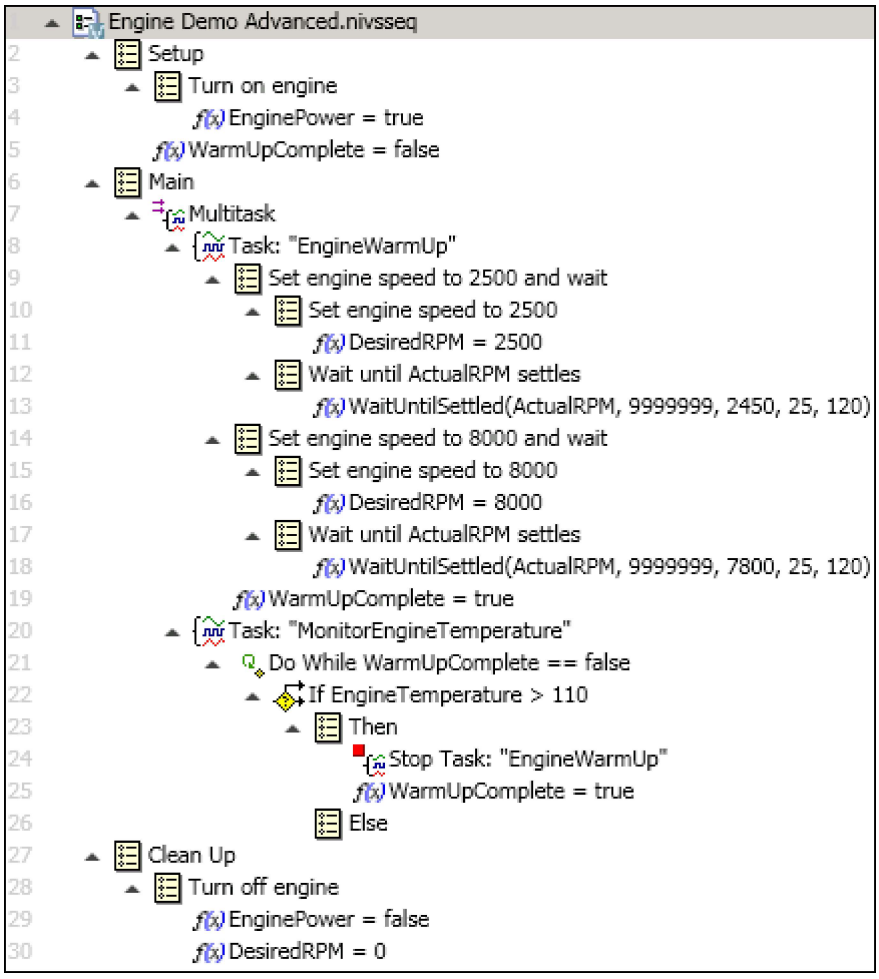


Note In the Primitives palette, expand **Structures** » **Loops** to find the **DoWhile Loop** and **Structures** » **Conditional** to find **If Else**.

As long as the WarmUpComplete variable is false, the MonitorEngineTemperature task monitors the engine temperature (EngineTemperature) to ensure it does not exceed 110 degrees. If the engine temperature exceeds 110 degrees, the engine shuts down and the sequence aborts. If the engine temperature remains below 110 degrees, the sequence completes its execution and turns off the engine.

9. **Save** the real-time sequence.

The real-time sequence code will look like the following image.



After creating this real-time sequence, add it to a stimulus profile.

Creating a Multitasking Stimulus Profile

Configure a stimulus profile to execute a multitasking real-time sequence.

Before you begin, create a real-time sequence that performs multitasking.

1. In the Stimulus Profile Editor, click the **Start Page** tab.
2. Click **New Stimulus Profile**.
3. Save the stimulus profile as Engine Demo Advanced tutorial in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Advanced directory.
4. Add a step to launch the VeriStand Editor.

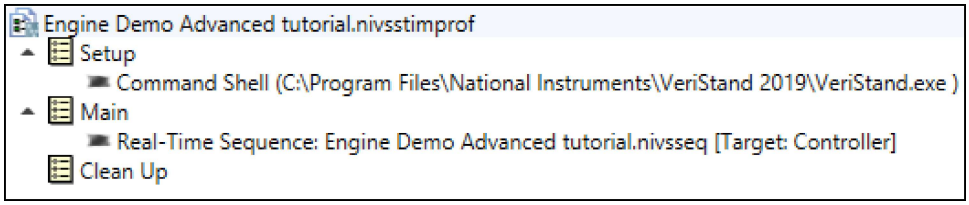
a. In the Steps palette, expand **Other** and drag **Command Shell** into **Setup**.
b. In the Property Browser, specify the full path to VeriStand.exe as the **Filename**.

> **Note** You can add arguments in the Property Browser for this step to specify the VeriStand project, system definition file, and VeriStand gateway IP address to connect to.

When the stimulus profile runs, this step opens the VeriStand Editor or Workspace so you can watch the stimulus profile execute.

5. Create a step to call the sequence you created.
   a. In the Steps palette, expand **Real-Time Sequences** and drag **Real-Time Sequence Call** into **Main**.
   b. Select the **Real-Time Sequence Call** step in the stimulus profile code to specify the real-time sequence to call.
   c. In the Property Browser, browse the **File Path** to the real-time sequence you created.
   d. Click the **Target Name** pull-down and select **Controller** to execute the real-time sequence on the Controller target.
6. **Save** the stimulus profile.

The stimulus profile code will look like the following image.



After creating the stimulus profile, run it.

Configuring Failure Notification Using the Stimulus Profile Editor Tutorial

Modify a real-time test by adding a return value that reflects the success or failure of a warm-up task.

If the engine temperature does not exceed 110 degrees, the warm-up task completes and the sequence returns true, which indicates that the warm-up task succeeded. If the engine temperature exceeds 110 degrees, the engine shuts down and the return value is false, which indicates that the warm-up task failed.

1. Deploy the Engine Demo—Deploy the engine demo's system definition before running a stimulus profile.
2. Create a real-time sequence to return a pass/fail value—Update an existing sequence to pass a Boolean return value.
3. Create a stimulus profile to execute after a step fails—Define the actions a stimulus profile performs to stop execution after a step fails.
4. Run the stimulus profile—Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

When the Message Box step executes in the stimulus profile, a dialog box with the message you specified appears in the Stimulus Profile Editor.

Creating a Real-Time Sequence to Return a Pass/Fail Value

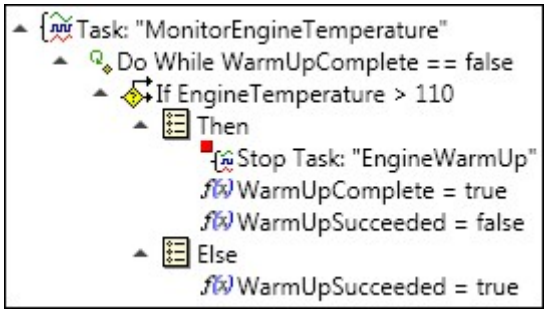
Update an existing sequence to pass a Boolean return value.

Before you begin, deploy the engine demo's system definition.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. Click **File** » **Open**, navigate to <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Advanced, and double-click the Engine Demo Advanced sequence.
3. Save the sequence as Engine Demo Return Value tutorial in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Return Value directory.
4. In the Primitives palette, expand **Variables** and drag **Boolean** to the Variables pane under **Return Value** to change the value from a Void Return Value.

5. In the Property Browser, change the **Identifier** to WarmUpSucceeded.
6. From the Variables pane, drag **WarmUpSucceeded** into the sequence code under **MonitorEngineTemperature**, inside **Then**.
   If the Then section of the code executes, the return value will be false. This indicates that the warm-up task failed because the engine temperature exceeded 110 degrees.
7. Drag another **WarmUpSucceeded** into the sequence code under **MonitorEngineTemperature**, inside **Else**, and modify it in the Property Browser to be true.
   If the Else section of the code executes, the return value will be true. This indicates that the warm-up task succeeded because the engine temperature remained below 110 degrees.
8. **Save** the real-time sequence.

The altered real-time sequence code will look like the following image.



After creating this real-time sequence, add it to a stimulus profile.

Creating a Stimulus Profile with Execution Behavior after Step Failure

Define the actions a stimulus profile performs to stop execution after a step fails.

Before you begin, create a real-time sequence that returns a pass/fail value.

1. In the Stimulus Profile Editor, click the **Start Page** tab.
2. Click **New Stimulus Profile**.
3. Save the stimulus profile as Engine Demo Return Value tutorial in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Return Value directory.

4. Add a step to launch the VeriStand Editor.
    a. In the Steps palette, expand **Other** and drag **Command Shell** into **Setup**.
    b. In the Property Browser, specify the full path to VeriStand.exe as the **Filename**.

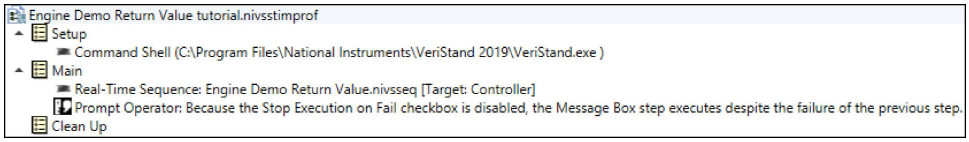
> **Note** You can add arguments in the Property Browser for this step to specify the VeriStand project, system definition file, and VeriStand gateway IP address to connect to.

    When the stimulus profile runs, this step opens the VeriStand Editor or Workspace so you can watch the stimulus profile execute.
5. Create a step to call the sequence you created.
    a. In the Steps palette, expand **Real-Time Sequences** and drag **Real-Time Sequence Call** into **Main**.
    b. Select the **Real-Time Sequence Call** step in the stimulus profile code to specify the real-time sequence to call.
    c. In the Property Browser, browse the **File Path** to the real-time sequence you created.
    d. Click the **Target Name** pull-down and select **Controller** to execute the real-time sequence on the Controller target.
6. Create pop-up dialog box.
    a. In the Steps palette, expand the **Other** and drag **Message Box** into Main.
    b. In the Property Browser, enter the following into **Message**:
    Because the Stop Execution on Fail checkbox is disabled, the Message Box step executes despite the failure of the previous step.
    c. Enter the following text into **Dialog Title**:
    Checked/Unchecked - Stop Execution on Fail
    d. In **Default Text**, enter OK.

Because the Message Box step executes after the Real-Time Sequence Call step, you only see this message if the Real-Time Sequence Call step executes without failure, or if Stop Execution on Fail is disabled.

7. Click **Engine Demo Return Value tutorial.nivsstimprof** and in the Property Browser disable **Stop Execution on Fail**.
8. **Save** the stimulus profile.

The stimulus profile code will look like the following image.



After creating the stimulus profile, run it.

Logging Data to a File using a Stimulus Profile

Log channel data from a running stimulus test to a TDMS file.

1. Deploy the Engine Demo—Deploy the engine demo's system definition before running a stimulus profile.
2. Create a stimulus profile to log data—Create a stimulus profile that logs RPM and Temperature data as a TMDS file.
3. Run the stimulus profile—Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

You can open the created TDMS file in the VeriStand Editor by clicking **Tool Launcher** » **TDMS File Viewer** or by using the Workspace TDMS File Viewer tool.

Creating a Stimulus Profile to Log Data

Create a stimulus profile that logs RPM and Temperature data as a TMDS file.

Before you begin, deploy the engine demo's system definition.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.

2. In the Stimulus Profile Editor, click **New Stimulus Profile**.
3. Save the stimulus profile as Engine Demo Logging tutorial.nivsstimprof in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Logging directory.
4. Add a step to launch the VeriStand Editor.
    a. In the Steps palette, expand **Other** and drag **Command Shell** into **Setup**.
    b. In the Property Browser, specify the full path to VeriStand.exe as the **Filename**.

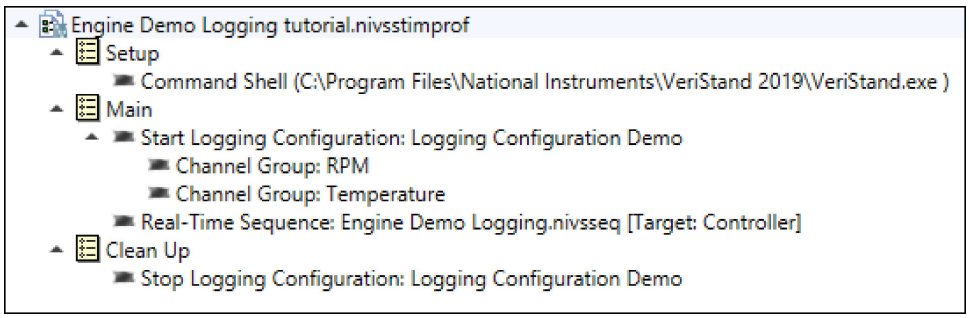
    > **Note** You can add arguments in the Property Browser for this step to specify the VeriStand project, system definition file, and VeriStand gateway IP address to connect to.

    When the stimulus profile runs, this step opens the VeriStand Editor or Workspace so you can watch the stimulus profile execute.
5. In the Steps palette, expand **Logging** and drag **Start Logging** into Main. A Channel Group step automatically appears under Start Logging. This step logs execution data for the channels you specify in each Channel Group that appears under this step.
6. From the Steps palette, drag another **Channel Group** into **Start Logging** so it contains two groups.
7. Click the Start Logging step and, in the Property Browser, configure it.
    a. Enter the **Configuration Name** as Logging Configuration Demo.
    b. Browse the **File Path** to <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Logging, enter the **File name** as Log File, and click **Save**.
    c. Enable **Replace Existing File**.
    d. Enter the **Log Rate [Hz]** as 100.
    e. Click the **Trigger Condition** pull-down and select **none**.
    f. Click the **Segment Options** pull-down and select **DoNotSegment**.

8. Click the first Channel Group and, in the Property Browser, configure it.
    a. Enter the **Channel Group Name** as RPM.
    b. Browse the **Channels**, expand **Aliases**, and enable **ActualRPM** and **DesiredRPM**.
9. Click the second Channel Group and, in the Property Browser, configure it.
    a. Enter the **Channel Group Name** as Temperature.
    b. Browse the **Channels**, expand **Aliases**, and enable **EngineTemp**.

    VeriStand logs data from the channels in the RPM and Temperature groups to Log File.tdms.
10. In the Steps palette, expand **Real-Time Sequences** and drag **Real-Time Sequence Call** into **Main**.
11. Configure the Real-Time Sequence Call.
    a. In the Property Browser, browse the **File Path** to <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Logging\Engine Demo Logging.nivsseq.
    b. Click the **Target Name** pull-down and select **Controller**.
12. In the Steps palette, drag **Stop Logging** into Clean Up.
13. In the Property Browser of the step, enter the **Configuration Name** as Logging Configuration Demo.
14. **Save** the stimulus profile.

The stimulus profile code will look like the following image.



After creating the stimulus profile, run it.

Calling a CSV File as a Real-Time Sequence

Use Comma Separated Values (.csv) files within stimulus profiles to stimulate, fault, and evaluate channels.

The stimulus profile you create in this tutorial executes a CSV file that turns on an engine and increases the RPM to 2500 within 10 seconds. After this CSV file completes its execution, the stimulus profile calls a sequence from the built-in library of sequences that causes the system to wait until the RPM parameters settle. Finally, the stimulus profile executes another CSV file that sets the engine RPM to various values over the course of 70 seconds.

To run in a stimulus profile, a CSV file must define inputs, values to assign to those inputs, and timestamps at which to update the input values. The CSV file must include this data under specific column headers that VeriStand expects.

1. Deploy the Engine Demo—Deploy the engine demo's system definition before running a stimulus profile.
2. Create a stimulus profile to call the CSV file—Create a stimulus profile that calls two Comma Separated Values (.csv) files like a real-time sequence.
3. Run the stimulus profile—Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

Creating a Stimulus Profile to call CSV Files

Create a stimulus profile that calls two Comma Separated Values (.csv) files like a real-time sequence.

Before you begin, deploy the engine demo's system definition.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. In the Stimulus Profile Editor, click **New Stimulus Profile**.
3. Save the stimulus profile as Engine Demo CSV File Replay tutorial.nivsstimprof in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo CSV File Replay directory.
4. Add a step to launch the VeriStand Editor.

a. In the Steps palette, expand **Other** and drag **Command Shell** into **Setup**.
b. In the Property Browser, specify the full path to VeriStand.exe as the **Filename**.

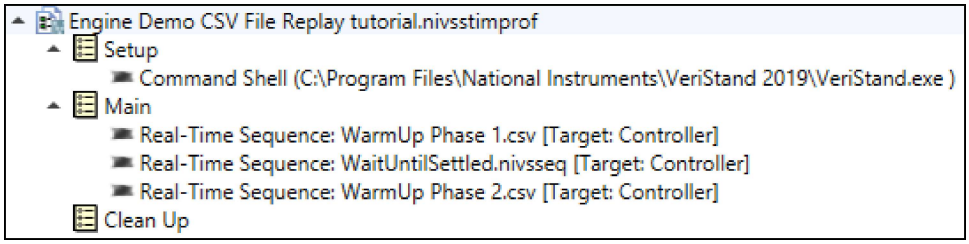
> **Note** You can add arguments in the Property Browser for this step to specify the VeriStand project, system definition file, and VeriStand gateway IP address to connect to.

When the stimulus profile runs, this step opens the VeriStand Editor or Workspace so you can watch the stimulus profile execute.

5. Expand **Real-Time Sequences** and drag **Real-Time Sequence Call** step into **Main**.
6. In the Property Browser, next to File Path, click **Browse** and select <Common Data>\Examples\Stimulus Profile\Engine Demo\Stimulus Profiles\Engine Demo CSV File Replay\WarmUp Phase 1.csv.
7. Click the **Target Name** pull-down and select **Controller** to execute the CSV file on the Controller target.
8. In the Sequences palette, expand **Real-Time Sequence Library** » **Standard** » **Timing** and drag **WaitUntilSettled** into **Main**.
9. In the Property Browser, configure the WaitUntilSettled sequence.
    a. Click the **Target Name** pull-down and select **Controller**.
    b. Click the **Type** pull-down and select **AlwaysPass**
    c. Next to Signal, select **Channel** and click **Browse** to display the system definition channel tree.
    d. Click **View aliases** to display the aliases defined in the system definition.
    e. Double-click **ActualRPM** to assign this alias to the parameter.
    f. Next to UpperLimit, select **Constant** and enter 500000.
    g. Next to LowerLimit, select **Constant** and enter 2400.
    h. Next to Timeout, select **Constant** and enter 120.

10. Drag another **Real-Time Sequence Call** step into **Main**.
11. In the Property Browser, configure the step to call WarmUp Phase 2.csv, located in the same directory as WarmUp Phase 1.csv, and set the **Target Name** to **Controller**.
12. **Save** the stimulus profile.

The stimulus profile code will look like the following image.



After creating the stimulus profile, run it.

Playing Back Previously Recorded Test Data Using the Stimulus Profile Editor

Create and run a stimulus profile that plays back data from a previously recorded VeriStand macro file.

A **macro file** is a recording of the commands sent to the target.

1. Deploy the Engine Demo—Deploy the engine demo's system definition before running a stimulus profile.
2. Create a stimulus profile—Create a stimulus profile to call a macro file that contains data from a previous engine test.
3. Run the stimulus profile—Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

Creating a Stimulus Profile to Play a Macro File

Create a stimulus profile to call a macro file that contains data from a previous engine test.

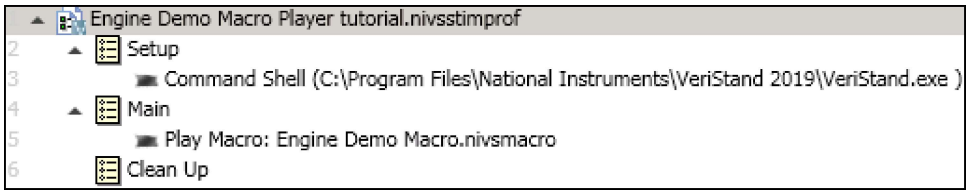
Before you begin, deploy the engine demo's system definition.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. In the Stimulus Profile Editor, click **New Stimulus Profile**.
3. Save the stimulus profile as Engine Demo Macro Player tutorial.nivsstimprof in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Macro Player directory.
4. Add a step to launch the VeriStand Editor.
    a. In the Steps palette, expand **Other** and drag **Command Shell** into **Setup**.
    b. In the Property Browser, specify the full path to VeriStand.exe as the **Filename**.

    > **Note** You can add arguments in the Property Browser for this step to specify the VeriStand project, system definition file, and VeriStand gateway IP address to connect to.

    When the stimulus profile runs, this step opens the VeriStand Editor or Workspace so you can watch the stimulus profile execute.
5. In the Steps palette, expand **Other** and drag **Macro Player** to **Main**.
6. In the Property Browser next to VeriStand Macro File, click **Browse** and select <Common Data>\Examples\Stimulus Profile\Engine Demo\Stimulus Profiles\Engine Demo Macro Player\Engine Demo Macro.nivsmacro.
7. Click the **Playback Mode** pull-down and select **UseTiming** to play back the macro file using the timing information embedded in the file.
8. **Save** the stimulus profile.

The stimulus profile code will look like the following image.



After creating the stimulus profile, run it.

Updating Model Parameters During Test Execution Using the Stimulus Profile Editor Tutorial

Call text files in a stimulus profile to update the values of engine model parameters while a test is running.

Use the Update Model Parameters from File step in a stimulus profile to call text files. This step has similar functionality to the VeriStand Editor Model Parameter Manager tab and the Model Parameter Manager Workspace tool. The step can update the parameters of a deployed and running model.

Using the step removes the manual process of launching the tool, updating values, and applying those values. Also, because Update Model Parameters from File is a step, it executes inline with the rest of your stimulus profile code. This ensures that you can update your model at known points within the execution of the stimulus profile.

1. Deploy the Engine Demo—Deploy the engine demo's system definition before running a stimulus profile.
2. Update the stimulus profile to use a text file—Configure an existing stimulus profile to use a text file to update model parameters without having to create and run multiple profiles.
3. Run the stimulus profile—Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

Updating a Stimulus Profile to Use a Text File for Model Parameters

Configure an existing stimulus profile to use a text file to update model parameters without having to create and run multiple profiles.

Before you begin, deploy the engine demo's system definition.

1. In the VeriStand Editor, click **Tool Launcher** » **Stimulus Profile Editor**.
2. Click **File** » **Open**, navigate to <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Engine Demo Return Value, and double-click the Engine Demo Return Value stimulus profile.

3. Save the stimulus profile as Update Model Parameters tutorial in the <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Update Model Parameters directory.
4. **Optional:** If you want to run the stimulus profile in the VeriStand Editor, update the Setup step.
    a. Right-click **Open VeriStand Workspace** and select **Delete**.
    b. In the Steps palette, expand **Other** and drag **Command Shell** into the **Setup**.
    c. In the Property Browser, specify the full path to VeriStand.exe as the **Filename**.

    > **Note** You can add arguments in the Property Browser for this step to specify the VeriStand project, system definition file, and VeriStand gateway IP address to connect to.

5. In the Steps palette, expand **VeriStand Control** » **Workspace** and drag **Update Model Parameters from File** into **Main**, above the Real-Time Sequence Call.
6. In the Property Browser, set **Source** to <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Update Model Parameters\OriginalParameterValues.txt
   If you open this file in a text editor, you can see that it resets the three key parameters for testing purposes back to their initial values.

```
a      [-7/9 0.5; -2/3 0]
{environment temperature (C)}      25
{idle speed (RPM)}      900
```

   In this code, a is the A matrix for the engine state-space model, {environment temperature (C)} is the temperature of the environment in which the engine operates, and {idle speed (RPM)} is the RPM the engine maintains while idle.

> **Note** VeriStand expects parameter names to start with a letter and contain only alphanumeric characters or underscores. If a parameter

> name does not fit this convention, you can enclose it in curly braces ({ }) to indicate that the string is a model parameter.

7. Select the Prompt Operator step that appears after the Real-Time Sequence Call and change the **Message** to Model parameters set to default values and the **Dialog Title** to New Parameter Values.
8. Add a second Update Model Parameters from File after the Prompt Operator, and set the Source to <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Update Model Parameters\ParameterUpdate1.txt. This file changes the environment temperature and the idle speed of the engine to very high values:

```
{environment temperature (C)}     75
{idle speed (RPM)}     2000
```

9. Right-click the Prompt Operator you configured and select **Copy**.
10. Right-click **Main**, and select **Paste** to add another operator prompt to the end of section. Update its message to Environment temperature set to 75 and Idle RPM set to 2000.
11. Copy the Real-Time Sequence Call and paste it after the prompt.
Because you are running the same test on each update of the model, you do not need to configure the sequence.
12. Add a third Update Model Parameters from File after the Prompt Operator, and set the Source to <Common Data>\VeriStand Projects\Engine Demo\Stimulus Profiles\Update Model Parameters\ParameterUpdate2.txt. This file demonstrates some more advanced operations:

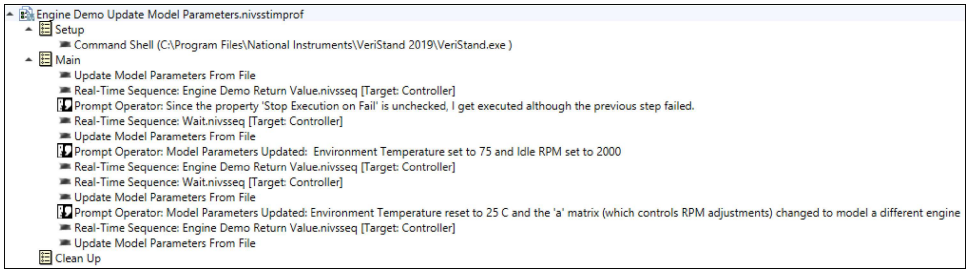
```
tempConversionFactor     0.5
{environment temperature (C)}     50 * tempConversionFactor
subscript     subfile.txt
```

The first line declares a temporary variable, tempConversionFactor. The second line uses this variable in a calculation. Temporary variables are local to the file in which you create them.
The third line uses the subscript command to call another text file subfile.txt. This subscript sets a few more parameter values. When you call a file as a

subscript, VeriStand inserts the contents of the subscript file into the calling file at the line that contains the subscript call.

13. Create another Prompt Operator with the Message Environment temperature reset to 25 C and the 'a' matrix (which controls RPM adjustments) changed to model a different engine.
14. Copy and paste another call to the real-time sequence.
15. **Optional:** If you want to ensure you leave your system in a known state, add another Update Model Parameters from File that calls OriginalParameterValues.txt.
16. Click **Update Model Parameters tutorial.nivsstimprof** and in the Property Browser disable **Stop Execution on Fail**.
17. **Save** the stimulus profile.

The stimulus profile code will look like the following image.



After updating the stimulus profile, run it. Each a time a test completes, click OK in the operator prompt to advance to the next model update.

Note Some tests, such as the extreme conditions in ParameterUpdate1.txt, fail quickly.

Running a Stimulus Profile

Compile and run the stimulus profile to see the real-time sequence interact with the Engine Demo.

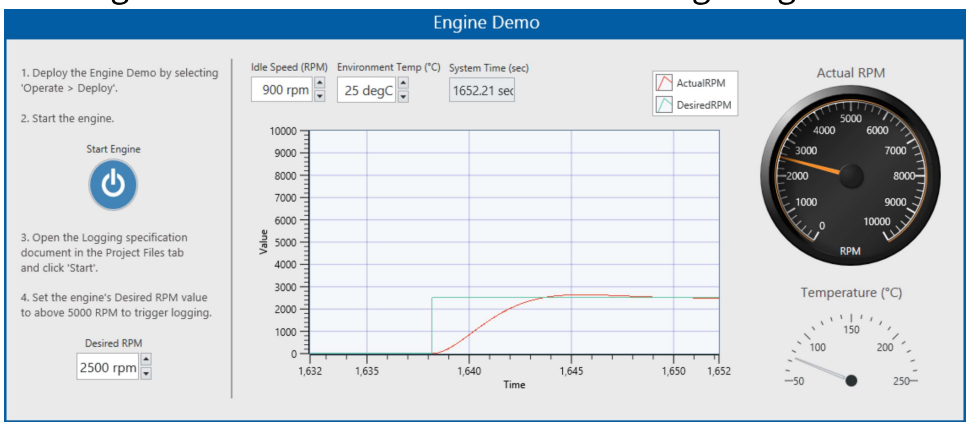
1. On the stimulus profile, expand **Execution Results**.

After each step executes, Execution Results displays the name of the step and the result of the step execution.

2. Click **Run** to compile all the open and referenced real-time sequence files and run the stimulus profile.
   You can observe the user interface's graph to see the output change as the real-time sequence executes.

> **Note** The Warnings and Errors pane displays any errors, warnings, or messages that you must resolve before the stimulus profile can successfully execute.

The engine demo will run like in the following image.



When the stimulus profile completes its execution, the ATML Test Report opens. This report displays details about the stimulus profile execution. If you run the stimulus profile again, the previous results are overwritten.

Debugging the System

Remedy system-related issues.

After receiving errors in VeriStand, you must resolve them to continue running the system.

Note For more help debugging the system, refer to NI KnowledgeBase for help resolving the error.

1. VeriStand Error Codes
   VeriStand returns an error code when it encounters a problem while executing.
2. Viewing Real-Time Target Logs
   Display and monitor logs from a real-time target.
3. Related Documentation
   Use the following documents for more help with VeriStand.
4. Maximizing System Performance
   Increase the efficiency of VeriStand by following best practices for your system definition, controllers, hardware, models, and reflective memory.

VeriStand Error Codes

VeriStand returns an error code when it encounters a problem while executing.

Use the following table to locate an error code message.

Note For more information on correcting VeriStand errors, refer to the KnowledgeBase.

Error code	Message
-307500	The FPGA configuration file (.fpgaconfig) is invalid or contains an error.
-307501	The specified operation is unable to acquire the memory necessary to execute. To correct this

Error code	Message
	error, increase the amount of memory in the system or reduce the amount of memory that this operation requires. You can reduce the amount of memory required by the operation by uninstalling unneeded components from the real-time (RT) target or by reducing the number of devices, channels, and/or stimulus generators used.
-307502	An FPGA argument or parameter is invalid. This might be a problem with the FPGA configuration file (.fpgaconfig) or FPGA bitfile. This problem also might be caused by an invalid FPGA target handle.
-307503	An unknown exception occurred while running the NI FPGA device. Verify that NI-RIO and the VeriStand software are installed correctly on the real-time (RT) target.
-307504	An FPGA argument or parameter is outside the expected range. Update the FPGA configuration file (.fpgaconfig).
-307505	The FPGA bitfile has changed since the system definition file was last compiled. Open the system definition file in System Explorer and refresh the FPGA configuration file (.fpgaconfig).
-307510	Framework with the given name already exists.
-307511	EESPort with the given name already exists.
-307512	An unexpected step error has occured.
-307525	Cannot open selected file. The file was saved in a later version of VeriStand.
-307526	The XML file provided is invalid.
-307548	Legacy Stimulus Profile Editor is not supported anymore.
-307549	Cannot find the wave file that an alarm response is configured to play when the alarm is triggered. Verify that wave files are located at their specified locations.

Error code	Message
-307550	You have provided an invalid username or password.
-307551	Internal error. The buffered data system failed to find required information in the transaction's variant data.
-307552	Internal error. An attempt was made to open a 2D waveform read session.
-307553	Unexpected waveform data type. This VI requires the specified waveform nodes to be of a specific data type.
-307554	The specified stream condition is not valid for the data type of the waveform. This error might occur if you try to set the streaming condition to be within a range of values for a waveform whose data type is a complex double (CDB). The in-range streaming condition supports only waveforms whose data type is double (DBL).
-307555	Cannot open a waveform session using an empty array of waveform references. Use the Get Waveform Data Reference VI to generate valid data references with which you can open a waveform session.
-307556	Cannot open waveform session because waveform data references are not valid. Use the Get Waveform Data Reference VI to generate valid data references with which you can open a waveform session.
-307557	Custom device cannot perform requested operation because VeriStand Engine is shutting down.
-307559	The size of an input array does not match the number of waveforms in the streaming session. When you write to a multiple-waveform session, the number of rows in the 2D **Data** array must match the number of waveforms in the session. When you start a multiple-waveform session, the size of the 1D **t0s** array must match the number of waveforms in the session.

Error code	Message
-307561	Cannot write to waveform. Another waveform write session is already open for this waveform reference, and only one writer can access the waveform reference at a time.
-307562	The waveform session reference cannot be used if it is not open. Use the Open Waveform Session VI to open a waveform session reference.
-307600	The specified VI is broken and cannot run. Open the VI in LabVIEW for more information.
-307601	The XML file is invalid according to the XML schema document (XSD).
-307602	The GUID could not be found. If the requested page is a custom device, contact the creator of the custom device.
-307603	The requested custom device does not have a main page specified. Contact the creator of the custom device to correct this error.
-307604	VeriStand cannot mutate the system definition file to the current version.
-307605	The requested simulation model could not be found. Navigate to the Model Configuration page in System Explorer and choose a different model.
-307606	The file does not contain a valid file format.
-307607	VeriStand could not mutate the system definition file. The system definition file version is different from the official released version of VeriStand.
-307608	The custom device does not provide a valid source distribution for the target specified. Specify a different target or contact the creator of the custom device for further support.
-307609	The formula contains an unused variable declaration.
-307610	The formula contains an invalid function or variable name.

Error code	Message
-307611	The associated NI-XNET database path cannot be found. Update either the alias or the database path.
-307612	One or more chassis are unidentified. Open NI MAX and identify the type of the chassis for each controller that is configured in System Explorer.
-307613	You cannot add channels of the type you selected. The DAQ device either contains no channels of this type, or all available channels of this type are already in the system definition.
-307614	The DAQ support files are missing for the channel type selected. Try repairing your installation of VeriStand.
-307615	This DAQ device does not support reference clock synchronization. In System Explorer on the DAQ Device Configuration page, click the **PXI Backplane Reference Clock** drop-down and select **None** or **Automatic**.
-307616	The associated NI-XNET signal has been deleted from the XNET database or it changed so it no longer matches the system definition. If the database no longer contains the signal, delete the signal from the system definition. If the signal changed in the database, reimport the signal.
-307617	The system definition contains aliases that are not linked to any channel. Click on the Error/ Warning icon on System Explorer to get a list of aliases that are not linked and either link them to a channel or delete them.
-307621	The request at the specified index does not have the same number of signals as the signal_num parameter. Call the GetNextCompletedRequest method to get a valid index and signal number of a completed request.

Error code	Message
-307622	The specified request was not completed. Call the GetNextCompletedRequest method to get a valid index to a completed request.
-307623	The specified request index is not valid. Call the GetNextCompletedRequest method to get a valid index to a completed request
-307624	No proper request was found. Call the GetNextCompletedRequest method after the model executes a step to get the index of a completed request.
-307625	An error occurred while reading the signal values.
-307626	One or more of the specified signals are not scalar. Only scalar signals can be viewed.
-307627	One or more of the specified signal indexes are not valid. Use signal indexes that are in the model.
-307628	There is not enough space on the target to read the signal values. Ensure you are not viewing signal values from paused models.
-307629	A required input or output parameter for this method is null. Use allocated variables for parameters.
-307630	The model refnum is not valid. Use the InitializeLibrary function to return a valid refnum.
-307650	An unexpected error has occurred in the VeriStand API.
-307651	The VeriStand API does not support this function.
-307652	Failed to open a connection to VeriStand.exe.
-307653	The input file parameter is invalid or does not exist. Ensure that this path is correct and that the file has not been removed.
-307654	Cannot perform this request because VeriStand is not running a system definition file. Deploy a system definition file, and try again.

Error code	Message
-307655	Invalid request. VeriStand is missing an engine component that is required to process the request.
-307656	Invalid request. VeriStand does not have a system definition file loaded.
-307657	The specified password does not match the current deployed configuration password.
-307658	VeriStand is unable to run the system definition file when a system definition file is already running.
-307659	Invalid password input parameter. The password cannot be an empty string.
-307660	Timeout occurred while deploying a new system definition file.
-307661	VeriStand failed to deploy the system definition file.
-307662	Node is not found in the system.
-307663	Timeout occurred while processing request.
-307664	Channel does not have read access.
-307665	Channel does not have write access.
-307666	The specified channel cannot be faulted.
-307667	The requested operation on this item reference is invalid.
-307668	The specified channel cannot be scaled.
-307669	The license is unavailable, invalid, or VeriStand failed to initialize the licensing component.
-307670	The VeriStand Gateway cannot process the requested operation. The VeriStand Gateway currently is busy performing another operation.
-307671	The execution host is running a different system definition file than the file specified. You must deploy the system definition file in order to connect to the execution host.
-307672	One or more targets failed to start within the specified timeout. Verify that any start trigger or clock signals are configured correctly.

Error code	Message
-307673	Another VeriStand Gateway already is connected to the execution host. Only one VeriStand Gateway can be connected to an execution host at a time.
-307674	The Run Workspace function is no longer supported. Use the Connect to System function instead, which has a system definition file parameter instead of a workspace file.
-307675	The VeriStand Gateway cannot complete the specified operation because the number of streamed channels for one or more targets exceeds the maximum size. Stop data logging configurations or graphs to reduce the number of streamed channels. You also can increase the maximum streamed channel count for the targets by editing the system definition file in System Explorer.
-307676	VeriStand cannot open the specified project file because another project is already open.
-307677	The reference to the project file is no longer valid. The referenced project is no longer open.
-307678	The specified workspace tool VI is not open.
-307679	The VeriStand Gateway is no longer connected to the targets. One of the targets stopped running the system definition file.
-307680	An exception occurred when the model parameter file was being parsed.
-307681	Failed to read the VeriStand service configuration file. Repair your VeriStand installation.
-307682	Could not complete one or more requests because a target is offline. A target is no longer connected and is not running the system definition. Any future requests for this target will not be completed.
-307683	Cannot deploy or connect to the system definition because all targets in the system are

Error code	Message
	disabled. Enable one or more targets in the system definition, and then try to deploy again.
-307684	Cannot start server because the port address is already in use by another process. To change the port address for the server, select **Tools** » **Options**, browse to the Port Settings page, and change the value of the corresponding control.
-307685	Cannot initialize model parameter(s) defined in model parameter file because parameter was not found in system. 1) Ensure the expression of the parameter is spelled correctly in the file. 2) If the file defines a temporary variable, ensure the system definition allows temporary variables. 3) If the file contains aliases, ensure the path to an alias file is correct in the system definition. Verify the contents of the file and the settings on the Simulation Models configuration page in System Explorer, and try again.
-307686	The specified target is not defined in the deployed system definition.
-307687	The given number of channels and number of values do not match.
-307688	Timeout occurred while reconnecting a target to the gateway.
-307689	The specified target is disabled. Change the target settings in the system definition and redeploy if you wish to use this target.
-307690	To reconnect to a target without deploying, the restart action for real-time (RT) targets must be set to Run System Definition. To enable the Run System Definition restart action, select the **System Initialization** node in the System Explorer and select **Run System Definition**. You must redeploy the system definition to make use of these changes.
-307691	Import operation failed for the specified SLSC chassis.

Error code	Message
-307692	VeriStand encountered an unknown internal error while using WebDAV. Check your target's disk to determine if it is full.
-307700	One or more Model Execution Loops failed to shut down properly.
-307701	Timeout occurred while writing a parameter value to the VeriStand Engine.
-307702	The size of the imported model data in the system definition file conflicts with the size in the specified model file. This error can occur if the model file contains a different number of inports, outports, signals, or parameters than when it was imported. This error also can occur if two or more models contain a global parameter with the same name but different dimensions. To correct this error, reload the model in the System Explorer and verify the dimensions of any global parameters that multiple models contain. Alternatively, on the Parameters page for a model in the Scope for Global Parameters drop-down, select **Model** to avoid conflicts caused when the model shares the global parameter with other models.
-307703	The specified model is incompatible with VeriStand. If you want to deploy the model to an RT target, launch the Console Viewer tool to display the console output of the target, which includes information about the source of the error.
-307704	The specified model is not compatible with the specified execution host.
-307705	The specified model has an unsupported 32-bit architecture.
-307710	The specified item is not an inline custom device. Pass in a valid reference to an inline custom device.
-307711	A channel and the alias that references it both specify a data source mapping. A channel can

Error code	Message
	only have one data source mapping. Remove the mapping for either the channel or its alias.
-307718	Setting Scan Interface mode failed. Before using this mode, make sure NI RIO IO Scan is installed on the target.
-307719	Unable to discover XNET interface(s) before timeout elapsed. Contact NI support at ni.com/support for more information about resolving this error.
-307720	Failed to route PXI_Trig0 from bus segment of chassis master device to other bus segments. To address this issue, open MAX and select the chassis. On the Triggers pane for the chassis, clear any reservation for PXI_Trig0 on the appropriate bus, set routing for PXI_Trig0 to **Dynamic**, and try again.
-307721	Cannot enable Slow Background Conversion Mode if hardware-timed single-point sample mode is disabled for analog input channels. If you want to enable Slow Background Conversion Mode for this device, you must first enable hardware-timed single-point support for its AI channels.
-307722	A DAQ channel of an unexpected type was found.
-307723	A property of a DAQ channel has an invalid value. Change the value the error message lists. If you used the System Definition API to change the units property of a channel, restore the units to the original value.
-307724	A DAQ channel initialization VI encountered an unexpected property name.
-307725	The value does not match the dimensions of the specified channel.
-307726	A calculated channel failed to compile correctly and cannot be executed.
-307727	The requested DAQ device, the master timing source for the timed loop, is unavailable.

Error code	Message
-307728	One or more channel mappings have incompatible dimensions. Matrix channels only can be mapped to matrix channels with the same dimension.
-307729	The system definition file is missing. Deploy a valid system definition file.
-307730	One or more asynchronous custom devices failed to shut down properly and the VeriStand Engine aborted them.
-307731	The system definition file is not saved in the current version of VeriStand. VeriStand cannot mutate system definition files on a real-time target. Use System Explorer to open and save the file.
-307732	The system has become unresponsive. The Primary Control Loop has been shut down. To correct this error, enable **Filter Watchdog Errors** on the Controller Configuration page of System Explorer. You can configure watchdog functionality by monitoring the Watchdog Timer system channel using alarms and procedures.
-307733	The version of the calibration file (HardwareCalibrationData.nivscal) is too old to mutate to the current version. Delete the file and manually recreate the calibration settings.
-307734	The scaling and calibration setting is invalid. The channel does not exist or is not scalable.
-307735	A required resource is disabled in the current version of the VeriStand Engine. For example, a DAQ board is the master timing device, but the current VeriStand Engine has disabled NI-DAQ. Remove the device or resource from the system definition file.
-307736	Unable to share data for the channel. Either the source or destination target does not have reflective memory support.

Error code	Message
-307737	Unable to create dynamic data sharing between targets. Either the source or destination target does not have a data sharing device.
-307738	Unable to share data between two targets. There is not enough dynamic data sharing space.
-307739	Data on reflective memory must have the correct byte alignment from memory address 0x0.
-307740	Invalid data range. The reflective memory network requires a valid data range.
-307741	VeriStand export data exceeds the allocated reflective memory address.
-307742	The VeriStand Engine could not be loaded. Make sure that VeriStand and the required drivers, such as NI-DAQmx, are installed correctly. Refer to the VeriStand readme.html for a list of required software and drivers.
-307743	The VeriStand Engine failed to start. A configured start trigger or external timing source did not occur within the specified timeout.
-307744	An alarm failed to compile correctly and cannot be executed.
-307745	You must specify a VISA device address for the reflective memory card.
-307746	The maximum Windows target rate is 1 kHz. To achieve higher rates, use an NI-DAQ device or NI FPGA target.
-307747	The system definition file contains two targets that are using the same IP address. Each target must have a unique IP address. Only one target can be a Windows target.
-307748	The VeriStand Engine rebooted while running the system definition file. You must reconnect to the execution host and possibly redeploy the system definition file.

Error code	Message
-307749	A simulated DAQ device cannot be used as a Timed Loop timing source or as a chassis master hardware synchronization device. In System Explorer, navigate to the Chassis Configuration page and choose a different DAQ device from the Chassis master hardware synchronization device pull-down menu. On the Controller Configuration page, choose a different Primary Control Loop timing source. Or, if all DAQ devices are simulated, disable hardware-timed single-point support for all of them or disable the DAQ device.
-307750	Deserialization error. Failed to load macro file.
-307751	The specified operation is not permitted while playing a macro.
-307752	The specified operation is not permitted until a macro is loaded.
-307753	The specified operation is not permitted unless running a macro.
-307754	Serialization error. Unable to save macro.
-307755	The specified operation is not permitted while recording a macro.
-307756	Internal error: The DAQmx waveform producer received a command which was not declared as a publicly available.
-307780	The specified feature is supported only if the VeriStand Gateway is running on the localhost machine.
-307781	The channel group cannot be found in the specified log file.
-307782	The channel cannot be found in the specified log file.
-307783	The data structure contains a dependencies cycle.
-307784	The STI file is not valid.
-307785	No error provider exists to handle the error signals.

Error code	Message
-307800	A procedure failed to compile correctly and cannot be executed.
-307824	The specified item is not a waveform.
-307825	The specified item is not a channel.
-307826	The specified item is not an alias for a channel.
-307827	The specified item is not an expected type inside the system storage hierarchy.
-307828	This dependency does not refer to a valid item. The item may have been deleted. Select a different dependency or re-add the invalid item and click **Update Dependencies**.
-307829	The specified node cannot be found.
-307830	The channel dimension does not match the channel default value length.
-307831	Unable to save file to the specified path.
-307832	The depending file path is improperly formed or cannot be found.
-307850	The target cannot be reached. Verify that the Ethernet cable is connected to the real-time (RT) target and that the RT target is enabled and running.
-307851	One or more system definition file dependency files differ from the originally imported version. This might cause unexpected behavior or errors. Cancel the current operation and resolve any conflicts in System Explorer.
-307852	The specified dependent file is contained within a LabVIEW library file (LLB), but the destination path on the target is not contained in an LLB. Individual files cannot be copied out of an LLB. The destination path on the target must have an LLB as the parent of the dependent file.
-307853	The VeriStand Gateway was unable to establish a connection with the target. Confirm that the target is running and that the VeriStand Engine successfully started. If you still cannot connect to the target, use MAX to reinstall the VeriStand

Error code	Message
	Run-Time Engine on the target. You may encounter this error if you attempt to connect to the target from a LabVIEW project, because a LabVIEW project can update the start-up application for the target. To deploy a system definition to a target, the VeriStand Run-Time Engine (VeriStand.rtexe) must be the start-up application.
-307860	The specified trigger channel does not exist.
-307861	The data logging configuration is invalid. One or more specified channels does not exist.
-307862	The specified data logging configuration resulted in a buffer size that exceeds the maximum allowed size. Reduce the buffer size by reducing the pre-trigger time, the logging rate, or the number of logged channels in the data logging configuration.
-307863	The specified property value is not a valid data type for a TDMS property.
-307864	The specified log file path is not valid. It might include invalid tags.
-307865	The specified data logging session does not exist.
-307866	An error occurred while logging data. Refer to the error description for more information.
-307867	The specified data logging session name is already in use. Specify a unique data logging session name.
-307868	The specified trigger condition contains syntax errors.
-307869	Data was lost during logging. The log files may not contain every sample. The start and stop triggers may have also missed data to evaluate.
-307870	A compatible version of NI DIAdem is not installed. You must have NI DIAdem installed to load data in NI DIAdem.

Error code	Message
-307871	Cannot retrieve information for the specified log file. Ensure the session name and filename are correct.
-307872	There was an error downloading one or more of the requested files. Please use the errors parameter for more detaield information.
-307881	Cannot use legacy Channel Scaling and Calibration tool/API because this project contains calibration features from the new Channel Calibration tool/API introduced in VeriStand 2012. This error occurs because the project contains a channel calibration file (.nivscf). If you want to use the legacy Channel Scaling and Calibration tool, delete the .nivscf file under the **Calibrations** in Project Explorer and redeploy. Otherwise, use the Channel Calibration Tool/API instead of the legacy Channel Scaling and Calibration tool/API.
-307882	Cannot use Channel Calibration tool/API because this project does not contain a channel calibration file (.nivscf). If you want to use the Channel Calibration tool/API, add a .nivscf file to the project under **Calibrations** in Project Explorer. Otherwise, you can use the legacy Channel Scaling and Calibration Tool/API instead of the Channel Calibration tool/API.
-307883	Cannot deploy project because the system definition file contains scales but the project does not contain a channel calibration file (.nivscf). Delete the scales from the system definition file and redeploy.
-307887	A polynomial scale mapped to a channel(s) does not have coefficients defined. Remove the mapping or update the scale to contain coefficients, and then try to deploy the system definition again.
-307889	A lookup table scale mapped to a channel(s) is empty. Remove the mapping or update the scale to contain pairs of pre-scaled values and

Error code	Message
	the corresponding values to which to scale them, and then try to deploy the system definition again.
-307891	VeriStand only supports devices whose calibration password is the default value. Use NI MAX or the NI System Configuration API to reset the calibration password on the device to the default value.
-307900	The number of generator mappings exceeds the maximum allowed. Reduce the number of mapped channels or increase the maximum number of mappings per generator using the Stimulus Configuration page.
-307901	The number of compiled stimulus generator steps exceeds the maximum allowed. Reduce the number of steps in the stimulus generator or increase the maximum number of steps per generator using the Stimulus Configuration page.
-307902	The specified stimulus profile is empty. It contains no compiled steps.
-307903	The number of data points exceeds the maximum auxiliary buffer size. Reduce the number of data points in the stimulus profile or increase the maximum auxiliary buffer size in the Stimulus Configuration page of System Explorer.
-307904	A stimulus profile failed to compile correctly. Verify that the stimulus profile links to valid channels.
-307905	Timeout occurred while writing data to the auxiliary buffer.
-307906	Timeout occurred while writing data to the analysis buffer.
-307907	Multi-point data does not have the correct timing. The data must start at time 0 and increment at a multiple of Delta t value.

Error code	Message
-307908	The stimulus profile manager currently is reserved by a client.
-307909	Unable to start stimulus profile generation. Stimulus profile file contains invalid profile steps or incorrect timing.
-307910	Unable to start stimulus profile generation. CAN logging requires that the CAN Bus Monitor is running.
-307911	Unable to start stimulus profile generation. One or more calibration files are not found.
-307912	The number of data points exceeds the maximum analysis buffer size. Reduce the number of data points in the stimulus profile or increase the maximum analysis buffer size using the Stimulus Configuration page of System Explorer.
-307913	The number of stimulus generators exceeds the maximum allowed. Reduce the number of stimulus generators in the stimulus profile or increase the maximum allowed using the Stimulus Configuration page of System Explorer.
-307914	The specified CSV file does not contain a Timestamp column. The file you specify must have a Timestamp column that contains values in milliseconds.
-307915	The operation could not be completed. A stimulus profile is running.
-307916	A stimulus profile generator contains a Set Variable step that attempts to set a variable on another target. A Set Variable step only can set a variable on the same target as the generator containing the step.
-307917	A stimulus profile generator contains channel mappings for channels on a different target than the target on which the generator will execute. A generator can only map to channels on the target on which it will execute. Confirm

Error code	Message
	that the generator is not mapped to channels from different targets, or that the generator does not specify an execution target that is different than one of the channel mappings.
-307925	The specified node is not compatible with this operation.
-307926	This node already exists in the system.
-307927	Invalid argument. The node reference is not a valid channel.
-307928	Found an existing FPGA target with the same RIO address configuration.
-307929	The mapping definition is invalid. The source channel must be readable, the destination channel must be writable, and both channels cannot be part of the stimulus generator.
-307930	A node name cannot be an empty string.
-307931	Model execution order group cannot be empty.
-307932	The formula type must depend on variables.
-307933	Attempted to write to a read-only property.
-307948	Unable to establish a connection between the VeriStand Model Simulation Server and client. The VeriStand MDL client version is incompatible with the currently loaded VeriStand Model Simulation Server.
-307949	The selected model (.mdl) is invalid. Verify that the model can be compiled and run properly, that you have added an NIVeriStandSignalProbe block to it, and that the stop time is set to inf.
-307950	Unable to connect to the VeriStand Model Simulation Server.
-307951	VeriStand requires that you compile a model (.mdl) before it can run on a real-time (RT) target.
-307970	An inline custom device attempted to access invalid data in the VeriStand Engine. The inline custom device attempted to read or write data using an invalid reference or a reference that

Error code	Message
	did not have the read or write access required for that operation.
-307971	A channel cannot have child items.
-307972	Error provider received an electrical error that contains a signal it cannot handle. Verify the error provider is implemented correctly and all errors contain the correct signals.
-307973	Error provider received an electrical error with an unexpected number of signals. Verify the error provider is implemented correctly and all errors contain the correct number of signals.
-307974	Error provider received an unexpected configuration. Verify the error provider is implemented correctly and the errors can be handled by the error provider.
-307980	The specified stimulus profile session is locked. Unlock the stimulus profile session to control sequences in the session.
-307981	The stimulus profile session does not contain a sequence with the specified name.
-307982	The stimulus profile session is not deployed.
-307983	The sequence containing the specified variable is currently running. You cannot probe a variable value while a sequence is running.
-307984	The stimulus profile session is already deployed.
-307985	The sequence did not complete the timestep within the specified timeout. This could be due to a sequence that contains too much processing or an infinite loop that does not yield. Evaluate the timing of your sequence or increase the timeout.
-307986	The specified parameter assignment is not valid. The real-time sequence parameter does not exist.
-307987	One or more real-time sequence parameters with a by-reference evaluation was not mapped

Error code	Message
	to. Assign a value to all real-time sequence parameters that have a by-reference evaluation type.
-307988	The compiler failed because the specified real-time sequence contains one or more errors.
-307989	The specified parameter assignment is not valid. The channel mapped to the parameter could not be found in the system.
-307990	A system definition channel has been mapped to two or more real-time sequence parameters that are not the same data type. A channel can only be mapped to multiple parameters if they share the same data type.
-307991	A real-time sequence parameter assignment has an invalid data type. Match the data type of the value assigned to the parameter to the parameter's data type.
-307992	The specified operation cannot be completed. There is no project file currently open.
-307993	The parameter dimension does not match the parameter default value length.
-307994	The specified parameter cannot be found.
-307995	VeriStand engine failed to retrieve data from the SLSC chassis status loop within the specified timeout. The System Definition was undeployed.
-307996	SLSC module not found in the specified slot. Check your SLSC hardware and make sure the custom device is inserted in the correct slot.
-307997	There are SLSC chassis added to the system definition that are not supported. Remove the SLSC chassis from the system definition.
-307998	The switch load signal conditioning (SLSC) chassis name or IP address/hostname is empty. Provide a SLSC chassis name or IP address/hostname.

Viewing Real-Time Target Logs

Display and monitor logs from a real-time target.

The Target Log Viewer collects system definition deployment information and debugging messages.

1. Open your project in the VeriStand Editor.
2. Deploy the system definition to a real-time target.
3. Click **Tool Launcher** » **Target Log Viewer**.
4. In the Target Log Viewer tab, click **Start Listing**.
   An output log will begin to appear in the tab.
5. Click **Stop Listening** and scroll on the log to review information from the beginning of the deployment.

> **Note** If you click **Start Listening** again, the log will update with information that was not recorded while stopped.

To view logs from previous deployments, access the machine serving as the target and navigate to the /c/ni-rt/NIVeriStand/Logs/ directory.

Related Documentation

Use the following documents for more help with VeriStand. Refer to ni.com/manuals for updated documentation resources.

Document	Description
VeriStand Release Notes	Provides system requirements, installation instructions, and important information about the version of VeriStand that you are installing. Access this file from ni.com.
VeriStand .NET API Help	Documents several of the .NET APIs included with VeriStand. Use these APIs to programmatically control various software operations from any .NET-compatible programming language or environment,

Document	Description
	including LabVIEW and TestStand. Access this file from the Windows Start menu.
LabVIEW Help	Provides information about LabVIEW palettes, menus, tools, VIs, and functions. It also includes step-by-step instructions for using LabVIEW features, and documentation on LabVIEW Real-Time Module and LabVIEW FPGA Module when you install those modules.
LabWindows/CVI Help	Provides information about LabWindows and CVI windows, functions, tools, and menus.
NI-DAQmx Help	Provides information on using NI-DAQmx to program your National Instruments device.
NI-XNET Hardware and Software Help	Describes how to install and configure the NI-XNET hardware and software. It also includes the NI-XNET LabVIEW and C API reference and summarizes the CAN, FlexRay, and LIN standards.
Hardware Documentation	Provides support for any hardware devices you want to use with a project.
MathWorks Simulink software documentation.	Provides support for Simulink tasks such as system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems.
MathWorks MATLAB software documentation.	Provides support for MATLAB tasks such as signal processing, event-based modeling, and application deployment.
MathWorks Simulink Coder software documentation.	Provides support generates and executes C and C++ code from Simulink models and MATLAB functions.

Maximizing System Performance

Increase the efficiency of VeriStand by following best practices for your system definition, controllers, hardware, models, and reflective memory.

Streamlining the System Definition

Decrease the complexity of your system definition by removing unused hardware I/O channels, maximizing the Convert Clock rate for multiplex sampling DAQ devices, and using hardware timing.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Depending on your goal, complete any of the following tasks.

Goal	Tasks	Rationale
Remove all unused hardware I/O channels.	a. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **DAQ** in the configuration tree. b. Select a DAQ device and delete any unused hardware I/O channels.	Unused hardware resources slow performance. VeriStand must read and write every I/O channel in the system definition regardless of whether or not the system uses the channel data.
Maximize the rate of the Convert Clock on DAQ devices that use multiplexed sampling.	a. Click **Targets** » **Controller** » **Hardware** » **Chassis** » **DAQ** in the configuration tree. b. Click a DAQ device and, in the Conversion rate drop-down, select **Maximum**.	Multiplexing can cause delays if the Convert Clock does not run fast enough. Changing the rate of the Convert Clock to maximum minimizes the delay. Relevant DAQ devices include the M Series, E Series, and some X Series. **Note** Using the maximum possible Convert Clock rate also reduces the

Goal	Tasks	Rationale
		accuracy of measurements.
Use hardware timing instead of software timing.	a. Click **Targets** » **Controller** in the configuration tree. b. In the Primary Control Loop timing source drop-down, select **Automatic**. c. Click **Hardware** » **Chassis** in the configuration tree. d. In the Chassis master hardware synchronization device drop-down, select **DAQ** or **FPGA**.	With hardware timing, a digital signal, such as a clock on your device, controls the rate at which signals are generated. With software timing, the software and operating system controls the rate instead of the measurement device. A hardware clock can run faster—and is more accurate—than a software loop.

4. Save the system definition file.

Related tasks:

- Synchronizing Hardware and Software

Configuring the BIOS Settings of the Controller

Increase the performance of your real-time controller by enabling turbo boost and reducing the number of enabled cores.

To configure the BIOS settings for VeriStand, complete any of the following tasks.

Task	Rationale
Enable Turbo Boost	Enabling Turbo Boost for Intel Core™ processors allows active processor cores to run faster than the base operating frequency for short durations while other cores are idle.

Task	Rationale
	Note Enabling Turbo Boost can also increase application jitter.
Reduce the number of enabled cores.	When you activate Turbo Boost, the maximum frequency of a specific processing core depends on the number of active cores. Manually reducing the number of cores ensures the active cores receive the maximum increase in clock frequency. If your system does not contain asynchronous components, enable only one core to provide the greatest frequency boost. For example, the Intel Core i7-820QM quad-core processor used in an NI PXIe-8133 embedded controller has a base clock frequency of 1.73 GHz. If an application requires only one CPU core, Turbo Boost automatically increases the clock frequency of the active CPU core on the Intel Core i7-820QM processor to 3.06 GHz.
Enable two cores for VeriStand systems with asynchronous components, such as an asynchronous custom device.	Enabling two cores allows you to assign the asynchronous components to another CPU while still providing greater clock frequency to both active cores.

For more information, refer to Top Eight Features of the Intel Core i7 Processors for Test, Measurement, and Control.

Configuring the Ethernet Settings of the Controller

Increase the performance of your real-time controller by using line interrupt packet detection.

Most NI Real-Time targets offer three options for packet detection: Line Interrupt, Polling, and Message Signal Interrupt. Line Interrupt provides the fastest performance as the device driver gets immediately notified when the target receives data.

Note Line Interrupt introduces the most jitter of the options.

1. Open NI MAX.
2. In the **Packet Detection** setting of the controller, select **Line Interrupt**.

> **Note** If **Line Interrupt** is not available, select **Polling** and change the **Polling Interval** to 1 millisecond. Polling at a high rate provides high performance while introducing less jitter than interrupt, but increases CPU utilization.

Optimizing Hardware Performance

Increase the performance of your VeriStand system by using hardware timing, simultaneous sampling, USB CAN devices, PXIe devices, and not using Real-Time Hypervisor.

1. Add a hardware device.
2. To optimize your hardware performance, complete any of the following tasks.

Task	Rationale
Use controllers that support hardware timing.	Software timing slows the system significantly and adds to CPU usage. Using controllers that support hardware timing allows for better system performance.
Choose DAQ devices that use simultaneous sampling.	Simultaneous sampling provides better performance than multiplexed sampling.
Use a USB CAN device instead of XNET ports or channels for bus monitoring only.	Using a USB CAN device on the host computer reduces the number of channels in the system. The fewer channels VeriStand reads, the better the performance.
Use PXIe devices and controllers.	PXIe devices generally contain newer technology and run at faster rates than other devices.
Do not use NI Real-Time Hypervisor for systems that require high performance.	Real-Time Hypervisor comes with dramatic real-time performance penalties. Switching to a real-time only PXI controller can potentially double the performance.

Improving Model Performance

Increase model performance by consolidating small models and preallocating arrays for LabVIEW models.

To optimize your model performance, complete any of the following tasks.

Task	Rationale
Consolidate small models into one large model.	Several small models use more memory than one large model.
Preallocate arrays for LabVIEW models instead of using Build Array functions.	Each Build Array function uses a shared resource. This may delay the model execution because both models cannot use the shared resource simultaneously. Preallocating arrays avoids potential delays. To preallocate an array, use a Case structure and the First Call? function. Replace the elements of the array at run time with the Replace Array Subset function.

Optimizing Reflective Memory

Improve the use of your reflective memory usage by reducing the dynamic data size, creating channel mappings between targets, and using data channels selectively with non-VeriStand systems.

Before you begin, add a reflective memory network.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. To optimize your reflective memory performance, complete any of the following tasks.

Goal	Tasks	Rationale
Reduce the dynamic data size of the data sharing network to 0.	a. Click **Data Sharing Network**.	Dynamic Data Size specifies the number of channels in reflective memory to reserve for dynamically mapping

Goal	Tasks	Rationale
	b. On the Data Sharing Network configuration page, enter the **Dynamic Data Size** as 0.	channel data at run time. Reflective memory can negatively impact performance. Reducing the number of channels to 0 avoids decreasing performance. **Note** This is only recommended for systems that do not execute stimulus profiles or perform data logging that references channels across multiple targets.
Create channel mappings between targets in the system definition file.	a. Create channel mappings between targets. b. Prevent reflective memory used by VeriStand from overlapping with any non-VeriStand traffic on the bus. a. Click **Configure Mappings**. b. In the System Configuration Mappings dialog box, click the **Network** drop-down and select **Reflective**	Rather than using data channels to read and write data between targets in a system definition, create channel mappings between targets. When you configure channel mappings, VeriStand uses optimized reflective memory.

Goal	Tasks	Rationale
	Memory Network . c. Click **OK**. d. Click **Data Sharing Network** » **Reflective Memory Network**. e. On the Reflective Memory Network Configuration page, enter a **Start Memory Address** and **Maximum End Address Block** to limit the range of addresses the systems can use.	
When sending data from a VeriStand system to a non-VeriStand system, manually select channels to add to reflective memory and read those channels from the non-VeriStand system.	a. Click **Controller** » **Hardware** » **Chassis** » **Data Sharing** » **Reflective Memory**. b. Click **Export Channels**. c. In the Export Channels to Reflective Memory dialog box, select channels to export and click **OK**. d. Click **Data Sharing Network** » **Reflective Memory Network**. e. On the Reflective Memory Network	VeriStand adds the channels you export to the direct memory access (DMA) block write for the target. This reduces CPU usage. After selecting the channels to export, you must configure the non-VeriStand systems to read the memory addresses of the exported channels. The Export memory table to file option creates a text file that contains the memory addresses when you deploy the system definition file.

Goal	Tasks	Rationale
	Configuration page, click **Export memory table to file**.	
When a VeriStand system must read data from a non-VeriStand system, add data channels to only the targets that require the data.	▪ Click **Controller** » **Hardware** » **Chassis** » **Data Sharing** » **Reflective Memory**. ▪ Click **Add Data Channel**.	Data channels allow you to specify the reflective memory addresses a VeriStand target reads. However, you should only add data channels you intend to use. VeriStand reads the memory addresses one at a time. The more addresses VeriStand reads, the slower the performance.

4. Save the system definition file.

Tailoring Channels

Manage channels for scaling, faulting, and more.

System Channels

Use system channels to monitor system parts, such as the host computer, the target, and the VeriStand Engine, while it is deployed and running.

You can access the system channels in System Explorer by clicking **Targets** » **Controller** » **System Channels**.

Use the following table for more information on each system channel.

Channel name	Units	Description
Absolute Time	sec	The current date and time relative to 12:00 a.m., Friday, January 1, 1904, Universal Time [01-01-1904 00:00:00]. VeriStand coerces this value from a 128-bit value to double precision. This change might impact resolution.
Actual Loop Rate	Hz	The execution rate of the Primary Control Loop.
Alarm Status	Enum	The alarm's current state. ▪ 0: Disabled ▪ 1: Enabled ▪ 2: Tripped ▪ 3: Delayed Trip ▪ 4: Indicate
Analysis State	N/A	The state of the analysis sub-system on the real-time target.
Command Rate	N/A	The decimation of the Data Processing Loop.

Channel name	Units	Description
DAQ Error	N/A	The last reported error code from a DAQmx function call. The value zero indicates no error.
Delta T	sec	The period of the Primary Control Loop.
Detailed Tracing Flag	N/A	The Boolean value that specifies whether detailed execution tracing is enabled on the real-time target.
Failure Count	N/A	The failure count of current analysis settings.
Host IP	N/A	The 32-bit integer IP Address of the connected host. A zero value indicates no host is connected to the VeriStand Engine.
HP Count	N/A	The number of times the Primary Control Loop reported being late.
HP Loop Duration	ns	The duration of the Primary Control Loop.
HP Loop Wakeup Status	Enum	The wakeup status of the Primary Control Loop. ▪ 0: Normal ▪ 1: Aborted ▪ 2: Asynchronous wakeup ▪ 3: Timing source error ▪ 4: Timed loop error ▪ 5: Timeout
HP-LP Overwrite	N/A	The number of times the Primary Control Loop timed out writing channel data to the Data Processing Loop.

Channel name	Units	Description
HS TCP Overflow Count	N/A	The number of times the streaming data gets overwritten.
Iteration	N/A	The iteration count of the Primary Control Loop.
Last Late Iteration	N/A	The iteration count of the Primary Control Loop that last recorded a late count. If the Primary Control Loop has not recorded a late count, this value is -1.
Log Status	N/A	The state of the logging sub-system on the real-time target.
LP Count	N/A	The number of times the Data Processing Loop reported being late.
LP Data Count	N/A	The number of times the Data Processing Loop timed out writing channel data to the Communication Send Loop.
LP Loop Duration	ns	The duration of the Data Processing Loop.
Max Streamed Channels	N/A	The maximum number of channels that the VeriStand Engine can stream to the host.
Model Count	N/A	The number of times the models have not completed their execution in time.
Real-Time (RT) Sequence Command	Enum	The command that specifies how to halt execution of real-time sequences: ▪ 0: None ▪ 1: Stop All—Stops real-time sequences and skips to their clean-up sections.

Channel name	Units	Description
		▪ 2: Abort All—Terminates sequence execution without performing any clean-up tasks.
RT Sequence Count	N/A	The number of real-time sequences currently running.
RT Engine Build	N/A	A value representing the build number of the VeriStand Engine.
Streamed Channel Count	N/A	The number of channels that the VeriStand Engine is currently streaming to the host.
Streamed Waveform Count	N/A	The number of waveforms that the VeriStand Engine is currently streaming to the host.
System Command	Enum	The command that internally directs the real-time system with the following numeric values: ▪ 0: None ▪ 1: Restart System ▪ 2: Reset System ▪ 3: Shut Down System ▪ 4: Reboot System
System Reserved X	N/A	These channels are reserved for use by VeriStand.
System Time	sec	The relative system time of the VeriStand Engine according to the iteration count and Delta T of the Primary Control Loop.
TCP Data Packet Loss	N/A	The number of times the VeriStand Engine fails to send a data packet to the host. A non-zero number can indicate data loss in logged data. The value is

Channel name	Units	Description
		reset every time a workspace connects to the VeriStand Engine.
Thread Tracing Flag	N/A	The Boolean value that specifies whether thread execution tracing is enabled on the real-time Target.
Trace Buffer Size	N/A	The size in bytes of the execution trace buffer on the RT target.
Trace Enabled Flag	N/A	The Boolean value that specifies whether execution tracing is currently active on the real-time target. Set to 1 to start tracing. Set to 0 to stop tracing and write result to disk. Set to -1 to stop tracing and send the result to the host. If no host is present, it is logged to disk.
VI Tracing Flag	N/A	The Boolean value that specifies whether VI execution tracing is enabled on the real-time Target.
Watchdog Timer	ns	The amount of time since the Watchdog Timer Loop last executed. The Watchdog Timer Loop is set to execute at a rate of 10Hz.
WPL Error Code	N/A	The last error code encountered by the Waveform Processing Loop.
WPL Error Count	N/A	The number of times the Waveform Processing Loop encountered an error.
WPL Overflow Count	N/A	The number of times the Waveform Processing Loop attempts to write to a waveform read session within

Channel name	Units	Description
		the VeriStand Engine and times out.
WPL TCP Overflow Count	N/A	The number of times the Waveform Processing Loop attempts to write to the TCP loop for a host waveform-stream session and times out.

Scaling a Channel on Hardware Devices

Create scales to convert hardware channel unit measurements to transducer/actuator scaled units.

Before you begin, you might need to scale a channel when using a sensor that requires scaling from its voltage or current output into engineering units. Use your sensor's documentation for more information on creating a scale.

For input channels, scales convert read samples from the hardware channel to scaled units. Scales on output channels convert written samples to pre-scaled units of the channel.

1. Depending on your goal, complete one of the following tasks.

Goal	Task
Create a lookup table scale.	Map an array of pre-scaled values to an array of corresponding scaled values.
Create a polynomial scale.	Convert values using a polynomial equation with up to ten coefficients.
Create a thermocouple scale.	Convert values to Kelvins or degrees Celsius, Fahrenheit, or Rankine.
Import scales from another application.	Import a scale created by another system definition file (.nivssdf) or NI-DAQmx.
Import scale values from a text file.	Import scale table values or coefficients from a text file.

2. Map the scale to the channel.

Related tasks:

- Importing and Exporting Scale Mappings

Creating a Lookup Table Scale

Map an array of pre-scaled values to an array of corresponding scaled values.

VeriStand applies linear interpolation to values between the table values. This ensures that values scale proportionally.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Right-click **Scales** and select **Add Scale** » **Lookup Table**.
4. On the new scale's Lookup Table Configuration page, enter the scale **Name** and **Units** to associate with the scaled values.

   Note The units you enter will supersede the units associated with the channel.

5. Enter and edit pairs of pre-scaled values and corresponding scaled values in the table.
6. Save the system definition file.

VeriStand clips samples that are outside the maximum and minimum scaled values found in the table.

Creating a Polynomial Scale

Convert values using a polynomial equation with up to ten coefficients.

VeriStand requires two **direction coefficients** for a polynomial scale. They are a **forward** polynomial to convert pre-scaled values to scaled values and a **reverse** polynomial to convert scaled values to pre-scaled values.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.

3. Right-click **Scales** and select **Add Scale** » **Polynomial Scale**.
4. On the new scale's Polynomial Scale Configuration page, enter the scale **Name** and **Units** to associate with the scaled values.

> **Note** The units you enter will supersede the units associated with the channel.

5. In the **Coefficients** drop-down menu, choose one of the following directions for the coefficients table.

Direction	Description
Forward	Coefficients table represents a polynomial from pre-scaled to scaled values.
Reverse	Coefficients table represents a polynomial from scaled to pre-scaled values.

6. Enter **Coefficients** for an order of the polynomial in the table control, where the coefficients are a0, a1, a2,...an for your polynomial scale y = a0 + a1x + a2x2 + … + anx.

> **Note** For forward coefficients, y is the scaled data and x is the raw data. For reverse coefficients, x is the scaled data and y is the raw data.

7. Save the system definition file.

After creating a direction coefficient, you can click **Generate** to use regression analysis to estimate coefficients for the opposite direction. In the dialog box, enter a **Minimum** and **Maximum** value for the raw range and click **OK**.

Creating a Thermocouple Scale

Convert values to Kelvins or degrees Celsius, Fahrenheit, or Rankine.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.

3. Right-click **Scales** and select **Add Scale** » **Thermocouple Scale**.
4. On the new scale's Thermocouple Scale Configuration page, enter the scale **Name**.

> **Note** The units you enter will supersede the units associated with the channel.

5. Select the **Thermocouple Type**,**Temperature Units**, **CJC Type** (cold-junction compensation device), and **CJC Sensor** locations.
6. Save the system definition file.

Mapping Scales to Channels

Use VeriStand to apply a scale when you deploy and run the system definition.

You can map a scale to multiple channels, but a channel can only have one scale applied.
Scales can map to the following:

- FPGA channels
- Single-point DAQ channels
- XNET signals and raw data frames
- Custom device channels that you flag as scalable

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Create a scale.
4. Right-click **Scales** and select **Map Scales**.
5. In the Scale Mappings dialog box, select one channel from **Sources** and one or more channels from **Destinations**.
6. Click **Connect** to map the scale to the channel(s).

> **Note** Connect is only enabled if it is valid to apply the selected scale to the selected channel(s).

7. Save the system definition file.

The mapped channels appear in Mappings.

Related tasks:

- Importing and Exporting Scale Mappings

Importing and Exporting Scale Mappings

Save and import the mappings of scales to channels using text files.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Right-click **Scales** and select **Map Scales**.
4. In the Scale Mappings dialog box, complete the following tasks.

Goal	Task
Import a scale mapping.	Click **Import** and select the file you want to import. **Note** The system definition file that you import scale mappings into must have matching scale and channel names.
Export a scale mapping.	Click **Export** and select a destination to save the text file.

5. Save the system definition file.

Importing Scales from Another Application

Import a scale created by another system definition file (.nivssdf) or NI-DAQmx.

The scale names you import must not duplicate scale names already in the system definition file.

You can import NI-DAQmx custom scales saved in MAX on the host computer. You can also import NI-DAQmx custom scales exported from MAX as an INI file. VeriStand does not support map range scales from MAX.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Right-click **Scales** and select **Import Scales**.
4. In the Import Scales dialog box, select the type of scale to import.
5. Save the system definition file.

Importing Scale Values from a Text File

Import scale table values or coefficients from a text file.

Before you import a text file, format it.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Create a new lookup table scale or polynomial scale.
4. On the Lookup Table Configuration or Polynomial Scale Configuration page, click **Import**.
5. Select a text file that contains scale values and click **OK**.
6. In the Import Table Values or Import Polynomial Coefficients dialog box, import values from the text file, and click **OK**.

   Note VeriStand imports polynomial coefficient values depending on the value in the Coefficients drop-down menu.

7. Save the system definition file.

Maintain the text file to allow reuse in multiple system definitions.

Defining Scale Values in a Text File

Format scale values in a text file to import to VeriStand.

VeriStand converts non-numeric characters in the import file to 0.

1. Create a text file.
2. Depending on the type of scale you want to import, use the format example in your text file.

Scale Type	Format Example

Lookup Table	Either column can serve as the pre-scaled values or the scaled values.
	`Value1 delimiter Value1`
	`Value2 delimiter Value2`
	The file can contain more than two columns of values
	`Value1 delimiter Value2 delimiter Value3`

Polynomial Coefficients	`Order0Coefficient`
	`Order1Coefficient`
	`.`
	`.`
	`.`
	`Order9Coefficient`

After creating and formatting the text file, you can import it to VeriStand.

Faulting a Channel to a Specific Value

Fault a channel with the **Channel Fault Manager** to test the behavior of a system when a channel reaches a specific value.

Before you begin, you must add and connect to a system definition file in order to use VeriStand Editor tools.

1. In the VeriStand Editor, click **View** » **Channel Fault Manager**.
2. In the Channel Fault Manager pane, click **Add Fault**, and select the channel you want to fault.

   Note You can select multiple channels and fault all the selected channels to a single value.

3. In the **Value** field, enter the value to which you want to force the channel, and then click **OK**.
   Channel Fault Manager saves the value as a pending value.
4. Verify the value and click **Apply Pending Changes** to apply the value.

VeriStand forces the channel to the value you specified.

To update fault values, click in the **Pending Value** column, enter the new value, and then click **Apply Pending Changes**.

Adding a User Channel

Store a single value as a user channel to use as a variable in procedures, stimulus profiles, and other operations.

Extend user channel functionality by mapping a channel to other channels. For example, use a single user channel to start multiple models simultaneously by mapping the user channel to the Model Command channel of each model. Use this technique to perform actions such as triggering alarms and dynamically changing a value in a real-time sequence.

Note The user channel's location in the system definition cannot be changed after the user channel is created.

1. Launch your project in the VeriStand Editor.
2. Click **View** » **System Definition**.

3. **Optional:** In the System Definition Palette, right-click **User Channels** and select **New** » **Folder** to create a sub-folder for your user channels.
4. Right click **User Channels** or the sub-folder you created and select **New** » **User Channel**.

> **Note** To add more than one user channel, select **New** » **Multiple user channels** and use the Create Multiple User Channels dialog box to configure the user channels.

5. In the Create a New User Channel dialog box, enter a **Name**, **Description**, **Unit**, and **Initial Value**.
6. Click **OK**.
7. Save the system definition file.

On the Mapping Diagram, expand the User Channels node to find the new user channel.

> **Note** In the System Definition Palette, right-click a user channel to **Delete** or change its **Properties**.

Creating an Alias

Set an alternate name for channels in a system definition file.

Before you begin, you must add a model to the system definition.

The VeriStand Editor will reference the alias instead of the full channel path.

> **Note** The alias' location in the system definition cannot be changed after the alias is created.

1. Launch your project in the VeriStand Editor.
2. Click **View** » **System Definition**.

3. **Optional:** In the System Definition Palette, right-click **Aliases** and select **New** » **Folder** to create a sub-folder for your alias.
4. Navigate to the channel you want to create an alias for.
5. Right click the channel and select **Create alias from selection**.
6. In the Select Folder for New Aliases dialog box, select the location you want the alias to appear in the system definition and click **OK**.
7. **Optional:** Configure the alias.
    a. Right click the alias you created and click **Properties**.
    b. In the Properties dialog box, enter a **Name** and **Description**.
    c. Select a new linked channel.
    d. Click **OK**.
8. Save the system definition file.

On the Mapping Diagram, expand the Aliases node to find the new alias.



Note In the System Definition Palette, right-click an alias to **Delete** it.

Mapping Channels and Aliases

Connect channels or aliases to one another.

Before you begin, add a channel or alias to the system definition.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, double click the system definition file (.nivssdf) to open the **Mapping Diagram**.
3. Click a channel or alias terminal and drag a wire to another channel or alias terminal to map them.



Note You can create mappings between terminals with matching names by right-clicking two nodes and selecting **Automap**. If the direction of the mapping cannot be automatically determined, use

> the Automap dialog box that appears to specify the source of the mappings.

4. Click **File** » **Save all**.

You can remove a mapping by selecting and deleting the wire.

Adding a Calculated Channel

Produce a new value based on calculations performed on other channels in the system using the **Mapping Diagram**.

1. Open VeriStand.
2. In the VeriStand Editor, click **Project Files**.
3. Right-click your system definition file (.nivsdf) and select **Open**.
4. Click the **Software** palette and drag one of the following calculated channels onto the diagram.

Calculated channel	Description
Acceleration	Calculates the acceleration and velocity of the input channel.
Average	Calculates the average value of the input channel.
Conditional	Compares X with Y based on the configured condition. This channel outputs W for true and Z for false.
Formula	Calculates the result of the formula you specify.
Lowpass Filter	Applies a lowpass Butterworth filter to the input channel.
Maximum	Compares two values and returns the larger value.
Minimum	Compares two values and returns the smaller value.
Peak and Valley	Calculates the peak, valley, and offset of a cyclical waveform produced by the input

Calculated channel	Description
	channel. This calculation is performed by running the incoming value through a lowpass Butterworth filter and comparing the filtered value to previous maximum and minimum values.

5. Use the **Configuration pane** to configure the calculated channel.
6. Save the system definition file.

Formula Calculated Channel Options

Use standard math operators and functions to create variables that customize calculated channels.

All formulas must adhere to the following rules:

- Every formula must contain a reference to at least one variable.
- Variables must be contained in brackets. For example, [variable].
- Element-wise calculation is done when one of the operands is a variable or the parameter is a variable.
- Decimal separators on literal numbers must be periods. Alternative separators, such as commas, are not supported.

Refer to the following table of supported formula formats for examples that you can use to configure a calculated variable.

Formula format	Description	Example
[**<variable>**] + x	Element-wise arithmetic addition	[Var0] + 1
[**<variable x>**] + [**variable y>**]		[Var0] + [Var1]
[**<variable>**] - x	Element-wise arithmetic subtraction	[Var0] - 1
[**<variable x>**] - [**<variable y>**]		[Var0] - [Var1]
[**<variable>**] * x	Element-wise arithmetic multiplication	[Var0] * 5
[**<variable x>**] * [**<variable y>**]		[Var0] * [Var1]
[**<variable>**] / x	Element-wise arithmetic division	[Var0] / 5
[**<variable x>**] / [**<variable y>**]		[Var0] / [Var1]
function([**<variable>**])	Apply the function to the variable	sin([Var0])

Formula format	Description	Example
x + y * z / [**<variable>**]	Arithmetic equation	2 + 3 * 4 / [Var0]
(x + y) * z / [**<variable>**]	Arithmetic equation	(2 + 3) * 4 / [Var0]
-[**<variable>**]	Unary minus	-[Var0]
+[**<variable>**]	Unary plus (a no-op)	+[Var0]
[**<variable>**]^x	Power operator	[Var0]^3

Note The data type for all numerics is double-precision, floating-point.

Refer to the following table of supported functions and operators for examples of formula elements that you can use to configure a calculated channel.

	Formula element	Definition
Supported Function	sin(x)	Calculates the sine of x. Enter x in radians.
	cos(x)	Calculates the cosine of x. Enter x in radians.
	tan(x)	Calculates the tangent of x. Enter x in radians.
	asin(x)	Calculates the inverse of sine of x. The result is in radians.
	acos(x)	Calculates the inverse of cosine of x. The result is in radians.
	atan(x)	Calculates the inverse of tangent of x. The result is in radians.
	abs(x)	Returns the absolute value of x.
	rand()	Generates a random number between 0 and 1.
	exp(x)	Computes the value of e raised to the x power.
	sqrt(x)	Calculates the square root of x.
	sign(x)	Returns 1 if x is greater or less than 0. Returns 0 if x is equal to 0.

	Formula element	Definition
	int(x)	Calculates the integer value of x.
Supported Operator	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
	( )	Parenthesis. The contents are evaluated first
	^	Calculate the base raised to the power of the exponent

Visualizing System State

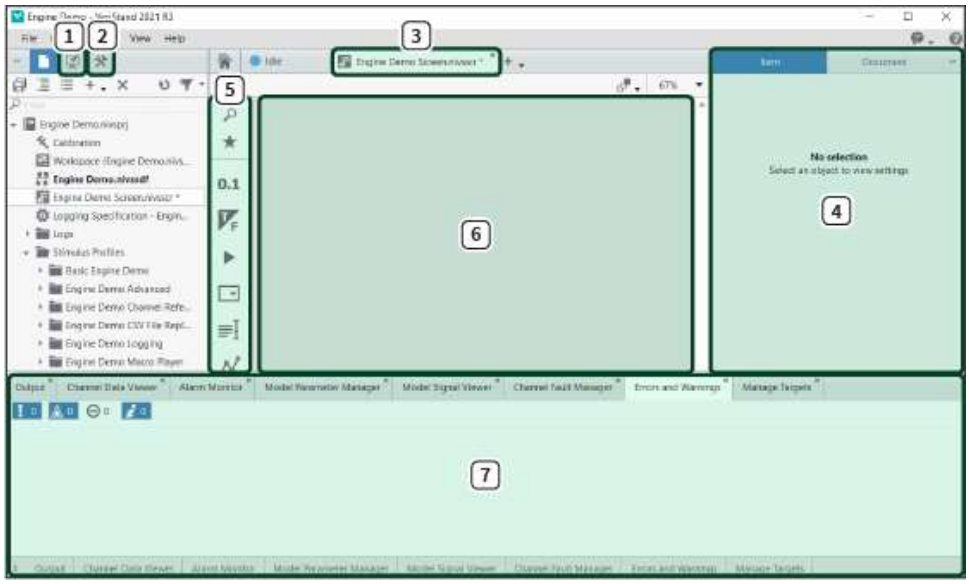
Create interfaces that an operator can use to interact with a VeriStand project.

User interfaces, called **screens**, allow an operator to easily control and monitor specific channels within a system definition at run-time. Use the VeriStand Editor to modify and operate screens.

Note For information on legacy interfaces, refer to Running the VeriStand Workspace.

Viewing, Creating, and Interacting with Screens

Create screens and projects in the **VeriStand Editor** to view, create, and interact with user interfaces.
The following image highlights the parts of the editor you use to open, access, and interact with screens and projects.



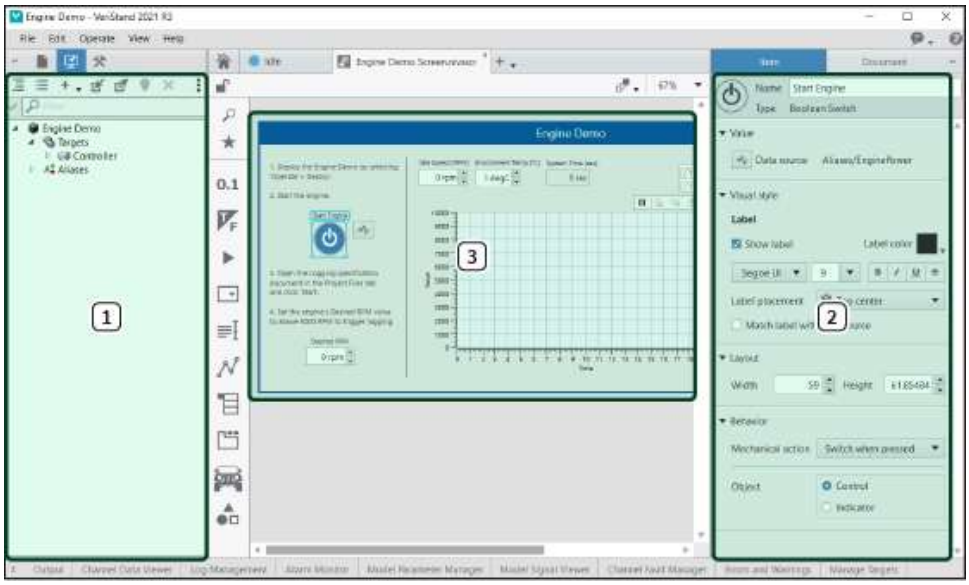
1	**Project Files**—Use this tab to open, create, and organize all the files within a project.

2	**Tools Launcher**—Use this tab to launch various tools for interacting with your project while it runs.
3	**Document**—After you open one or more screens, click one of these tabs to access the contents of a single screen.
4	**Configuration pane**—Use this pane to edit various aspects of your current project.
5	**Palette**—Organized hierarchy of all the controls and indicators you can add to the screen. By adding controls and indicators to the screen, you can modify the user interface an operator uses to interact with a project.
6	**Screen**—Use this area to view and modify the contents of a screen.
7	**Tools pane**—Use this pane to access various tools for interacting with your project while it runs. You can also use this pane to view errors and warnings that occur in your project.

Adding and Configuring Components of a Screen

Add controls and indicators to a Screen to modify the user interface of a VeriStand project.

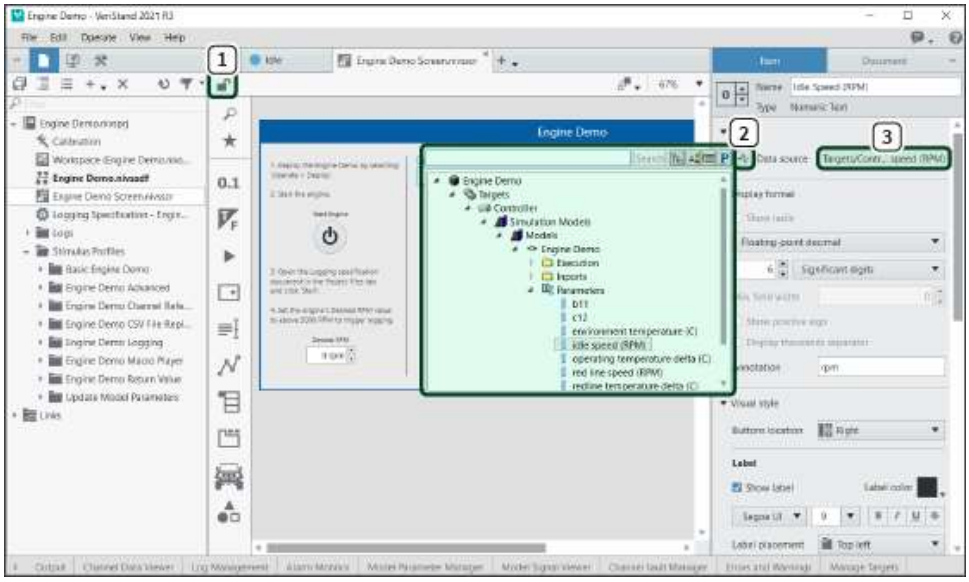
The following image highlights the parts of the VeriStand Editor you use to add and configure controls and indicators.



1	**System Definition palette**—A palette that displays the targets and channels in the current system definition file. You can drag channels from the System Definition palette and drop them onto the screen to add controls and indicators for those channels. VeriStand selects an appropriate control or indicator type for the channel.
2	**Controls and indicators**—Objects you add to the screen to either enter or view data from the system definition. Each control or indicator can accept or display a single type of data, such as a number, a TRUE/FALSE value, or a text string.
3	**Item tab**—Collection of options for customizing a selected control or indicator. The options on the **Item** tab appear only when you select a control or indicator on a screen.

Configuring Controls and Indicators to Send and Receive Data

Configure a screen to send a receive data by mapping controls and indicators to a channel in the system definition.
A screen contains various components that allow a user to send data to a system definition as well as receive data from it. These components are called **controls** and **indicators**.



1	**Lock**—Locks the screen, making it easier to send and receive data with controls and indicators by preventing the controls from being edited while you operate them. **Note** You can also operate controls while locked. However, you have to select the control before you can operate it.
2	**System definition tree**—Displays the configuration tree of the current system definition. You can navigate the tree to select the channel you want to map with the control or indicator.

3	**Data source**—Displays the path of the channel currently mapped to the control or indicator. The channel path is only visible when you select a control or indicator on the screen.

VeriStand Editor Tools

The **VeriStand Editor** provides tools that allow you to manipulate a project without having to open the system definition.

Goal	Task
Log data	Create logging specification (.nivslspec) files with Log Management to configure and execute host-side data logging.
View model values	Check signal values with the Model Signal Viewer without importing the signal as a channel or editing the system definition file.
Fault a channel to a specific value	Fault a channel with the Channel Fault Manager to test the behavior of a system when a channel reaches a specific value.
Import and manage batches of model parameters	Import model parameters with the Model Parameter Manager to apply values defined in an external .txt file to a model.
View channel values at run time	Monitor channels with the **Channel Data Viewer** to view the current values of several channels together.

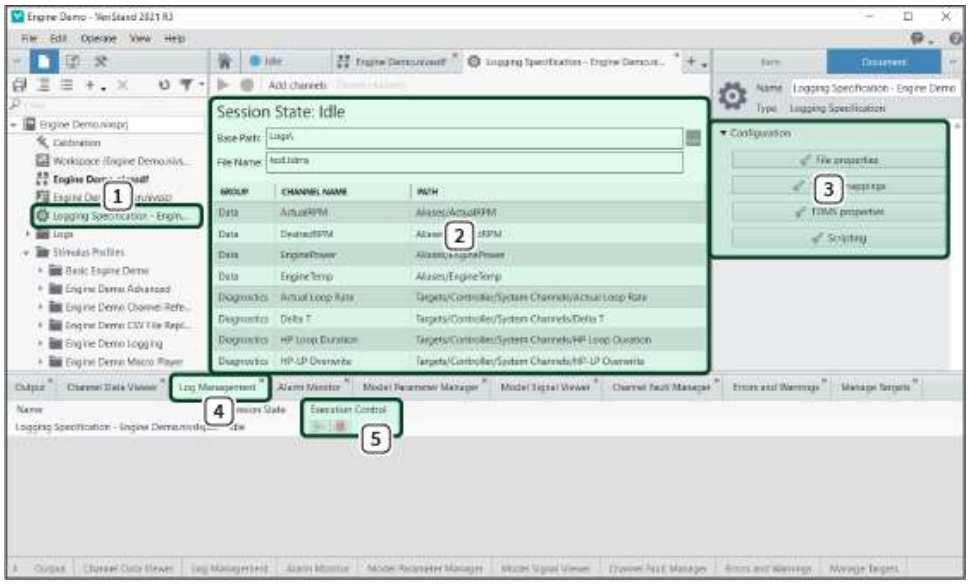
Logging Data with the VeriStand Editor

Create logging specification (.nivslspec) files with **Log Management** to configure and execute host-side data logging.

You can add and configure logging specifications after connecting to a deployed system definition. This enables you to adjust which channels to log, when to start and stop logging, and the format of log files produce by the control. You can also configure logging specifications to automate post-processing actions, such as merging log data or producing reports, for log files produced during a log session.

1. In the VeriStand Editor, click **View** » **Tool Launcher** » **Log Management**.

2. Use the following sections of the VeriStand Editor to add and configure a logging specification file.



1	**Logging Specification**—The logging specification file appears in the **Files** pane under the current project. Select the logging specification file you want to configure. To create a new logging specification file, right-click the project file and select **Add New** » **Logging Specification**.
2	**Logging specification file screen**—When you open a logging specification file in VeriStand, the file is displayed in a screen. Use the screen to add channels to log.
3	**Configuration**—Contains various options for configuring the log session, such as when to start and stop logging, how to save the log files, and options for post processing of log files.
4	**Log Management**—Displays all of the logging specification files in your project. Use the Log Management tool to view the

	state of logging specifications and to manually start and stop logging sessions. To display this tab, click **Log Management** in the **Tool Launcher**.
5	**Execution Control section**—Contains **Start** and **Stop** options for the selected logging specification file. Click **Start** to begin a log session. If you specified to log immediately, logging begins after you click **Start**. If you created a start trigger, logging begins after the specified condition is triggered.

3. Save the logging specification file.

Viewing Model Values in the VeriStand Editor

Check signal values with the **Model Signal Viewer** without importing the signal as a channel or editing the system definition file.

1. In the VeriStand Editor, click **View** » **Tool Launcher** » **Model Signal Viewer**.
2. In the Model Signal Viewer tab, click **Select Signals**.
3. In the Select Signals dialog box, click the signals you want to view and click **OK**.

> **Note** Only select signals that you need immediate information on. Adding too many signals will decrease the VeriStand Engine's execution loop rate. For every 500 signals displayed, there is an expected 1% drop in performance.

The **Path** and **Value** of the signals you selected will appear in the Model Signal Viewer tab.

Importing and Managing Batches of Model Parameters with the VeriStand Editor

Import model parameters with the **Model Parameter Manager** to apply values defined in an external .txt file to a model.

Before you can work with model parameters, you must deploy and connect to a system definition and understand model parameter syntax.

Maintaining model parameter values in external files allows you to quickly switch between batches of test parameters without manually entering the values. This tool is also useful for managing multiple parameters from a single interface.

Setting the value of a parameter is a two-step process: importing or calculating new values, and then applying the new values.

1. In the VeriStand Editor, click **View** » **Model Parameter Manager**.
2. In the Model Parameter Manager pane, click **Import File**.
3. In the **Configure Parameter Import** dialog box, select **Parameter File Path** and browse to the .txt file that contains parameter values.
    a. If aliases define the parameters in the file, select **Alias File Path** and browse to the alias file.
4. Click **OK** to import the values from the file as pending values.
5. Verify the values and click **Apply Pending Changes** to apply the values.

To update model parameter values, click in the **Pending Value** column, enter the new value, and click **Apply Pending Changes**.

Supported Syntax for Model Parameter Manager

The **Model Parameter Manager** supports simple text files (.txt) as model parameter files.

The Model Parameter Manager allows text files that conform to the model parameter file format that other VeriStand features support.

One limitation of using model parameter text files is that the manager imports and sets the result of the expression rather than the expression itself. The following parameter definitions demonstrate this limitation.

```
a 10
b a * 2
```

In this example, the Model Parameter Manager sets the value of b to 20, not a * 2. Changes to the value of a do not affect b because the original expression is no longer valid.

Viewing Channel Values at Run Time

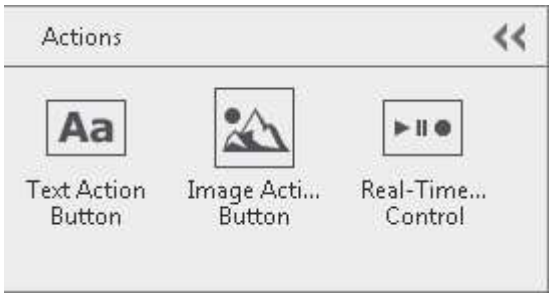
Monitor channels with the **Channel Data Viewer** to view the current values of several channels together.

1. In the VeriStand Editor, Click **View** » **Channel Data Viewer**.
2. In the Channel Data Viewer tab, click the **Channel Filter** pull-down menu and select the type of channels to display.

Type of channel	Description
Channels	Displays all channels in the system definition file.
Parameters	Displays model parameters.
Writable	Displays all writable channels. To update the value of a writable channel, click inside the **Pending Value** cell for the channel, enter the desired value, and then click **Apply Pending Changes**.
Aliases	Displays all aliases defined in the system definition file.
Custom	Allows you to display channels of your choosing. To select the channels to display, click **Select Channels** and use the system definition configuration tree to select the channels to display.

Actions Controls

Launch EXE, BAT, or real-time sequence files from the screen.



Launching an EXE or BAT File from the Screen

1. Add an action button to your screen.
2. Select the action button and use the **Action Configuration** section Item tab to configure the action button.
   You can define arguments to pass to the EXE or BAT file when you run it. For example, if you want to launch a web browser, you can specify a specific web page to launch.
3. To launch the EXE or BAT file, switch your screen to operate mode and press the action button.

You can also use the action buttons to open file formats other than EXE and BAT. VeriStand calls the default program to open the file you specify.

Configuring and Running a Real-Time Sequence

You can use any of the Actions controls to configure and run a real-time sequence. However, the Real-Time Sequence control gives you more control and oversight over sequence execution. You can pause and stop a sequence, view the sequence time, and visualize the return value of the sequence, which often indicates a pass/fail result for your test.

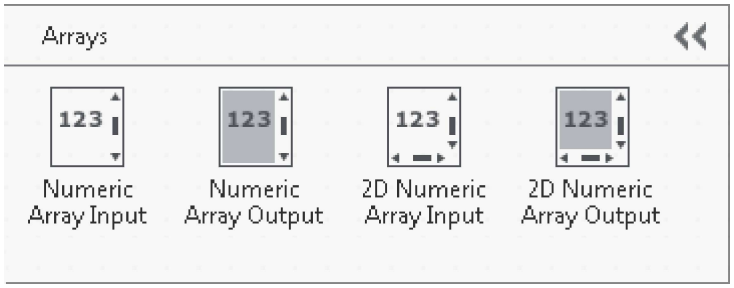
Complete the following steps to configure and run a real-time sequence by using the Real-Time Sequence control:

1. Add the Real-Time Sequence control to your screen.

2. On the Item tab, click **Sequence Path** to specify the real-time sequence file.
3. Click **Target** to select the target on which you want to run the real-time sequence.
4. Click **Configure Parameters** to update the parameters in the real-time sequence file.
5. Switch the screen to operate mode and click **Run** on the Real-Time Sequence control to run the real-time sequence.

Array Controls

Enter or display array data.



What Is an Array?

An **array** consists of elements and dimensions. **Elements** are the data that make up the array. A **dimension** is the length, height, or depth of an array. An array can have one or two dimensions.

How Do I Display an Array?

Drag an existing array channel from the System Definition palette onto the screen. The control will automatically populate with array values. The dimension and size of the array will match the channel.

How Do I Find an Array Element?

To locate a particular element in an array, you must use indices. Each dimension in the array has an index. Indices let you navigate through an array and retrieve elements, rows, columns, and pages from an array on the diagram.

Note Array indices are zero-based. The index of the first element in the array, regardless of dimension, is zero.

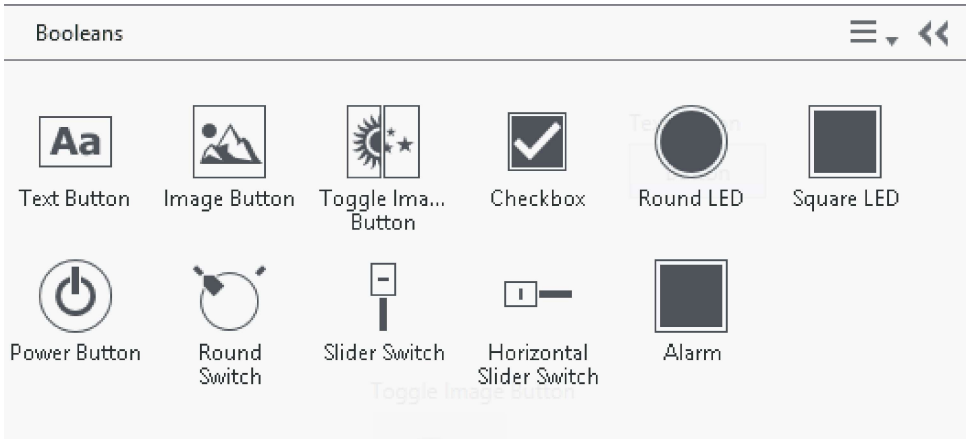
Automotive Controls

Create an automotive dashboard.



Booleans Controls

Enter or display true and false data.

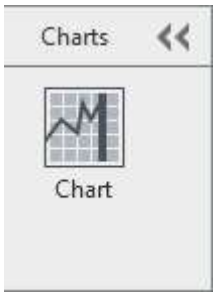


Which Boolean Control Should I Use?

Boolean Type	Use
Buttons	Enter true or false input.
Checkbox	Select an item or items from a list.
LEDs	Displays true or false data. You must select **Show Content** on the configuration pane for content to appear on the LED.
Power Button	Sets power state.
Switches	Enter true or false input in an application.
Alarm	Displays alarm data.

Charts Controls

Display data from channels or waveforms in the system definition file.



Configuring a Scale

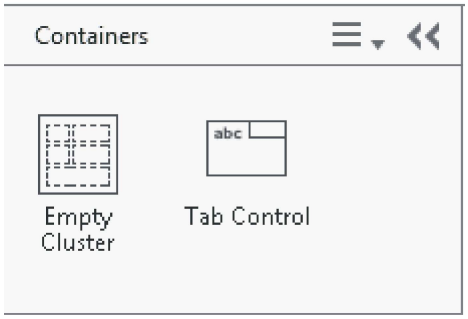
1. Select the Chart control on the Screen.
2. On the Item tab, select **Scale Legend** in the **Visual Style** section. The Scales box appears on the screen.
3. Select the scale you want to configure.
4. Update the settings for the scale on the Item tab.

Adding a Cursor

1. Select the Chart control on the Screen.
2. On the Item tab, select **Cursor Legend** in the **Visual Style** section. The Cursors box appears on the screen.
3. Click **New Cursor** to add a cursor.

Containers Controls

Group and organize controls and indicators.

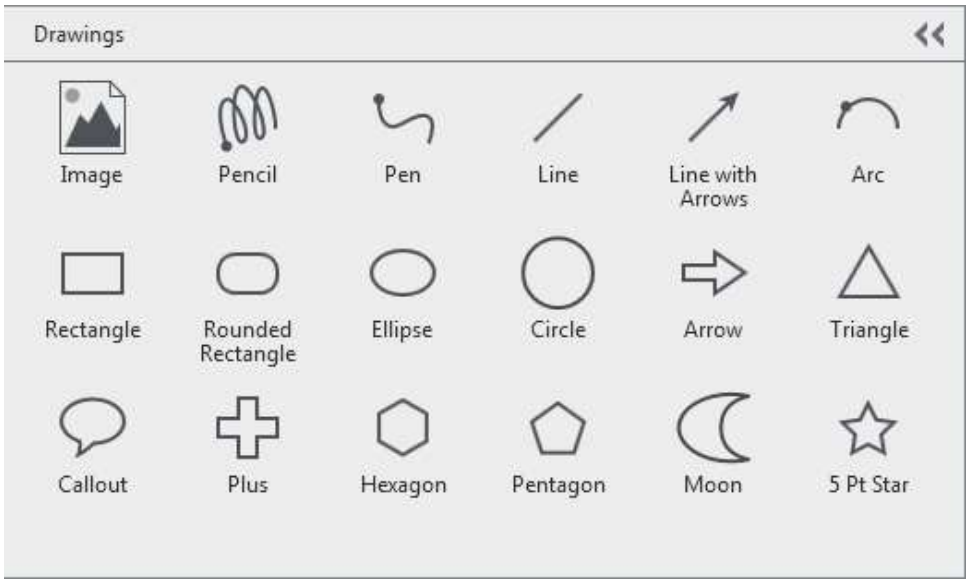


When Should I Use a Tab Control?

Tab controls are useful when you have several panel objects that are used together or during a specific phase of operation. For example, you might have an application that requires the user to first configure several settings before a test can start, then allows the user to modify aspects of the test as it progresses, and finally allows the user to display and store only pertinent data.

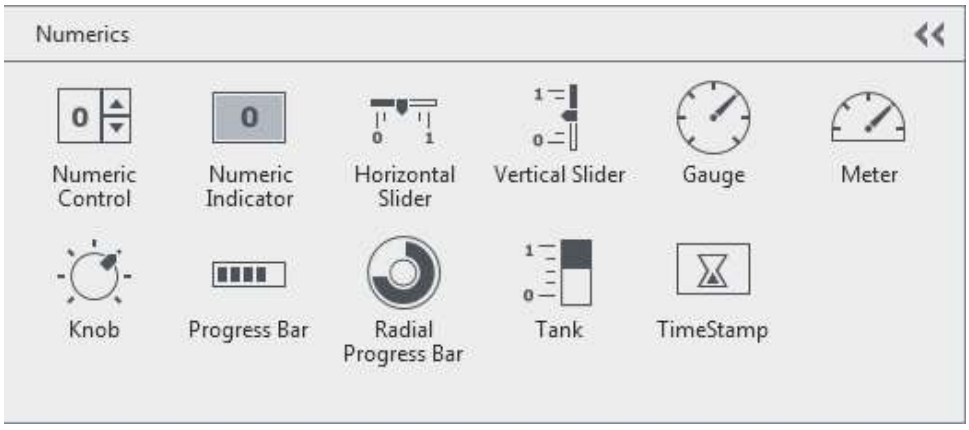
Drawings Controls

Decorate the user interface with images, lines, or shapes.



Numerics Controls

Enter and display numeric data.



Which Numerics Control Should I Use?

Control	Use
Numeric control and indicator	Enter or display numeric data.

Control	Use
Sliders	Display numeric data in a vertical or horizontal slide with a customizable scale and a pointer that helps you see the exact value.
Gauge, Meter, and Knob	Enter or display numeric data in a rotary scale.
Tank	Display numeric data in a vertical slide that resembles a real tank or thermometer instrument.
Timestamp	Enter or display a time and date value.
Progress bars	Show progress in a vertical bar or circle.

What Is the Difference between Significant Digits and Digits of Precision?

Significant digits specifies the number of significant digits to display in the numeric control.

Digits of precision specifies how many digits to display after the decimal point.

How Do I Use, View, and Store Absolute Time?

Use the Timestamp control to use, view, and store absolute time with high precision. This data type can accurately store 18 digits of precision in whole seconds and 19 digits of precision in fractions of a second. The Timestamp control displays values in local time. However, the timestamp wire data type stores values in UTC.

Although you can use a numeric control to display timestamp values, the numeric control holds a relative quantity. The Timestamp control holds an absolute quantity.

What Characters Do Numerics Controls Accept?

Character(s)	Description
Hexadecimal digits	0 through F
Octal digits	0 through 7
Binary digits	0 and 1
Decimal digits	1, 1.0, 2, 3.5, and so on
.	Decimal point
+	Positive symbol

Character(s)	Description
-	Negative symbol
E or e	For scientific or engineering notation format
Infinity	Infinity
NaN	Not a number
/	For use in absolute time format
:	For use in absolute time format
AM, am, PM, pm	For use in absolute time format
SI prefixes	
y	yocto (10)
z	zepto (10)
a	atto (10)
f	femto (10)
p	pico (10)
n	nano (10)
u	micro (10)
m	milli (10)
c	centi (10)
d	deci (10)
da	deka (10)
h	hecto (10)
k	kilo (10)
M	mega (10)
G	giga (10)
T	tera (10)
P	peta (10)
E	exa (10)
Z	zetta (10)
Y	yotta (10)

Pull Downs Controls

Create a list that a user can cycle through to make selections.

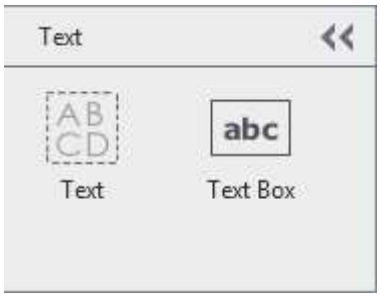


How Do I Assign Specific Values to the Pull Downs?

1. Select the control to which you want to assign specific values.
2. On the configuration pane, assign and arrange specific values in the **Items** section.

Text Controls

Create text entry boxes and labels.



Running the VeriStand Workspace

Launch the **Workspace** to run a project, view and modify the user interface, and to perform operations such as monitoring alarms, viewing channel data, scaling and calibrating channels, and running stimulus profiles.

1. Open a project in VeriStand.
2. In the **Project Files** tab, double-click **Workspace**.

3. Depending on your goal, complete any of the following tasks in the Workspace.

Goal	Task
Adding and configuring controls and indicators	Use the Workspace Controls palette to add and configure controls and indicators to create a user interface.
Modifying control mappings at run time	Change the channel mapping of a control or indicator while the Workspace runs.
Calibrating a hardware channel at run time	Use the **Channel Calibration** tool to calibrate hardware channels by adjusting the values they return to known values while a system definition runs.
Using channel value forcing	Use the **Channel Fault Manager** tool to test the behavior of a system when a channel reaches a certain value.
Logging test results with stimulus profiles	Use the **Stimulus Profile Editor** to create and execute a stimulus profile on your host machine to log test data acquired from real-time sequences performed on a target.
Recording commands sent to the target	Use the Macro Recorder tool to record commands, such as the setting of channel or parameter values, model execution states, and fault or alarm values, that VeriStand sends to the target and save them to a macro (.nivsmacro) file.
Playing back commands sent to the target	Use the Macro Player tool to review the commands that VeriStand sent to the target using the macro (.nivsmacro) file you recorded with the Macro Recorder tool.
Setting model parameter values in the Workspace	Use model calibration controls in the Workspace to view and modify the values for any model parameters in the system definition.
Importing and managing batches of model parameters	Use the Model Parameter Manager tool to import and apply model parameter values defined in external .m or .txt files to a model.

Goal	Task
Displaying waveform data in a graph	Use a waveform graph control to display data from one or more waveforms in the system definition file to monitor and verify acquired data.
Enhancing your Workspace to view data	Use other tools, such as the Alarm Monitor, TDMS File Viewer, and XNET Bus Monitor, to view data in the **Workspace**.
Configuring and Executing host-side logging	Use the **Data Logging** control to adjust log times, better format data files, select start and stop times, and more at run time.

Differences between Workspace and VeriStand Editor

The **Workspace** and VeriStand **Editor** contain features that are unique to their respective screen files.
The screen files for each application are not compatible with each other. You cannot convert files from one application to the other.
Use the following table to determine the screen application to use based on the task you want to accomplish.

Task	Supported Application
▪ Natively design a user interface at run-time. ▪ Access scripting capabilities. ▪ Operators can launch the application without opening VeriStand. ▪ View screens across multiple monitors. ▪ Use core controls such as rings and tabs. ▪ View the project configuration tree within the application.	VeriStand Editor
▪ Access the Channel Calibration tool. ▪ Access macro recording and playback capabilities.	Workspace

Task	Supported Application
▪ Access custom objects. ▪ Access API to automate the application. ▪ Use services to launch application tools upon connecting to a target. ▪ Sync services with the launch of the application window. ▪ Access add-ons. ▪ Map a vector channel to controls or indicators. ▪ Access model controls. ▪ Access a built-in TDMS file viewer. ▪ Use the application for benchmarking CPU and debugging the Console Viewer that ships with the Workspace. ▪ Access a built-in tool to launch the NI-XNET Bus Monitor. ▪ Access a tools menu that you can edit from Project Explorer.	
You can map execution channels to regular controls to achieve the same functionality as a model control within the VeriStand Editor. If you double-click a TDMS file in the VeriStand Editor, the file launches in your default viewer. You can use the browser-based console viewer in the VeriStand Editor. You can use an action button to launch the bus monitor application in the VeriStand Editor. You can use action buttons to launch custom tools from the VeriStand Editor.	

Related reference:

- VeriStand Environment

Adding and Configuring Controls and Indicators

Use the **Workspace Controls** palette to add and configure controls and indicators to create a user interface.

1. Open the Workspace.
2. Select **Screen** » **Edit Mode** to display the **Workspace Controls** palette and alignment grid.

3. Click the **Workspace Controls** palette and drag a control or indicator onto the alignment grid.
4. In the **Item Properties** dialog box, configure the control or indicator.

> **Note** If the Workspace contains simple or FFT graph controls and the project that owns the screen file connects to the VeriStand Gateway running on a remote target, you must set the **TTL** value to 128 on the **Port Settings** page of the Options dialog box. Otherwise, these graph controls will not update.

Modifying Control Mappings at Run Time

Change the channel mapping of a control or indicator while the **Workspace** runs.

1. Right-click a control or indicator.
2. In the Item Properties dialog box, click **Browse**.
3. In the Browse dialog box, select a channel in the configuration tree, and click **OK**.
4. In the **Control Label** field, rename the control or indicator, and click **OK**.

Calibrating a Hardware Channel at Run Time

Use the **Channel Calibration** tool to calibrate hardware channels by adjusting the values they return to known values while a system definition runs.

Before you begin, verify that you have the tool in your Workspace **Tools** menu. If you do not, refer to Adding a Standard or Custom Tools Menu Item.

You can calibrate channels according to polynomial equations you define, custom-defined channels marked as scalable, as well as FPGA and single-point DAQ channels in running system definitions.

> **Note** If a channel has a scale mapped to it, VeriStand applies the scale first and applies the calibration to the scaled channel values second.

VeriStand supports polynomial calibrations with up to ten coefficients. A simple polynomial calibration is a linear calibration, y = mx + b, where m is the first-order coefficient (a1) that serves as the scale and b is the constant (a0) that serves as an offset.

1. In the Workspace, select **Tools** » **Channel Calibration** to launch the tool.

   Note The name of this menu item might differ depending on how you named it in the Tools Properties dialog box.

2. In the Channel Calibration tool under **Scalable Channels**, select a channel you want to calibrate.
3. Click **Next**.
4. On the User Information page, enter metadata about the specific calibration procedure you want to perform.

   Note This information is used to store the history of calibrations performed for a channel. You can view the information by clicking **View History** on the main page of the **Channel Calibration** tool.

5. On the Polynomial Calibration Details page, enter coefficients to define the polynomial equation that will calibrate the channel.

   Note You can click **Build Table** and then complete a procedure for automatically calculating polynomial coefficients that best fit raw sensor values to known input values.

6. Click **Finish**.

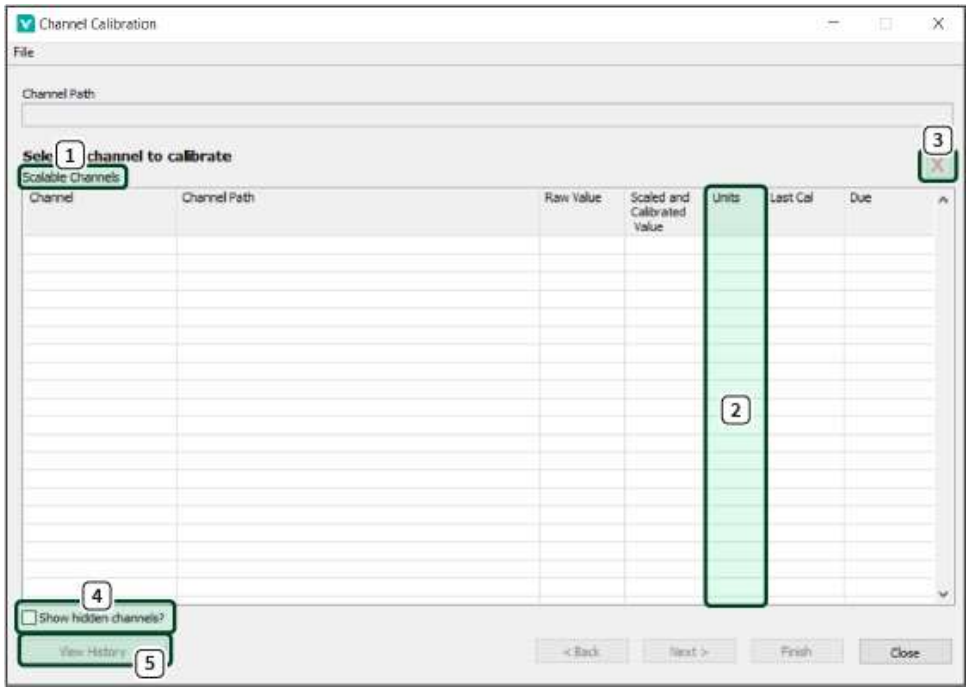
The Scaled and Calibrated Value column in the Scalable Channels table displays channel values after the calibration.

When you close the **Channel Calibration** tool, the calibration remains applied to the channel. Even if you close the Workspace or undeploy and redeploy the system definition, the calibration remains active until you remove it.

You also can use the Calibration VIs in the LabVIEW Execution API to automate applying calibrations and reading raw values from hardware channels.

Channel Calibration

The **Channel Calibration** tool displays channel information such as channel names, paths, values, and calibration dates you use to calibrate your hardware. The following image highlights the layout of the **Channel Calibration** tool you will interact with to calibrate values received from a hardware device.



1	**Scalable Channels**—Displays both the **Raw Value** the channel acquires or generates and the **Scaled and Calibrated Value**. Values in the latter column reflect both scales mapped to the channel in the system definition and calibrations you apply to the channel with this tool.
2	**Units**—Displays the units associated with the channel. If you map a scale to the channel, the units associated with the scale display.

3	**Delete Calibration**—Removes a selected channel. The button is disabled when a selected channel does not have a calibration applied to it.
4	**Show hidden channels?**—Displays hidden channels available for calibration in the **Scalable Channels** table.
5	**View History**—Displays information about previous calibrations performed for a channel, including the date and time it started and the calibration coefficients.

Using Channel Value Forcing

Use the **Channel Fault Manager** tool to test the behavior of a system when a channel reaches a certain value.

Before you begin, verify that you have the tool in your Workspace **Tools** menu. If you do not, refer to Adding a Standard or Custom Tools Menu Item.

1. In the Workspace, select **Tools** » **Channel Fault Manager** to launch the tool.

   Note The name of this menu item might differ depending on how you named it in the **Tools Properties** dialog box.

2. In the Channel Fault Manager tool, select a channel from the configuration tree and click **Add Fault**.
3. In the Fault Value dialog box, enter a fault value for the specified channel and click **OK**.

The faulted channel and value appear in the Channel Fault Manager tool.

Logging Test Results with Stimulus Profiles

Use the **Stimulus Profile Editor** to create and execute a stimulus profile on your host machine to log test data acquired from real-time sequences performed on a target.

If the Stimulus Profile Editor is not in the **Tools** menu, add it.

You can start multiple concurrent stimulus profile executions. Each stimulus profile execution performs sequential execution of one or more real-time sequences.

1. In the Workspace, select **Tools** » **Stimulus Profile Editor**.

   > **Note** The name of this menu item might differ depending on how you named it in the Tools Properties dialog box. The Stimulus Project Editor launches as a separate application and may take a few minutes to open.

2. On the **Start Page** tab, select **New Stimulus Profile**.
3. Configure your stimulus profile.
   a. From the Steps palette, drag steps to the relevant step group folders.
   b. From the Sequences palette, drag real-time sequences to the relevant group folders.
4. On the **Start Page** tab, select **New Real-Time Sequence**.
5. Configure your real-time sequence file.
   a. From the Primitives palette, drag statements to the real-time sequence.
   b. From the Sequences palette, drag sequences to the real-time sequence.
6. Save the real-time sequence.
7. In the Stimulus Profile File window, add a step to call the real-time sequence you just created from the stimulus profile.
8. Save the stimulus profile.
9. Click **Run**.

Once you have created a stimulus profile, you can deploy it to a real-time target based on your system definition. This profile is then run on the real-time target, and the current state of the profile is displayed in the Profile window on the host computer.

Recording Commands VeriStand Sends to the Target

Use the **Macro Recorder** tool to record commands, such as the setting of channel or parameter values, model execution states, and fault or alarm values, that VeriStand sends to the target and save them to a macro (.nivsmacro) file.

Before you begin, verify that you have the tool in your Workspace **Tools** menu. If you do not, refer to Adding a Standard or Custom Tools Menu Item.

> **Note** You can also use the Macro Recorder VIs to access the **Macro Recorder** and create macro files programmatically from LabVIEW. Use the Execution API to access the **Macro Recorder** from any .NET-compatible programming language.

1. In the Workspace, select **Tools** » **Macro Recorder** to launch the tool.

   > **Note** The name of this menu item might differ depending on how you named it in the Tools Properties dialog box.

2. In the **Macro Recorder** tool, click **Record** to begin recording.
3. Use controls on the **Workspace** to send commands to the target. The **Workspace Macro** list displays each command you send.
4. Click **Pause** to pause recording.

   > **Note** The Macro Recorder does not record any commands you send while it is paused. You can click **Resume** to continue appending commands to the same macro recording.

5. Click **Stop** to finish recording.
6. Click **File** » **Save As** to save the macro file.

You can use the **Macro Player** tool to play back macro files you create using the **Macro Recorder** tool.

Playing Back Commands Sent to the Target

Use the **Macro Player** tool to review the commands that VeriStand sent to the target using the macro (.nivsmacro) file you recorded with the **Macro Recorder** tool.

Before you begin, verify that you have the tool in your Workspace **Tools** menu. If you do not, refer to Adding a Standard or Custom Tools Menu Item.

> **Note** You can also use the Macro Player VIs to access the **Macro Player** and play back macro files programmatically from LabVIEW. Use the Execution API to access the **Macro Player** from any .NET-compatible programming language.

1. In the Workspace, select **Tools** » **Macro Recorder** to launch the tool.

   > **Note** The name of this menu item might differ depending on how you named it in the Tools Properties dialog box.

2. In the Macro Recorder tool, click **File** » **Open** and navigate to the macro file you want to play back.
3. Select a **Play Mode**.

Play Mode	Description
Ignore Timing	Plays back as fast as possible.
Use Timing	Plays back at the speed you recorded.

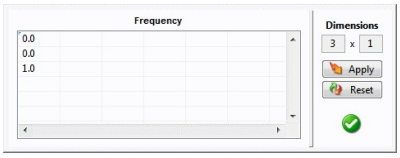
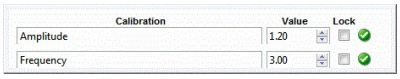
4. Click **Play**.
   The **Workspace Macro** list highlights each command as it is played back.
5. Click **Stop** to stop playback.

Setting Model Parameter Values in the Workspace

Use model calibration controls in the **Workspace** to view and modify the values for any model parameters in the system definition.

1. Open the Workspace.
2. Select **Screen** » **Edit Mode** to display the Workspace Controls palette and alignment grid.
3. Click the Workspace Controls palette.
4. Select **Model Calibration Control**.
5. Choose one of the following control types and drag it onto the alignment grid.

> **Note** Glyphs appear next to parameters to indicate whether their status is unlocked ( ) or locked ( ). The glyph appears on a control to indicate when you have made changes to a parameter but not applied them to the system.

Control Type	Description	Example
Medium Control	Medium controls are the basic numeric controls for modifying a single model parameter value. VeriStand applies value changes as soon as you enter them in a medium control	
Array Control	Array controls help manage the elements of parameters in vector form. Click **Apply** to commit the changes. Click **Reset** to replace any value changes you have not applied with the current system values.	
List Control	List controls provide access to multiple parameters. Select the parameters in the **Item Properties** dialog box for list controls.	

Control Type	Description	Example
	VeriStand applies value changes as soon as you enter them.	

6. Use the Item Properties dialog box to customize the control and click **OK**.

You can also lock the parameter value.

Importing and Managing Batches of Model Parameters in the Workspace

Use the **Model Parameter Manager** tool to import and apply model parameter values defined in external .m or .txt files to a model.

Before you begin, verify that you have the tool in your Workspace **Tools** menu. If you do not, refer to Adding a Standard or Custom Tools Menu Item.

Maintaining model parameter values in external files allows you to quickly switch between batches of test parameters without manually entering the values. This tool is also useful for managing multiple parameters from a single interface.
In order to set the value of a parameter, you must import or calculate the new values and then apply the new values.

1. Deploy the system definition with models to the target.

   Note You can only apply new values to deployed systems.

2. In the Workspace, select **Tools** » **Model Parameter Manager** to launch the tool.

   Note The name of this menu item might differ depending on how you named it in the **Tools Properties** dialog box.

3. In the Model Parameter Manager tool, click **Select Files**.
4. In the Select Calibration Files to Load dialog box, click **Add File**.
5. Browse to the file(s) that contain parameter values, and click **OK**.

> **Note** You can import parameter values from multiple model parameter files. However, only one file can define a specific parameter. Otherwise, VeriStand requires you to select the value to apply.

6. In the Select Calibration Files to Load dialog box, click **OK** to import the values from the files as new values.
7. In the Model Parameter Manager, review the **New Value** column for errors.

> **Note** If an error glyph (⚠) appears next to a **Parameter**, ensure your files use supported syntax.

8. Double-click a file in the **Selected Files** column to edit the contents of the file.
9. Click **Calc. New Values** to read or calculate new parameters values from files you loaded.

> **Note** This is useful for resetting parameters to their initial values.

10. Click **Apply New**.

Related tasks:

- Deploying the System Definition File to a Real-Time Target

Supported Formatting for Model Parameter Files

You need to format .txt and .m files correctly to use them in the **Model Parameter Manager** tool.

The following table displays the file types and how they should be formatted.

File Type	Description	Example Formatting	Limitations
.txt	Support for simple text files that conform to the model parameter file format that several	`a 10` `b a * 2`	The Model Parameter Manager imports and sets the result of the

File Type	Description	Example Formatting	Limitations
	other VeriStand features support. Refer to Supported Syntax in Model Parameter Files for more information about this type of file.		expression rather than the expression itself. The Model Parameter Manager sets the value of b to 20, not a * 2. Changes to the value of a do not affect b because the original expression is no longer valid.
.m	Support for .m model parameter files that perform one or both of the following actions: ▪ Assigns constant values to parameters. ▪ Defines parameter values as the result of simple expressions that include other parameters or variables defined in the .m file. This feature only supports simple assignments and expressions, not the entire .m file syntax.	`temp = 1;` `b = (temp * 3) / 2;`	▪ Expressions must depend on other parameters or variables. ▪ The names of parameters and variables used in expressions must start with a letter and contain only alphanumeric characters or underscores. Otherwise the Model Parameter Manager displays an error glyph (⚠).

Setting a Parameter Value Manually

Use the **Model Parameter Manager** tool to set a parameter value manually.

If the Model Parameter Manager tool is not in the **Tools** menu, add it.

To set the value of a parameter manually, you must set and then apply new values.

1. Deploy the system definition with models to the target.

   Note You can only apply new values to deployed systems.

2. In the Workspace, select **Tools** » **Model Parameter Manager** to launch the tool.

   Note The name of this menu item might differ depending on how you named it in the **Tools Properties** dialog box.

3. Double-click a parameter to launch the Edit Calibration Value dialog box.
4. Enter the new value you want to apply to the parameter in the **New Value** table.
5. Click **OK**.
6. Click **Copy System Values** to copy over any new values you have not applied.
7. Click **Apply New**.

Related tasks:

- Deploying the System Definition File to a Real-Time Target

Exporting Parameter Values

Use the **Model Parameter Manager** tool to export parameter values to reproduce certain behaviors in your model.

Before you begin, verify that you have the tool in your Workspace **Tools** menu. If you do not, refer to Adding a Standard or Custom Tools Menu Item.

1. Deploy the system definition with models to the target.

   Note You can only apply new values to deployed systems.

2. In the Workspace, select **Tools** » **Model Parameter Manager** to launch the tool.

> **Note** The name of this menu item might differ depending on how you named it in the **Tools Properties** dialog box.

3. Click **Save Values** to save the current system values to a new or existing model parameter file.

VeriStand exports values in the format that corresponds to the file extension you choose.

Related tasks:

- Deploying the System Definition File to a Real-Time Target

Displaying Waveform Data in a Graph

Use a waveform graph control to display data from one or more waveforms in the system definition file to monitor and verify acquired data.

Before you begin, set up timing and logging properties for waveform acquisitions.

You can only graph waveforms with a double-precision floating-point data type.

1. Open the Workspace.
2. Select **Screen** » **Edit Mode** to display the Workspace Controls palette and alignment grid.
3. Click the Workspace Controls palette.
4. Select **Graph** » **Waveform**and drag the waveform graph onto the alignment grid.
5. In the Waveform Graph Settings dialog box, select the waveforms to display in the graph and the conditions that you want to stream data.

To configure the x-axis to represent absolute time, right-click a waveform graph and uncheck **Ignore Time Stamps**.

Note By default, the waveform graph ignores the absolute time associated with samples in a waveform and plots relative time on the x-axis. This behavior is useful if the data has no time correlation, or if you want to graph multiple waveforms from two or more systems that are not time synchronized.

Enhancing Your Workspace to View Data

Use other tools, such as the **Alarm Monitor**, **TDMS File Viewer**, and **XNET Bus Monitor**, to view data in the **Workspace**.

Before you begin, verify that you have the tool in your Workspace **Tools** menu. If you do not, refer to Adding a Standard or Custom Tools Menu Item.

1. In the Workspace, select **Tools**.
2. Depending on your goal, use any of the following tools.

Goal	Tool
Manage alarms high limit, low limit, corresponding procedure name, delay duration, trip value, priority, state, and mode information.	Alarm Monitor
View the current values for several channels at a time.	Channel Data Viewer
View the contents of a .tdms file.	TDMS File Viewer
View and log messages sent by an NI-XNET device.	XNET Bus Monitor
Perform custom tasks specified by a LabVIEW VI.	Custom VI Tool

Configuring and Executing Host-Side Logging

Use the **Data Logging** control to adjust log times, better format data files, select start and stop times, and more at run time.

1. Open the Workspace.

2. Select **Screen** » **Edit Mode** to display the Workspace Controls palette and alignment grid.
3. Click the Workspace Controls palette.
4. Select **Logging** » **Logging Control** and drag the control onto the alignment grid.
5. Complete any of the following configuration goals in the Edit Settings dialog box.

Goal	Description	How to configure
Specify channels to log	Designate the channels to log data from.	Use the **Channels** page.
Start and stop a log session with triggers	Configure the Data Logging control to start and stop logging during a log session based on triggers. A trigger is a condition-based formula, such as EngineTemp>5000. When you configure a start trigger, the Data Logging control waits after a log session begins for the start trigger to evaluate to TRUE before logging. You can also configure a stop trigger. When you configure a stop trigger, the Data Logging control stops logging when the stop trigger evaluates to TRUE.	Use the **Start Trigger** and **Stop Trigger** pages.
Capture custom information about a log session	Capture specific information from an operator at the start of a log session using properties. Any captured properties are added as metadata to all log files generated during the log session. For example, you can prompt an operator to enter	Use the **Properties** page.

Goal	Description	How to configure
	information, such as the their name, the unit under test, and the test set point before running a log session.	
Automate post-processing of log data	Configure the Data Logging control to automatically run post-processing actions for log files produced by the control during a log session. This allows you to automate post-processing actions, such as passing log files to Excel or generating a PDF report from the log files.	Use the **Post-Processing** page. See Running a Command Line Script and Loading a File in DIAdem.
Download log files produced by targets	Configure the Data Logging control to automatically download files produced by a target and to include these files in post-processing.	Use the **Target Logs** page. See Including Log Files Produced on a Target in Post-Processing.

Related tasks:

- Logging and Documenting Sessions

Running a Command Line Script

Execute a command line script for each file the control generates during a log session to complete tasks such as passing log files to a batch file or to an executable, like Excel, for additional post-processing.

1. Click **Edit settings** on the data logging control.
2. Select **Post-Processing**.
3. From the Action to take at end of log session dull-down menu, select **Run Command Line Script**.
4. Enter the command line script you want to run.

Loading a File in DIAdem

Load each data file generated during a log session in DIAdem to complete a variety of tasks such as generating graphs and tables.

This feature requires NI DIAdem. For more information about NI DIAdem, visit ni.com/diadem.

1. Click **Edit settings** on the data logging control.
2. Select **Post-Processing**.
3. From the Action to take at end of log session drop-down, select **Load File in DIAdem**.
4. Configure the options on this page to meet your needs.

Related tasks:

- Automating Post-Processing in DIAdem with a VBScript
- Generating a PDF or HTML Report with DIAdem
- Including Log Files Produced on a Target in Post-Processing

Automating Post-Processing in DIAdem with a VBScript

Automate the post-processing actions in DIAdem to analyze and process data by running a VBScript (.VBS).

1. Click **Edit settings** on the data logging control.
2. Select **Post-Processing**.
3. Select **Run script**.
4. In **Script path**, specify the path to the VBScript file you want to run.

Generating a PDF or HTML Report with DIAdem

Use a pre-configured DIAdem report file (.TDR) to automatically generate a PDF or HTML report from log files.

A DIAdem report file can display graphs, texts, and images based on the data from the loaded log files.

1. Click **Edit settings** on the data logging control.
2. Select **Post-Processing**.
3. From the **Report generation options** drop down menu, select **Generate HTML report** or **Generate PDF report**.
4. Specify the **Report template path** to the report template and the **Report export path** where you want to save the PDF or HTML report.

Including Log Files Produced on a Target in Post-Processing

Automatically download files produced by a target to include these files in post-processing.

You must start and stop the logs on the targets from outside of the **Data Logging** control.

1. Click **Edit settings** on the data logging control.
2. Select **Target Logs**.
3. Click **Download target log files to host**.
4. Click **Add** to add channels.
5. Select the targets from which you want to download files.
6. Specify the location to which to download the files in **Destination** and click **OK**.
7. Select **Include downloaded target-log files in post processing**.

When you stop logging, the log control downloads all files closed by the target during logging. If you choose to use DIAdem for post-process, the log files merge in DIAdem below the host-side log files.

Logging and Documenting Sessions

Control when a session log executes and document your observations about the session.

Note You can also launch the **File History** dialog box from the control to view past log sessions and the files associated with them.

1. Add and configure the Logging Control on the Workspace.
2. Complete any of the following goals.

Goal	Description	How to Operate
Start a log session	The log session starts when you click **Start Logging** or based on a start trigger.	Click **Start Logging** on the **Data Logging** control.
Stop a log session	The log session stops when you click **Stop Logging**, after a specified amount of time has elapsed, or based on a stop trigger.	Click **Stop Logging** on the **Data Logging** control.
Capture notes about a log session	Add notes to a log session to document observations, such as errors or peculiar behaviors, at the time they occur. The note saves the information you enter and a timestamp of when you click **Enter Note**.	Click **Enter Note** on the **Data Logging** control while logging.
View the results of a log session	View information about past log sessions and any files generated during those log sessions with the **File History** dialog box. **Note** If you do not have a specified application for viewing TDMS files, you can view	Click **History** on the **Data Logging** control while logging.

Goal	Description	How to Operate
	the file in the **TDMS Filer Viewer** workspace tool. To do so, right-click the file and select **View File in TDMS File Viewer.** Each log session has a unique SessionGUID that appears as a root file property in the TDMS log files and user notes files generated during the log session. You can use the SessionGUID to link all files generated during the same log session when analyzing data.	

Adding Custom Software

Integrate add-ons, plug-ins, and custom devices.

You can use LabVIEW to create various types of custom add-ons and plug-ins to extend VeriStand.

Add-on features for VeriStand include:

- Custom user interface objects for the Workspace.
- Custom devices that add support for additional hardware interfaces.
- Real-time engine functions.

The following table lists different types of add-ons.

Type of add-on	Description
Installed with Veristand	Use NI supported custom devices included with the VeriStand suite installer. For example, the Embedded Data Logger is available from the suite. These custom devices are available for both offline and online installation.
Installed by another NI product	Purchase software packages that add specialized features to VeriStand. For example, the ECU Measurement and Calibration Toolkit adds the XCP or CCP Master custom device and workspace controls to VeriStand.
Available for download	Download VeriStand Add-ons from NI and the VeriStand community.
User created	Use LabVIEW to create various types of custom add-ons for VeriStand. VeriStand installs a full palette of VeriStand VIs and the VeriStand .NET APIs. You can access both from LabVIEW. **Note** To develop add-ons, you must install matching versions of LabVIEW and VeriStand. For example, you

Type of add-on	Description
	must use LabVIEW 2020 to develop add-ons for VeriStand 2020.

System Requirements for Common Add-ons

Certain add-ons require additional LabVIEW modules, toolkits, or features not included with the Base Development System, such as the Application Builder.

The following table lists common add-ons you can develop using LabVIEW and any additional features they require.

Add-on	Requirements
Custom devices	Application Builder
Custom timing and sync devices	
Custom FPGA configuration files and bitfiles	▪ Application Builder ▪ LabVIEW FPGA Module
Custom Workspace tools	—
Custom Workspace controls and indicators	
Compiled models	▪ Application Builder ▪ LabVIEW Control Design and Simulation Module

The Application Builder is included with the LabVIEW Professional Development System. If you use the LabVIEW Base Development System or Full Development System, you can purchase the Application Builder.

For more information on how to use LabVIEW to create custom workspace objects, refer to Creating Custom Workspace Objects for VeriStand.

Logging Target Data with the Embedded Data Logger

Use the VeriStand Embedded Data Logger custom device to log data on a target instead of the host.

Before you begin, download the Embedded Data Logger from GitHub.

Use the Embedded Data Logger to log more data than the connection bandwidth between the target(s) and the host allows. For example, to log data on several real-time targets, use the Embedded Data Logger instead of streaming the data from each target back to the host.

The Embedded Data Logger also allows you to log data after you disconnect the host machine. This is useful if you want to deploy your system definition to a real-time target, disconnect your host computer, and let the target run over several days.

1. Launch your project in the VeriStand Editor.
2. In the Project Files pane, left-click a system definition file (.nivssdf) and select **Configure in System Explorer**.
3. Click **Targets** » **Controller** in the configuration tree.
4. Right-click **Custom Devices** and click **National Instruments** » **Embedded Data Logger**.
5. Add a log file.
    a. Right click **Embedded Data Logger** and select **Add Log File**.
    b. Edit the settings of the log file on its configuration page.
6. Specify the channels you want to log in each log file by group.
    a. Click **Channel Groups**.
    b. Click **Add Channel Group**.
    c. Click **Add Channels**.
    d. In the Select Channels window, specify the channels you want to log in the channel group.
    e. Click **OK**.
7. Save the system definition file.

Related concepts:

- Components of a VeriStand Project

Creating Custom Devices

Extend the functionality of VeriStand by packaging LabVIEW code into a form that you can add to the system definition file and deploy to a target.

Before you begin, you should determine if you have the necessary experience to create a custom device. You should also check the VeriStand Add-ons page or with your hardware vendor for pre-built custom devices.



Note For a detailed guide to developing custom devices, refer to the Custom Device Handbook.

You can use a custom device to execute on real-time systems and perform deterministic hardware-in-the-loop and real-time test procedures. With a custom device, you can control the configuration, execution, and user interface of a project. Custom devices appear in the System Explorer configuration tree.

A custom device can run either inline or in parallel (asynchronously) with the VeriStand Engine's Primary Control Loop, and can function as a timing and sync device. All custom devices communicate with the VeriStand Engine using configurable channels and properties.

1. Custom Device FAQs
   Answers to common questions about creating a custom device.
2. Custom Device Framework
   Follow the custom device framework to ensure your LabVIEW code interacts correctly with VeriStand.
3. Planning a Custom Device
   It is important to plan the major components of a custom device before writing any LabVIEW code.
4. Implementing a Custom Device
   Develop a custom device to fit your requirements.
5. Building a Custom Device
   Build a custom device for distribution to VeriStand.

6. Distributing Custom Devices
   Package a custom device to manually distribute it to VeriStand.

Custom Device FAQs

Answers to common questions about creating a custom device.

Am I qualified to create a custom device?

Creating a custom device requires specific knowledge and skills. The following table displays the specialized experience areas you need to successfully create a custom device.

Note You can gain this experience through taking NI educational courses and earning certifications.

Experience area	Description
LabVIEW Application Development	To develop a custom device, you must thoroughly understand LabVIEW programming and application architectures. NI recommends a Certified LabVIEW Developer (CLD) level of expertise before beginning development of a custom device.
LabVIEW Real-Time Application Development	As custom devices execute within real-time systems, you must be familiar with programming for real-time operating systems (RTOS) and specialized LabVIEW development techniques for developing real-time applications.
VeriStand Background	To develop a custom device, you must fully understand the VeriStand Engine.

When do I need a custom device?

VeriStand supports most real-time testing applications. Before pursuing a custom device, you should first try to meet your needs with the built-in functionality. If you cannot meet your needs using the built-in VeriStand features, you can extend the

functionality with a custom device. The following table lists situations where a custom device is best suited.

Situation	Rationale
Integrating VeriStand with third-party hardware	If the hardware you need to integrate with VeriStand is not natively supported, you may be able to integrate it by creating a custom device.
Implementing a measurement or generation mode that VeriStand does not support	If VeriStand does not support the measurement or generation mode you need for your hardware type, you may be able to implement it using a custom device. For example, VeriStand supports single-point hardware-timed analog acquisition using DAQmx. However, VeriStand does not support force or torque measurements for analog DAQ channels. You can implement this measurement mode as a custom device.
Implementing additional features that VeriStand does not support	A VeriStand project may require a feature that VeriStand does not provide. You can extend VeriStand to meet your needs through a variety of methods. Custom devices are best suited for implementing features that require or use VeriStand channel data on the execution host. For example, the Embedded Data Logger allows you to log VeriStand channels to a TDMS file without first sending channel data back to the Workspace, as with high-speed streaming. However, if you need to display the previous test results on the workspace while running a new test, a custom workspace object may be more appropriate than a custom device.

If you do not need the full range of custom device functionality, you can fulfill your requirements by converting a LabVIEW VI into a compiled model. Other alternatives include utilizing workspace tools, implementing custom FPGA bitfiles, and exploring the various LabVIEW and .NET APIs that ship with VeriStand.

When do I need a hardware custom device?

Before you begin developing a custom device to interact with unsupported or third-party hardware, NI recommends you evaluate the data requirements of the device and the availability of device drivers and APIs. To support third-party hardware, a custom device must call a hardware or instrument driver. If a hardware or instrument driver does not exist, you will need to either create the driver yourself or ask the vendor for that device for a driver.

Answering the following questions can help you determine whether a custom device is feasible for a specific hardware device:

- Does a LabVIEW instrument driver exist for the device?

> **Note** You can search for instrument drivers on the Instrument Driver Network and NI Hardware Drivers pages.

- Is a hardware driver/API available for the device and easy to use?
- If necessary, is the hardware driver executable in LabVIEW Real-Time Module?

> **Note** Refer to Verify Your DLL Is Executable in LabVIEW Real-Time for more information about testing DLLs for real-time support.

- Can you meet the hardware requirements by passing LabVIEW 64-bit double-precision floating point numbers to and from the custom device during steady state operation?

> **Note** If the hardware driver returns a vector, structure, or any non-DBL data, you cannot pass the data directly from the custom device to VeriStand. You must coerce the data or design an alternative communication mechanism to pass data from the custom device to the rest of the system.

Custom Device Framework

Follow the custom device framework to ensure your LabVIEW code interacts correctly with VeriStand.
The custom device framework consists of type definitions, specifically named controls and indicators, template VIs, and a LabVIEW API. These items form the rules, or framework, that allow your code to interact with VeriStand.
The following table describes the components of the framework.

Component	Description
Custom Device XML file	The **Custom Device XML file** enables you to define parts of the custom device in System Explorer, specify which VIs to call, and select the dependencies to deploy to an RT target.
Custom Device API Library	The **Custom Device API library** is a LabVIEW library that contains type definitions, template VIs, and the LabVIEW API a custom device needs to interact with VeriStand.
Custom Device Library	The **Custom Device library** is a LabVIEW library that contains the configuration and engine VIs for a custom device. The configuration and engine VIs may optionally be distributed in different LabVIEW libraries.
Build Specifications	The **Build Specifications** in a custom device LabVIEW project includes the Configuration and Engine source distributions.

Custom Device XML File

The **Custom Device XML file** enables you to define parts of the custom device in **System Explorer**, specify which VIs to call, and select the dependencies to deploy to an RT target.
When VeriStand launches, it uses the Custom Device XML file in the <Common Data>\Custom Devices directory to determine how to load, configure, display, and run custom devices. This XML file provides basic information about a custom device, including the type of custom device, paths to its VIs and dependencies, and pages, glyphs, buttons, and menu items associated with the custom device.

Every custom device must have a Custom Device XML file in the directory. A properly formatted XML file correctly implements the tags defined by the Custom Device XSD file located in the same directory. The XML file must have its custom device type defined in its title before the device name. The following are examples of properly named XML files:

- Custom Device <Device Name>.xml
- SLSC Module <Device Name>.xml
- Timing and Sync <Device Name>.xml

Note You must restart VeriStand to recognize new or changed Custom Device XML files. After you restart, VeriStand will report errors in the XML.

The <Common Data>\Custom Devices directory also includes Custom Device XML files for the custom devices that install with VeriStand. You can use these XML files for reference when structuring your XML.

Caution Do not directly edit the contents of the included Custom Device XML files. Edits to the file can break the custom devices.

Custom Device XML Tags

XML tags define settings for a custom device.

These elements, and non-standard element types, are defined in the Custom Device.xsd schema located in the <Common Data>\Custom Devices directory. You can open the file in an XML or text editor to read the schema and view the hierarchy. For an example of how to implement tags in your XML file, refer to the pre-built custom device XML files that install with VeriStand.

Caution Do not directly edit the contents of the included Custom Device XML files. Edits to the file can break the custom devices.

The following table displays the XML tags you can use in a custom device XML file.

Element	Required?	Element type	Min/Max occurrences	Description
<CustomDevice>	Yes	complex	1/1	Opening tag for a custom device definition.
<XSDVersion>	No	VersionType	0/1	Specifies the version of the Custom Device.xsd file the XML is using. The version must match the version of VeriStand you are using.
>> <AddMenu>	Yes	LocString	1/1	Specifies the device name to display in the shortcut menu when an operator right-clicks the Custom Devices node in System Explorer.
>> <Dependency>	No	Dependency	0/1	Specifies the path to a dependency file, such as a DLL or VI, that the custom device requires and that you

Element	Required?	Element type	Min/Max occurrences	Description
				want to deploy to the target along with the custom device.
>> <Version>	Yes	xs:string	1/1	Specifies version information for the custom device.
>> <Type>	Yes	xs:string	1/1	Specifies the device type or execution mode of the custom device: ▪ Asynchronous ▪ Inline HW Interface ▪ Inline Model Interface ▪ Inline Timing and Sync ▪ Asynchronous

Element	Required?	Element type	Min/Max occurrences	Description
				Timing and Sync
>> <MaxOccurrence>	Yes	xs:int	1/1	Specifies the maximum number of instances of the custom device to allow in a system definition file.
>> <MainPageGUID>	Yes	xs:string	1/1	Specifies the GUID of the configuration page to associate with the top-level custom device item.
>> <TimingSource>	No	complex	0/1	Configures a custom timing source for the Primary Control Loop.
>>> <HasTimingSourceCapability>	Yes*	xs:boolean	1/1	Enables or disables the use of the timing source VI you specify as the timing source for the Primary

Element	Required?	Element type	Min/Max occurrences	Description
				Control Loop.
>>> <Paths>	No	complex	1/1	These elements are obsolete. Use <SourceDistribution> instead.
>>>> <Source>	Yes*	Path	1/1	
>>>> <RealTimeSystemDestination>	Yes*	xs:string	1/1	
>>> <SourceDistribution>	No	complex	0/1	Contains information about the source distribution for the timing source VI.
>>>> <Source>	Yes*	complex	1/ unbounded	Contains information about the files in the source distribution.
>>>>> <SupportedTarget>	Yes*	Target	1/ unbounded	Specifies the target operating system(s) on which the custom device runs.
>>>>> <Location>	Yes*	Path	1/1	Specifies the location of the timing source VI on the host computer.
>>>>> <RealTimeSystemDestination>	Yes*	xs:string	1/1	Specifies the destination

Element	Required?	Element type	Min/Max occurrences	Description
				path for the timing source VI on the target.
>>>>> <Version>	No	xs:string	0/1	Specifies version information for the source distribution.
>> <InitializationVI>	Yes	complex	1/1	Contains information about the Initialization VI that runs when you add the custom device to the system definition file.
>>> <Type>	Yes	xs:string	1/1	Specifies how the VI runs: ▪ Action—Specifies that the VI runs silently in the background. ▪ VI—Specifi

Element	Required?	Element type	Min/Max occurrences	Description
				es that the VI runs in an interactive mode with the front panel visible.
>>> <Execution>	No	xs:string	1/1	Specifies additional information about the execution of the VI: ▪ Silent—Specifies that the VI runs silently in the background. ▪ Modal—Specifies that the VI runs in a modal

Element	Required?	Element type	Min/Max occurrences	Description
				window. ▪ Floating—Specifies that the VI runs in a floating window. ▪ Default—Specifies that the VI runs in the default mode for the <Type> you specify.
>>> <Position>	No	xs:string	1/1	Specifies where to position the front panel window, if displayed, on VI launch: ▪ Centered—Specifies to

Element	Required?	Element type	Min/Max occurrences	Description
				center the window on the default monitor. ▪ Mouse pointer—Specifies to position the origin of the window on the mouse pointer.
>>> <Item2Launch>	Yes	Path	1/1	Specifies the VI to launch as the Initialization VI.
>> <CustomDeviceVI>	Yes	complex	1/1	Contains information about the RT Driver VI that runs on the target.
>>> <Source>	No	Path	0/1	These elements are obsolete.
>>> <RealTimeSystemDestination>	No	xs:string	0/1	

Element	Required?	Element type	Min/Max occurrences	Description
				Use <SourceDistribution> instead.
>>> <SourceDistribution>	No	complex	0/1	Contains information about the source distribution for the RT driver VI.
>>>> <Source>	Yes*	complex	1/ unbounded	Contains information about the files in the source distribution.
>>>>> <SupportedTarget>	Yes*	Target	1/ unbounded	Specifies the target operating system(s) on which the custom device runs.
>>>>> <Source>	Yes*	Path	1/1	Specifies the location of the RT driver VI on the host computer.
>>>>> <RealTimeSystemDestination>	Yes*	xs:string	1/1	Specifies the destination path for the RT driver VI on the target.
>> <Dependencies>	Yes	complex	1/1	Contains information

Element	Required?	Element type	Min/Max occurrences	Description
				about dependencies of the custom device.
>>> <Dependency>	No	complex	0/ unbounded	Contains information about a specific dependency.
>>>> <SupportedTarget>	No	Target	0/1	Specifies the target operating system(s) on which the custom device runs.
>>>> <Source>	Yes*	Path	1/1	Specifies the location of the dependency on the host computer.
>>>> <RealTimeSystemDestination>	Yes*	xs:string	1/1	Specifies the destination path for the dependency on the target.
>>>> <ForceDownload>	No	xs:boolean	0/1	Specifies whether to force the download of the dependency.
>>>> <Version>	No	xs:string	0/1	Specifies version information

Element	Required?	Element type	Min/Max occurrences	Description
				for the dependency.
>> <Pages>	Yes	complex	1/1	Contains information about System Explorer configuration pages associated with the custom device.
>>> <Page>	Yes	complex	1/ unbounded	Contains information about a specific page.
>>>> <Name>	Yes	LocString	1/1	Specifies the name of the page.
>>>> <DisallowRenaming>	No	xs:boolean	0/1	Disallows renaming of the item with which the page is associated.
>>>> <DeleteProtection>	No	xs:boolean	0/1	Disallows deleting the item with which the page is associated.
>>>> <AllowMultiSelection>	No	xs:boolean	0/1	Allows an operator to select the item with which the

Element	Required?	Element type	Min/Max occurrences	Description
				page is associated during a multi-select operation.
>>>> <ExcludeFromAlphabeticalOrder>	No	xs:boolean	0/1	Excludes a page's children from any alphabetical sorting operations.
>>>> <Copy>	No	xs:string	0/1	Configures copying of the item: ▪ Copy—Enables copying. ▪ Disabled—Disables copying.
>>>> <Paste>	No	xs:string	0/1	Configures pasting of the item: ▪ Create—Pastes by creating a new

Element	Required?	Element type	Min/Max occurrences	Description
				instance of the item. ▪ CreateIfNotExists_GUID—Pastes only if an item with the same GUID does not already exist. ▪ CreateIfNotExists_Name—Pastes only if an item with the same name does not already exist. ▪ Replace—

Element	Required?	Element type	Min/Max occurrences	Description
				Pastes by replacing an item. ▪ Dialog—Prompts the operator with a dialog before pasting.
>>>> <ParentGUIDs>	No	complex	0/1	Contains information about the GUIDs of possible parent items. When <Paste> is CreateIfNotExists_GUID, VeriStand checks any GUIDs listed here in addition to the page GUID.
>>>>> <ParentGUID>	No	xs:string	0/unbounded	Specifies the GUID of a parent item.
>>>> <GUID>	Yes	xs:string	1/1	Specifies the page GUID.

Element	Required?	Element type	Min/Max occurrences	Description
>>>> <Glyph>	Yes	Path	1/1	Specifies the default icon that appears next to the item in System Explorer.
>>>> <InactiveGlyph>	No	Path	0/1	Specifies the icon that appears next to the item in System Explorer when the item is inactive.
>>>> <BrokenGlyph>	No	Path	0/1	Specifies the icon that appears next to the item in System Explorer when the item is broken.
>>>> <Item2Launch>	Yes	Path	1/1	Specifies the VI to launch as the page.
>>>> <RunTimeMenu>	No	complex	0/1	Contains information about the shortcut menu an operator can access by right-clicking the

Element	Required?	Element type	Min/Max occurrences	Description
				item at run time.
>>>>> <MenuItem>	Yes*	complex	1/ unbounded	Contains information about a specific shortcut menu item.
>>>>>> <GUID>	Yes*	xs:string	1/1	Specifies the GUID for the menu item.
>>>>>> <Type>	Yes*	xs:string	1/1	Specifies the type of the menu item: ▪ Action—Specifies that the item launches a VI that runs silently in the background. ▪ VI—Specifies that the item launches a VI and displa

Element	Required?	Element type	Min/Max occurrences	Description
				ys the front panel so the operator can interact with the VI. ▪ Separator—Specifies that the item is a menu separator. ▪ Custom—Specifies that the menu item has a custom type.
>>>>>> <Execution>	No	xs:string	0/1	Specifies additional information about the execution of the VI associated

Element	Required?	Element type	Min/Max occurrences	Description
				with the menu item. ▪ Silent—Specifies that the VI runs silently in the background. ▪ Modal—Specifies that the VI runs in a modal window. ▪ Floating—Specifies that the VI runs in a floating window. ▪ Default—Specifies that

Element	Required?	Element type	Min/Max occurrences	Description
				the VI runs in the default mode for the <Type > you specify .
>>>>>> <Position>	No	xs:string	0/1	Specifies where to position the front panel window, if displayed, on VI launch. ▪ Centered—Specifies to center the window on the default monitor. ▪ Mouse pointer—Specifies to position the origin

Element	Required?	Element type	Min/Max occurrences	Description
				of the window on the mouse pointer.
>>>>>> <Behavior>	No	xs:string	0/1	Specifies whether the menu item does nothing (None) or launches a VI (OpenFrontPanel).
>>>>>> <MinNrOfChilds>	No	xs:int	0/1	Specifies a minimum number of children for the menu item.
>>>>>> <Name>	Yes*	LocString	1/1	Specifies the name of the menu item.
>>>>>> <Item2Launch>	Yes*	Path	1/1	Specifies the VI to launch when an operator selects the menu item.
>>>>>> <Dependency>	No	Dependency	0/1	Specifies the path to a dependency file.
>>>>>> <CustomPopulation>	No	Path	0/1	Specifies a file to associate with the

Element	Required?	Element type	Min/Max occurrences	Description
				menu item if <Type> is Custom.
>>>> <ButtonList>	No	complex	0/1	Contains information about buttons that appear in the System Explorer when an operator displays the page.
>>>>> <Button>	Yes*	complex	1/ unbounded	Contains information about a specific button.
>>>>>> <ID>	Yes*	xs:string	1/1	Specifies a unique ID to associate with the button.
>>>>>> <Glyph>	Yes*	Path	1/1	Specifies the icon that appears on the button.
>>>>>> <Type>	No	xs:string	0/1	Specifies the type of the button: ▪ Action—Specifies that the item launch

Element	Required?	Element type	Min/Max occurrences	Description
				es a VI that runs silently in the background. ▪ Dialog—Specifies that the button displays a dialog with which the operator can interact. ▪ Page—Specifies that the button displays a new configuration page. ▪ Notificatio

Element	Required?	Element type	Min/Max occurrences	Description
				n—Specifies that the button sends a notification to the currently loaded configuration page. This option passes the button ID to the page. ▪ Separator—Specifies that the button is actually a separator on the button

Element	Required?	Element type	Min/Max occurrences	Description
				toolbar.
>>>>>> <ReferencedGUID>	Yes	xs:string	0/1	Specifies a GUID to reference when an operator clicks the button.
>>>>>> <ButtonText>	No	LocString	0/1	Specifies text to display on the button.
>>>>>> <Caption>	Yes*	LocString	1/1	Specifies a caption for the button. Captions appear in Context Help.
>>>>>> <TipStrip>	Yes*	LocString	1/1	Specifies a tip to display when an operator hovers over the button.
>>>>>> <Documentation>	Yes*	LocString	1/1	Specifies the description that appears in Context Help.
>>>>>> <Dependency>	No	Dependency	0/1	Specifies the path to a dependency file.
>>>> <ActionVIOnDelete>	No	Path	0/1	Specifies a VI to run when

Element	Required?	Element type	Min/Max occurrences	Description
				an operator deletes the item associated with the page from System Explorer.
>>>> <ActionVIOnLoad>	No	Path	0/1	Specifies a VI to run when VeriStand loads the item associated with the page.
>>>> <ActionVIOnSystemShutdown>	No	Path	0/1	Specifies a VI to run when VeriStand shuts down.
>>>> <ActionVIOnSave>	No	Path	0/1	Specifies a VI to run when an operator saves the system definition file.
>>>> <ActionVIOnDownload>	No	Path	0/1	Specifies a VI to run when an operator downloads the custom device to the target.
>>>> <ActionVIOnPaste>	No	Path	0/1	Specifies a VI to run when an operator pastes the

Element	Required?	Element type	Min/Max occurrences	Description
				item associated with the page.
>>>> <ActionVIOnTargetTypeChange>	No	Path	0/1	Specifies a VI to run when an operator changes the type of target on which the custom device runs.
>>>> <ActionVIOnDeleteRequest>	No	Path	0/1	Specifies a VI to run when a delete request is received for the item associated with the page.
>>>> <ActionVIOnCompile>	No	Path	0/1	Specifies a VI to run when the item associated with the page is compiled.
>>>> <Help>	No	complex	0/1	Contains information about help content for the item associated with the page.

Element	Required?	Element type	Min/Max occurrences	Description
>>>>> <Item2Launch>	Yes*	Path	1/1	Specifies the file to launch when an operator selects the Help option for the item.
>>>>> <FileType>	Yes*	xs:string	1/1	Specifies the help file type. Valid values are chm or other.
>>>>> <Section>	No	xs:string	0/1	Specifies a section of the help file to display when an operator selects the Help option.
>>>> <AdditionalInformation>	No	xs:string	0/1	Specifies additional information to store with the item.
>> <CustomXML>	No	xs:string	0/1	Contains custom XML that you add to the file. Code you enter between <CustomXML> tags is not validated against the Custom

Element	Required?	Element type	Min/Max occurrences	Description
				Device.xsd file.
*Required sub-tags of optional parent tags are only required if you use the parent tag.				

Custom Device Non-Standard XML Element Types

Non-standard element types are defined in the Custom Device.xsd schema.

Dependency

The **Dependency** element type describes a dependency of the custom device, such as a DLL or VI, that the custom device requires and that you want to deploy to the target with the custom device.

Element	Required?	Element type	Min/Max Occurrences	Description
<Type>	Yes	xs:string	1/1	Specifies if the path you specify is relative or absolute. Valid values are Absolute, To Base, To Common Doc Dir, or To Application Data Dir.
<Path>	Yes	xs:string	1/1	Specifies the path to the dependency on the host computer.
<Behavior>	No	xs:string	0/1	Specifies whether the path is Static or Dynamic.

LocString

The **LocString** element type specifies English and localized strings to display in menu items.

Element	Required?	Element type	Min/Max Occurrences	Description
<eng>	Yes	xs:string	1/1	VeriStand does not currently use this string. The <loc> string appears in the shortcut menu.
<loc>	Yes	xs:string	1/1	Specifies a string for a custom shortcut menu. You can use two colons (::) to separate elements and create nested menu items.

Path

The **Path** element type describes a path to a file and whether the path is relative or absolute.

Element	Required?	Element type	Min/Max Occurrences	Description
<Type>	Yes	xs:string	1/1	Specifies whether the path you specify is relative or absolute. Valid values are Absolute, To Base, To Common Doc Dir, or To

Element	Required?	Element type	Min/Max Occurrences	Description
				Application Data Dir.
<Path>	Yes	xs:string	1/1	Specifies the path to the file.

Target

The Target element type describes the target to which you want to deploy the custom device.

Element type	Enumerations
xs:string	▪ All ▪ Pharlap ▪ Windows ▪ Linux_32_ARM ▪ Linux_x64 ▪ PharlapWindows

VersionType

The **VersionType** element type specifies version information.

Attribute	Required?	Attribute type
Major	Yes	xs:unsignedInt
Minor	Yes	xs:unsignedInt
Fix	Yes	xs:unsignedInt
Build	Yes	xs:unsignedInt

Custom Device API Library

The **Custom Device API library** is a LabVIEW library that contains type definitions, template VIs, and the LabVIEW API a custom device needs to interact with VeriStand.

Note To access the Custom Device API library, you must first install a compatible version of LabVIEW. When installing VeriStand, under Additional items you may wish to install, select **VeriStand Support for LabVIEW**. This will install the VeriStand APIs, including the Custom Device API library.

The components of the library allows the custom device to communicate seamlessly in the VeriStand Engine. You can find the library in the labview\vi.lib\NI VeriStand\Custom Device API directory.
The following table displays the template VIs included in the Custom Device API library. You can access most of the VIs in this library from the Custom Device palette and subpalettes in LabVIEW. The connector panes of these VIs are configured to work with VeriStand, and the front panels and block diagrams include documentation to help you configure VIs that fit the needs of your custom device.

Note If you add any VIs created from the following templates to a custom device project, you must define the item in the Custom Device XML file using valid Custom Device XML tags.

Template VI	Usage
Initialization VI Template.vit	Initialization VI.
Page Template.vit	Main Page VI or additional Page VI.
Asynchronous Custom Device Driver VI Template.vit	RT Driver VI for an asynchronous custom device.
Inline Custom Device Driver VI Template (HW Interface).vit	RT Driver VI for an Inline Hardware Interface custom device.
Inline Custom Device Driver VI Template (Model Interface).vit	RT Driver VI for an Inline Model Interface custom device.
Timing Source Initialization VI Template.vit	Timing Source Initialization VI for the Primary Control Loop. For VeriStand to call this VI, you must select the custom device as the Master Custom Device on the Controller configuration page.
RunTimeMenu Custom Item 2 Launch.vit	Runs when the user selects a custom shortcut menu item.

Template VI	Usage
RunTimeMenu Custom Population.vit	Creates a custom shortcut menu.
RunTimeMenu Dependency.vit	Shows, removes, enables, or disables items in a custom shortcut menu.
ActionVIOnCompile Template.vit	Runs when an item is compiled.
ActionVIOnDelete Template.vit	Runs when a user successfully deletes an item.
ActionVIOnDownload Template.vit	Runs when the custom device downloads to the target.
ActionVIOnDeleteRequest Template.vit	Runs when a user attempts to delete an item.
ActionVIOnLoad Template.vit	Runs when VeriStand loads a system definition containing a custom device in System Explorer.
ActionVIOnPaste Template.vit	Runs when a user pastes an item in System Explorer.
ActionVIOnSave Template.vit	Runs when a user saves the system definition file.
ActionVIOnShutdown Template.vit	Runs when the VeriStand system shuts down.

Custom Device Library

The **Custom Device library** is a LabVIEW library that contains the configuration and engine VIs for a custom device. The configuration and engine VIs may optionally be distributed in different LabVIEW libraries.
Custom device library VIs configure the initialization behavior of the custom device, its interaction with the VeriStand Engine, and the user interface for the custom device in System Explorer.

Note Custom devices also include a separate library, called the Custom Device API library, which contains the type definitions, template VIs, and the LabVIEW API that a custom device needs to interact with VeriStand.

The Custom Device library contains the following sets of VIs to modify your custom device.

VIs for Configuring the Custom Device

The configuration of a custom device defines how you add and configure the custom device within the system definition file. Together, these VIs form the Configuration.llb for the custom device.



Note In a custom device, pages are the configuration pages that appear when you select the custom device or various sub-items of the device in System Explorer. The configuration page that an operator sees is the front panel of a page VI.

VI	Description
Initialization VI	Prepares the custom device for first use. This VI defines the initial list of channels and/or sections that appear in System Explorer, as well as sets their initial properties. The VI runs in the background every time an operator adds the custom device to the system definition file. It reads information from the Custom Device XML file and uses the data to add an instance of the custom device to System Explorer. The instance runs in the execution mode you specify and loads the correct dependency files. If the operator adds multiple instances of the device to the same system definition file, this VI runs for each new instance.
Main Page VI	Runs when the operator selects the custom device in System Explorer. The front panel of this VI serves as the configuration page for the custom device. You can modify this VI to add additional controls and indicators to the configuration page, and to configure responses to user events, such as when the operator enters a new item property value.
Extra Page VIs	Provides custom configuration pages for the sections and channels that appear under the

VI	Description
	custom device in System Explorer. You can configure these VIs in the same way you configure the Main Page VI. Every extra page you configure must have an entry in the <Pages> section of the Custom Device XML file. If you do not specify an extra page for a section or channel, VeriStand displays a default section or channel page when you select the item. You can use the Set Item GUID VI to change the page associated with an item at run time.
Action VIs	Allows the custom device to perform custom actions when specific events occur, such as when the system definition is loaded in System Explorer.
Shortcut Menus	Appears in a shortcut menu when a custom device item is right-clicked in the System Explorer. These may also appear as toolbar buttons.

VIs for Customizing the Custom Device Engine

The custom device engine defines the real-time behavior of the custom device on the target. The RT Driver VI runs on the target regardless of the target's operating system.

You can have more than one RT Driver VI depending on the needs of your custom device. For example, if your custom device must support multiple real-time operating systems, you can add additional RT Driver VIs for each additional operating system the custom device must support.

VI name	Description
RT Driver VI	Runs after the operator deploys the system definition file. You can add code to this VI to handle any run-time requirements of the custom device. Use this code to configure initialization, steady-state, and shutdown behavior for the custom device engine. This VI

VI name	Description
	handles timing information and the exchange of data between the custom device and the rest of the VeriStand system.

Custom Device Build Specifications

The **Build Specifications** in a custom device LabVIEW project includes the Configuration and Engine source distributions.
The **Configuration** source distribution contains the source files that specify the configuration of the custom device when added to a VeriStand system definition. It also defines the user interface for the custom device in System Explorer.
This build specification must include the following items:

- Initialization VI
- Main Page VI
- Custom Device XML file
- Any additional page VIs
- Any VIs the custom device configuration calls dynamically

The **Engine** source distribution contains the source files that configure how the custom device runs on the target. This build specification minimally includes the RT Driver VI. This specification also can include additional driver VIs and a timing source VI.
Both build specifications always include the Custom Device library, even though they do not both include every VI within that library.

Optional Source Distributions

In most cases, the standard Engine source distribution can work for a variety of targets. You can create a separate Engine distribution for each target. For example, your custom device project can contain all of the following source distributions:

- Configuration
- Engine (Windows)
- Engine (Phar Lap)
- Engine (Linux x64)

- Engine (Linux ARM)

After you configure the source distributions, you can choose between distributing the source files manually or using the LabVIEW Application Builder to build an installer.

Planning a Custom Device

It is important to plan the major components of a custom device before writing any LabVIEW code.

Before you begin planning, you should understand the custom device framework.

Understand the following components of a custom device.



Note If the custom device must interact with unsupported or third-party hardware, consider the availability of drivers and other resources to determine the feasibility of the custom device.

Component	Description
Channels and waveforms	The inputs and outputs of the custom device.
Properties	Configuration data for the custom device.
Hierarchy	The organization and appearance of the custom device in System Explorer.
Pages	The configuration pages that appear in System Explorer.
Device Type	The execution mode of the custom device in terms of interaction with the rest of VeriStand.

After planning your custom device, you can implement the custom device.

Custom Device Channels and Waveforms

A custom device uses channels and waveforms to exchange data with the rest of the VeriStand system.
Waveforms differ from channels in that VeriStand always sends channel data to the host computer, but VeriStand only streams waveform data upon request. You should acquire signals as waveforms when you need to read at rates faster than the rate at which the system runs.

Channels perform single-point acquisition, an immediate, non-buffered operation that occurs at the rate at which the system runs. All channels are either 64-bit floating point numbers (LabVIEW DBLs) or waveforms. No other data types are currently supported. However, as you write code for a custom device, you can use LabVIEW's various data conversion functions to convert other types of data to 64-bit floating point numbers. For example, if the LabVIEW API for a third-party device calls for Boolean data to enable a channel or filter, you can use a DBL channel with the assumption that 0 = FALSE and !0 = TRUE.



Note You can also use the Register Custom Device Engine Events VI to configure engine events and design a communication mechanism to handle data of different types. Alternatively, you can use other remote communication methods, such as TCP and UDP.

You can create input and output channels. Input channels get data from the rest of the VeriStand system. Output channels send data to the rest of the system. An operator can map each input channel to a single data source. However, an output channel can map to any number of sinks, such as simulation model inputs.
There are three common use cases for a custom device channel:

1. Handling data generated by the custom device after deployment.
2. Handling data generated elsewhere in the VeriStand system and used by the custom device after deployment.
3. Implementing dynamic properties.

Best practice is to implement channels with general use-cases in mind. For example, if you are writing a custom device to interface with third-party hardware, consider adding custom device channels for every physical channel of the hardware device and give the custom device operator the ability to remove channels while configuring the custom device in the system definition file.

Custom Device Item Properties

Custom device item properties store and communicate state information about a custom device item, such as a section or channel.
A **custom device item** is anything that appears in System Explorer, such as a channel within a custom device or the custom device itself. Use **properties** to

transfer configuration and state information from the configuration to the engine after an operator deploys the system definition file to the target. After deployment, the engine can read properties on the target, but it cannot write properties or exchange properties with the host computer.
For example, if you are creating a custom device for third-party hardware and you need to implement a range option for the channels you will use to communicate with the hardware, you should implement the range setting as a property. You can then customize the page VIs for those channels to accept a value for range, allowing you to define the value when configuring the channels in System Explorer. When the custom device is deployed, VeriStand will read the value for range and set the range of that channel when the system definition is deployed. After it is deployed, you can no longer change the value for range.
Unlike a channel, which must be a 64-bit floating point number, a property can be any standard LabVIEW data type. However, property names are case-sensitive strings. VeriStand saves property names and values with the system definition file. If you save and close the file or project, VeriStand retains the properties the next time you access the system definition file.

Custom Device Pages

In a custom device, **pages** are the configuration pages that appear when you select the custom device or various sub-items of the device in **System Explorer**.
The configuration page that an operator sees is simply the front panel of a page VI. Every custom device must include a Main Page VI that runs when the operator selects the top-level custom device item in the configuration tree. Sub-items of the custom device, such as sections and channels, appear with simple default configuration pages that allow a user to set descriptions for the associated items.

Extra Pages

You can create additional pages with extended functionality and use those pages in place of the default pages.

Plan an extra page for each item in a custom device that you want to customize differently. For example, if you want to use the same custom configuration page for all channels in a custom device, you only need to create one extra page.

The following table displays the items VeriStand requires to override a default page with an extra page in the custom device.

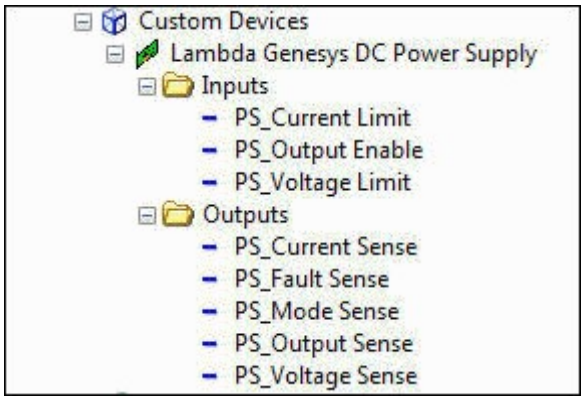
Required item	Description	How to create
Page VI	Defines the functions and appearance of the extra page. The front panel of the page VI serves as the configuration page in System Explorer.	Use the page template in the Custom Device API library.
Globally Unique Identifier (GUID)	Links to the extra page using an ID. When you associate an extra page with a channel or section, you override the default page by referencing the GUID of the extra page.	Use the GUID Generator VI in the labview\vi.lib\NI Veristand\Custom Device Tools\Custom Device Template Tool directory. You can set the GUID of an extra page by using the GUID terminal of either the Add Custom Device Section VI or Add Custom Device Channel VI. For more information on how to use the GUID Generator VI, refer to the topic on adding extra pages after creating the custom device project.
XML Declaration	Associates the GUID with the page VI in the custom device XML.	Edit the custom device XML to declare the extra page and its GUID. For more information, refer to the topic on adding extra pages after creating the custom device project.
Build Specification	Specifies the extra page and any of its dynamically called dependencies.	Edit the Configuration build specification to include the extra page and any of its dynamically called dependencies in the initialization library. For more information, refer to the topic on adding extra pages after creating the custom device project.

Custom Device Hierarchy

A logically configured hierarchical structure allows operators to efficiently use your custom device.
The **hierarchy** of a custom device is its appearance in System Explorer. The top level of the custom device hierarchy is the device itself, which appears under Custom Devices section of the configuration tree. Under the device, you can add any number of sections and channels to build the hierarchy.
Sections are groups of items in the hierarchy. You can group similar channels in a section. For example, you can group input channels into a Inputs section.

Note All items in the hierarchy must be either sections or channels.

The following example shows the hierarchy of a typical power supply custom device.



Custom Device Types

The type of a custom device refers to its execution mode, which defines how the device interacts with the VeriStand Engine.
Depending on your requirements, you can create custom devices that run inline or in parallel with the Primary Control Loop (PCL). You also can create timing and sync devices.

Note Timing and sync devices are the same as regular custom devices, but you can configure them as the hardware synchronization master to

drive RTSI0. They appear in System Explorer under **Hardware** » **Chassis** » **Timing and Sync**.

The following table displays the pre-defined custom device types that are included in VeriStand.

Device type	Basic architecture	Data interface	Timing	Benefits	Caveats	Use cases
Asynchronous	Two loops, one for receiving commands and one for data transfer	FIFOs	Variable	Unlikely to affect timing of the rest of the VeriStand system, can run faster or slower than the PCL	1-cycle latency due to FIFOs	Shared resources, background processes, non-deterministic hardware/ protocols, system health monitoring, logging, offline analysis
Inline Hardware Interface	State machine with two-phase execution	Channel references	Inline	Sends data to engine before other components execute, receives data from engine after other components execute	Can adversely affect PCL timing	Most hardware, deterministic operation, two-phase operations (for example, stimulus-response)
Inline Model Interface	State machine with one-phase execution	Channel references	Inline	Sends data to engine with low latency	Can adversely affect PCL timing	Low latency calculations, such as PID, interpolation, and so on
Inline Timing and Sync	State machine	Channel references	Inline	Same as Inline	Can adversely	Inline hardware

Device type	Basic architecture	Data interface	Timing	Benefits	Caveats	Use cases
	with two-phase execution			Hardware Interface, can function as hardware synchronization master device to drive RTSI 0 line	affect PCL timing	synchronization master device
Asynchronous Timing and Sync	Single loop	FIFOs	Variable	Same as Asynchronous, but can function as hardware synchronization master device to drive RTSI 0 line	1-cycle latency due to FIFOs	Asynchronous hardware synchronization master device

Note Both inline and asynchronous custom devices have advantages and limitations. Consider launching asynchronous loop(s) within an inline custom device to take advantage of the best features of both while overcoming many of their limitations.

Custom Device Type Selection

Choose the type of custom device that most closely resembles the functionality you want to configure. For example, if you want to read and write data to and from a hardware device at the same rate as the PCL, select an inline hardware interface custom device type. However, if you have a serial hardware device, you should select an asynchronous custom device, as serial devices are slow and could affect the PCL timing. Understanding the execution steps of the PCL also can help you select the appropriate custom device type.

Note Consider using an inline mode or hardware interface custom device and, if an asynchronous loop is required, launch an asynchronous loop within that device. Inline model and hardware interface custom devices give you more control and allow you to access channels outside of the custom device.

Custom Device Type Configuration

You configure the custom device type by selecting the appropriate template VI from the Custom Device API library. The type of custom device you select will determine the architecture of the RT Driver VI, but the VIs for configuring the custom device remain the same across all types.

A custom device is not limited to a single device type, and you can alter the code in the template VIs to fit your needs. For example, you can create both inline and asynchronous engines for the same custom device, and use the Set Custom Device Drivers VI to switch between them. However, you must maintain the connector pane, controls, and indicators that the tool and templates include to ensure correct interaction with VeriStand.

Asynchronous Custom Devices

An **asynchronous** custom device executes in a parallel loop with the VeriStand Engine's Primary Control Loop (PCL) and uses RT FIFOs to exchange channel data with the rest of VeriStand.
You can create a standard asynchronous custom device from a template in the Custom Device API library. This also applies to the Asynchronous Timing and Sync device type, which is an asynchronous device that you add to the system definition file as a timing and sync device.

Timing of an Asynchronous Custom Device

The rate at which an asynchronous custom device executes depends on how you configure it. By default, the asynchronous custom device RT Driver VI template uses a While Loop, meaning your asynchronous custom device will execute as fast as possible. You can change the default While Loop to a Timed Loop, and then

configure the Timed Loop to use a specific timing source, such as the timing source for a hardware device.

You can synchronize an asynchronous custom device with the Primary Control Loop by using the Device Clock control as the timing source of your Timed Loop. **Device Clock** is a timing sourced ticked for every iteration of the Primary Control Loop after custom device FIFOs have been updated. If you synchronize your device with the PCL, the dt of your Timed Loop will be in ticks of the PCL. So if you set the dt as 3, your Timed Loop will execute every 3 ticks of the PCL.

Decimation of an Asynchronous Custom Device

You can use Set Custom Device Decimation VI in the initialization code of your asynchronous custom device to change the decimation rate of your device. In an asynchronous custom device, the decimation affects when the Primary Control Loop reads and writes the FIFOs it uses to communicate with the custom device. For example, if you set the Decimation parameter of Set Custom Device Decimation VI to 4, the Primary Control Loop reads and writes the FIFOs on every fourth iteration.

Latency Due to FIFOs in Asynchronous Custom Device

Because asynchronous devices run in parallel with the PCL and pass channel data via RT FIFOs, there is a minimum of one cycle delay from when data leaves the PCL and when it enters the custom device, and vice versa. Additionally, asynchronous devices might not always execute at the same time with respect to the other components of the VeriStand. For example, the first iteration might execute before the PCL processes alarms, the second and third iterations after, and so on.

Using the Asynchronous Custom Device Driver Template

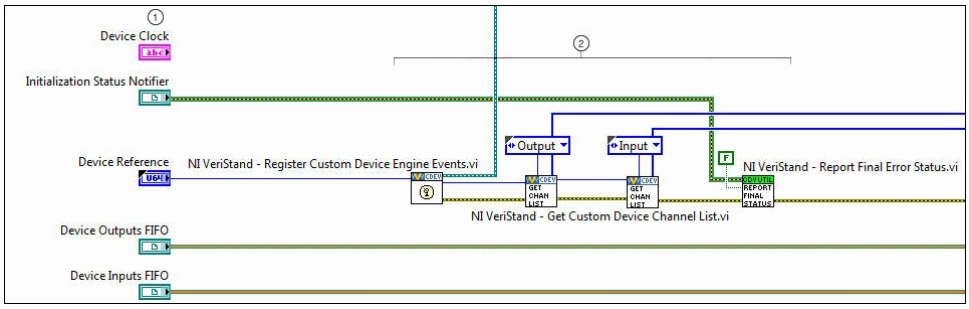
Use the asynchronous custom device driver template to build the RT Driver VI for a custom device.

You can use most of the Custom Device API VIs when building an asynchronous RT Driver VI. The Asynchronous Device Properties VIs and Asynchronous Device Channels VIs configure functionality exclusive to asynchronous custom devices.

An asynchronous custom device uses a two-loop architecture, one loop for receiving commands and one for transferring data, with sections of code for initialization and cleanup before and after the loops, respectively. The template VI uses a While Loop for data transfer, but you also can use a Timed Loop.

In general, you use a While Loop if timing is not important or if you want the loop to run as fast as it can. You use a Timed Loop if you need the loop to execute deterministically or run pseudo-synchronously with the Primary Control Loop.

1. Navigate to the labview\vi.lib\NI VeriStand\Custom Device API directory and open Asynchronous Custom Device Template.vit.
2. Modify the following code to set up your controls and initialization for the custom device.



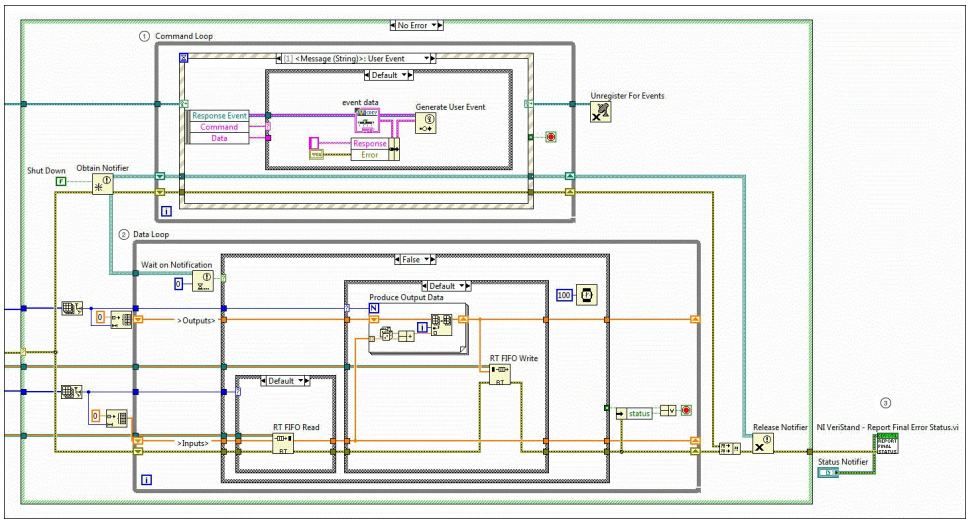
1	Controls—The input controls are specially named controls that the VeriStand Engine will use to provide the asynchronous custom device loops with data. To function, the name of each control must match the following names. ▪ Device Clock—Device Clock specifies the name of a timing source that is ticked for every iteration of the Primary Control Loop after Custom Device FIFOs have been updated. If you change the data loop of your custom device to a Timed Loop, you can use Device Clock as the timing source of the Timed Loop to closely synchronize your asynchronous custom device with

the Primary Control Loop. Device Clock is only populated if you set the Use Device Clock input of the Set Loop Type VI to True in one of the VIs for configuring the custom device.

- Initialization Status Notifier—You can use this optional input to send the VeriStand Engine the final status of the custom device initialization process. If this control exists on the custom device front panel, the VeriStand Engine will wait for a status update before starting up. If the custom device reports an error, that will abort the execution of the current configuration in the VeriStand Engine.
- Device Reference—Device Reference is an auto-populated reference to the custom device. Use it to read configuration properties, get a list of channels, etc.
- Device Outputs FIFO—The array of outputs sent to the system on the Device Outputs FIFO corresponds one-to-one to the Outputs array the Get Custom Device Channel List VI returns. By default, the VeriStand Engine reads the Device Outputs FIFO every iteration of the PCL.
- Device Inputs FIFO—The array of inputs received from the system on the Device Inputs FIFO corresponds one-to-one to the Inputs array the Get Custom Device Channel List VI returns. The VeriStand engine pushes data to the Device Inputs FIFO every iteration of the PCL. If the FIFO is full, the new data packet will overwrite the oldest data packet.
- Status Notifier—Notifies the engine of the last state of the custom device and indicates when the device completes execution. If you do not use this control, the device returns a default No Error value

	when it completes execution. By default, VeriStand does not check this error until shutdown, but you can use an output channel to send more immediate status values to the system.
2	Initialization Code—The template includes initialization code to do the following: a. Register custom engine events. b. Get the number of input and output channels and set up data buffers for the RT FIFOs. c. Read a final error status for the asynchronous custom device. If you use a Timed Loop for your data loop, you can also add code here to configure the Timed Loop.

3. Modify the following code to set up your command loop, data loop, and cleanup code for the custom device.



1	Command Loop—The command loop allows you to send commands to and receive data from your custom device that you can not easily do using a DBL channel value. By

	using the Send Custom Device Message VI in a LabVIEW VI or calling a corresponding .NET method, you can use, for example, a custom workspace object or NI TestStand automation script to send a command to your custom device, which can then execute a response to that command. For example, a generic custom device for logging. If you need to change configuration data, such as the file path to which to save log files, at run time, you could create a custom workspace control to send this data to the command loop of the device, and then configure the command loop to update the configuration data when the data is received. The command loop contains the following three events: a. Message (Byte Array)—Receives and sends data as a byte array of 8-bit unsigned integer values. b. Message (String)—Receives and sends data as a string. c. Shut Down—The VeriStand Engine sends this command to indicate that the custom device should shut down. For an example custom device that uses Send Custom Device Message VI in a LabVIEW VI to communicate directly with the custom device, refer to the labview\examples\NI Veristand\Custom Devices\Communication Example directory.
2	Data Loop—Use this loop to read input data from the Device Inputs FIFO, update the data, and send the updated data via the Device Outputs FIFO to the rest of VeriStand. The template data loop contains code that reads the input data, adds it to a random number, and writes it back to the output channels. The data loop also executes

	shutdown if it receives a shut down notification from the command loop.
3	Cleanup Code—Use the optional Status Notifier control to publish the final error state of your device regardless of errors. If a Status Notifier control is present in the RT driver VI, VeriStand this as an indication that the device has shut down. Otherwise the VeriStand provides default status notification for the device.

Inline Hardware Interface Custom Devices

An **inline hardware interface** custom device executes inline with the VeriStand Engine's Primary Control Loop (PCL) and enables you to read and write data from and to a hardware device.
You can create a standard inline hardware interface custom device from a template in the Custom Device API library. This also applies to the Inline Timing and Sync device type, which is an inline hardware interface device that you add to the system definition file as a timing and sync device.
An inline hardware interface custom device executes as a state machine, or action-engine. The device contains a Case structure, and the PCL calls each case at a specific time with respect to other components of the VeriStand Engine. Within the device, an uninitialized Feedback Node handles iterative data transfer between states.



Note You can use additional Feedback Nodes or other storage mechanisms, such as functional global variables, in an inline custom device.

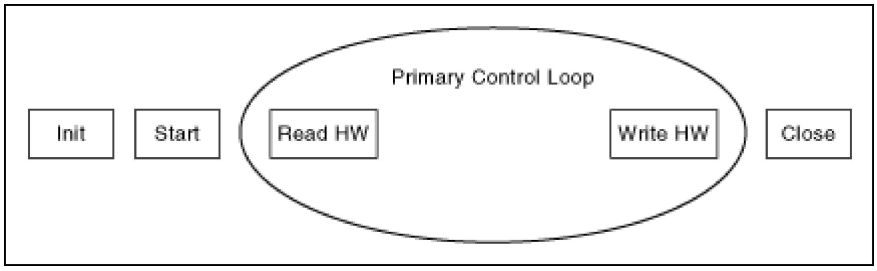
The inline hardware interface custom device is similar to the inline model interface custom device. An inline hardware interface custom device has two cases, or steps, that execute within an iteration of the PCL. An inline model interface custom device has only one step that executes within a PCL iteration.

Using the Inline Hardware Interface Custom Device Driver Template

Use the inline hardware interface custom device driver template to build the RT Driver VI for a custom device.

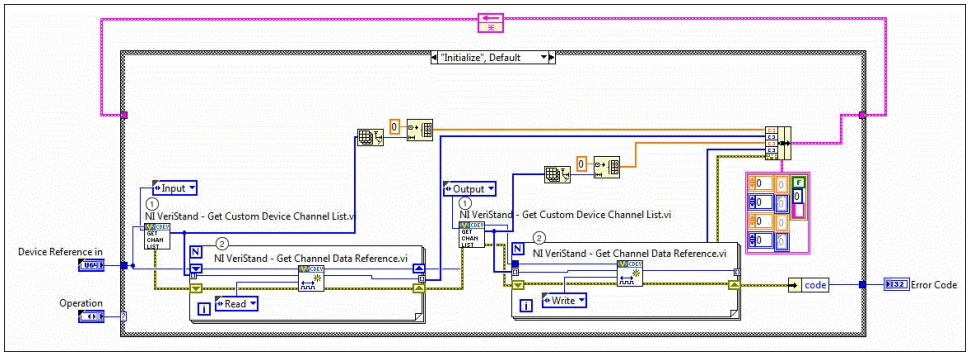
The block diagram of the Inline Custom Device Driver VI Template (HW Interface).vit template VI contains a case structure with five cases to which you can add code to customize the device.

The following image displays the execution order of those cases with respect to the Primary Control Loop (PCL).



1. Navigate to the labview\vi.lib\NI VeriStand\Custom Device API directory and open Inline Custom Device Driver VI Template (HW Interface).vit.
2. Modify the following code to set up the Initialization case that executes before the PCL starts running.

> **Note** Because the Initialize case executes before the PCL starts, you cannot read or write channel values in this case.

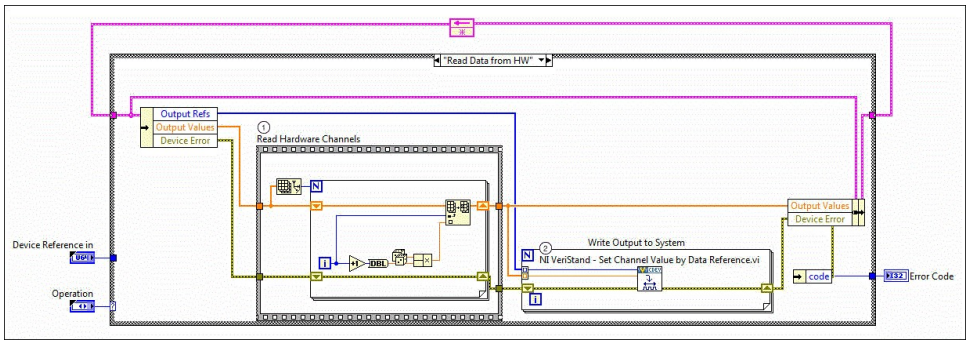


1	Get Custom Device Channel List VI—Gets a list of all the input and output of channels of the custom device.
2	Get Channel Data Reference VI—Compiles a list of channel data references for the channels. In this case, you can also read device configuration information from properties that use a reference to the device. **Note** The Get Channel Data Reference VI does not appear on the Functions palette but belongs to the Custom Device API library. To avoid causing system instability or errors, do not call this VI or the Set Channel Data Reference VI outside of an inline RT Driver VI.

3. Modify the Start case that executes after initialization but before the PCL starts running.
   If necessary, you can use this case to start device tasks, such as DAQ tasks, or to wait for start triggers. Because the Start case executes before the PCL starts, you cannot read or write channel values in this case.

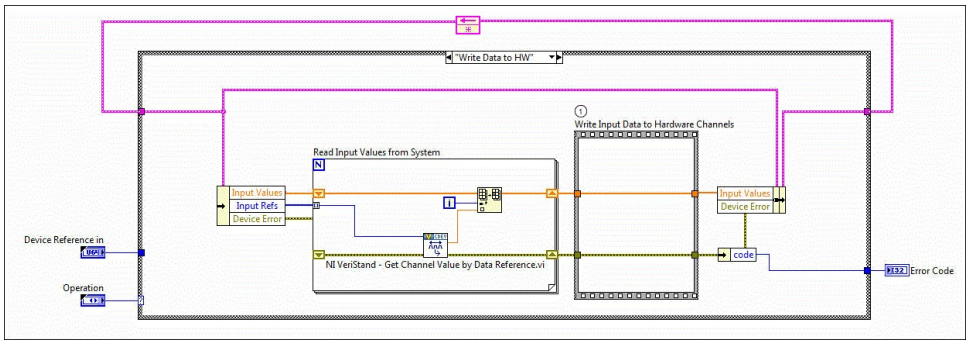
   Note If you use a start trigger in the Start case, you should specify a timeout for waiting on the trigger. Failing to specify a timeout can cause your system to wait indefinitely if the start trigger does not occur as expected.

4. Modify the following code to set up a the Read Data from HW case that executes at the beginning of each iteration of the PCL before other components such as faults, alarms, and procedures.



1	Read Hardware Channels—You can replace this flat sequence structure or the code inside it with the code necessary to obtain data from a hardware device. For example, you can use a hardware device's API calls to request an A/D sample.
2	Set Channel Value by Data Reference VI—Writes the data to a specified channel, making the data available to the other components of VeriStand for the remainder of the PCL iteration.

5. Modify the following code to set the Write Data to Hardware case that executes at the end of each iteration of the PCL after other components such as faults, alarms, and procedures.



1	Write Input Data to Hardware Channels—You can replace the code in this flat sequence

	structure with the code necessary to send data to a hardware device.

6. Modify the Close case that executes after the PCL finishes executing. Use this case to close references and release resources. Because the close case executes after the PCL terminates, you cannot read or write channel values in this case.

Inline Model Interface Custom Devices

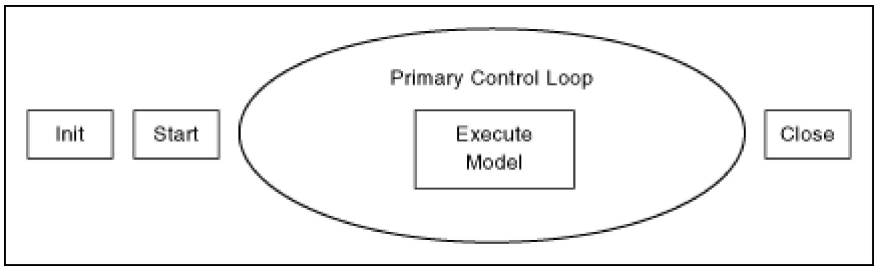
An **inline model interface** custom device executes inline with the VeriStand Engine's Primary Control Loop (PCL), which processes data acquired from hardware inputs and sends the processed values to hardware outputs without latency.
You can create an inline model interface custom device from a template in the Custom Device API library.
An inline model interface custom device executes as a state machine, or action-engine. The device contains a Case structure, and the PCL calls each case at a specific time with respect to other components of the VeriStand Engine. Within the device, an uninitialized Feedback Node handles iterative data transfer between states.
The inline model interface custom device is very similar to the inline hardware interface custom device. An inline model interface custom device has one case, or step, that executes within an iteration of the PCL, while an inline hardware interface custom device has two steps that execute within a PCL iteration.

Using the Inline Model Interface Custom Device Driver Template

Use the inline model interface custom device driver template to build the RT Driver VI for a custom device.

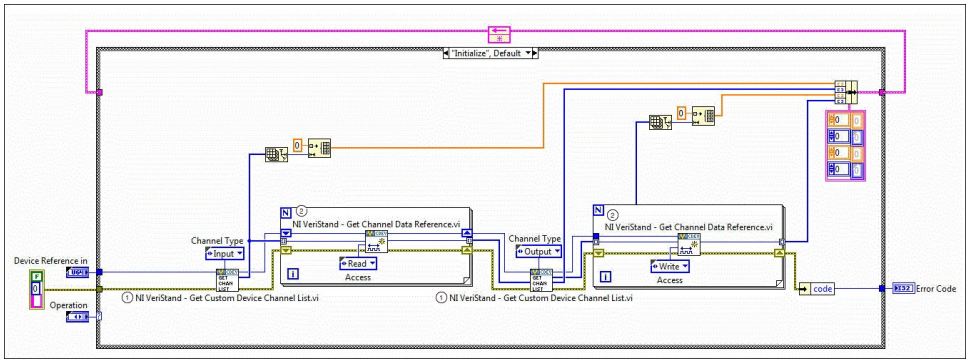
The block diagram of the Inline Custom Device Driver VI Template (Model Interface).vit template VI contains a case structure with four cases to which you can add code to customize the device.

The following image displays the execution order of those cases with respect to the Primary Control Loop (PCL).



1. Navigate to the labview\vi.lib\NI VeriStand\Custom Device API directory and open Inline Custom Device Driver VI Template (Model Interface).vit.
2. Modify the following code to set up the Initialization case that executes before the PCL starts running.

> **Note** Because the Initialize case executes before the PCL starts, you cannot read or write channel values in this case.



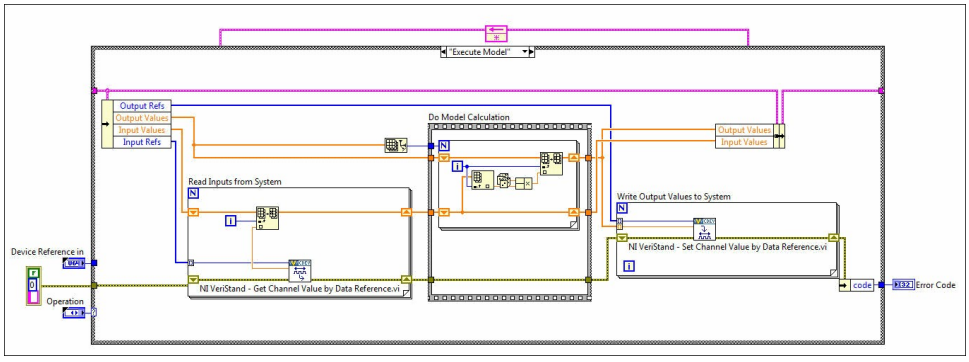
1	Get Custom Device Channel List VI—Gets a list of all the input and output of channels of the custom device.
2	Get Channel Data Reference VI—Compiles a list of channel data references for the channels. In this case, you can also read device configuration information from properties that use a reference to the device.

Note The Get Channel Data Reference VI does not appear on the Functions palette but belongs to the Custom Device API library. To avoid causing system instability or errors, do not call this VI or the Set Channel Data Reference VI outside of an inline RT Driver VI.

3. Modify the Start case that executes after initialization but before the PCL starts running.
   If necessary, you can use this case to start device tasks, such as DAQ tasks, or to wait for start triggers. Because the Start case executes before the PCL starts, you cannot read or write channel values in this case.

   Note If you use a start trigger in the Start case, you should specify a timeout for waiting on the trigger. Failing to specify a timeout can cause your system to wait indefinitely if the start trigger does not occur as expected.

4. Modify the following code to set up the Execution case that executes in the middle of the PCL iteration.



This case reads input data, executes the model, and then writes output data to the rest of VeriStand. **Model** refers to a mathematical function. You may need to average channel data, or you can execute a LabVIEW or other model using the LabVIEW Model Interface Toolkit.

5. Modify the Close case that executes after the PCL finishes executing.
   Use this case to close references and release resources. Because the close case executes after the PCL terminates, you cannot read or write channel values in this case.

If your custom device needs to read or write channel data for multiple channels at a time, consider using block data references in your code.

Implementing Channel Block Reading and Writing in Inline Custom Devices

Modify an inline model interface custom device to use block data references to read and write channel data.

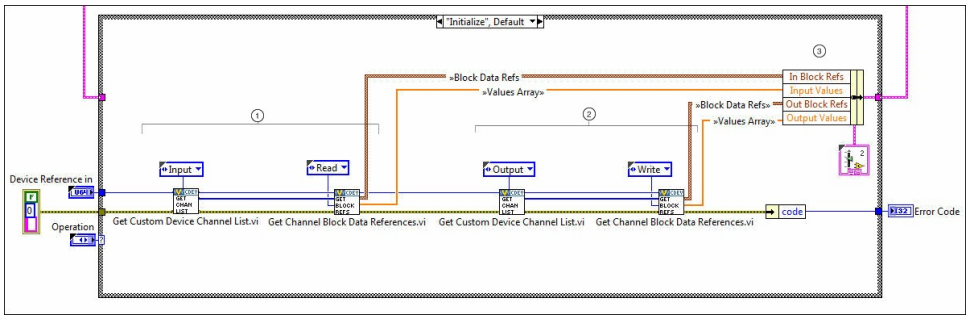
Before you begin, create an inline model interface custom device.

In an inline custom device, you can read or write channel data for multiple channels at a time using block data references. Block reading and writing is useful for custom devices with a large number of channels, as this technique runs faster than channel-by-channel access. Block reading and writing also simplifies your code, as referencing, reading and writing to a large number of channels individually can become large and complex.

Use the following VIs from the Custom Device API library to work with block references:

- Get Channel Block Data References VI
- Get Channel Values by Block Data Reference VI
- Set Channel Values by Block Data Reference VI

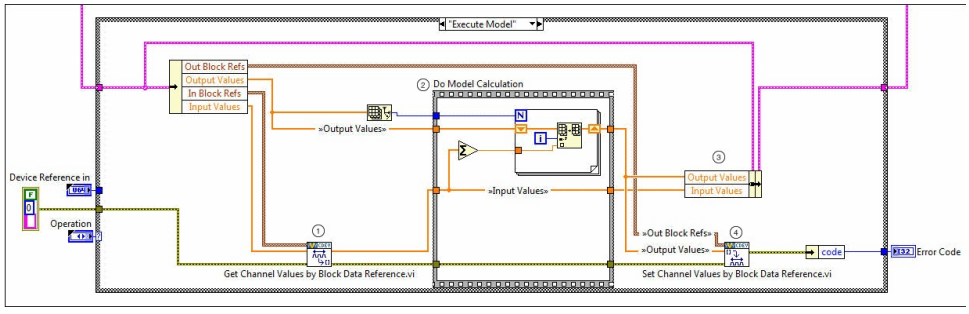
1. Modify the Initialize case to obtain block data references.



1	Get Custom Device Channel List VI and Get Channel Block Data References VI—Replaces the For Loop and Get Channel Data Reference VI that attains Read Access channel data references in the inline model interface custom device template. Get Custom Device Channel List VI and Get Channel Block Data References VI work together as follows: a. Get Custom Device Channel List VI returns an array of all the input channel references of the custom device. b. Get Channel Block Data References VI uses this array to generate Block Data Refs for the input channels, or an array of 32-bit data references, and sets the Access to Read, allowing you to use the references to get channel data. c. Get Channel Block Data References VI also returns the initial channel values as Values Array, which is used later in the Execute Model case. d.
2	Get Custom Device Channel List VI and Get Channel Block Data References VI—Replaces the For Loop and Get Channel Data Reference VI that attains Write Access channel data references in the inline model

	interface custom device template. Works similarly for output channels. Get Custom Device Channel List VI and Get Channel Block Data References VI work together as follows: a. Get Custom Device Channel List VI returns an array of all the output channel references of the custom device. b. Get Channel Block Data References VI uses this array to generate Out Block Refs for the input channels, or an array of 32-bit data references, and sets the Access to Write, allowing you to use the references to write data to the channels. c. Get Channel Block Data References VI also returns the initial channel values as Values Array, which is used later in the Execute Model case.
3	Block Data Refs and Values Array—Stored in a cluster for use in the Execute Model case. The refs and array are of both the input and output channels.

2. Modify the Execute Model case to read and write channel data with block references.



1	Get Channel Values by Block Data Reference VI—Uses the stored In Block Refs of the input

	channels to simultaneously get the value for each input channel.
2	Do Model Calculation Flat Sequence Structure—Sums all values in the Input Values array and saves them to the Output Values array.
3	Input Values and Output Values—Bundles arrays.
4	Set Channel Values by Block Data Reference VI—Uses the Block Data Refs of the output channels to simultaneously write the updated values, which are contained in Output Values array to the output channels.

Timing and Sync Custom Devices

A **timing and sync** custom device is any device that has the capability to drive the RTSI 0 line to serve as the chassis master hardware synchronization device for a system.

The **RTSI 0 line** is a digital line that sends a clock signal to synchronize all hardware I/O devices in the system. You can plan and build a timing and sync custom device from a LabVIEW project. You must modify the VIs of the project to conform to the custom device framework and build the device for distribution to VeriStand.

When you distribute the device, operators can add the device to VeriStand by copying the contents of the Build directory you create into the <Common Data>\Timing and Sync directory on their host computer. VeriStand parses this directory for timing and sync devices when it launches.

After an operator has added the custom timing and sync device to the directory, they can add the device to the system definition file and configure the device as the chassis master hardware synchronization device. Timing and sync devices appear in System Explorer under **Hardware** » **Chassis** » **Timing and Sync**.

Inline Custom Devices with Asynchronous Loops

The RT Driver VI of an inline custom device can communicate channel data with VeriStand while launching an asynchronous loop(s) to handle nondeterministic operations.

One example of a nondeterministic operation is writing data to a log file. The RT Driver VI of the inline custom device communicates with the asynchronous loop(s) using RT FIFOs.
The following table displays the advantages and limitations of using this architecture when compared to inline and asynchronous custom devices.

Device Type	Advantages	Limitations
Inline custom devices	Allow you to read and write data to and from VeriStand in each iteration of the Primary Control Loop (PCL). Allow you to access VeriStand system channels outside of your custom device.	Can introduce latency into the PCL as inline custom devices run inline with the PCL.
Asynchronous custom devices	Allow you to execute large operations without introducing latency into the PCL.	While you can synchronize the loop to the PCL, making the custom device pseudo-synchronous, pseudo-synchronous loops are not guaranteed to iterate once per iteration of the PCL nor are they guaranteed to iterate deterministically with respect to the PCL.
Inline custom devices with asynchronous loops	Allow you to read and write data to and from VeriStand in each iteration of the Primary Control Loop (PCL). Asynchronous loop(s) handle nondeterministic operations, such as writing data to a log file, without introducing latency into the PCL.	Data must be consumed from the RT FIFOs at a fast enough rate or the mechanism will overflow.

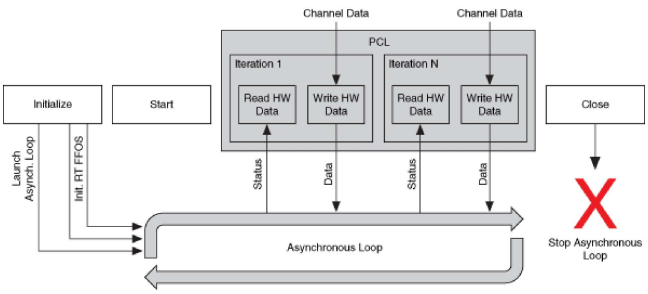
Example: Embedded Data Logger

The Embedded Data Logger is an inline hardware interface custom device that reads and writes data to and from VeriStand in each iteration of the PCL. The Embedded

Data Logger launches an asynchronous loop to log the data it receives to a file to avoid causing latency in the PCL that is associated with file I/O.

The following table displays actions the Embedded Data Logger takes to operate under this architecture.

Note You can find the Embedded Data Logger source code on GitHub.

Action	Description
Initializing RT FIFOs and launching an asynchronous loop	The Initialize case of the Embedded Data Logger RT Driver VI creates two RT FIFOs and launches an asynchronous loop. One RT FIFO communicates channel data from the Embedded Data Logger RT Driver VI to the asynchronous loop. The other RT FIFO communicates state information from the asynchronous loop to the Embedded Data Logger RT Driver VI. The following diagram illustrates how the Embedded Data Logger and asynchronous loop execute with respect to the PCL and how the Embedded Data Logger communicates to the asynchronous loop via RT FIFOs. 
Reading the status of the asynchronous loop in the inline custom device	One of the RT FIFOs created in the Initialize case passes status information from the asynchronous loop to the Embedded Data Logger RT Driver VI. You can see how the Embedded Data Logger RT Driver VI reads data from the RT FIFO by examining the Read Data from HW case. In this case, the Embedded Data

Action	Description
	Logger checks the RT FIFO for a change in the error value.
Sending channel data from the inline custom device to the asynchronous loop	The other RT FIFO sends channel data from the Embedded Data Logger RT Driver VI to the asynchronous loop. This communication takes place in the Write Data to HW case, specifically in the Sample Group Data VI that executes in this case. This VI is responsible for getting the values of the channels to log from the PCL and writing them to the RT FIFO so the asynchronous loop can access them.

Implementing a Custom Device

Develop a custom device to fit your requirements.

Before you begin, you should plan your custom device.

1. Depending on your goal, complete any of the following tasks to configure the appearance and components of your custom device.

Goal	Task
Add custom device channels and waveforms	Add channels and waveforms to a custom device by using the appropriate VI within a Custom Device Library VI that runs on the host computer.
Add custom device item properties	Use VIs on the Item Properties palette to get and set properties of custom device items.
Add custom device pages	Add pages to a custom device by creating the extra page VIs and GUIDs, adding the extra pages to the custom device build specifications, and defining the pages in the Custom Device XML file.
Implement a custom device hierarchy	Implement a flat or nested hierarchy for your custom device.
Add custom glyphs, shortcut menus, and toolbar buttons	Use elements in the Custom Device XML to configure custom user interface

Goal	Task
	components, such as glyphs, toolbar buttons, and shortcut menus.
Add custom error codes	Build custom error codes for your custom device using the General Error Handler VI or the Error Code File Editor.
Automate responses to user actions	Use action VI templates with the custom device XML file to automate responses to user actions.

2. Depending on your goal, complete any of the following tasks to customize the custom device engine by completing the following tasks.

Goal	Task
Sync an asynchronous custom device with the PCL	Configure an asynchronous custom device to run synchronously with VeriStand by configuring the custom device to use the same timing source as the Primary Control Loop (PCL).
Read and write waveforms in the custom device engine	Use waveforms in custom devices to publish waveform data or read waveform data from other sources in the VeriStand Engine.

3. Use tools to benchmark and debug the custom device.

After implementing your custom device, build the custom device.

Adding Custom Device Channels and Waveforms

Add channels and waveforms to a custom device by using the appropriate VI within a Custom Device Library VI that runs on the host computer.

Before you begin, you should understand custom device channels and waveforms.

Typically, you add channels in the Initialization VI so they appear when an operator adds the custom device to the system definition.



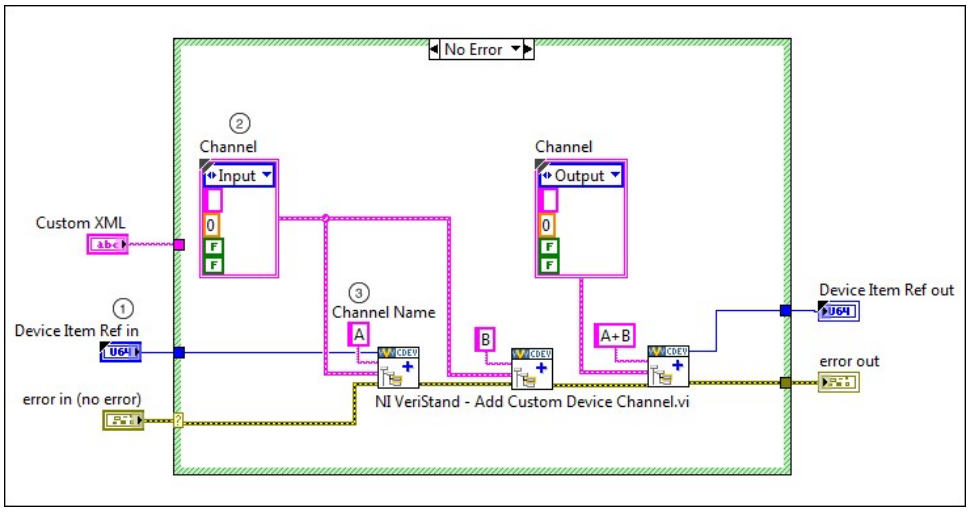
Note Channels can also be added when an operator takes an action, such as using a shortcut menu or toolbar button.

1. Open a Custom Device Library VI that runs on the host computer.
2. Based on the data exchange mechanism you want the custom device to use, add a VI from the Configurations VIs palette.

Mechanism	VI Type
Channel	Add Custom Device Channel VI
Waveform	Add Custom Device Waveform VI

3. Modify the following code in the VI.

Note The following image displays the Add Custom Device Channel VI customized to add two input channels and one output channel. The customizing process for the Add Custom Device Waveform VI is very similar.



1	Device Item Ref in—Provides each instance of the Add Custom Device Channel VI with the reference to the custom device to which to add the channels.
2	Channel cluster—Defines the various properties of the channel, including the type, units, default value, faultability, and

	scalability of the channel. In this example, two input channels and one output channel will be created.
3	Channel Name—Specifies the name of the new channel. In this example, the channel names are A, B, and A+B. Each channel must have a unique name. If the name you specify already exists, this VI overwrites the existing channel settings.

With this configuration, the custom device will create three channels, A, B, and A+B, when you add it to a system definition.

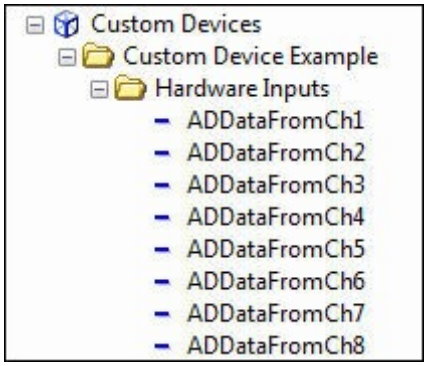
Adding Custom Device Item Properties

Use VIs on the Item Properties palette to get and set properties of custom device items.

Before you begin, you should understand custom device item properties.

While you can call Item Properties VIs from any VI within the custom device, you should call them from VIs that run before the engine is deployed, such as in a page VI or your Initialization VI. You cannot pass property information from the configuration to the engine after the engine is deployed.

The following image displays the hierarchy of a custom device modified to set and read properties.

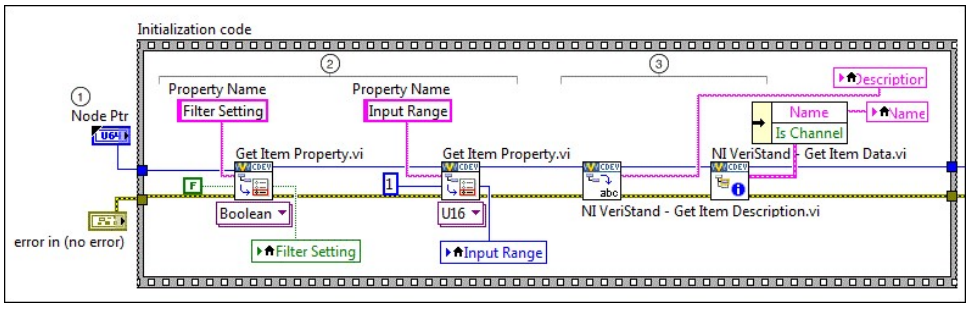


This custom device contains several hardware input channels, each with two item properties. They are Filter Setting and Input Range. The custom device has the following functionality regarding its properties:

- When a user selects a channel from the system definition, the configuration page for that channel initializes Filter Setting and Input Range. However, if the properties already have set values, the configuration page displays the values of the properties.
- The configuration page allows operators to enter new values for each property.
- When an operator deploys the custom device, the RT Driver VI gets the value for each item property so the RT Driver VI can use the values.

To implement this functionality, you have to modify your custom device code.

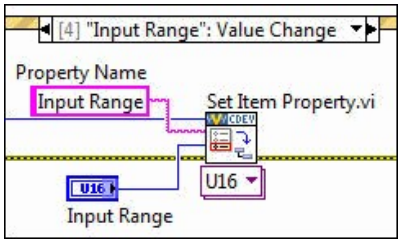
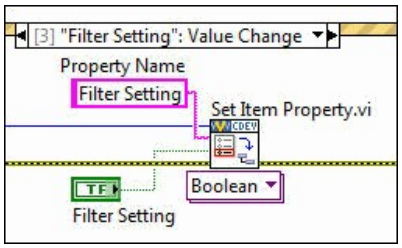
1. Modify the following code to set up the Initialization case to initialize the item properties Filter Setting and Input Range in a page VI for a selected channel. When an operator selects a custom device channel in System Explorer, the page VI associated with that channel or section runs. The page VI then uses a reference to the selected channel or section to get and set the properties for that channel. If the properties are already initialized, the page VI gets the current values for the properties and displays them on the front panel.



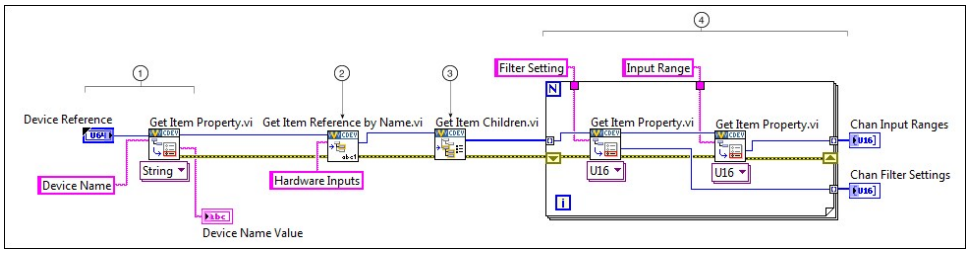
1	Node Ptr—Reference to the channel an operator selects in System Explorer.
2	Get Item Property VIs—Gets the values for the Filter Setting and Input Range properties of the selected channel and then populates the controls on the configuration page with that data. If a value does not already exist, Get Item Property VI initializes the property with Default Value. In this example, the

	default values are False for Filter Setting and 1 for Input Range. **Note** The names of properties are case sensitive strings. To reduce the risk of error, consider storing property names as global variables.
3	Get Item Description VI and Get Item Data VI—Get the description and name of the channel or section, and send that information to indicators on the front panel. These VIs are part of the template page VI.

2. Modify the following code to set up the cases that detect a change of an item property value in a page VI.
   The page VI contains two Value Change event cases, one for the Input Range control and one for the Filter Setting control. If an operator changes the value of the item property using one of these controls, the page VI detects the change in value and the Set Item Property VI sets the new value. The following table shows these cases.

Control	Basic Architecture	Purpose
Input Range		Determine the appropriate range of each channel.
Filter Setting		Determines whether to filter each channel

3. Modify the following code to set up a RT Driver VI to get the property values of Filter Setting and Input Range for each channel.



1	Device Reference and Get Item Property VI—References custom device and gets the Device Name item property to pass to the Get Item Reference by Name VI.
2	Get Item Reference by Name VI—Searches under the custom device for the Hardware Inputs section and outputs the reference to that section.
3	Get Item Children VI—Returns an array containing all the references for all of the children, or items, under the Hardware Inputs section. In the hierarchy, channels ADDataFromCh<1...8> appear under Hardware Inputs, so the Get Item Children VI returns an array of references for the channels.
4	Get Item Property VIs—Returns the values for the Filter Setting and Input Range properties of each channel using the array of channel references.

Adding Custom Device Pages

Add pages to a custom device by creating the extra page VIs and GUIDs, adding the extra pages to the custom device build specifications, and defining the pages in the Custom Device XML file.

Before you begin, you should understand custom device pages.

1. Create a page VI with the required reference using the template VI included in the Custom Device API library.
    a. Open the LabVIEW project for your custom device.
    b. In Project Explorer, browse to **My Computer** » **Custom Device API.lvlib** » **Templates** » **Subpanel Page VI** » **Page Template.vit**.
    c. Right-click **Page Template.vit** and select **New from Template**.
    d. From the front panel of the new VI, save the VI in the folder containing the other VIs for your custom device, such as RT Driver VI and Main Page VI.
    e. Close the VI.
    f. In Project Explorer, drag the new VI to the Custom Device library.
2. Declare the page in the XML file of the custom device.
    a. Open the LabVIEW project for your custom device.
    b. In Project Explorer, browse to **My Computer** » **Custom Device <Name>.xml** and open the XML file of your custom device.
    c. Under the Pages section of the XML file, locate the Page section for the main page of your custom device.
    The declarations for the main page should be the first listed under the Pages section. The name of the main page corresponds to the name of the custom device.
    d. Copy the information between the Page tags, including the <Page> and </Page> declarations, and paste it between the <Pages> and </Pages> declarations.
    e. Replace the information between the <eng> and <loc> decelerations with the name of the new page.
    For example, if you saved your page VI as ExtraPage.vi, enter ExtraPage.
    f. Replace the information between the Path tags with the file path to the new page.

> **Note** You should only need to replace the last token in the path, <Name> Main Page.vi, with your page VI.

g. Change the GUID between the <GUID> tags.

> **Note** To reduce the risk of error when working with GUID string constants, consider using a LabVIEW global variable that is read only or creating a combo box control and saving it as a type definition.

h. Save and close the XML file.

> **Note** Every item with a unique page VI must have a page entry in the Custom Device XML file and a unique GUID. However, items with different GUIDs can reference the same page VI. If you want to create several configuration pages that are only slightly different, you can use the same page VI for each item by configuring the VI to check the associated item GUID at run time. For example, you can add a Case structure to the page VI with a case for each GUID.

3. Add the page to the build specifications.
   a. Open the LabVIEW project for your custom device.
   b. In Project Explorer, expand **Build Specifications**, and double-click **Configuration**.
   c. In the Category menu, select **Source Files**.
   d. In Project Files, expand **<Name> Custom Device.lvlib**.
   e. Select the new page VI, and click **Add Item**.
   The VI is added to the Always Included section.
   f. In the Category menu, select **Source Files Settings**.
   g. In Project Files, expand the **<Name> Custom Device.lvlib**.
   h. Select the new page VI, and from the Destination drop down box, select **<Name> Configuration LLB**.

i. Click **OK** to close the build specification.
j. Save the LabVIEW project.

4. Restart VeriStand.

If your new page is not created properly, you will receive a custom device page error when adding your custom device to the system definition:

Implementing a Custom Device Hierarchy

Implement a flat or nested hierarchy for your custom device.

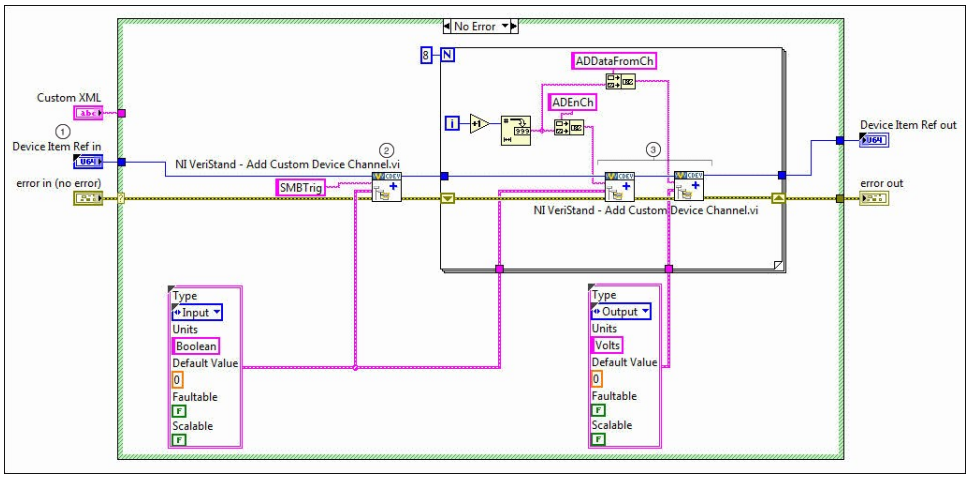
Before you begin, you should understand custom device hierarchies.

A **flat hierarchy** is a hierarchy in which all of the channels appear under one section in the configuration tree. A **nested hierarchy** includes additional sections under the main section, allowing you to organize your channels.

1. Use the following table to determine the type of hierarchy you want in your custom device.

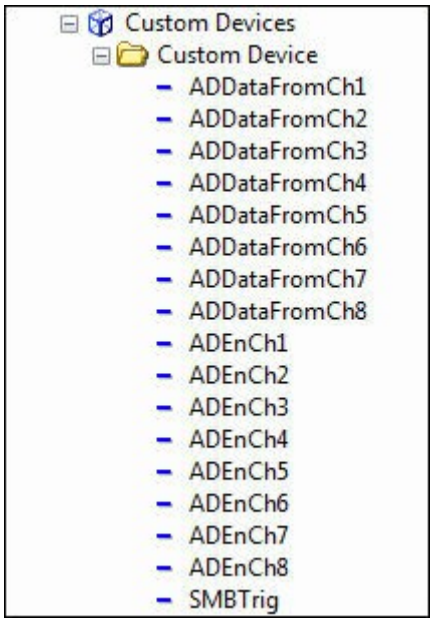
Hierarchy type	Use case
Flat	Best suited for custom devices with a low number of channels.
Nested	Best suited for custom devices with many channels. A nested hierarchy makes the device easier to understand and operate for users.

2. **Optional:** Modify the following code to create a flat hierarchy.

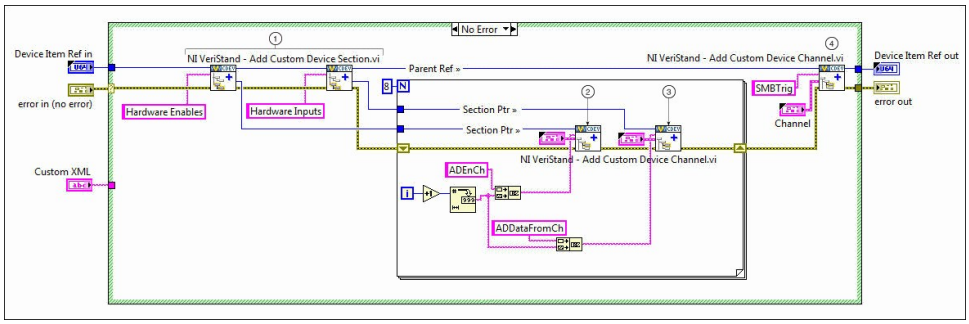


1	The Device Item Ref—Reference to the custom device, and specifies that each of the Add Custom Device Channel VIs will place the channels they create under the main section of the custom device in System Explorer.
2	Add Custom Device Channel VI—Creates an input channel named SMBTrig under the main section of the custom device.
3	Add Custom Device Channel VIs—Creates eight input channels named ADEnCh<1...8> and eight output channels named ADDataFromCh<1...8> under the main section of the custom device.

All channels will appear under the main custom device section when an operators adds the custom device to the system definition file. The following image displays what this hierarchy will look like in System Explorer.



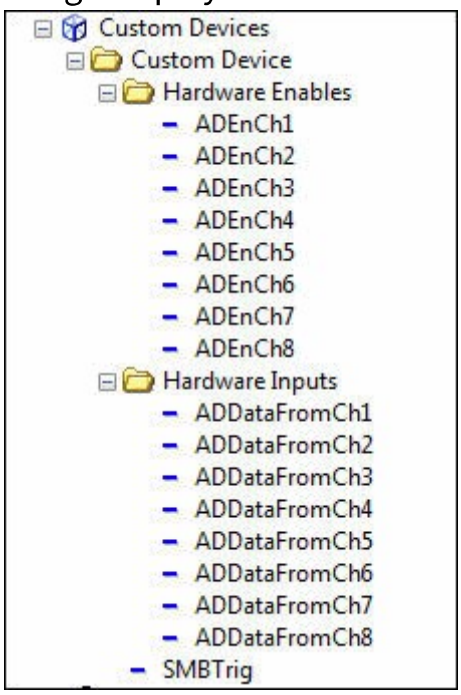
3. **Optional:** Modify the following code to create a nested hierarchy.



1	Add Custom Device Section VIs—Creates two sections, Hardware Enables and Hardware Inputs, under the main section in System Explorer.
2	Add Custom Device Channel VI—Uses the Section Ptr reference to create eight channels, ADEnCh <1...8>, under the Hardware Enables section.
3	Add Custom Device Channel VI—Uses the Section Ptr reference to create eight

	channels, AddDataFromCh <1...8>, under the Hardware Inputs section.
4	Add Custom Device Channel VI—Creates SMBTrig to use Device Item Ref in. Therefore, SMBTrig will appear under the main custom device section.

All channels will appear under the main custom device section when an operators adds the custom device to the system definition file. The following image displays what this hierarchy will look like in System Explorer.



Adding Custom Glyphs, Shortcut Menus, and Toolbar Buttons

Use elements in the Custom Device XML to configure custom user interface components, such as glyphs, toolbar buttons, and shortcut menus.

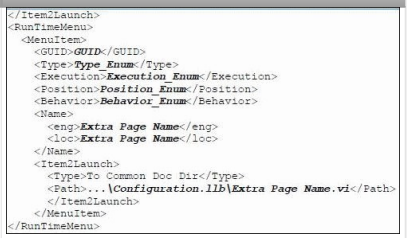
Before you begin, you should understand the custom device XML and the custom device specific XML tags.

1. Open the LabVIEW project for your custom device.
2. In Project Explorer, browse to **My Computer** » **Custom Device <Name>.xml** and open the XML file of your custom device.

3. Depending on your goal, complete any of the following tasks to configure the user interface of your custom device.

Goal	Description	Task
Adding glyphs	A **glyph** is the icon that appears next to an item System Explorer.	Within the <Page> tags for an item, you can use the <Glyph> tag to configure a custom glyph to display for the item. You can use any PNG file as a glyph. A collection of glyphs that install with VeriStand is available in the <Application Data>\System Explorer\Glyphs directory.
Adding toolbar buttons	A **Toolbar button** appears in the toolbar of System Explorer. These buttons only appear when displaying the configuration page associated with the button.	Within the <Page> tags for an item, you can use the <ButtonList> tag to configure the toolbar buttons that appear with the item's configuration page. Each <Button> must include a unique <ID> string that identifies the button. The toolbar button displays by default. However, in each page VI, you can use the Disable Dynamic Button VI and the Enable Dynamic Button VI to dynamically disable and enable a button for that page based on its unique ID. These VIs are useful when you want the toolbar button to appear only when certain conditions are true. These VIs are located in the labview\vi.lib\NI VeriStand\Custom Device API directory.

Goal	Description	Task
		The following is an example framework you can use to implement a toolbar button. 
Adding shortcut menus	A **shortcut menu** for an item is the menu that appears when you right-click the item in **System Explorer**.	Within the <Page> tags for an item, you can use the <RunTimeMenu> tag to configure the shortcut menu for the item. Each <MenuItem> you add under <RunTimeMenu> includes an <Item2Launch> section that specifies a VI to run when an operator selects the menu item. The Custom Device API library includes a template for this VI, RunTimeMenu Custom Item 2 Launch.vit, in the labview\vi.lib\NI VeriStand\Custom Device API directory. The following is an example framework you can use to implement a shortcut menu.

Goal	Description	Task
		

```
</Item2Launch>
<RunTimeMenu>
  <MenuItem>
    <GUID>GUID</GUID>
    <Type>Type_Enum</Type>
    <Execution>Execution_Enum</Execution>
    <Position>Position_Enum</Position>
    <Behavior>Behavior_Enum</Behavior>
    <Name>
      <eng>Extra Page Name</eng>
      <loc>Extra Page Name</loc>
    </Name>
    <Item2Launch>
      <Type>To Common Doc Dir</Type>
      <Path>...\Configuration.llb\Extra Page Name.vi</Path>
      </Item2Launch>
    </MenuItem>
</RunTimeMenu>
```

4. Save and close the XML file.

Adding Custom Error Codes in a Custom Device

Build custom error codes for your custom device using the General Error Handler VI or the Error Code File Editor.

1. Create the error code file in LabVIEW.
2. Move the error code file to the VeriStand errors folder located at <Base>\National Instruments\Shared\Errors\English.

If VeriStand encounters your custom error code, it will display the message you defined in the file.

Automating Responses to User Actions with Action VIs

Use action VI templates with the custom device XML file to automate responses to user actions.

Before you begin, you should understand the action VI templates and the custom device XML file.

Action VIs are dynamically called VIs that execute when a user performs a specific action in a custom device. For example, you can use an action VI to notify a user of the implications of removing a specific custom device item before they delete it.

1. Create a new VI from an action VI template.
    a. From the custom device LabVIEW project, navigate to **Custom Device API.lvlib** » **Templates** » **Action VI**.
    b. Right-click the action VI you want to customize and select **New from Template**.

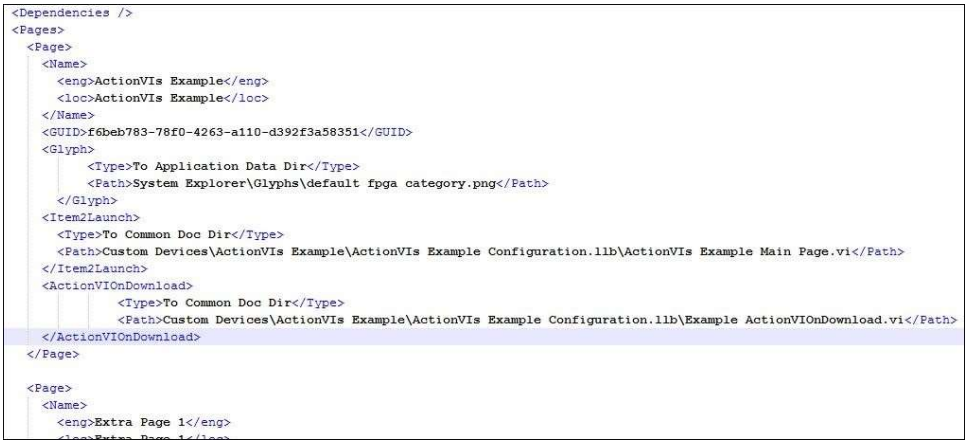
c. Save the new action VI to the same directory as the custom device project.

2. Declare the action VI in the Custom Device XML.
   You declare the action VI within the declaration for a specific page. The action VI executes when a user performs the triggering action on that page.

   a. From the custom device LabVIEW project, open the custom device XML file.

   b. Locate the <Page> tags for the custom device item from which you want to call the action VI.

   c. Declare the action VI anywhere beneath the <Item2Launch> tag.
   The following image gives an example of how to declare the action VI.



3. Customize the action VI to meet your needs.
4. Add the action VI to the Configuration build specification of the custom device.

   a. From the custom device LabVIEW project, double-click **Build Specifications** » **Configuration**.

   b. Select Source Files from the Category menu.

   c. Under the Project Files menu, locate the action VI and move it to the Always Included list.

   d. In the Category menu, select **Source File Settings**.

   e. In the Project Files menu, select the action VI.

f. From the Destination pull-down menu, select the Configuration .llb file.

g. Click **OK** to save the new settings.

Action VI Templates

VeriStand contains eight action VI templates that are triggered by different actions.

The following table displays the action VI templates provided by VeriStand in the Custom Device API library.

Action VI	Description
ActionVIOnLoad	Executes when VeriStand loads a custom device item into memory. This template helps create action VIs that launch background processes. For example, if your custom device requires large amounts of data, you can customize this template to start a daemon that runs processes or gathers data in the background.
ActionVIOnDeleteRequest	Executes when a user tries to delete an item from the custom device. This template helps create action VIs that prevent a user from deleting a custom device item or warn a user of the implications of deleting a custom device item. The template has the following unique parameters. ▪ Item Ref—The reference to the custom device item whose XML declaration calls this action VI. ▪ Refs that are about to get deleted—A 1D array of references to the items to be deleted. However, the 1D array will only contain one reference, as users can only delete one item at a time in the System Explorer window. ▪ Discard reason—An output you can use to capture the user's reason for deleting the item.

Action VI	Description
	▪ Discard delete request?—Allows you to discard the delete request. After the action VI finishes executing, VeriStand will evaluate this output to determine whether or not to delete the item. If True, VeriStand will not delete the item. If False, VeriStand will delete the item. ▪ Additional items to delete—An array of references to additional items you want to delete. For example, if other custom device items depend on the item the user wants to delete, you can use this output to automatically delete those items.
ActionVIOnDelete	Executes after a user deletes an item from the custom device. You can customize this template to alert users which channel mappings break when they delete the custom device item. You can also customize this template to reconfigure hardware. For example, if the user deletes a page that specifies custom configuration data for your hardware, you can customize the template to return the configuration to default settings.
ActionVIOnSystemShutdown	Executes when System Explorer closes. You can customize this template to close hardware connections or to close daemons you launch from an ActionVIOnLoad action VI. The template has the following unique parameters. ▪ Device Item Ref—Reference to the custom device item whose XML declaration calls this action VI. ▪ Unload SDF?—Indicates whether or not the system definition file was unloaded. Unload SDF? is always True.

Action VI	Description
	▪ Saved?—Indicates whether or not a user saved the system definition file before closing System Explorer. ▪ Path—Path on disk to the system definition file. ▪ System Explorer Shutdown?—Indicates whether or not System Explorer closed. This parameter is always True.
ActionVIOnSave	Executes when a user saves the system definition file. For example, you can customize this template to log each time the custom device is saved.
ActionVIOnDownload	Executes when a user deploys the system definition file containing the custom device to a real-time target. This action VI does not execute if a user deploys the system definition to a Windows target. This template helps create action VIs that finalize the target configuration after you deploy the system definition. You can also customize this template to deploy any additional files or dependencies your custom device requires. For example, if your custom device reads and writes to shared variables, you can deploy those variables. The template has the following unique parameters. ▪ Device Item Ref in—Reference to the custom device item whose XML declaration calls this action VI. ▪ ftp session—Open FTP session used to download the system definition to the target. You can use this open session to move additional files to the target. ▪ System Definition Dir—Path to the system definition file on disk.

Action VI	Description
	▪ IP Address—IP address of the target. ▪ ftp session out—Open FTP session used to download the system definition file to the target.
ActionVIOnPaste	Executes when a user pastes a custom device item. This template helps create action VIs that check channel properties. For example, if the user pastes a page that configures a target, you can create an action VI to ensure that the new page does not attempt to reconfigure the target. You can also customize this template to prompt a user to enter new values for the pasted item. For example, if a user pastes a page that will conflict with existing pages, you can prompt the user to enter new values for the page. The template has the following unique parameters. ▪ Ptr in—Reference to the custom device item whose XML declaration calls this action VI. ▪ Parent—Reference to the parent of the custom device item whose XML declaration calls this action VI. ▪ All Ptrs—Array of references to the items the user pasted. You can only select one item to copy. This array only contains one reference that matches the Ptr in reference.
ActionVIOnCompile	Executes when VeriStand compiles the system definition file. **Note** If you deploy the system definition, then undeploy it, and then redeploy it without making changes, this template does not

Action VI	Description
	execute because the system definition does not recompile. You can customize this template to finish configuring your hardware. The system definition file compiles when a user deploys the system definition, so you can configure your hardware based on the final settings from the system definition. You can also customize the template to quickly gather host-side settings. For example, often the custom device RT Engine VI uses properties set in the system definition. You can customize this template to read the values on the host side, which is much faster than reading them from the real-time target. You can then gather the properties into a single cluster, convert that cluster to a data variant, and write the variant as a single item property.

Synchronizing an Asynchronous Custom Device with the Primary Control Loop

Configure an asynchronous custom device to run synchronously with VeriStand by configuring the custom device to use the same timing source as the Primary Control Loop (PCL).

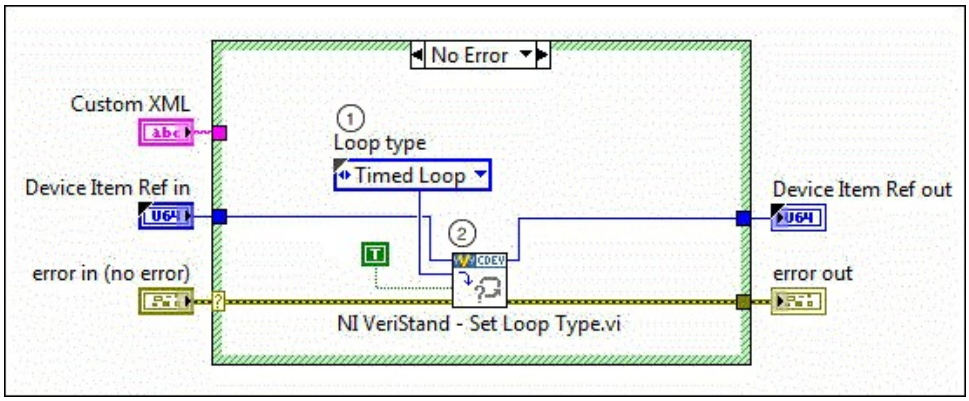
Before you begin, you should understand asynchronous custom devices and the VeriStand Engine.

VeriStand can use a DAQ device as the timing source for the PCL. You can configure the RT Driver VI of your asynchronous custom device to use the same timing source.

When you synchronize your custom device to the PCL, your custom device will not delay PCL if it finishes late.

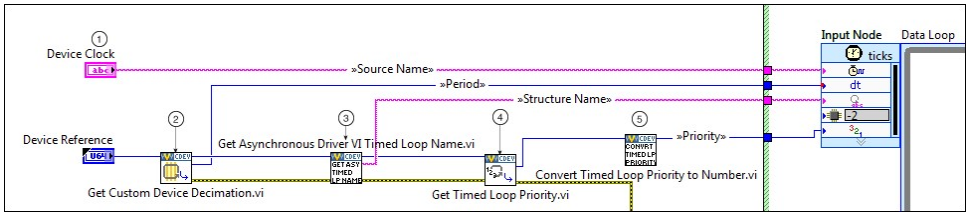
1. Set up an asynchronous custom device.
2. Modify the following code to add and configure the Set Loop Type VI in the Initialization VI.

You can add and configure the Set Loop Type VI in any VI for configuring the custom device, such as a page VI. For example, if you want a user to be able to specify whether or not to synchronize the custom device to the PCL, you could add and configure the Set Loop Type VI in the Main Page VI of the custom device.



1	Loop type—Specifies that the custom device will use a Timed Loop.
2	Set Loop Type VI—When Use Device Clock (Timed Loops only) input is True, VeriStand passes the timing source from the PCL to the Device Clock control in the RT Driver VI.

3. Modify the following code to configure the RT Driver VI to use the PCL Timing Source.
   Before configuring your asynchronous RT Driver VI you must change the device's Data Loop from the default While Loop to a Timed Loop. You can do this by right-clicking the While Loop and selecting **Replace with Timed Loop**.



1	Device Clock—Specifies the name of a timing source that is ticked for every iteration of the PCL after the Custom Device FIFOs have been updated. VeriStand passes this timing source to the custom device. By wiring Device Clock to Source Name, the Timed Loop will run according to the same timing source as the PCL.
2	Get Custom Device Decimation VI—Returns the decimation factor of the custom device, or how many iterations of the PCL occur between calls to the custom device. This decimation factor serves as the Period of the Data Loop Timed Loop. The default value, 1, specifies no decimation, meaning the PCL will call the custom device on every iteration. This configuration allows you to use the Set Custom Device Decimation VI in one of the VIs for configuring your custom device, such as Initialization VI or a page VI, to specify a decimation.  **Note** For asynchronous custom devices, the decimation only affects when the Primary Control Loop reads and writes the FIFOs it uses to communicate with the custom device.
3	Get Asynchronous Driver VI Timed Loop Name VI—Returns the System Explorer path to the custom device as a string. This string serves as the unique Structure Name of the Data Loop Timed Loop, ensuring the VeriStand Engine synchronizes the start of this Timed Loop with the start of the PCL.
4	Get Timed Loop Priority VI—Outputs the priority of the Timed Loop. The priority can be low, medium, or high.

5	Convert Timed Loop Priority VI—Converts this enumeration value to a numeric value that the Priority terminal of the Timed Loop input node accepts.

Reading and Writing Waveforms in the Custom Device Engine

Use waveforms in custom devices to publish waveform data or read waveform data from other sources in the VeriStand Engine.

Before you begin, add a waveform to your custom device.

1. Open your waveform custom device in LabVIEW.
2. Use the VIs on the Waveform Data palette to open a write session to the waveform in the RT Driver VI of your custom device.

VI	Description
Get Waveform Data Reference	Generates a data reference to a specific waveform. If you want to write multiple waveforms, call this VI once per waveform and build an array of data references.
Open Waveform Session	Opens a write session for one or multiple waveforms. You should specify the delta t (dt), or time interval in seconds between any two points in the signal. For example, the dt of a 10 kHz waveform is 0.0001 seconds.
Start Waveform(s)	Provides a start time (t0) that defines when the first sample is written. Readers of the waveform in the VeriStand Engine and on the host computer will receive this value. If you want to read data from another waveform or from the DAQmx API, the t0 is provided. However, if you want to read waveform data from an FPGA, you must calculate the t0 yourself.
Write Waveform(s)	Writes an array of waveform samples. Call this VI repeatedly as you generate data for the waveform. You can write as much or as little data at a time as you want. If you opened a write session for multiple

VI	Description
	waveforms, this VI writes to each waveform simultaneously.
Close Waveform Session	Closes the write session.

3. Use the VIs on the Waveform Data palette to implement a read session from the waveform in the RT Driver VI of your custom device.

VI	Description
Get Waveform Data Reference	Generates a data reference to a specific waveform. If you want to read multiple waveforms, call this VI once per waveform and build an array of data references.
Open Waveform Session	Opens a read streaming session for one or multiple waveforms.
Read Waveform(s)	Returns an array of values from the waveform whenever any waveform source publishes data. Call this VI repeatedly to read data as it is published.
Close Waveform Session	Closes the waveform streaming session.

For an example of a custom device that reads and writes waveforms, on a computer with LabVIEW installed, browse to the labview\examples\NI Veristand\Custom Devices\Waveform Analysis and Generation directory and open Waveform Analysis and Generation Custom Device Project.lvproj.

After you have set up your custom device, review the other considerations for reading and writing waveforms in a custom device engine.

Other Considerations for Reading and Writing Waveforms in a Custom Device Engine

When reading and writing waveforms in a custom device, you should try to avoid data loss, identify the value source when reading from multiple waveforms, and calculate the start time of a read operation.

Avoiding Data Loss When Reading and Writing Data

VeriStand uses queues to transfer data between custom devices and the Waveform Processing Loop (WPL), which transfers the waveform data through the system. By

default, VeriStand automatically calculates the size of the queue used to hold waveform data. The calculations, described as follows, depend on whether you open a read or write session with the Open Waveform Session VI:

- Write sessions—The queue size is three elements, so the custom device can write three times without the WPL reading from the queue before data loss occurs.
- Read sessions—The queue size is three times the number of waveforms being read. For example, if you read from ten waveforms, the queue size is 30.

If you notice data loss, you can specify a larger, custom queue size for the VeriStand Engine to use. When you open a read or write session with the Open Waveform Session VI, use the elements of the Communication Properties input cluster to define a custom queue size. If the CPU cannot read or write as quickly as the custom device requires, changing the queue size will not resolve data loss, only delay it.

For read sessions, you can monitor the WPL Overflow Count system channel to determine if a queue overflows. For write sessions, use the timed out? output of the Write Waveform(s) VI to indicate when data loss occurs.

Identifying the Source of Values When You Read from Multiple Waveforms

If you open a read session for multiple waveforms, the Read Waveform(s) VI executes whenever any waveform returns data, and the VI returns data from only that waveform. You can identify which waveform the data came from during a particular execution using the Data reference element that the VI returns in a cluster output called Properties. Compare the Data reference to the data references you generated before you opened the waveform session.

Calculating the Start Time of a Read Operation

When you read from a waveform with the Read Waveform(s) VI, the VI returns a cluster that includes a t0 at start element. t0 at start is the start time of the first sample in the waveform, not the start time of the first sample in the current array of read values. You can calculate the start time of the first sample read during a particular read operation using the following equation:

```
current t0 = t0 at start + (dt * offset from start (samples))
  where  t0 at start is the start time of the first sample in the waveform,
         dt is the time between samples, and
         offset from start (samples) is the number of samples by which the
first sample in the array of read values is offset from the first sample in the
waveform.
```



Note The Read Waveform(s) VI returns dt, offset from start (samples), and t0 at start in a cluster output.

Custom Device Benchmarking and Debugging

You can use tools provided by LabVIEW and VeriStand to benchmark and debug your custom device.
You should perform benchmarking on a system that is similar to the target. Other components of the VeriStand system, such as models, calculated channels, alarms, and procedures, affect the ultimate execution speed of the system.
The following table displays various tools you can use to benchmark and debug VeriStand custom devices.

Tool	Purpose	Granularity	Location	Details
LabVIEW Debugging Tools	Debugging	N/A	LabVIEW	You can merge debugged VIs into the Custom Device Framework manually. Timing can differ between standalone VIs and VIs in the Custom Device Framework. This tool is not available after VeriStand integration. For more information, refer to the Debugging

Tool	Purpose	Granularity	Location	Details
				Techniques topic in **LabVIEW Help**.
Console Viewer	Benchmarking (CPU) and debugging	Low	VeriStand	A VeriStand workspace tool that displays system definition details, CPU usage, and debugging messages. The tool takes periodic snapshots and prints messages sent by the Print Debug String VI. This tool is only available on PharLap real-time (RT) targets. CPU spikes and transients might not appear.
Custom Error Codes	Debugging	N/A	LabVIEW and VeriStand	You can define custom error codes in LabVIEW and distribute the codes to VeriStand with a custom device. Copy the custom errors.txt file to VeriStand in the <Base>\National Instruments\Shared\Errors\English directory and add the file as a dependency in custom device

Tool	Purpose	Granularity	Location	Details
				and Custom Device XML file. For RT targets, deploy the errors.txt file to the error directory on target to display error descriptions in Console Viewer. For more information, refer to the Defining Custom Error Codes to Distribute throughout Your Application topic in the **LabVIEW Help**
Print Debug String VI	Debugging	NA	LabVIEW	This VI prints messages to the RT console and the VeriStand data log. This tool works on Windows and RT targets.
NI Distributed System Manager	Benchmarking (CPU and RAM)	Medium	Installs with LabVIEW.	The manager works with network variables and manages remote target settings and the status of the Shared Variable Engine. This tool takes periodic snapshots.

Tool	Purpose	Granularity	Location	Details
				To use this tool, you must install System State Publisher on the RT target. CPU spikes and transients might not appear.
System Channels	Benchmarking and debugging	High	VeriStand	Useful system channels include: ▪ HP Count ▪ HP Loop Duration ▪ LP Count ▪ Model Count You can use these channels with alarms or procedures.

Building a Custom Device

Build a custom device for distribution to VeriStand.

Before you begin, you should start implementing your custom device. It will take several iterations of implementing and building to create your custom device.

The process for building the source distribution for a custom device is the same as the process for building any source distribution in LabVIEW. When you build a custom device, the Build Specifications item in the LabVIEW project must include source distributions called Configuration and Engine.

Note For more information on building source distributions, refer to the **LabVIEW Help** by selecting **Help** » **LabVIEW Help** in LabVIEW.

1. Open your LabVIEW project.

2. Create a Configuration source distribution.
    a. In Project Explorer, right-click **Build Specifications** and select **New** » **Source Distribution**.
    b. In the My Source Distribution Properties dialog box, click **Information**, and enter the **Build specification name** as Configuration.
    c. Enter a **Destination directory**.
    d. Click **Source Files** and determine which files to include and exclude. All dependency files that the custom device accesses on the target must exist in the LabVIEW project and appear in the Always Included list for the appropriate source distribution. You should include any dynamically-loaded dependencies, such as pages and the Initialization VI.

    > **Note** You must include the Custom Device XML file in the Configuration source distribution.

    e. Click **Destinations** and set the Destination type to **LLB**.
    f. Click **Additional Exclusions**, and disable **Modify project library file after removing unused members**.

    > **Note** This option can cause an error when you load the custom device in VeriStand.

    g. Click **OK**.
3. Create an Engine source distribution.
    a. In Project Explorer, right-click **Build Specifications** and select **New** » **Source Distribution**.
    b. In the My Source Distribution Properties dialog box, click **Information**, and enter the **Build specification name** as Engine.
    c. Enter a **Destination directory**.



Note Use the same Destination directory as the Configuration source distribution. Using one directory makes it easier to distribute the custom device.

a. Click **Source Files** and determine which files to include and exclude.
b. Click **Destinations** and set the Destination type to **LLB**.
c. Click **Additional Exclusions**, and disable **Modify project library file after removing unused members**.
d. Click **OK**.

4. Save the LabVIEW project.
5. Right-click **Build Specifications** and select **Build All**.

After building the custom device, distribute it.

Distributing Custom Devices

Package a custom device to manually distribute it to VeriStand.

Before you begin, you must build the custom device.



Note For information on building installers, refer to the **LabVIEW Help**.

1. On your development machine, create a top-level directory to contain the custom device files.
2. Within the new directory, create two sub-directories and label them Build and Source.
3. Copy the source distributions you built from the custom device project into the Build directory.
   Include both the LLBs and the Custom Device XML file.
4. Copy the LabVIEW project you used to create the custom device into the Source directory, along with any supporting files and dependencies.

5. On the same level as the Build and Source directories, create a readme file that includes any instructions an operator needs to install, license, use, and/or modify the custom device.
6. Create a ZIP file from the top-level directory.

You now can distribute the custom device ZIP file to any operator using a corresponding version of VeriStand. The operator can add the custom device to VeriStand by copying the contents of the Build directory into the <Common Data>\Custom Devices directory or, for timing and sync devices, the <Common Data>\Timing and Sync directory on their host computer. VeriStand parses these directories for custom devices when it launches.

Note The path to the shared directory varies based on your operating system.

To enable an operator with LabVIEW access to modify and rebuild the custom device, you can provide the files in the Source directory.

Customizing an FPGA Target

To run a project on an FPGA target, you need a bitfile (.lvbitx) and an FPGA configuration file (.fpgaconfig).

VeriStand includes bitfiles and FPGA configuration files for many FPGA targets. The default bitfiles and configuration files are sufficient for many applications. If you want to use additional digital I/O lines, use more than two PWM outputs, or have digital filtering built into the FPGA target, you must create a custom bitfile and configuration file.

Note Install the LabVIEW FPGA Module to create these files.

1. Copy the sample FPGA VI and project—Create a template by making a copy of a sample FPGA VI and project.
2. Customize the FPGA VI—Modify an FPGA VI to match your hardware device.
3. Compile the FPGA VI into a bitfile—Prepare to use your FPGA VI in an FPGA target by compiling the VI into a bitfile.

4. Create an FPGA configuration file—Create an FPGA configuration file for the host computer.

Copying the Sample FPGA VI and Project

Create a template by making a copy of a sample FPGA VI and project.

To develop an FPGA VI and project, you must install matching versions of LabVIEW and VeriStand. For example, you must use LabVIEW 2020 to develop for VeriStand 2020.

1. Browse to the <Common Data>\FPGA\Templates directory.
2. Create a copy of NI VeriStand IO PXI-7854R.lvproj in the same directory and open the copy in LabVIEW.

> **Note** If you are using a CompactRIO FPGA target, create a copy of NI VeriStand FPGA IO cRIO.lvproj instead.

3. In Project Explorer, under My Computer, expand **FPGA Target (PXI-7854R)**.
4. **Optional:** To define a target other than PXI-7854R, add it to the project.
    a. Right-click **My Computer** and select **New** » **Targets and Devices** from the shortcut menu.
    b. In the Targets and Devices dialog box, select the **New target or device**.
    c. Select the device type from the list and click **OK**.
    d. In Project Explorer, under the FPGA Target (PXI-7854R), drag **NI VeriStand FPGA DMA IO.vi** to the new target.
    e. Under the FPGA Target (PXI-7854R) target, drag the **DMA_WRITE** FIFO and **DMA_READ** FIFO to the new target.
    f. Right-click the new target and select **New** » **FPGA I/O** to add the connectors available on the new target.

> **Note** The FPGA VI displays broken wires from any FPGA I/O Nodes with undefined channels. For more information on

> adding I/O to a project, refer to the **LabVIEW FPGA Module Help**.

5. In Project Explorer, double-click NI VeriStand FPGA DMA IO.vi.
6. In the NI VeriStand FPGA DMA IO VI, select **File** » **Save As**.
7. Enable **Substitute copy for original** and click **Continue**.
8. Rename the VI and click **Save**.
9. In Project Explorer, save the project.

After creating your FPGA VI and project, customize the VI.

Customizing an FPGA VI

Modify an FPGA VI to match your hardware device.

Before you begin, copy a sample FPGA VI and project and learn about FPGA customization guidelines and defaults.

1. Open the FPGA VI in LabVIEW.
2. Add or remove FPGA I/O items depending on the device and the needs of the project.
   By default, the sample FPGA VI only uses the first 40 lines on connectors 1 and 2. You can add more FPGA I/O items to this project if you want to expose addition I/O lines on your target.

   > **Note** For FPGA targets with no analog inputs or outputs, you can remove the analog I/O items from the project and the corresponding FPGA I/O Nodes from the FPGA VI.

   Similarly, the default sample FPGA VI defines the digital lines on connector 0 as 8 PWM inputs and 8 PWM outputs. You may need more or fewer PWM channels. You can add other custom I/O not defined in the sample FPGA VI.
3. **Optional:** If the FPGA VI displays broken wires to FPGA I/O nodes, update the corresponding I/O nodes with the correct pins available on the target.
4. **Optional:** If the number of packets in either the DMA_READ or DMA_WRITE FIFO is greater than 15, update the FIFO size.

a. In Project Explorer window, right-click a FIFO I/O item and select **Properties**.
b. In the FPGA FIFO Properties dialog box under General, change the **Number of Elements**, and click **OK**.

5. Save the FPGA VI.

After customizing the FPGA VI, compile the VI into a bitfile.

FPGA VI Customization Guidelines and Defaults

When modifying an FPGA VI, be aware of the guidelines and defaults.

Guidelines

Use the following guidelines to avoid creating errors.

- Do not modify, remove, or rename block diagram objects in the gray areas of the sample FPGA VI.
- Do not modify the read or write code except to change the number of packets or to change the size of the array constant for the DMA read operation of the DMA_WRITE FIFO.
- Ensure that the name of each control is unique within the VI.
- Do not use the following control/indicator names: Loop Rate (usec), Write to RTSI, Use External Timing, Reset, Start, or Generate IRQ.



Note For more information on creating FPGA VIs and bitfiles for an FPGA target, refer to the **LabVIEW Help** by selecting **Help** » **LabVIEW Help** in LabVIEW.

Defaults

The process of creating a custom FPGA VI differs depending on the hardware devices you are using. The default project defines the following FPGA I/O items for the PXI-7854R device:

- analog input channels 0–7

- analog output channels 0–7
- digital lines 0–39 on connectors 1 and 2
- digital lines 0–15 on connector 0

VeriStand uses direct memory access (DMA) FIFOs to transfer data between the host computer and FPGA target. The **DMA_READ FIFO** sends data read from the FPGA inputs to the host computer. The **DMA_WRITE FIFO** transfers data from the host computer to the FPGA outputs. The data is stored in packets that each can contain up to 64 bits. For example, you can pack four 16-bit signed (I16) integer values into a single 64-bit packet. You can pack values of different data types together in the same packet. If you add a channel to the FPGA VI, you also must add the channel to a packet that is written to the FIFO.



Note Use the Join Numbers or Split Number function to construct packets.

Compiling a Custom FPGA VI into a Bitfile

Prepare to use your FPGA VI in an FPGA target by compiling the VI into a bitfile.

Before you begin, create a customized FPGA VI.

FPGA VIs define the analog, digital, and pulse width modulation (PWM) inputs and outputs of an FPGA target. A **bitfile** contains the information that FPGA targets need to function as the FPGA VI specifies.



Note For more information on compiling FPGA VIs, refer to the **LabVIEW Help** by selecting **Help** » **LabVIEW Help** in LabVIEW.

1. Open LabVIEW.
2. In Project Explorer, right-click the FPGA VI, and select **Compile**.

The compiler places the bitfile in an FPGA Bitfiles subdirectory relative to the project file directory. By default, the bitfile name is <name of project>_<name of FPGA VI>.lvbitx.

After creating a bitfile, create an FPGA configuration file.

Creating a Custom FPGA Configuration File

Create an FPGA configuration file for the host computer.

Before you begin, compile your custom bitfile.

For each FPGA bitfile, VeriStand requires an FPGA configuration file. The **FPGA configuration file** is an XML file that the host computer uses to determine the DMA FIFO properties and how the FPGA device appears in System Explorer.

1. Navigate to the <Common Data>\FPGA directory and create a copy of an existing default configuration file.
2. Modify the FPGA configuration file using XML tags.
3. Save the file to the <Common Data>\FPGA directory using the .fpgaconfig file extension.
   If you choose to save the file in a different directory, save a copy of the NI VeriStand FPGA DMA.xsd file, NI VeriStand FPGA DMA.xsl file, and the associated .lvbitx file in the same directory. If these files are not saved in the same directory, VeriStand cannot load the FPGA configuration file.

> **Note** FPGA configuration files must have at least one space in the file name.

FPGA Configuration File XML Tags

Use XML tags and structure to customize an FPGA configuration file.

The first line of the FPGA configuration file includes the XML version. All other tags inside the file must be enclosed in <FPGADMAChannelData> tags.

```
<?xml version="1.0"?>
<FPGADMAChannelData>
<!-- Insert tags here -->
</FPGADMAChannelData>
```

XML schema files (.xsd) are definition files that constrain an XML file to a certain format. You can add a schema file to most XML editing tools when writing an XML file. Use the NI VeriStand FPGA DMA.xsd schema file in the <Common Data>\FPGA

directory to minimize syntax and formatting errors when you create an FPGA configuration file.

The following table displays the XML tags you can use in the FPGA configuration file. To see how to implement these tags, refer to the example FPGA Configuration File Structure.

Tag	Required?	Parent tag	Number of child tags	Description
FPGADMAChannel Data	Yes	—	—	Contains all channel definitions.
Version	Yes	FPGADMAChannel Data	1	Defines which version of the channel tags you used to create the FPGA configuration file. For example, if you used the tags defined in this table, you must set the Version tag to 2.0.
Bitfile	No	FPGADMAChannel Data	1	Specifies the name of the corresponding bitfile (.lvbitx). The bitfile must be saved in the same directory as the FPGA configuration file. The default value is <name of FPGA configuration file>.lvbitx.
Categories	No	FPGADMAChannel Data	1	Contains multiple category definitions and

Tag	Required?	Parent tag	Number of child tags	Description
				describes the hierarchy of the channels visible in System Explorer. You cannot nest <Categories>.
Category	Yes	Categories	0 or more	Defines a single level of the configuration tree hierarchy in System Explorer. You can nest <Category>. If you do not specify <Category> elements, the configuration tree hierarchy is inferred based on the <Category> tags contained within individual channels.
Name	Yes	Category	1	Defines the name of <Category> in System Explorer configuration tree. <Name> must be unique within its set of siblings.
Description	No	Category	1	Specifies the description of <Category> in System Explorer.
Symbol	No	Category	1	Defines the symbol for <Category> in the System Explorer

Tag	Required?	Parent tag	Number of child tags	Description
				window configuration tree. You can select from the following values: ▪ Default ▪ AI ▪ AO ▪ DI ▪ DO ▪ PWM In ▪ PWM Out
DMA_Read	Yes	FPGADMAChannelData	1	Contains packet definitions. Specifies the content of the DMA_Read FIFO.
DMA_Write	Yes	FPGADMAChannelData	1	Contains packet definitions. Specifies the content of the DMA_Write FIFO.
Packets	Yes	DMA_XXX*	1	Defines the number of unsigned 64-bit packets contained in the DMA FIFO. If the number of <Packet> elements is less than the number specified in <Packets>, VeriStand ignores the last <Packet>.

Tag	Required?	Parent tag	Number of child tags	Description
Packet	No	DMA_XXX	1 or more	Specifies the content of a single unsigned 64-bit channel in the DMA FIFO. You do not have to use all of the available bits in a packet. You also can specify an empty packet using the <Packet/> tag with no closing tag. An empty <Packet> element specifies a packet that is to be ignored. You might want to use an empty packet for the first DMA_Read packet because the first bit of the first DMA_Read packet contains a Late Status field by default. If you specify an empty packet for the first DMA_Read packet, this unusable Late Status bit does not appear in System Explorer. If you want the Late Status bit to be visible in System Explorer, specify it as a Boolean

Tag	Required?	Parent tag	Number of child tags	Description
				channel in the first DMA_Read packet.
I8	No	Packet	0 or more	Specifies a signed 8-bit channel in the DMA FIFO.
U8	No	Packet	0 or more	Specifies an unsigned 8-bit channel in the DMA FIFO.
I16	No	Packet	0 or more	Specifies a signed 16-bit channel in the DMA FIFO.
U16	No	Packet	0 or more	Specifies an unsigned 16-bit channel in the DMA FIFO.
I32	No	Packet	0 or more	Specifies a signed 32-bit channel in the DMA FIFO.
U32	No	Packet	0 or more	Specifies an unsigned 32-bit channel in the DMA FIFO.
I64	No	Packet	0 or more	Specifies a signed 64-bit channel in the DMA FIFO.
U64	No	Packet	0 or more	Specifies an unsigned 64-bit channel in the DMA FIFO.
Boolean	No	Packet	0 or more	Specifies a Boolean channel in the DMA FIFO.
FXPI32	No	Packet	0 or more	Specifies a fixed-point signed 32-bit channel in the DMA FIFO. Use this

Tag	Required?	Parent tag	Number of child tags	Description
				data type if the word length you specify in <FXPWL> is less than or equal to 32 bits. The channel occupies the full 32 bits in the packet, but only use the bits corresponding to the word length.
FXPU32	No	Packet	0 or more	Specifies a fixed-point unsigned 32-bit channel in the DMA FIFO. Use this data type if the word length you specify in <FXPWL> is less than or equal to 32 bits. The channel still occupies the full 32 bits in the packet, but only the bits corresponding to the word length are used.
FXPI64	No	Packet	0 or more	Specifies a fixed-point signed 64-bit channel in the DMA FIFO. Use this data type if the word length you specify in <FXPWL> is greater than 32

Tag	Required?	Parent tag	Number of child tags	Description
				bits. The channel still occupies the full 64 bits in the packet, but only the bits corresponding to the word length are used.
FXPU64	No	Packet	0 or more	Specifies a fixed-point unsigned 64-bit channel in the DMA FIFO. Use this data type if the word length you specify in <FXPWL> is greater than 32 bits. The channel still occupies the full 64 bits in the packet, but only the bits corresponding to the word length are used.
PWM	No	Packet	0 or more	Specifies a pulse width modulation (PWM) channel in the DMA FIFO. Consists of two 32-bit numbers, totaling 64 bits. The lower 32 bits represent the low time of the PWM channel. The higher 32 bits represent the high

Tag	Required?	Parent tag	Number of child tags	Description
				time of the PWM channel.
Void	No	Packet	0 or more	Specifies unused bits in the middle of a packet.
Size	Yes	Void	1	Specifies the number of bits to ignore in a packet.
Name	Yes	any data type	1	Defines the name of the channel in System Explorer. This tag must be unique within its category.
Description	No	any data type	1	Specifies the description of the channel in System Explorer.
Category	No	any data type	1	Specifies the full path of the category where the channel should appear. If you do not specify <Category>, the default is Input for channels in the DMA_Read FIFO and Output for channels in the DMA_Write FIFO.
InitialValue	No	any data type	1	Specifies the value of the channel until its value is set. The default value is 0.

Tag	Required?	Parent tag	Number of child tags	Description
				Use <InitialValue> for output channels.
Scale	No	any data type	1	Specifies the range of the scale in engineering units. For PWM and fixed-point data types, the default value is 1. For all other data types, the default value is the full positive range of the data type. For example, the full positive range of the signed 8-bit data type is 127, and the full positive range of the unsigned 8-bit data type is 255.
Offset	No	any data type	1	Specifies the offset of the scale in engineering units. The default value is 1.
Unit	No	any data type	1	Specifies the units of the channel. If you do not specify the units, the channel has no units.
Symbol	No	any data type	1	Defines the symbol of the channel in System Explorer. You can

Tag	Required?	Parent tag	Number of child tags	Description
				select from the following values: ▪ Default ▪ AI ▪ AO ▪ DI ▪ DO ▪ PWM In ▪ PWM Out
FXPWL	No	any fixed-point data type	1	Specifies the fixed-point word length.
FXPIWL	No	any fixed-point data type	1	Specifies the fixed-point integer word length. The default value is 0.
PWMPeriod	No	PWM	1	Specifies the pulse width modulation (PWM) period for output channels. The default value is 100000.
Parameters	No	any data type	1	Specifies the parameters associated with the parent channel.
any data type except for Void	Yes	Parameters	1 or more	Defines a parameter associated with the parent channel and specifies the data type of that parameter. <Parameters>

Tag	Required?	Parent tag	Number of child tags	Description
				accepts the same data types as <Packet>. However, Void is not a valid data type for <Parameters>. If you use the Boolean data type, or the signed or unsigned 8-bit, 16-bit, 32-bit, or 64-bit data types, the associated FPGA VI control must be of the same data type. If you use the fixed-point or PWM data types, the data type of the associated FPGA VI control must correspond as follows: ▪ FXPI32 — I32 ▪ FXPI64 — I64 ▪ FXPU32 — U32 ▪ FXPU64 — U64 ▪ PWM — U64
Name	Yes	any data type except for Void	1	Defines the name of the parameter

Tag	Required?	Parent tag	Number of child tags	Description
				on the channel configuration page in System Explorer.
ControlName	No	any data type except for Void	1	Specifies the name of the associated control in the corresponding FPGA VI. The default is the same as <Name>. The control referenced in the parameter must exist in the FPGA VI.
InitialValue	No	any data type except for Void	1	Specifies the parameter value when System Explorer loads the FPGA configuration file. The default value is 0. You can change parameter values in System Explorer. You cannot change parameter values at run time.
Scale	No	any data type except for Void	1	Specifies the range of the scale in engineering units. For PWM and fixed-point data types, the

Tag	Required?	Parent tag	Number of child tags	Description
				default value is 1. For all other data types, the default value is the full positive range of the data type. For example, the full positive range of the signed 8-bit data type is 127, and the full positive range of the unsigned 8-bit data type is 255.
Offset	No	any data type except for Void	1	Specifies the offset of the scale in engineering units. The default value is 1.
* DMA_XXX denotes both DMA_Read and DMA_Write.				

DMA Scale and Offset

Scale and offset are useful for converting the DMA FIFO value (DFV) to a usable value that corresponds to a real-world measurement.
The following table displays equations that show how scale and offset convert DFV to voltage value (VV) for read channels of different data types, and VV to DFV for write channels of different data types.

Data type	Read or Write	Equation
I8, U8, I16, U16, I32, U32, I64, U64, Boolean	Read	VV = [DFV × (scale ÷ P) + offset]
I8, U8, I16, U16, I32, U32, I64, U64, Boolean	Write	DFV = (VV – offset) × (P ÷ scale)
FXPI32, FXPU32,FXPI64, FXPU64	Read	VV = (FXPV × scale) + offset
FXPI32, FXPU32,FXPI64, FXPU64	Write	FXPV = (VV – offset) ÷ scale

Data type	Read or Write	Equation
PWM	Read	VV = [HT ÷ (LT + HT)] × (scale + offset)
PWM	Write	HT = [(VV – offset) ÷ scale] × PWMperiod LT = PWMperiod – HT

VV
—Represents voltage value.
DFV
—Represents DMA FIFO value.
P
—Represents the positive range of the data type.
FXPV
—Represents the converted fixed-point value.
HT
—Represents high time.
LT
—Represents low time.

Example FPGA Configuration File Structure

An FPGA configuration file needs to be structured correctly in XML to function properly.

The following code displays an example of how to implement the FPGA configuration file XML tags.

```
<?xml version='1.0'>
<?xml-stylesheet type="text/xsl" href='NI VeriStand FPGA DMA.xsl'?>
<FPGADMAChannelData xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "NI VeriStand FPGA DMA.xsd">
      <Version>2.0</Version>
      <Bitfile>FPGADMABitfile.lvbitx</Bitfile>

      <Categories>
          <Category>
              <Name>TopLevel1</Name>
              <Description>Description</Description>
              <Symbol>Default</Symbol>
              <Category>
                  <Name>NestedCategory1</Name>
                  <Description>Category with AI Symbol contained in TopLevel1</
Description>
                  <Symbol>AI</Symbol>
```

```
            </Category>
            <Category>
                <Name>NestedCategory2</Name>
                <Description>Category with DI Symbol contained in TopLevel1</
Description>
                <Symbol>DI</Symbol>
            </Category>
        </Category>
    </Categories>

    <DMA_Read>
        <Packets>5</Packets>
        <Packet/>
        <Packet>
            <I8>
                <Name>Signed8Channel</Name>
                <Description>Description of channel</Description>
                <Category>TopLevel1\NestedCategory1</Category>
                <InitialValue>25</InitialValue>
                <Scale>128</Scale>
                <Offset>5</Offset>
                <Unit>Volts</Unit>
                <Symbol>AI</Symbol>
            </I8>
            <U8>
                <Name>Unsigned8Channel</Name>
                <Description>Description of channel</Description>
                <Category>TopLevel1\NestedCategory2</Category>
                <InitialValue>25</InitialValue>
                <Scale>128</Scale>
                <Offset>5</Offset>
                <Unit>Volts</Unit>
                <Symbol>AI</Symbol>
            </U8>
            <I16>
                <Name>Signed16Channel</Name>
            </I16>
            <Boolean>
                <Name>BooleanChannel</Name>
            </Boolean>
            <Void>
                <Size>7</Size>
            </Void>
        </Packet>
```

```
        <Packet>
            <FXPI32>
                <Name>Fixed Point Channel 1</Name>
                <Scale>2</Scale>
                <Offset>5</Offset>
                <Unit>Volts</Unit>
                <Symbol>AI</Symbol>
                <FXPWL>20</FXPWL>
                <FXPIWL>5</FXPIWL>
            </FXPI32>
        </Packet>
        <Packet>
            <PWM>
                <Name>PWM In 0</Name>
                <Description>PWM input</Description>
                <Category>Input\PWM</Category>
                <Scale>100</Scale>
                <Unit>%</Unit>
                <Symbol>PWM In</Symbol>
                <PWMPeriod>40000</PWMPeriod>
            </PWM>
        </Packet>
        <Packet>
            <U8>
                <Name>Channel with one parameter</Name>
                <Parameters>
                    <I16>
                      <Name>UI Parameter name</Name>
                      <ControlName>Name of Control</ControlName>
                      <InitialValue>25</InitialValue>
                      <Scale>100</Scale>
                      <Offset>0</Offset>
                    </I16>
                </Parameters>
            </U8>
        </Packet>
    </DMA_Read>

    <DMA_Write>
        <Packets>4</Packets>
        <Packet>
            <U8>
                <Name>Unsigned8Out</Name>
                <Category>TopLevel2</Category>
```

```
                    <Symbol>DO</Symbol>
                </U8>
            </Packet>
        </DMA_Write>
</FPGADMAChannelData>
```

Glossary

VeriStand uses unique terms for completing tasks while creating a project.

Use the following table to learn more about a term used in VeriStand.

Alphabetical order	Term	Description
A	**Alarm**	A notification that the value of a particular channel has gone outside a specified range of values. An alarm triggers the execution of a specified procedure.
	Alias	An alternate name for a channel in a system definition file.
B	**Bitfile**	A LabVIEW-generated file that defines the available I/O on the FPGA. A bitfile is a compiled version of an FPGA VI.
	Block diagram	A pictorial description or representation of a program or algorithm. In LabVIEW, the block diagram that consists of executable icons called nodes and wires that carry data between the nodes. The block diagram is the source code for the VI. The block diagram resides in the block diagram window of the VI.
C	**Calculated channel**	A channel that produces a new value based on calculations performed on other channels in the system.
	Calibration	The process of determining the accuracy of an instrument. In a

Alphabetical order	Term	Description
		formal sense, calibration establishes the relationship of an instrument's measurement to the value provided by a standard. When that relationship is known, the instrument may then be adjusted (calibrated) for best accuracy.
	CAN (Controller Area Network)	A serial bus finding increasing use as a device-level network for industrial automation. CAN was developed by Bosch to address the needs of in-vehicle automotive communications.
	Chassis master hardware synchronization device	A hardware device that controls the synchronization of all hardware in a PXI chassis or across multiple PXI chassis. The chassis master hardware synchronization device must be an NI-DAQ device with at least one analog input or output channel, any NI FPGA, or a timing and sync device that has the capability to drive the RTSI 0 line.
	Custom device	A virtual instrument that executes user-defined actions, such as third-party hardware control.
D	**Differential measurement system**	A way to configure a device to read signals, in which you do not need to connect either input to a fixed reference, such as the earth ground or a building ground.
	DLL (Dynamic Link Library)	A compiled model.

Alphabetical order	Term	Description
	DMA (Direct Memory Access)	A method by which data can be transferred to/from computer memory from/to a device or memory on the bus while the processor does something else. DMA is the fastest method of transferring data to/from computer memory.
	Driver	Software that controls a specific hardware device.
F	**FIBEX** (FIeld Bus EXchange)	A vendor-independent exchange format for embedded network data. It is an XML-based text format. For NI-XNET, NI adopted the ASAM FIBEX standard as a database storage format.
	FIFO (First-In-First-Out memory buffer)	The first data stored is the first data sent to the acceptor.
	FIFO sink	The output of a FIFO. You can use the VeriStand Custom Device APIs to set the buffer size at the source and sink of the FIFOs that an asynchronous custom device uses to share data with the real-time engine.
	FIFO source	The input of a FIFO. You can use the VeriStand Custom Device APIs to set the buffer size at the source and sink of the FIFOs that an asynchronous custom device uses to share data with the real-time engine.
	FlexRay	A new, deterministic, fault-tolerant, and high-speed bus system developed in conjunction with automobile manufacturers and leading suppliers.

Alphabetical order	Term	Description
	FPGA (Field-Programmable Gate Array)	A semi-conductor device that contains a large quantity of gates (logic devices), which are not interconnected, and whose function is determined by a wiring list, which is downloaded to the FPGA. The wiring list determines how the gates are interconnected, and this interconnection is performed dynamically by turning semiconductor switches on or off to enable the different connections.
	FPGA configuration file	An XML-based file that specifies the content of DMA FIFOs.
	FPGA VI	A configuration that is downloaded to the FPGA and that determines the functionality of the hardware.
	Frames	Messages sent across an embedded network. Frames are sorted into clusters within an NI-XNET database.
H	**HIL** (Hardware-In-the-Loop)	A simulation configuration in which you test a controller implementation with a simulated system.
	Host computer	The computer that runs the VeriStand Gateway and hosts the screen file.
I	**Interface**	The interface represents a single CAN, FlexRay, or LIN connector on an NI hardware device. Within NI-XNET, the interface is the software object used to communicate with external hardware described in the database.

Alphabetical order	Term	Description
L	**LabVIEW**	A graphical programming language.
	LIN (Local Interconnect Network)	A standard for low-cost, low-end multiplexed communication in automotive networks. LIN provides cost-efficient communication in applications where the bandwidth and versatility of CAN are not required.
M	**Mapping**	A connection between two channels.
	MAX (Measurement & Automation Explorer)	Provides a centralized location for configuration of NI hardware products. MAX also provides many useful tools for interaction with hardware.
	MIO (Multifunction I/O)	A DAQ module that designates a family of data acquisition products that have multiple analog input channels, digital I/O channels, timing, and optionally, analog output channels. An MIO product can be considered a miniature mixed signal tester, due to its broad range of signal types and flexibility. Also known as multifunction DAQ.
N	**NRSE** (Non-Referenced Single-Ended mode)	All measurements are made with respect to a common (NRSE) measurement system reference, but the voltage at this reference can vary with respect to the measurement system ground.
O	**Offline**	A simulation configuration in which you use software to simulate the controller and the

Alphabetical order	Term	Description
		system you want to control. No hardware is involved in an offline simulation.
P	**PCI** (Peripheral Component Interconnect)	An industry-standard, high-speed databus.
	Phar Lap ETS	A real-time operating system designed optimized for devices based on the Intel x86 architecture.
	Port	In regard to NI-XNET, port refers to the connector on an NI hardware device. The physical connector includes the transceiver cable if applicable.
	Port width	Refers to the number of lines in a port. For example, E Series devices have one port with eight lines; therefore, the port width is eight.
	Procedure	A set of actions that the VeriStand Engine executes.
	Project file	The .nivsprj file that defines high-level settings in an VeriStand project, such as the screen and system definition files to run, the IP address of the VeriStand Gateway, etc.
Q	**Quadrature encoder**	An encoding technique for a rotating device where two tracks of information are placed on the device, with the signals on the tracks offset by 90° from each other. This makes it possible to detect the direction of the motion.
R	**RAM** (Random-Access Memory)	The generic term for the read/write memory that is used in computers. RAM allows bits and

Alphabetical order	Term	Description
		bytes to be written to it as well as read from.
	RCP (Rapid Control Prototype)	A simulation configuration in which you test plant hardware with a software model of the controller.
	RT (Real-Time)	Pertaining to the performance of a computation during the actual time that the related physical process transpires so results of the computation can be used in guiding the physical process.
	Real-time sequence	A program that can deploy to a target with a system definition file and read/write channels defined in the system definition file. Real-time sequences can feature a wide array of programming constructs, including while loops, for loops, variables, and conditional statements. Real-time sequences execute on the target.
	Reflective memory network	A means of sharing data between two independent systems in a deterministic manner. Reflective memory devices are connected together using fiber optic cables. This reflective memory system forms a deterministic network that operates like a dual-ported memory system.
	RSE (Referenced Single-Ended configuration)	All measurements are made with respect to a common reference measurement system or ground. Also called a

Alphabetical order	Term	Description
		grounded measurement system.
	RTSI bus (Real-Time System Integration)	The NI timing bus that interconnects data acquisition devices directly by means of connectors on top of the devices for precise synchronization of functions.
S	**Screen file**	A .nivscreen or .nivsscr file that defines the configuration and settings for the screens and display items you view in the VeriStand Editor or Workspace.
	SCXI (Signal Conditioning eXtensions for Instrumentation)	The NI product line for conditioning low-level signals within an external chassis near sensors so that only high-level signals are sent to DAQ devices in the noisy PC environment.
	Service	A LabVIEW VI that runs on the host computer when VeriStand connects to a target. Services are typically Workspace tools that you want to launch as soon as you connect to a target, or that you want to synchronize with the launch of the Workspace window.
	Single-point	Data acquisition in which the software reads a single point of data from one or more analog input channels and immediately returns the value.
	Stimulus profile	A test executive that can call real-time sequences, open and close VeriStand projects, and perform data-logging and pass/fail analysis. It also connects real-time sequences to system

Alphabetical order	Term	Description
		definition files to bind channel data within the system definition file to variables in the real-time sequence. Stimulus profiles execute on the host computer.
	Stimulus Profile Editor	A development environment you use to create, modify, and execute tests.
	System channel	A channel that monitors the state and condition of various internal aspects of VeriStand.
	System definition file	A .nivssdf file you configure primarily in System Explorer. A system definition file contains the configuration settings of the VeriStand Engine.
T	**Target**	The desktop PC or real-time target on which you run the system definition file and VeriStand Engine.
	TestStand	NI test executive for sequencing and managing automatic test programs.
	Timing and sync device	A virtual instrument that synchronizes more than one chassis.
U	**User channel**	A channel that stores a single value.
V	**VeriStand Engine**	The non-visible execution mechanism that controls the timing of the entire system as well as the communication between the target and the host computer.
	VeriStand Gateway	The non-visible mechanism that creates a TCP/IP communication channel which

Alphabetical order	Term	Description
		facilitates communication with the VeriStand Engine over the network. The VeriStand Gateway receives channel values from the VeriStand Engine and stores these values in a table that can be viewed using the Channel Data Viewer.
	VeriStand LabVIEW Model Generator	A tool that generates a compiled model from a LabVIEW VI or simulation subsystem. This tool is accessible from the **Tools** menu in LabVIEW 2010 or later and generates files of the type .lvmodel or .lvmodelso. **Note** You must install additional software to enable LabVIEW models for targets running a Linux Real-Time OS. For more information about how to use LabVIEW models with Linux, see Creating Models in LabVIEW for Use in VeriStand.
	VI (Virtual Instrument)	A LabVIEW program.
X	**XNET**	A suite of products that provide connectivity to Controller Area Network (CAN), Local Interconnect Network (LIN), and FlexRay networks.
	XNET database	A standardized file, such as CANdb (.dbc) or NI-CAN (.ncd)

Alphabetical order	Term	Description
		for CAN or FIBEX (.xml) for FlexRay that NI-XNET applications use to understand hardware communications in the embedded system. The database contains many object classes, each of which describes a distinct entity in the embedded system.

© 2023 National Instruments Corporation.